Electronic Supplementary Material (ESI) for CrystEngComm.

Supporting Information for:

Data Mining the Cambridge Structural Database for Hydrate-Anhydrate Pairs with SMILES Strings

Jen E. Werner and Jennifer A. Swift

Georgetown University, Department of Chemistry, Washington, DC 20057-1227

Table of Contents for Scripts

```python
#1 Find hydrates with water SMILES strings

from ccdc import io, search
import argparse
import os
import glob

#Defines restrictions for structures that are pulled from the CSD
#There are no limitations on the number of structures to pull, the R-factor of a structure, or the disorder of a structure
#Structures with reported errors, organometallic or polymeric entities, or 2D coordinates are not permitted
class Runner(argparse.ArgumentParser):
    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default='CSD',
            help='input database [CSD]'
        )
        self.add_argument(
            '-o', '--output', default='entries_with_water_smiles_string.gcd',
            help='output file [entries_with_water_smiles_string.gcd]'
        )
        self.add_argument(
           '-m', '--maximum', default=0, type=int,
           help='Maximum number of structures to find [all]'
        )
        self.add_argument(
            '-R', '--r_factor', default=100.0, type=float,
            help='Maximum acceptable R-factor [100.0]'
        )
        self.add_argument(
            '-E', '--errors', default=True, action='store_false',
            help='Whether structures with errors are acceptable [No]'
        )
        self.add_argument(
            '-M', '--organometallic', default=True, action='store_false',
            help='Whether organometallic structures are acceptable [No]'
        )
        self.add_argument(
            '-P', '--polymeric', default=True, action='store_false',
            help='Whether polymeric structures are acceptable [No]'
        )
        self.add_argument(
            '-T', '--two_d', default=True, action='store_false',
            help='Whether 2d structures are acceptable [No]'
        )

        args = self.parse_args()


        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum
        self.settings.max_r_factor = self.args.r_factor
        self.settings.no_errors = self.args.errors
        self.settings.only_organic = self.args.organometallic
        self.settings.not_polymeric = self.args.polymeric
        self.settings.has_3d_coordinates = self.args.two_d
        #Any structures containing the following elements were not pulled from the CSD
        self.settings.must_not_have_elements = ['Be', 'Mg', 'Ca', 'Sr', 'Ba', 'Ra', 'B', 'Si', 'As', 'Se', 'Te', 'At',
                                                'He', 'Ne', 'Ar', 'Kr', 'Xe', 'Rn']

#The entry reader object is used to generate the SMILES string for each structure
    def run(self):
        if self.args.input == 'CSD':
```

```
65              reader = io.EntryReader()
66          else:
67              database = self.args.input
68              reader = io.EntryReader(database, format='identifiers')
69
70          count = 0
71          total = 0
72
73          with io.EntryWriter(self.args.output) as hydrates_writer:
74              with io.EntryWriter("entries_with_no_water_smiles_string.gcd") as writer:
75                  for i, entry in enumerate(reader):
76                      #The number of structures with water SMILES strings found so far is
                        printed for every 10,000 structures analyzed
77                      if i and i % 10000 == 0:
78                          print 'Analysed %d structures from %d so far...' % (count, i)
79
80                      if not self.settings.test(entry):
81                          continue
82                      total += 1
83
84                      try:
85                          molecule = entry.molecule
86                      except RuntimeError:
87                          continue
88
89                      #The SMILES string for water is searched for
90                      #Structures without a water SMILES string but that meet all other
                        search criteria form the OTHER list
91                      if any(component.smiles == 'O' for component in molecule.components):
92                          hydrates_writer.write(molecule)
93                          count += 1
94                      else:
95                          writer.write(molecule)
96
97                  #A final count of structures with water SMILES strings are printed
98                  print 'Found %d hydrates from %d valid structures' % (count, total)
99
100
101 if __name__ == '__main__':
102     # This runs the script
103     r = Runner()
104     r.run()
105
```

```
#2 Find solvates and hydrates with no water SMILES strings

from ccdc import io, search
import argparse
import os
import glob
import re

#Input is no longer the CSD but the output files of the previous script
filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\entries_with_water_smiles_string.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\entries_with_no_water_smiles_string.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='solvated_hydrates.gcd',
            help='output file [solvated_hydrates.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()
        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

    entry_reader1 = io.EntryReader(filepath1, format='identifiers')
    entry_reader2 = io.EntryReader(filepath2, format='identifiers')

    with io.EntryWriter(self.args.output) as solvate_writer:
        with io.EntryWriter("hydrates.gcd") as writer1:
            with io.EntryWriter("solvated_waterless_forms.gcd") as writer2:
                with io.EntryWriter("waterless_forms.gcd") as writer3:

                    count1 = 0
                    total1 = 0
                    #The title of each entry with a water SMILES string is searched for the word "solvate"
                    #Solvated hydrates are separated from non-solvated hydrates based on the presence of "solvate" in the chemical name
                    for a in range(len(entry_reader1)):
                        entry1 = entry_reader1[a]
                        if entry1.chemical_name != None:
                            title1 = entry1.chemical_name
                            if 'solvate' in title1:
                                solvate_writer.write(entry_reader1[a])
                                count1 += 1
                            else:
                                writer1.write(entry_reader1[a])
                        elif entry1[a].identifier == 'KIPWOY' or entry1[a].identifier == 'KIPWUE':
                            solvate_writer.write(entry_reader1[a])

                        total1 += 1
                    print 'Found %d solvated structures from %d hydrate structures' %
```

```
                    (count1, total1)
63
64                  count2 = 0
65                  count3 = 0
66                  total2 = 0
67                  #The title of each entry without a water SMILES string is searched
                    for "hydrate" and "deuterium oxide"
68                  #The formula of each entry without a water SMILES string is
                    searched for "H2 O1" and "D2 O1"
69                  #Once again structures with "solvate" in their chemical name are
                    separated from those without "solvate"
70                  for b in range(len(entry_reader2)):
71                      entry2 = entry_reader2[b]
72                      if entry2.chemical_name != None:
73                          title2 = entry2.chemical_name
74                          pieces2 = entry2.formula.split(',')
75                          if 'deuterium oxide' in title2 and any(x == 'D2 O1' for x
                            in pieces2) and 'solvate' in title2 or 'deuterium oxide' in
                            title2 and any('(D2 O1)' in x for x in pieces2) and
                            'solvate' in title2:
76                              solvate_writer.write(entry_reader2[b])
77                              count3 += 1
78                          elif 'deuterium oxide' in title2 and any(x == 'D2 O1' for x
                            in pieces2) or 'deuterium oxide' in title2 and any('(D2
                            O1)' in x for x in pieces2):
79                              writer1.write(entry_reader2[b])
80                          elif 'hydrate' in title2 and any(x == 'H2 O1' for x in
                            pieces2) and 'solvate' in title2 or 'hydrate' in title2 and
                            any('(H2 O1)' in x for x in pieces2) and 'solvate' in title2:
81                              solvate_writer.write(entry_reader2[b])
82                              count3 += 1
83                          elif 'hydrate' in title2 and any(x == 'H2 O1' for x in
                            pieces2) or 'hydrate' in title2 and any('(H2 O1)' in x for
                            x in pieces2):
84                              writer1.write(entry_reader2[b])
85                          #Structures that are hydrates but are not indicated as such
                            by both "H2 O1" or "D2 O1" in the formula and "hydrate" in
                            the chemical name are accounted for here
86                          #FIFYUQ and QIMROV have "H2 O1" in the formula but no
                            "hydrate" in the chemical name
87                          elif entry_reader2[b].identifier == 'FIFYUQ' or
                            entry2[b]entry_reader2[b].identifier == 'QIMROV':
88                              solvate_writer.write(entry_reader2[b])
89                          #KIJFAN and LOYXAA have "hydrate" in the chemical name but
                            no "H2 O1" or "D2 O1" in the formula
90                          elif entry_reader2[b].identifier == 'KIJFAN' or
                            entry_reader2[b].identifier == 'LOYXAA':
91                              writer1.write(entry_reader2[b])
92                          elif 'solvate' in title2:
93                              writer2.write(entry_reader2[b])
94                              count2 += 1
95                          else:
96
                                writer3.write(entry_reader2[b])

97                      else:
98                          print entry_reader2[a].identifier
99
100                     total2 += 1
101                 print 'Found %d solvated hydrate structures from %d waterless
                    structures' % (count3, total1)
102                 print 'Found %d solvated structures from %d waterless structures' %
                    (count2, total2)
103
104
105 if __name__ == '__main__':
106     r = Runner()
```

```
107         r.run()
```

```
#3 Find forms without smiles strings

from ccdc import io, search
import argparse
import os
import glob
import re

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\solvated_hydrates.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\solvated_waterless_forms.txt"
filepath3 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\hydrates.txt"
filepath4 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\waterless_forms.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='smiles_solvated_hydrates.gcd',
            help='output file [smiles_solvated_hydrates.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')
        entry_reader2 = io.EntryReader(filepath2, format='identifiers')
        entry_reader3 = io.EntryReader(filepath3, format='identifiers')
        entry_reader4 = io.EntryReader(filepath4, format='identifiers')

        total = 0
        count = 0

        with io.EntryWriter(self.args.output) as SMILES_writer:
            with io.EntryWriter("smiles_solvated_waterless_forms.gcd") as writer1:
                with io.EntryWriter("smiles_hydrates.gcd") as writer2:
                    with io.EntryWriter("smiles_waterless_forms.gcd") as writer3:
                        with io.EntryWriter("None_solvated_hydrates.gcd") as writer4:
                            with io.EntryWriter("None_solvated_waterless_forms.gcd") as writer5:
                                with io.EntryWriter("None_hydrates.gcd") as writer6:
                                    with io.EntryWriter("None_waterless_forms.gcd") as writer7:

                                        loop = 0
                                        entry1 = entry_reader1
                                        while loop < 4:
                                            #The entry reader is used to generate an entry SMILES string
                                            #Any SMILES strings that return as None are
```

```
                                        separated as hydrate and anhydrate
                                        structures without SMILES strings
61                                      #None will be returned for both absent
                                        SMILES strings and incomplete SMILES
                                        strings (at least one molecule SMILES
                                        string is present)
62                                      for a in range(len(entry1)):
63                                          if loop == 0:
64                                              if entry1[a].molecule.smiles == None:
65                                                  writer4.write(entry1[a])
66                                              else:
67                                                  SMILES_writer.write(entry1[a])
68                                          if loop == 1:
69                                              if entry1[a].molecule.smiles == None:
70                                                  writer5.write(entry1[a])
71                                              else:
72                                                  writer1.write(entry1[a])
73                                          if loop == 2:
74                                              if entry1[a].molecule.smiles == None:
75                                                  writer6.write(entry1[a])
76                                              else:
77                                                  writer2.write(entry1[a])
78                                          if loop == 3:
79                                              if entry1[a].molecule.smiles == None:
80                                                  writer7.write(entry1[a])
81                                              else:
82                                                  writer3.write(entry1[a])
83                                      loop += 1
84                                      if loop == 1:
85                                          entry1 = entry_reader2
86                                      elif loop == 2:
87                                          entry1 = entry_reader3
88                                      elif loop == 3:
89                                          entry1 = entry_reader4
90
91  if __name__ == '__main__':
92      # This runs the script
93      r = Runner()
94      r.run()
95
```

```python
#4 Find forms with metals

from ccdc import io, search
import argparse
import os
import glob
import re

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\complete_SMILES_hydrates.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\complete_None_hydrates.txt"
filepath3 = "C:\Users\jenwe\Hydrates Manuscript\complete_SMILES_waterless_forms.txt"
filepath4 = "C:\Users\jenwe\Hydrates Manuscript\complete_None_waterless_forms.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='hydrates_SMILES_without_metals.gcd',
            help='output file [hydrates_SMILES_without_metals.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')
        entry_reader2 = io.EntryReader(filepath2, format='identifiers')
        entry_reader3 = io.EntryReader(filepath3, format='identifiers')
        entry_reader4 = io.EntryReader(filepath4, format='identifiers')

        with io.EntryWriter(self.args.output) as new_lists_writer:
            with io.EntryWriter("hydrates_None_without_metals.gcd") as writer1:
                with io.EntryWriter("waterless_forms_SMILES_without_metals.gcd") as
                writer2:
                    with io.EntryWriter("waterless_forms_None_without_metals.gcd") as
                    writer3:

                        loop = 0
                        entry1 = entry_reader1
                        while loop < 4:
                            #The formula of each structure was split into the
                            corresponding molecules
                            for a in range(len(entry1)):
                                if entry1[a].identifier != "EQOPAD01" and
                                entry1[a].identifier != "EQOPAD02":
                                    molecules = entry1[a].formula.split(',')
                                    found = 0
                                    #The formula of each molecule was split into the
                                    corresponding elements
                                    #Stoichiometry values greater than one indicated by
                                    parenthese were removed (ex. 3(C1 H4 O1) to C1 H4 O1)
                                    for b in range(len(molecules)):
                                        if '(' and ')' in molecules[b]:
                                            start = molecules[b].index('(')
```

```
                                        end = molecules[b].index(')')
                                        molecules[b] = molecules[b][start+1:end]
                                    elements = molecules[b].split(' ')
                                    if found == 0:
                                        for c in range(len(elements)):
                                            if found == 0:
                                                #Any charge components of a
                                                molecule were omitted from being
                                                screened
                                                if '+' not in elements[c] and '-'
                                                not in elements[c]:
                                                    string1 = elements[c]
                                                    #Each element was changed to
                                                    just the letters (ex. Na1 to Na)
                                                    string1 = re.sub(r'[0-9]+', '',
                                                    string1)
                                                    #Any instance of resulting
                                                    elements that are not C, H, D,
                                                    O, N, S, Se, P, Cl, Br, I, or F
                                                    led to that structure being
                                                    placed in a new list of
                                                    structures with metals
                                                    #Structures in this new list
                                                    are later removed from the
                                                    lists of hydrate and anhydrous
                                                    form structures
                                                    if string1 != 'C' and string1
                                                    != 'H' and string1 != 'D' and
                                                    string1 != 'O' and string1 !=
                                                    'N' and string1 != 'S' and
                                                    string1 != 'P' and string1 !=
                                                    'Cl' and string1 != 'Br' and
                                                    string1 != 'I' and string1 !=
                                                    'F':
                                                        found += 1
                                if found == 0:
                                    if loop == 0:
                                        new_lists_writer.write(entry1[a])
                                    if loop == 1:
                                        writer1.write(entry1[a])
                                    if loop == 2:
                                        writer2.write(entry1[a])
                                    if loop == 3:
                                        writer3.write(entry1[a])

                        loop += 1
                        if loop == 1:
                            entry1 = entry_reader2
                        if loop == 2:
                            entry1 = entry_reader3
                        if loop == 3:
                            entry1 = entry_reader4



if __name__ == '__main__':
    r = Runner()
    r.run()
```

```
#5 Check for SMILES of solvent molecule in solvates

from ccdc import io, search
import argparse
import os
import glob
import re

#This script finds missing solvent SMILES strings from entries tagged in the chemical name as a "solvate"
filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\smiles_solvated_hydrates.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\smiles_solvated_waterless_forms.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='solvates.gcd',
            help='output file [solvates.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')
        entry_reader2 = io.EntryReader(filepath2, format='identifiers')

        total = 0
        count = 0

        with io.EntryWriter(self.args.output) as smiles_check_writer:

            #The following dictionaries contain solvent names for molecules who are missing in at least one entry SMILES string
            solvents_dictionary = {hash('but-2-ene'):'CC=CC',
            hash('1,1,2-dichloroethane'):'ClCC(Cl)Cl',
            hash('4-aminopyridine'):'Nc1ccncc1', hash('N-methylurea'):'CNC(N)=O',
            hash('2-methylphenol'):'Cc1ccccc1O', hash('tetraglyme'):'COCCOCCOCCOCCOC',
            hash('isophorone'):'CC1=CC(=O)CC(C)(C)C1',
            hash('L-carvone'):'CC(=C)C1CC=C(C)C(=O)C1',
            hash('2-cyanopyridine'):'N#Cc1ccccn1', hash('biuret'):'NC(=O)NC(N)=O',
            hash('styrene'):'C=Cc1ccccc1', hash('carbondisulfide'):'S=C=S',
            hash('tricyanomethanide'):'N#C[C-](C#N)C#N',
            hash('methyl-cyclohexane'):'CC1CCCCC1',
            hash('pyridine-3-carboxamide'):'NC(=O)c1cccnc1',
            hash('3,7-dimethyl-3,7-dihydro-1H-purine-2,6-dione'):'CN1C=NC2=C1C(=O)NC(=O)N2C', hash('nitroethane'):'CCN(=O)=O',
            hash('dimethylaminobenzaldehyde'):'CN(C)c1ccc(C=O)cc1',
            hash('3-(1H-pyrrol-1-yl)propanenitrile'):'C1=CN(C=C1)CCC#N',
            hash('trifluoromethanol'):'OC(F)(F)F', hash('methoxyethane'):'CCOC',
            hash('methylanisole'):'COc1ccccc1C', hash('xylene'):'Cc1ccc(C)cc1',
```

```
hash('petrol'):'CCCCCC', hash('n-alkane'):'CCCCC',
hash('dichloromethanal'):'ClC(Cl)=O',
hash('1-fluoro-4-methylbenzene'):'Cc1ccc(F)cc1',
hash('1,2,3-trimethoxybenzene'):'COc1cccc(OC)c1OC',
hash('1,3,5-trimethoxybenzene'):'COc1cc(OC)cc(OC)c1',
hash('chloroform'):'C1C(Cl)Cl', hash('methanol'):'CO',
hash('hexane'):'CCCCCC', hash('ethanol'):'CCO',
hash('acetonitrile'):'CC#N', hash('toluene'):'Cc1ccccc1',
hash('tetrahydrofuran'):'C1CCOC1', hash('acetone'):'CC(C)=O',
hash('dichloromethane'):'ClCCl', hash('1,2-dichloroethane'):'ClCCCl',
hash('dimethylsulfoxide'):'CS(C)=O', hash('1,4-dioxane'):'C1COCCO1',
hash('1,2-dimethoxyethane'):'COCCOC', hash('o-xylene'):'Cc1ccccc1c',
hash('propanol'):'CCO', hash('dioxane'):'C1COCCO1',
hash('n-pentane'):'CCCCC', hash('4-cyanopyridine'):'N#Cc1ccncc1',
hash("4,4'-bipyridine"):'c1cc(ccn1)c1ccncc1', hash('isopropanol'):'CC(C)O',
hash('1-butanol'):'CCCCO', hash('nitromethane'):'CN(=O)=O',
hash('n-hexane'):'CCCCCC', hash('dimethylformamide'):'CN(C)C=O',
hash('N,N-dimethylformamide'):'CN(C)C=O', hash('cyclohexane'):'C1CCCCC1',
hash('hydroquinone'):'Oc1ccc(O)cc1', hash('benzene'):'c1ccccc1',
hash('octane'):'CCCCCCCC', hash('cyclohexanone'):'O=C1CCCCC1',
hash('isobutanol'):'CC(C)CO', hash('p-xylene'):'Cc1ccc(C)cc1',
hash('pentane'):'CCCCC', hash('pyridine'):'c1ccncc1',
hash('t-butanol'):'CC(C)(C)O', hash('m-xylene'):'Cc1cccc(C)c1',
hash('phenol'):'Oc1ccccc1', hash('o-cresol'):'Cc1ccccc1O',
hash('p-cresol'):'Cc1ccc(O)cc1',
hash('2,4,6-trimethylpyridine'):'Cc1cc(C)nc(C)c1',
hash('1-methylpyrrolidin-2-one'):'CN1CCCC1=O',
hash('1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8-hexadecafluoro-1,8-diiodooctane'):'FC(F
)(I)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)I',
hash('trifluoroethanol'):'OCC(F)(F)F', hash('dichloroethane'):'ClCCCl',
hash('1,1,2,2-tetrachloroethane'):'ClC(Cl)C(Cl)Cl',
hash('2-propanol'):'CC(C)O', hash('chlorobenzene'):'Clc1ccccc1',
hash('naphthalene'):'c1ccc2ccccc2c1', hash('propan-2-ol'):'CC(C)O',
hash('benzonitrile'):'N#Cc1ccccc1', hash('nitrobenzene'):'O=N(=O)c1ccccc1',
hash('dodecane'):'CCCCCCCCCCCC',
hash('1,2-dichlorobenzene'):'Clc1ccccc1Cl', hash('n-heptane'):'CCCCCCC',
hash('methanesulfonate'):'CS(=O)(=O)[O-]',
hash('3-methylbutan-1-ol'):'CC(C)CCO',
hash('1,1,2-trichloroethane'):'ClCC(Cl)Cl', hash('n-butane'):'CCCC',
hash('dichlorine'):'ClCl', hash('o-dichlorobenzene'):'Clc1ccccc1Cl',
hash('N,N-dimethylacetamide'):'CN(C)C(C)=O',
hash('4-methylaniline'):'Cc1ccc(N)cc1', hash('morpholine'):'C1COCCN1',
hash('1,3-dimethylimidazolidin-2-one'):'CN1CCN(C)C1=O)',
hash('1,3-dimethoxybenzene'):'COc1cccc(OC)c1', hash('n-propanol'):'CCCO',
hash('mesitylene'):'Cc1cc(C)cc(C)c1',
hash('acetylacetone'):'CC(=O)CC(C)=O', hash('diethylether'):'CCOCC',
hash('hydrazine'):'NN', hash('dimethylamine'):'CNC',
hash('1,2-difluorobenzene'):'Fc1ccccc1F', hash('cycloheptane'):'C1CCCCCC1',
hash('formamide'):'NC=O', hash('2-butanol'):'CCC(C)O',
hash('ethylacetate'):'CCOC(C)=O', hash('butan-1-ol'):'CCCCO',
hash('bromomethane'):'CBr', hash('piperazine'):'C1CNCCN1',
hash('n-nonane'):'CCCCCCCCC',
hash('1,2,4-trimethoxybenzene'):'COc1ccc(OC)c(OC)c1',
hash('1,2-diaminoethane'):'NCCN', hash('n-butanol'):'CCCCO',
hash('N-methylformamide'):'CNC=O', hash('pyrrolidine'):'C1CCNC1',
hash('actone'):'CC(C)=O', hash('n-octane'):'CCCCCCCC',
hash('(S)-2-methyl-1-butanol'):'CCC(C)CO',
hash('N,N-diethylformamide'):'CCN(CC)C=O',
hash('tetracyanoethylene'):'N#CC(C#N)=C(C#N)C#N',
hash('fluorobenzene'):'Fc1ccccc1', hash('heptane'):'CCCCCCC',
hash('tetrachloroethane'):'ClC(Cl)C(Cl)Cl',
hash('methylcyclohexane'):'CC1CCCCC1',
hash('tetrachloromethane'):'ClC(Cl)(Cl)Cl', hash('1-propanol'):'CCCO',
hash('acetamide'):'CC(N)=O', hash('propane'):'CCC',
hash('diethylamine'):'CCNCC',
hash('hexafluorophosphate'):'F[P-](F)(F)(F)(F)F',
hash('hexafluoropropan-2-ol'):'OC(C(F)(F)F)C(F)(F)F',
```

```
hash('1-chloro-4-methylbenzene'):'Cc1ccc(Cl)cc1',
hash('acetaldehyde'):'CC=O', hash('propiononitrile'):'CCC#N',
hash('bromobenzene'):'Brc1ccccc1'}
deuterated_dictionary = {hash('dideutero-dichloromethane'):'ClCCl',
hash('perdeutero-toluene'):'Cc1ccccc1', hash('octadeuterofuran'):'C1CCOC1',
hash('deutero-ethanol'):'CCO', hash('deuterochloroform'):'C1C(Cl)Cl',
hash('deutero-chloroform'):'C1C(Cl)Cl', hash('deuterobenzene'):'c1ccccc1',
hash('deuteromethanol'):'CO', hash('hexadeutero-benzene'):'c1ccccc1',
hash('perdeuteroacetone'):'CC(C)=O',
hash('dideuterodichloromethane'):'ClCCl', hash('perdeuteromethanol'):'CO',
hash('hexadeuterobenzene'):'c1ccccc1'}
acids_dictionary = {hash('trifluoroacetic'):'OC(=O)C(F)(F)F',
hash('acetic'):'CC(O)=O', hash('formic'):'OC=O',
hash('sulfuric'):'OS(O)(=O)=O', hash('maleic'):'OC(=O)C=CC(O)=O',
hash('benzoic'):'OC(=O)c1ccccc1', hash('succinic'):'OC(=O)CCC(O)=O'}
ethers_dictionary = {'t-butyl methyl ether':'COC(C)(C)C',
hash('di-isopropyl'):'CC(C)OC(C)C', hash('dipropyl'):'CCCOCCC',
hash('diethyl'):'CCOCC', hash('diisopropyl'):'CC(C)OC(C)C',
hash('isopropyl'):'CC(C)OC(C)C', hash('t-butyl'):'COC(C)(C)C',
hash('acetic'):'CCOC(C)=O'}
acetates_dictionary = {hash('ethyl'):'CCOC(C)=O',
hash('amyl'):'CCCCCOC(C)=O', hash('methyl'):'COC(C)=O',
hash('butyl'):'CCCCOC(C)=O', hash('isobutyl'):'CC(C)COC(C)=O'}
glycol_dictionary = {hash('ethylene'):'OCCO'}
oxalates_dictionary = {hash('dipropyl'):'CCCOC(=O)C(=O)OCCC'}
acid_neutral_dictionary = {hash('oxalic'):'OC(=O)C(O)=O',
hash('nitric'):'ON(=O)=O', hash('trans-but-2-enedioic'):'OC(=O)C=CC(O)=O',
hash('but-2-enedioic'):'OC(=O)C=CC(O)=O',
hash('fumaric'):'OC(=O)C=CC(O)=O', hash('phosphonic'):'OP(O)=O',
hash('perchloric'):'O[Cl](=O)(=O)=O', hash('phosphoric'):'OP(O)(O)=O',
hash('pyridine-2,6-dicarboxylic'):'OC(=O)c1cccc(n1)C(O)=O',
hash('benzene-1,2,4,5-tetracarboxylic'):'OC(=O)c1cc(C(O)=O)c(cc1C(O)=O)C(O)=O
', hash('trifluoroacetatic'):'OC(=O)C(F)(F)F'}
acid_charged_dictionary = {hash('oxalic'):'OC(=O)C(=O)[O-]',
hash('nitric'):'O=N(=O)[O-]',
hash('trans-but-2-enedioic'):'OC(=O)C=CC(=O)[O-]',
hash('but-2-enedioic'):'OC(=O)C=CC(=O)[O-]',
hash('fumaric'):'OC(=O)C=CC(=O)[O-]', hash('phosphonic'):'OP(=O)[O-]',
hash('perchloric'):'O=[Cl](=O)(=O)[O-]',
hash('phosphoric'):'OP(O)(=O)[O-]',
hash('pyridine-2,6-dicarboxylic'):'OC(=O)c1cccc(n1)C(=O)[O-]',
hash('benzene-1,2,4,5-tetracarboxylic'):'OC(=O)c1cc(C(=O)[O-])c(cc1C(=O)[O-])
C(O)=O', hash('trifluoroacetatic'):'FC(F)(F)C(=O)[O-]'}

partial_name = [hash('ketone'), hash('propanoate'), hash('carbonate'),
hash('formate'), hash('iodine'), hash('tetrachloride'), hash('sulfoxide'),
hash('dichloride'), hash('chloride'), hash('benzoate'), hash('disulfide'),
hash('oxide')]
full_solvate_name_dictionary = {'methyl isobutyl ketone':'CC(C)CC(C)=O',
'deuterium oxide':'O', 'ethyl propanoate':'CCOC(=O)CC', 'dimethyl
carbonate':'COC(=O)OC', 'ethyl formate':'CCOC=O', '12]cycloparaphenylene
iodine':'II', 'carbon tetrachloride':'ClC(Cl)(Cl)Cl', 'dimethyl
sulfoxide':'CS(C)=O', 'methylene dichloride':'ClCCl', 'methylene
chloride':'ClCCl', 'ethyl benzoate':'CCOC(=O)c1ccccc1', 'carbon
disulfide':'S=C=S'}

#These lists keep track of the solvent names where all the evaluated
entries contain the corresponding solvent SMILES string
complete_smiles_partial_name = [hash('ether'), hash('peroxide'),
hash('oxide'), hash('acid'), hash('glycol'), hash('oxalate'),
hash('oxide)'), hash('sulfoxide'), hash('alcohol'), hash('anhydride'),
hash('acetate'), hash('disulfide'), hash('dioxide'), hash('triamide'),
hash('fluoride'), hash('dulfoxide'), hash('sulfide'),
hash('hydroperoxide)'), hash('propionate'), hash('acrylate'),
hash('ather'), hash('chloride'), hash('benzoate'), hash('triacetate'),
hash('propanoate'), hash('acid)'), hash("2,2'-oxydiacetate"),
hash('ester'), hash('2-methylbutanoate')]
```

```
62  complete_smiles_full_solvate_name_dictionary = {'trideutero-oxonium
    deuterium oxide':'[D+]([OH2])[OH2]', 'hydrogen peroxide':'OO', 'propionic
    acid':'CCC(O)=O', '(S)-propylene glycol' :'CC(O)CO', 'acrylic
    acid':'OC(=O)C=C', 'diethyl oxalate':'CCOC(=O)C(=O)OCC', 'dimethyl
    oxalate':'COC(=O)C(=O)OC', 'di(deuterium oxide)':'O', 'butyric
    acid':'CCCC(O)=O', 'dimethyl ether':'COC', 'perdeutero-dimethyl
    sulfoxide':'CS(C)=O', 'picric acid':'Oc1c(cc(cc1N(=O)=O)N(=O)=O)N(=O)=O',
    'propanoic acid':'CCC(O)=O', '(trifluoromethyl)benzyl
    alcohol':'OC(c1ccccc1)C(F)(F)F', 't-butyl alcohol':'CC(C)(C)O', 'acetic
    anhydride':'CC(=O)OC(C)=O', 't-butyl acetate':'CC(=O)OC(C)(C)C',
    'chloroacetic acid':'OC(=O)CCl', 'isoamyl acetate':'CC(C)CCOC(C)=O',
    't-butylcarboxylic acid':'CC(C)(C)C(O)=O', 'benzyl alcohol':'OCc1ccccc1',
    'ethanedioic acid':'OC(=O)C(O)=O', 'diphenyl ether':'O(c1ccccc1)c1ccccc1',
    "(2,7-diethylbenzo[1,2-b:6,5-b']bisthiene-4,5-diyl)C60fullerene
    disulfide":'S=C=S', 'n-butyl acetate':'CCCCOC(C)=O', 'carbon
    dioxide':'O=C=O',
    '2-amino-3-(2-hydroxy-2-phenylethyl)pyrazolo[1,5-a]pyrimidin-5(4H)-one
    acetate':'CC(O)=O', 'pentanedioic acid':'OC(=O)CCCC(O)=O', 'dimethy
    sulfoxide':'CS(C)=O', 'isopropyl alcohol':'CC(C)O', 'isoamyl
    alcohol':'CC(C)CCO', 'bis(trideuteromethyl) sulfoxide':'CS(C)=O',
    'trifluoromethanesulfonic acid':'OS(=O)(=O)C(F)(F)F',
    '2-amino-6-(4-chlorophenyl)-5-((3-chlorophenyl)diazenyl)nicotinate
    acetate':'CC(O)=O', 'pivalic acid':'CC(C)(C)C(O)=O', '2,6-dihydroxybenzoic
    acid':'OC(=O)c1c(O)cccc1O', '1-hydroxynaphthalene-2-carboxylic
    acid':'OC(=O)c1ccc2ccccc2c1O', 'decanedioic acid':'OC(=O)CCCCCCCCC(O)=O',
    'hydroxy(phenyl)acetic acid':'OC(C(O)=O)c1ccccc1', 'sulfur
    dioxide':'O=S=O', 'salicylic acid':'OC(=O)c1ccccc1O', 'butanoic
    acid':'CCCC(O)=O', 'phthalic acid':'OC(=O)c1ccccc1C(O)=O', '2-phenylacetic
    acid':'OC(=O)Cc1ccccc1', 'hexamethylphosphoric acid
    triamide':'CN(C)P(=O)(N(C)C)N(C)C', '2,5-dihydroxybenzoic
    acid':'OC(=O)c1cc(O)ccc1O', 'squaric acid':'OC1=C(O)C(=O)C1=O',
    'hyponitrous acid':'ON=NO', '2,2,2-trifluoroacetic acid':'OC(=O)C(F)(F)F',
    'trichloroacetic acid':'OC(=O)C(Cl)(Cl)Cl', 'sulfamic
    acid':'[NH3+]S(=O)(=O)[O-]', 't-butyl(methyl) ether':'COC(C)(C)C',
    '4-aminobenzoic acid':'Nc1ccc(cc1)C(O)=O', 'hydrogen fluoride':'F',
    'methanoic acid':'OC=O', '2,3,5-triiodobenzoic
    acid':'OC(=O)c1cc(I)cc(I)c1I', 'phenyl sulfoxide':'CS(=O)c1ccccc1',
    '3,4-dimethylphenyl sulfoxide':'Cc1ccc(cc1C)S(C)=O', 'o-tolyl
    sulfoxide':'Cc1ccccc1S(C)=O', 'tartaric acid':'OC(C(O)C(O)=O)C(O)=O',
    'dimethyl dulfoxide':'CS(C)=O', 'propylene sulfide':'CC1CS1', 'isopropyl
    acetate':'CC(C)OC(C)=O', 'carbonic acid':'OC(O)=O', '4-methoxybenzoic
    acid':'COc1ccc(cc1)C(O)=O', '1,4-phenylenediacetic
    acid':'OC(=O)Cc1ccc(CC(O)=O)cc1', '(1,4-dioxan-2-yl
    hydroperoxide)':'OOC1COCCO1', 'triethylamine oxide':'CCN(=O)(CC)CC',
    "diethyl 2,2'-oxydiacetate":'CCOC(=O)COCC(=O)OCC', 'hexadeuterodimethyl
    sulfoxide':'CS(C)=O', 'methyl propionate':'CCC(=O)OC', 'methyl
    acrylate':'COC(=O)C=C', 'bis(2-methoxyethyl) ether':'COCCOCCOC',
    'tris(pyrrolidino)phosphine oxide':'O=P(N1CCCC1)(N1CCCC1)N1CCCC1',
    'pyromellitic anhydride':'O=C1OC(=O)c2cc3C(=O)OC(=O)c3cc12', 'diethyl
    ather':'CCOCC', 't-butyl chloride':'CC(C)(C)Cl', 'p-nitrobenzoic
    acid':'OC(=O)c1ccc(cc1)N(=O)=O', 'methyl benzoate':'COC(=O)c1ccccc1',
    't-butylbenzoic acid':'CC(C)(C)c1ccc(cc1)C(O)=O', 'aminosulfonic
    acid':'[NH3+]S(=O)(=O)[O-]', 'ethanoic acid':'CC(O)=O', 'perdeuterodimethyl
    sulfoxide':'CS(C)=O', 'propane-1,2,3-triyl
    triacetate':'CC(=O)OCC(COC(C)=O)OC(C)=O', 'hydrochloric acid':'Cl',
    'isobutyric acid':'CC(C)C(O)=O', 'methyl propanoate':'CCC(=O)OC', 'n-butyl
    alcohol':'CCCCO', '2-hydroxybenzoic acid':'OC(=O)c1ccccc1O',
    'orthophosphoric acid':'OP(O)(O)=O', 'hexanedioic acid':'OC(=O)CCCCC(O)=O',
    'pimelic acid':'OC(=O)CCCCCC(O)=O', 'octanedioic
    acid':'OC(=O)CCCCCCC(O)=O', 'bis(trichloroacetic
    acid)':'OC(=O)C(Cl)(Cl)Cl', 'propylene glycol':'CC(O)CO', 'di-imino-oxalic
    acid diethyl ester':'CCOC(=N)C(=N)OCC', 'ethyl
    2-methylbutanoate':'CCOC(=O)C(C)CC'}
63  complete_smiles_dictionary =
    {hash('1,1,2,2,3,3,4,4,5,5,6,6-dodecafluoro-1,6-diiodohexane'):'FC(F)(I)C(F)(
    F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)I',
    hash('2,2,2-trifluoroethanol'):'OCC(F)(F)F',
```

```
hash('1,1,2,2-tetrachloro-1,2-dideuteroethane'):'ClC(Cl)C(Cl)Cl',
hash("2,2'-bipyrimidine"):'c1cnc(nc1)c1ncccn1',
hash('2-methyl-2,4-pentanediol'):'CC(O)CC(C)(C)O',
hash('1,1,1,3,3,3-hexafluoropropan-2-ol'):'OC(C(F)(F)F)C(F)(F)F',
hash('pyrazine'):'c1cnccn1', hash('2-methyl-3-butanol'):'CC(C)C(C)O',
hash('2-pentanol'):'CCCC(C)O', hash('2-hexanol'):'CCCCC(C)O',
hash('2-heptanol'):'CCCCCC(C)O', hash('ethane-1,2-diol'):'OCCO',
hash('N,N-diethylethanamine'):'CCN(CC)CC',
hash('4-ethylpyridine'):'CCc1ccncc1', hash('2-methylpyridine'):'Cc1ccccn1',
hash('acidN,N-dimethylformamide'):'CN(C)C=O',
hash('dimethylsulfone'):'CS(C)(=O)=O',
hash('2-methylpentane-2,4-diol'):'CC(O)CC(C)(C)O',
hash('7-methyl-2-oxo-3,6,8-trioxadecane'):'CCOC(C)OCCOC(C)=O',
hash('4-nitrophenol'):'Oc1ccc(cc1)N(=O)=O',
hash('1,10-phenanthroline'):'c1cnc2c(c1)ccc1cccnc21',
hash('1-methyl-2-pyrrolidinone'):'CN1CCCC1=O',
hash('(S)-butane-1,2-diol'):'CCC(O)CO',
hash('(S)-propane-1,2-diol'):'CC(O)CO',
hash('perdeuterodimethylsulfoxide'):'CS(C)=O',
hash('(S)-butan-2-ol'):'CCC(C)O', hash('2-methylpropanol'):'CC(C)(C)O',
hash('trans-1,4-cyclohexanediol'):'OC1CCC(O)CC1',
hash('quinoline'):'c1ccc2ncccc2c1', hash('1-undecanol'):'CCCCCCCCCCCO',
hash('acetylene'):'C#C', hash('1-naphthol'):'Oc1cccc2ccccc12',
hash('2-naphthol'):'Oc1ccc2ccccc2c1',
hash('2-methoxy-2-methylpropane'):'COC(C)(C)C',
hash('hexane-1,6-diol'):'OCCCCCCO',
hash('1,3,5-trimethylbenzene'):'Cc1cc(C)cc(C)c1',
hash("N,N'-dimethylformamide"):'CN(C)C=O',
hash("N,N'-dimethylacetamide"):'CN(C)C(C)=O',
hash('2-ethylhexanol'):'CCCCC(CC)CO', hash('octan-1-ol'):'CCCCCCCCO',
hash('4-amino-1,2,4-triazole'):'NN1C=NN=C1',
hash('perdeutero-dimethylsulfoxide'):'CS(C)=O', hash('methanamine'):'CN',
hash('chloroacetonitrile'):'ClCC#N', hash('malonate'):'CCOC(=O)CC(=O)OCC',
hash('succinate'):'CCOC(=O)CCC(=O)OCC', hash('n-butan-1-ol'):'CCCCO',
hash('ethane-1,2-diol'):'OCCO', hash('4-methylpiperidine'):'CC1CCNCC1',
hash('2-butanone'):'CCC(C)=O', hash('ethyleneglycol'):'OCCO',
hash('diiodine'):'II', hash('acetotnitrile'):'CC#N',
hash('nonane'):'CCCCCCCCC',
hash('bis(phthalate)-N,N-dimethylformamide'):'CN(C)C=O',
hash('butylamine'):'CCCCN', hash('n-decanol'):'CCCCCCCCCCO',
hash('n-pentanol'):'CCCCCO', hash('m-cresol'):'Cc1cccc(O)c1',
hash('pentan-2-ol'):'CCCC(C)O',
hash('1-(4-fluorophenyl)ethanol'):'CC(O)c1ccc(F)cc1',
hash('2-ethoxyethanol'):'CCOCCO', hash('2-propoxyethanol'):'CCCOCCO',
hash('hexanol'):'CCCCCCO', hash('pentanol'):'CCCCCO',
hash('1,6-diamino-hexane'):'NCCCCCCN', hash('phenylmethanol'):'OCc1ccccc1',
hash("5,5'-bipyrimidine"):'c1ncc(cn1)c1cncnc1',
hash('6-nitrobenzimidazole'):'O=N(=O)c1ccc2NC=Nc2c1',
hash('bromo(trichloro)methane'):'ClC(Cl)(Cl)Br',
hash('3,4-Diacetyl-15,21-dioxatetracyclo[23.4.0.02,7.06,11]nonacosa-1(29),2,4
,6,8,10,25,27-octaene-14,22-dione'):'CC(=O)c1cc2c3CCC(=O)OCCCCCOC(=O)CCc4cccc
c4c(c2ccc3)c1C(C)=O', hash('butanol-1,2-diol'):'CCC(O)CO',
hash('ethylbenzene'):'CCc1ccccc1', hash('1-pentanol'):'CCCCCO',
hash('undecanol'):'CCCCCCCCCCCO', hash('iodomethane'):'CI',
hash('1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9,10,10-icosafluoro-1,10-diiododecane
'):'FC(F)(I)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)I'
,
hash('1,1,2,2,3,3,4,4-octafluoro-1,4-diiodobutane'):'FC(F)(I)C(F)(F)C(F)(F)C(
F)(F)I', hash('1,2-fluorobenzene'):'Fc1ccccc1F',
hash('ethanedihydrazide'):'NNC(=O)C(=O)NN', hash('bromoform'):'BrC(Br)Br',
hash('p-bromophenol'):'Oc1ccc(Br)cc1',
hash('N-methylacetamide'):'CNC(C)=O', hash('n-propylamine'):'CCCN',
hash('4-methylbenzenesulfonamide'):'Cc1ccc(cc1)S(N)(=O)=O',
hash('methoxyethanol'):'COCCO', hash('tetrabromide'):'BrC(Br)(Br)Br',
hash('N-methylpyrrolidone'):'CN1CCCC1=O', hash('ethoxyethane'):'CCOCC',
hash('pentan-3-one'):'CCC(=O)CC', hash('ethane'):'CC',
hash('methylcyclopentane'):'CC1CCCC1',
```

```
hash('hexafluorobenzene'):'Fc1c(F)c(F)c(F)c(F)c1F',
hash('1,4-butanediol'):'OCCCCO',
hash('1-chloro-2-methylbenzene'):'Cc1ccccc1Cl',
hash('dimethyl-formamide'):'CN(C)C=O', hash('butan-2-one'):'CCC(C)=O',
hash('aniline'):'Nc1ccccc1', hash('1,4-dichlorobutane'):'ClCCCCCl',
hash('1,4-difluorobutane'):'FCCCCF',
hash('1,2,4-trichlorobenzene'):'Clc1ccc(Cl)c(Cl)c1',
hash('iodoethane'):'CCI', hash('hexadeutero-dimethylsulfoxide'):'CS(C)=O',
hash('hydroxyacetonitrile'):'OCC#N', hash('butan-2-amine'):'CCC(C)N',
hash('(S)-butan-2-amine'):'CCC(C)N', hash('(R)-butan-2-amine'):'CCC(C)N',
hash('dibromomethane'):'BrCBr', hash('1,2-dibromobenzene'):'Brc1ccccc1Br',
hash('perdeuterodimethylsulphoxide'):'CS(C)=O',
hash('2-methylprop-1-ene'):'CC(C)=C',
hash('1-bromonaphthalene'):'Brc1cccc2ccccc12',
hash('n-decane'):'CCCCCCCCCC',
hash('2-methylpropane-2-peroxol'):'CC(C)(C)OO',
hash('2-phenylpropane-2-peroxol'):'CC(C)(OO)c1ccccc1',
hash('2-isopropoxypropane'):'CC(C)OC(C)C',
hash('1,2-dibromoethene'):'BrC=CBr', hash('1,2-dibromoethane'):'BrCCBr',
hash('2-methylcyclohexanone'):'CC1CCCCC1=O', hash('isopentane'):'CCC(C)C',
hash('1,3,5-trinitrobenzene'):'O=N(=O)c1cc(cc(c1)N(=O)=O)N(=O)=O',
hash('2,4-pentanedione'):'CC(O)=CC(C)=O',
hash('4-methylpyridine'):'Cc1ccncc1', hash('acetonitrle'):'CC#N',
hash('butane-1,4-diol'):'OCCCCO',
hash('1,3-diphenylacetone'):'O=C(Cc1ccccc1)Cc1ccccc1',
hash('2-ethylpyridine'):'CCc1ccccn1', hash('1,2,4-triazole'):'N1C=NC=N1',
hash('1-methylpyrrolidine'):'CN1CCCC1',
hash('trichloroethane'):'ClCC(Cl)Cl', hash('piperidine'):'C1CCNCC1',
hash('but-2-yne-1,4-diol'):'OCC#CCO', hash('methanedithione'):'S=C=S',
hash('2-t-butyl-4-methylphenol'):'Cc1ccc(O)c(c1)C(C)(C)C',
hash('aniline'):'Nc1ccccc1', hash('(methylsulfinyl)methane'):'CS(C)=O',
hash('1,1,3,3-tetramethylurea'):'CN(C)C(=O)N(C)C',
hash('cyclooctane'):'C1CCCCCCC1',
hash('1,3-dimethyltetrahydropyrimidin-2(1H)-one'):'CN1CCCN(C)C1=O',
hash('2,2-difluoroethanol'):'OCC(F)F',
hash('trideuteroacetonitrile'):'CC#N',
hash('1-methyl-3-(methylsulfinyl)benzene'):'Cc1cccc(c1)S(C)=O',
hash('1-(ethylsulfinyl)-4-methylbenzene'):'CCS(=O)c1ccc(C)cc1',
hash('dimethylaminobenzene'):'CN(C)c1ccccc1', hash('ammonia'):'N',
hash('perdeuterobenzene'):'c1ccccc1',
hash('perdeuterochloroform'):'ClC(Cl)Cl', hash('benzylamine'):'NCc1ccccc1',
hash('cyclopentanone'):'O=C1CCCC1',
hash('2-nitroaniline'):'Nc1ccccc1N(=O)=O',
hash('1-butoxybutane'):'CCCCOCCCC', hash('propanenitrile'):'CCC#N',
hash('hexamethylbenzene'):'Cc1c(C)c(C)c(C)c(C)c1C',
hash('1,2,3-trimethylbenzene'):'Cc1cccc(C)c1C',
hash('1,2,4-trimethylbenzene'):'Cc1ccc(C)c(C)c1',
hash('1,4-dimethylbenzene'):'Cc1ccc(C)cc1',
hash("N,N,N',N'-tetramethylethane-1,2-diamine"):'CN(C)CCN(C)C',
hash('ethylenediamine'):'NCCN', hash('p-hydroquinone'):'Oc1ccc(O)cc1',
hash('N,N-Dimethylformamide'):'CN(C)C=O', hash('urea'):'NC(N)=O',
hash('butane-2,3-diol'):'CC(O)C(C)O',
hash('benzene-1,4-diol'):'Oc1ccc(O)cc1',
hash('furan-2-carbaldehyde'):'O=CC1=CC=CO1',
hash('2,6-dimethylpyridine'):'Cc1cccc(C)n1',
hash('perdeuterobenzene'):'c1ccccc1',
hash('hexafluoroisopropanol'):'OC(C(F)(F)F)C(F)(F)F',
hash('3,6-dioxaoctane'):'CCOCCOCC',
hash('4-methoxytoluene'):'COc1ccc(C)cc1',
hash('1,4-dichlorobenzene'):'Clc1ccc(Cl)cc1',
hash('N-ethylethanamine'):'CCNCC', hash('iso-butanol'):'CC(C)CO',
hash('4-hydroxy-4-methylpentan-2-one'):'CC(=O)CC(C)(C)O',
hash('2-methylpropan-1-ol'):'CC(C)CO', hash('propane-1,2-diol'):'CC(O)CO',
hash('iodoform'):'IC(I)I', hash('1-phenylethanol'):'CC(O)c1ccccc1',
hash('benzylamine'):'NCc1ccccc1',
hash('1,2-bis(4-pyridyl)ethene'):'c1cc(ccn1)C=Cc1ccncc1',
hash('1,4-dihydroxybenzene'):'Oc1ccc(O)cc1',
```

```
hash('ethoxycarbonylisothiocyanate'):'CCOC(=O)N=C=S',
hash('tribromo(nitro)methane'):'BrC(Br)(Br)N(=O)=O',
hash('2,2,2-tribromoacetamide'):'NC(=O)C(Br)(Br)Br',
hash('tetrahydrafuran'):'C1CCOC1',
hash('N-methylbenzamide'):'CNC(=O)c1ccccc1',
hash('phenylacetylene'):'C#Cc1ccccc1',
hash('isonicotinamide'):'NC(=O)c1ccncc1',
hash('benzotriazole'):'N1N=Nc2ccccc12', hash('dichlormethane'):'ClCCl',
hash('N-methyl-2-pyrrolidone'):'CN1CCCC1=O', hash('lactone'):'O=C1CCCO1',
hash('diiodomethane'):'ICI',
hash('4-hydroxy-5-methylpentan-2-one'):'CC(=O)CC(C)(C)O',
hash('dichlorofluoromethane'):'FC(Cl)Cl',
hash('1,4-cyclohexanedione'):'O=C1CCC(=O)CC1',
hash('hexadeutero-acetone'):'CC(C)=O',
hash('1,2-bis(methoxy)benzene'):'COc1ccccc1OC',
hash('hexafluoro-sulfane'):'FS(F)(F)(F)(F)F', hash('methylamine'):'CN',
hash('2-n-butyl-2-t-butyl-4,6-diphenyl-1H-1,3,5-triazine'):'CCCCC1(NC(=NC(=N1
)c1ccccc1)c1ccccc1)C(C)(C)C', hash('hexanedinitrile'):'N#CCCCCC#N',
hash('2,3.11,12-dibenzo-1,4,10,13,16-hexaoxocyclo-octadeca-2,11-diene'):'C1CO
c2ccccc2OCCOCCOc2ccccc2OCCO1', hash('cyclopentanol'):'OC1CCCC1',
hash('(methylsulfinyl)benzene'):'CS(=O)c1ccccc1',
hash('2-methylpropane'):'CC(C)C',
hash('pentafluorobenzaldehyde'):'Fc1c(F)c(F)c(C=O)c(F)c1F',
hash('ethanaol'):'CCO', hash('1-pentene'):'CCCC=C',
hash('1-chlorobutane'):'CCCCCl', hash('triiodomethane'):'IC(I)I',
hash('tetramethylethane-1,2-diamine'):'CN(C)CCN(C)C',
hash('3-ethyltoluene'):'CCc1cccc(C)c1', hash('neopentane'):'CC(C)(C)C',
hash('butane'):'CCCC', hash('1,2-diaminoethenol'):'NC=C(N)O',
hash('dimethylisosorbide'):'COC1COC2C(COC12)OC',
hash('5-chloropentanenitrile'):'ClCCCCC#N',
hash('propanedinitrile'):'N#CCC#N', hash('o-toluidine'):'Cc1ccccc1N',
hash('p-toluidine'):'Cc1ccc(N)cc1',
hash('hexachloroethane'):'ClC(Cl)(Cl)C(Cl)(Cl)Cl',
hash('1,4-dibromobutane'):'BrCCCCBr', hash('cyclohexenone'):'O=C1CCCC=C1',
hash('tetrahydro-2H-pyran'):'C1CCOCC1',
hash('1-(4-chlorophenyl)ethanol'):'CC(O)c1ccc(Cl)cc1',
hash('1-phenylethanone'):'CC(=O)c1ccccc1',
hash('4-ethyltoluene'):'CCc1ccc(C)cc1', hash('tetrahydropyran'):'C1CCOCC1',
hash('benzoquinone'):'O=C1C=CC(=O)C=C1',
hash('terephthalaldehyde'):'O=Cc1ccc(C=O)cc1', hash('acetontrile'):'CC#N',
hash('2methylpyridine'):'Cc1ccccn1', hash('3-chloropyridine'):'Clc1cccnc1',
hash('1,1-dichloroethane'):'CC(Cl)Cl',
hash('1-(4-methylphenyl)ethanol'):'CC(O)c1ccc(C)cc1',
hash('3-ethylpentan-3-ol'):'CCC(O)(CC)CC',
hash('(S)-2-methylbutan-1-ol'):'CCC(C)CO',
hash('1,3,5-trifluoro-2,4,6-triiodobenzene'):'Fc1c(I)c(F)c(I)c(F)c1I',
hash('N,N,4-trimethylbenzene-1-sulfonamide'):'CN(C)S(=O)(=O)c1ccc(C)cc1',
hash('iso-propanol'):'CC(C)O', hash('deuterodimethylsulfoxide'):'CS(C)=O',
hash('dibenzylamine'):'C(NCc1ccccc1)c1ccccc1',
hash('hexamethylphosphortriamide'):'CN(C)P(=O)(N(C)C)N(C)C',
hash("2,2'-oxydiethanol"):'OCCOCCO',
hash('trichlorofluoromethane'):'FC(Cl)(Cl)Cl',
hash('dichlorodifluoromethane'):'FC(F)(Cl)Cl',
hash('chlorodifluoromethane'):'FC(F)Cl',
hash('1,1,1-trifluoroethane'):'CC(F)(F)F',
hash('2,2-dichloropropane'):'CC(C)(Cl)Cl',
hash('2,4,6-trimethylphenol'):'Cc1cc(C)c(O)c(C)c1',
hash('1,1,1,3,3,3-hexafluoroisopopanol'):'OC(C(F)(F)F)C(F)(F)F',
hash('cyclo-octylamine'):'NC1CCCCCCC1', hash('aetonitrile'):'CC#N',
hash('propan-2-one'):'CC(C)=O', hash('biphenyl'):'c1ccc(cc1)c1ccccc1',
hash('18-crown-6'):'C1COCCOCCOCCOCCOCCO1',
hash('2,5-dihydrofuran'):'C1OCC=C1',
hash('carbazole'):'N1c2ccccc2c2ccccc12', hash('indole'):'N1C=Cc2ccccc12',
hash('(N,N-dimethylformamide)'):'CN(C)C=O', hash('di-iodomethane'):'ICI',
hash('2,6-dichlorophenol'):'Oc1c(Cl)cccc1Cl',
hash('1-(morpholin-4-yl)ethanone'):'CC(=O)N1CCOCC1',
hash('4-methylbenzaldehyde'):'Cc1ccc(C=O)cc1',
```

```
248                     entry1 = entry_reader2
249
250
251
252
253
254 if __name__ == '__main__':
255     # This runs the script
256     r = Runner()
257     r.run()
258
```

```python
#6 Find potential duplicate none hydrate structures with formula match

from ccdc import io, search
import argparse
import os
import glob
import re
from decimal import Decimal

#All the hydrate structures without entry SMILES strings will be compared to each other and hydrates with entry SMILES strings
#This script finds all the hydrate structures with the same entry formula as the hydrate structures without entry SMILES strings
filepath1 = "C:\Users\jenwe\Hydrates Manuscript\hydrates_None_without_metals.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\hydrates_SMILES_without_metals.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='hydrates_None_all_potential_duplicates.gcd',
            help='output file [hydrates_None_all_potential_duplicates.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')
        entry_reader2 = io.EntryReader(filepath2, format='identifiers')

        with io.EntryWriter(self.args.output) as new_lists_writer:

            #Two lists of formulas are created; one for hydrate structures with entry SMILES strings and one for hydrate structures without entry SMILES strings
            None_formulas = []
            formulas = []
            positions = []
            numbers = []
            for a in range(len(entry_reader1)):
                component_formulas = entry_reader1[a].formula.split(',')
                if entry_reader1[a].identifier == 'LOYXAA':
                    component_formulas.append('H2 O1')
                if entry_reader1[a].identifier != 'KIJFAN':
                    component_formulas.append('x(H2 O1)')
                if all('x(' not in t for t in component_formulas) == True and all('n(' not in s for s in component_formulas) == True:
                    factors = []
                    for z in range(len(component_formulas)):
                        if '(' and ')' in component_formulas[z]:
                            pin = component_formulas[z].index('(')
                            if component_formulas[z][:pin].isdigit() == True:
                                factors.append(int(component_formulas[z][:pin]))
                            else:
```

```
                                factors.append(Decimal(component_formulas[z][:pin]))
                            else:
                                factors.append(1)
                        normalized_factors = []
                        for y in range(len(factors)):
                            normalized_factors.append(str(factors[y]/min(factors)))
                        if factors != normalized_factors:
                            for x in range(len(normalized_factors)):
                                if normalized_factors[x] == '1':
                                    if '(' and ')' in component_formulas[x]:
                                        start = component_formulas[x].index('(')
                                        end = component_formulas[x].index(')')
                                        component_formulas[x] =
                                        component_formulas[x][start+1:end]
                                else:
                                    if '(' and ')' in component_formulas[x]:
                                        start = component_formulas[x].index('(')
                                        component_formulas[x] = normalized_factors[x] +
                                        component_formulas[x][start]
                                    else:
                                        component_formulas[x] = normalized_factors[x] + "("
                                        + component_formulas[x] + ")"

                pre_formulas = []
                #The stoichiometry of each molecule in the entry was excluded when
                comparing the formulas
                #The stoichiometry of structures that are considered duplicates was
                checked later on
                #Structures with different stoichiometry are considered distinct entries
                for b in range(len(component_formulas)):
                    if '(' and ')' in component_formulas[b]:
                        start = component_formulas[b].index('(')
                        end = component_formulas[b].index(')')
                        letter_formula = component_formulas[b][start+1:end]
                        elements = letter_formula.split(' ')
                    #The elements of each molecular formula was screened to change
                    deuterium to hydrogen
                    #Structures with the same packing that exist in non-deuterated and
                    deuterated forms are considered duplicate entries
                    else:
                        elements = component_formulas[b].split(' ')
                    deuteriums = [i for i in elements if re.sub(r'[0-9]+', '', i) == 'D']
                    if deuteriums != []:
                        if len(deuteriums) != 1:
                            print entry_reader1[a].identifier
                        hydrogens = [i for i in elements if re.sub(r'[0-9]+', '', i) ==
                        'H']
                        if hydrogens == []:
                            spot = elements.index(deuteriums[0])
                            deuteriums[0] = deuteriums[0].replace('D', 'H')
                            elements[spot] = deuteriums[0]
                        else:
                            #In cases where there are hydrogen and deuterium atoms in a
                            molecule, the stoichiometry for each is used to determine
                            the stoichiometry of the resulting hydrogen only formula
                            if len(hydrogens) != 1:
                                print entry_reader1[a].identifier
                            spot = elements.index(deuteriums[0])
                            spot2 = elements.index(hydrogens[0])
                            stoichiometry1 = re.sub(r'[A-Z]+', '', deuteriums[0])
                            stoichiometry2 = re.sub(r'[A-Z]+', '', hydrogens[0])
                            total = int(stoichiometry1) + int(stoichiometry2)
                            hydrogens[0] = hydrogens[0].replace(stoichiometry2,
                            str(total))
                            elements[spot2] = hydrogens[0]
                            elements.remove(elements[spot])
                        if '(' and ')' in component_formulas[b]:
```

```
                            letter_formula = ' '.join(elements)
                            component_formulas[b] =
                            component_formulas[b].replace(component_formulas[b][start+1:e
                            nd], letter_formula)
                        else:
                            component_formulas[b] = ' '.join(elements)
                    pre_formulas.append(hash(component_formulas[b]))

                pre_formulas = list(set(pre_formulas))
                pre_formulas = sorted(pre_formulas)
                None_formulas.append(pre_formulas)
                formulas.append(pre_formulas)
                #The position of each structure in the input file was recorded to
                update the list of formulas later on in the script
                positions.append(a)
                #The spacegroup number of each structure was also recorded in a
                separate list
                #Structures that could not generate the spacegroup number were listed
                separately with their number determined manually in Mercury
                if hash(entry_reader1[a].identifier) == hash('KECYBU15'):
                    numbers.append(87)
                else:

                    numbers.append(entry_reader1[a].crystal.spacegroup_number_and_setting
                    [0])

            #The same steps are taken for hydrate structures with entry SMILES strings
            for d in range(len(entry_reader2)):
                component_formulas = entry_reader2[d].formula.split(',')
                if entry_reader2[d].identifier == 'LOYXAA':
                    component_formulas.append('H2 O1')
                if entry_reader2[d].identifier != 'KIJFAN':
                    component_formulas.append('x(H2 O1)')
                if all('x(' not in t for t in component_formulas) == True and all('n('
                not in s for s in component_formulas) == True:
                    factors = []
                    for z in range(len(component_formulas)):
                        if '(' and ')' in component_formulas[z]:
                            pin = component_formulas[z].index('(')
                            if component_formulas[z][:pin].isdigit() == True:
                                factors.append(int(component_formulas[z][:pin]))
                            else:
                                factors.append(Decimal(component_formulas[z][:pin]))
                        else:
                            factors.append(1)
                    normalized_factors = []
                    for y in range(len(factors)):
                        normalized_factors.append(str(factors[y]/min(factors)))
                    if factors != normalized_factors:
                        for x in range(len(normalized_factors)):
                            if normalized_factors[x] == '1':
                                if '(' and ')' in component_formulas[x]:
                                    start = component_formulas[x].index('(')
                                    end = component_formulas[x].index(')')
                                    component_formulas[x] =
                                    component_formulas[x][start+1:end]
                            else:
                                if '(' and ')' in component_formulas[x]:
                                    start = component_formulas[x].index('(')
                                    component_formulas[x] = normalized_factors[x] +
                                    component_formulas[x][start]
                                else:
                                    component_formulas[x] = normalized_factors[x] + "("
                                    + component_formulas[x] + ")"

                pre_formulas2 = []
                for e in range(len(component_formulas)):
```

```
                    if '(' and ')' in component_formulas[e]:
                        start = component_formulas[e].index('(')
                        end = component_formulas[e].index(')')
                        letter_formula = component_formulas[b][start+1:end]
                        elements = letter_formula.split(' ')
                    else:
                        elements = component_formulas[e].split(' ')
                    deuteriums = [i for i in elements if re.sub(r'[0-9]+', '', i) == 'D']
                    if deuteriums != []:
                        if len(deuteriums) != 1:
                            print entry_reader2[d].identifier
                        hydrogens = [i for i in elements if re.sub(r'[0-9]+', '', i) ==
                        'H']
                        if hydrogens == []:
                            spot = elements.index(deuteriums[0])
                            deuteriums[0] = deuteriums[0].replace('D', 'H')
                            elements[spot] = deuteriums[0]
                        else:
                            if len(hydrogens) != 1:
                                print entry_reader2[d].identifier
                            spot = elements.index(deuteriums[0])
                            spot2 = elements.index(hydrogens[0])
                            stoichiometry1 = re.sub(r'[A-Z]+', '', deuteriums[0])
                            stoichiometry2 = re.sub(r'[A-Z]+', '', hydrogens[0])
                            total = int(stoichiometry1) + int(stoichiometry2)
                            hydrogens[0] = hydrogens[0].replace(stoichiometry2,
                            str(total))
                            elements[spot2] = hydrogens[0]
                            elements.remove(elements[spot])
                        if '(' and ')' in component_formulas[b]:
                            letter_formula = ' '.join(elements)
                            component_formulas[e] =
                            component_formulas[b].replace(component_formulas[e][start+1:e
                            nd], letter_formula)
                        else:
                            component_formulas[e] = ' '.join(elements)
                    pre_formulas2.append(hash(component_formulas[e]))

                pre_formulas2 = list(set(pre_formulas2))
                pre_formulas2 = sorted(pre_formulas2)
                formulas.append(pre_formulas2)
                positions.append(d+718)
                if hash(entry_reader2[d].identifier) == hash('MTYHFB03'):
                    numbers.append(29)
                else:

                    numbers.append(entry_reader2[d].crystal.spacegroup_number_and_setting
                    [0])

            #A while loop is used to iterate through the list of formulas for hydrate
            structures without an entry SMILES string
            while None_formulas != []:
                print len(None_formulas)
                formulas_to_remove = []
                positions_to_write = []
                numbers_to_compare = []
                formulas_to_remove.append(formulas[0])
                positions_to_write.append(positions[0])
                numbers_to_compare.append(numbers[positions[0]])
                #The formula of the first hydrate structure without an entry SMILES
                string is compared to all the other hydrate structure formulas
                #Matching formulas are appended to two lists: one has their position in
                the corresponding input file, the other has their spacegroup number
                #The formula list includes all hydrate structures, the beginning
                formulas are for hydrate structures without an entry SMILES string
                for c in range(1, len(formulas)):
                    if formulas[0] == formulas[c]:
```

```
                        positions_to_write.append(positions[c])
                        numbers_to_compare.append(numbers[positions[c]])

                #If there are no matches, the length of the numbers_to_compare list will only contain the hydrate structure being compared
                #Otherwise, the stoichiometry of the water molecules in each hydrate structure with that formula is determined
                if len(numbers_to_compare) != 1:
                    #Only cases where the spacegroup number is the same for at least two structures are checked for duplicates
                    if all(x == numbers_to_compare[0] for x in numbers_to_compare):
                    elif any(numbers_to_compare.count(x) > 1 for x in numbers_to_compare):
                        positions_to_remove = []
                        #Each potential duplicate is compared to all the other structures in the list
                        #If the potential duplicate does not match both the water stoichiometry and spacegroup number of another structure in the list it is placed in the lists to be removed
                        while len(positions_to_write) != len(positions_to_remove):
                            for j in range(len(positions_to_write)):
                                if positions_to_write[j] not in positions_to_remove:
                                    positions_to_remove.append(positions_to_write[j])
                                    if positions_to_write[j] < 625:
                                        match = 0
                                        for k in range(len(positions_to_write)):
                                            if positions_to_write[k] not in positions_to_remove:
                                                #If the spacegroup number is the same, the structures are put in lists to be analyzed in part2 of this script
                                                if numbers_to_compare[j] == numbers_to_compare[k]:
                                                    if match == 0:

                                                        new_lists_writer.write(entry_reader1[positions_to_write[j]])
                                                    if positions_to_write[k] < 625:

                                                        new_lists_writer.write(entry_reader1[positions_to_write[k]])
                                                    else:

                                                        new_lists_writer.write(entry_reader2[positions_to_write[k]-625])
                                                    match += 1

                                                    positions_to_remove.append(positions_to_write[k])
                                        if match != 0:

                                            new_lists_writer.write(entry_reader1[positions_to_write[j]])

                #The lists of formulas and positions is updated so structures that have been evaulated are removed
                #The list of spacegroup numbers is generated for each analysis and therefore does not need to be updated
                None_formulas = [i for i in None_formulas if i not in formulas_to_remove]
                formulas = [i for i in formulas if i not in formulas_to_remove]
                positions = [j for j in positions if j not in positions_to_write]



if __name__ == '__main__':
    r = Runner()
```

```
277         r.run()
278
```

```python
#7 Find potential duplicate none hydrate structures with same identifier prefix

from ccdc import io, search
import argparse
import os
import glob
import re
from decimal import Decimal

#All the hydrate structures that share the same component stoichometry and spacegroup number found in the previous script are analyzed here
filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\hydrates_None_all_potential_duplicates.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='hydrates_None_all_manual_check.gcd',
            help='output file [hydrates_None_all_manual_check.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')

        with io.EntryWriter(self.args.output) as new_lists_writer:

            #Lists of the hydrate identifiers and entry readers are generated
            solvated_hydrates_hash = []
            solvated_hydrates_entry = []
            for a in range(len(entry_reader1)):
                solvated_hydrates_hash.append(hash(entry_reader1[a].identifier))
                solvated_hydrates_entry.append(entry_reader1[a])

            #The previous output file printed the first hydrate structure identifier in the list of duplicates twice
            #For example "ABCDEF, ABCDEF01, ABCDEF"
            #That way the duplicates can be found by searching for the next occurence of the first identifier
            while len(solvated_hydrates_hash) != 0:
                print len(solvated_hydrates_hash)
                identifiers = []
                identifiers.append(solvated_hydrates_hash[0])
                for d in range(1,len(solvated_hydrates_hash)):
                    if identifiers[0] != solvated_hydrates_hash[d]:
                        identifiers.append(solvated_hydrates_hash[d])
                    else:
                        del solvated_hydrates_hash[d]
                        del solvated_hydrates_entry[d]
                        break

```

```
                    #The first six letters in each identifer are appended to a list
                    nomenclature = []
                    for e in range(len(identifiers)):
                        pin = solvated_hydrates_hash.index(identifiers[e])
                        nomenclature.append(solvated_hydrates_entry[pin].identifier[:6])

                    #Two structures with the same first six letters in their identifiers
                    are considered potential duplicates
                    #If all the structures have the same first six letters in their
                    identifiers they are written to the output file of potential duplicates
                    if all(x == x[0] for x in nomenclature):
                        pin1 = solvated_hydrates_hash.index(identifiers[0])
                        new_lists_writer.write(solvated_hydrates_entry[pin1])
                        for f in range(1, len(identifiers)):
                            pin2 = solvated_hydrates_hash.index(identifiers[f])
                            new_lists_writer.write(solvated_hydrates_entry[pin2])
                        new_lists_writer.write(solvated_hydrates_entry[pin1])
                    else:
                        #If not, they are iterated through to check for subsets of
                        structures with the same first six letters in their identifier
                        remove_identifiers = []
                        while len(identifiers) != len(remove_identifiers):
                            for h in range(len(identifiers)):
                                if identifiers[h] not in remove_identifiers:
                                    remove_identifiers.append(identifiers[h])
                                    if nomenclature.count(nomenclature[h]) > 1:
                                        pin3 = solvated_hydrates_hash.index(identifiers[h])
                                        new_lists_writer.write(solvated_hydrates_entry[pin3])
                                        #Any cases where at least two structures share the
                                        same first six letters are written to the output
                                        file of potential duplicates
                                        for i in range(len(identifiers)):
                                            if identifiers[i] not in remove_identifiers:
                                                if nomenclature[h] == nomenclature[i]:
                                                    remove_identifiers.append(identifiers[i])
                                                    pin4 =
                                                    solvated_hydrates_hash.index(identifiers[
                                                    i])

                                                    new_lists_writer.write(solvated_hydrates_
                                                    entry[pin4])
                                        new_lists_writer.write(solvated_hydrates_entry[pin3])

                    #Each set of structures that has been analyzed is removed from the list
                    of structures before the next set is tested
                    for j in range(len(identifiers)):
                        pin5 = solvated_hydrates_hash.index(identifiers[j])
                        solvated_hydrates_hash.remove(solvated_hydrates_hash[pin5])
                        solvated_hydrates_entry.remove(solvated_hydrates_entry[pin5])



if __name__ == '__main__':
    r = Runner()
    r.run()

```

```
#8 Find potential duplicate SMILES hydrate structures with formula and SMILES string match

from ccdc import io, search
import argparse
import os
import glob
import re
import itertools
from decimal import Decimal
from ccdc.crystal import PackingSimilarity

similarity_engine = PackingSimilarity()
similarity_engine.settings.distance_tolerance = 0.25
similarity_engine.settings.angle_tolerance = 25.

#This script finds hydrate structures with SMILES strings that have the same formula as other hydrate structures with SMILES strings
filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\complete_smiles_hydrates.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='hydrates_smiles_all_potential_duplicates.gcd',
            help='output file [hydrates_smiles_all_potential_duplicates.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')

        with io.EntryWriter(self.args.output) as duplicates_writer:

            solvents_dictionary = {hash('but-2-ene'):'CC=CC',
            hash('1,1,2-dichloroethane'):'ClCC(Cl)Cl',
            hash('4-aminopyridine'):'Nc1ccncc1', hash('N-methylurea'):'CNC(N)=O',
            hash('2-methylphenol'):'Cc1ccccc1O', hash('tetraglyme'):'COCCOCCOCCOCCOC',
            hash('isophorone'):'CC1=CC(=O)CC(C)(C)C1',
            hash('L-carvone'):'CC(=C)C1CC=C(C)C(=O)C1',
            hash('2-cyanopyridine'):'N#Cc1ccccn1', hash('biuret'):'NC(=O)NC(N)=O',
            hash('styrene'):'C=Cc1ccccc1', hash('carbondisulfide'):'S=C=S',
            hash('tricyanomethanide'):'N#C[C-](C#N)C#N',
            hash('methyl-cyclohexane'):'CC1CCCCC1',
            hash('pyridine-3-carboxamide'):'NC(=O)c1cccnc1',
            hash('3,7-dimethyl-3,7-dihydro-1H-purine-2,6-dione'):'CN1C=NC2=C1C(=O)NC(=O)N2C', hash('nitroethane'):'CCN(=O)=O',
            hash('dimethylaminobenzaldehyde'):'CN(C)c1ccc(C=O)cc1',
            hash('3-(1H-pyrrol-1-yl)propanenitrile'):'C1=CN(C=C1)CCC#N',
            hash('trifluoromethanol'):'OC(F)(F)F', hash('methoxyethane'):'CCOC',
            hash('methylanisole'):'COc1ccccc1C', hash('xylene'):'Cc1ccc(C)cc1',
```

```
hash('petrol'):'CCCCCC', hash('n-alkane'):'CCCCC',
hash('dichloromethanal'):'ClC(Cl)=O',
hash('1-fluoro-4-methylbenzene'):'Cc1ccc(F)cc1',
hash('1,2,3-trimethoxybenzene'):'COc1cccc(OC)c1OC',
hash('1,3,5-trimethoxybenzene'):'COc1cc(OC)cc(OC)c1',
hash('chloroform'):'C1C(Cl)Cl', hash('methanol'):'CO',
hash('hexane'):'CCCCCC', hash('ethanol'):'CCO',
hash('acetonitrile'):'CC#N', hash('toluene'):'Cc1ccccc1',
hash('tetrahydrofuran'):'C1CCOC1', hash('acetone'):'CC(C)=O',
hash('dichloromethane'):'ClCCl', hash('1,2-dichloroethane'):'ClCCCl',
hash('dimethylsulfoxide'):'CS(C)=O', hash('1,4-dioxane'):'C1COCCO1',
hash('1,2-dimethoxyethane'):'COCCOC', hash('o-xylene'):'Cc1ccccc1c',
hash('propanol'):'CCO', hash('dioxane'):'C1COCCO1',
hash('n-pentane'):'CCCCC', hash('4-cyanopyridine'):'N#Cc1ccncc1',
hash("4,4'-bipyridine"):'c1cc(ccn1)c1ccncc1', hash('isopropanol'):'CC(C)O',
hash('1-butanol'):'CCCCO', hash('nitromethane'):'CN(=O)=O',
hash('n-hexane'):'CCCCCC', hash('dimethylformamide'):'CN(C)C=O',
hash('N,N-dimethylformamide'):'CN(C)C=O', hash('cyclohexane'):'C1CCCCC1',
hash('hydroquinone'):'Oc1ccc(O)cc1', hash('benzene'):'c1ccccc1',
hash('octane'):'CCCCCCCC', hash('cyclohexanone'):'O=C1CCCCC1',
hash('isobutanol'):'CC(C)CO', hash('p-xylene'):'Cc1ccc(C)cc1',
hash('pentane'):'CCCCC', hash('pyridine'):'c1ccncc1',
hash('t-butanol'):'CC(C)(C)O', hash('m-xylene'):'Cc1cccc(C)c1',
hash('phenol'):'Oc1ccccc1', hash('o-cresol'):'Cc1ccccc1O',
hash('p-cresol'):'Cc1ccc(O)cc1',
hash('2,4,6-trimethylpyridine'):'Cc1cc(C)nc(C)c1',
hash('1-methylpyrrolidin-2-one'):'CN1CCCC1=O',
hash('1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8-hexadecafluoro-1,8-diiodooctane'):'FC(F
)(I)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)I',
hash('trifluoroethanol'):'OCC(F)(F)F', hash('dichloroethane'):'ClCCCl',
hash('1,1,2,2-tetrachloroethane'):'ClC(Cl)C(Cl)Cl',
hash('2-propanol'):'CC(C)O', hash('chlorobenzene'):'Clc1ccccc1',
hash('naphthalene'):'c1ccc2ccccc2c1', hash('propan-2-ol'):'CC(C)O',
hash('benzonitrile'):'N#Cc1ccccc1', hash('nitrobenzene'):'O=N(=O)c1ccccc1',
hash('dodecane'):'CCCCCCCCCCCC',
hash('1,2-dichlorobenzene'):'Clc1ccccc1Cl', hash('n-heptane'):'CCCCCCC',
hash('methanesulfonate'):'CS(=O)(=O)[O-]',
hash('3-methylbutan-1-ol'):'CC(C)CCO',
hash('1,1,2-trichloroethane'):'ClCC(Cl)Cl', hash('n-butane'):'CCCC',
hash('dichlorine'):'ClCl', hash('o-dichlorobenzene'):'Clc1ccccc1Cl',
hash('N,N-dimethylacetamide'):'CN(C)C(C)=O',
hash('4-methylaniline'):'Cc1ccc(N)cc1', hash('morpholine'):'C1COCCN1',
hash('1,3-dimethylimidazolidin-2-one'):'CN1CCN(C)C1=O)',
hash('1,3-dimethoxybenzene'):'COc1cccc(OC)c1', hash('n-propanol'):'CCCO',
hash('mesitylene'):'Cc1cc(C)cc(C)c1',
hash('acetylacetone'):'CC(=O)CC(C)=O', hash('diethylether'):'CCOCC',
hash('hydrazine'):'NN', hash('dimethylamine'):'CNC',
hash('1,2-difluorobenzene'):'Fc1ccccc1F', hash('cycloheptane'):'C1CCCCCC1',
hash('formamide'):'NC=O', hash('2-butanol'):'CCC(C)O',
hash('ethylacetate'):'CCOC(C)=O', hash('butan-1-ol'):'CCCCO',
hash('bromomethane'):'CBr', hash('piperazine'):'C1CNCCN1',
hash('n-nonane'):'CCCCCCCCC',
hash('1,2,4-trimethoxybenzene'):'COc1ccc(OC)c(OC)c1',
hash('1,2-diaminoethane'):'NCCN', hash('n-butanol'):'CCCCO',
hash('N-methylformamide'):'CNC=O', hash('pyrrolidine'):'C1CCNC1',
hash('actone'):'CC(C)=O', hash('n-octane'):'CCCCCCCC',
hash('(S)-2-methyl-1-butanol'):'CCC(C)CO',
hash('N,N-diethylformamide'):'CCN(CC)C=O',
hash('tetracyanoethylene'):'N#CC(C#N)=C(C#N)C#N',
hash('fluorobenzene'):'Fc1ccccc1', hash('heptane'):'CCCCCCC',
hash('tetrachloroethane'):'ClC(Cl)C(Cl)Cl',
hash('methylcyclohexane'):'CC1CCCCC1',
hash('tetrachloromethane'):'ClC(Cl)(Cl)Cl', hash('1-propanol'):'CCCO',
hash('acetamide'):'CC(N)=O', hash('propane'):'CCC',
hash('diethylamine'):'CCNCC',
hash('hexafluorophosphate'):'F[P-](F)(F)(F)(F)F',
hash('hexafluoropropan-2-ol'):'OC(C(F)(F)F)C(F)(F)F',
```

```
            hash('1-chloro-4-methylbenzene'):'Cc1ccc(Cl)cc1',
            hash('acetaldehyde'):'CC=O', hash('propiononitrile'):'CCC#N',
            hash('bromobenzene'):'Brc1ccccc1'}
49          deuterated_dictionary = {hash('dideutero-dichloromethane'):'ClCCl',
            hash('perdeutero-toluene'):'Cc1ccccc1', hash('octadeuterofuran'):'C1CCOC1',
            hash('deutero-ethanol'):'CCO', hash('deuterochloroform'):'C1C(Cl)Cl',
            hash('deutero-chloroform'):'C1C(Cl)Cl', hash('deuterobenzene'):'c1ccccc1',
            hash('deuteromethanol'):'CO', hash('hexadeutero-benzene'):'c1ccccc1',
            hash('perdeuteroacetone'):'CC(C)=O',
            hash('dideuterodichloromethane'):'ClCCl', hash('perdeuteromethanol'):'CO',
            hash('hexadeuterobenzene'):'c1ccccc1'}
50          acids_dictionary = {hash('trifluoroacetic'):'OC(=O)C(F)(F)F',
            hash('acetic'):'CC(O)=O', hash('formic'):'OC=O',
            hash('sulfuric'):'OS(O)(=O)=O', hash('maleic'):'OC(=O)C=CC(O)=O',
            hash('benzoic'):'OC(=O)c1ccccc1', hash('succinic'):'OC(=O)CCC(O)=O'}
51          ethers_dictionary = {'t-butyl methyl ether':'COC(C)(C)C',
            hash('di-isopropyl'):'CC(C)OC(C)C', hash('dipropyl'):'CCCOCCC',
            hash('diethyl'):'CCOCC', hash('diisopropyl'):'CC(C)OC(C)C',
            hash('isopropyl'):'CC(C)OC(C)C', hash('t-butyl'):'COC(C)(C)C',
            hash('acetic'):'CCOC(C)=O'}
52          acetates_dictionary = {hash('ethyl'):'CCOC(C)=O',
            hash('amyl'):'CCCCCOC(C)=O', hash('methyl'):'COC(C)=O',
            hash('butyl'):'CCCCOC(C)=O', hash('isobutyl'):'CC(C)COC(C)=O'}
53          glycol_dictionary = {hash('ethylene'):'OCCO'}
54          oxalates_dictionary = {hash('dipropyl'):'CCCOC(=O)C(=O)OCCC'}
55          acid_neutral_dictionary = {hash('oxalic'):'OC(=O)C(O)=O',
            hash('nitric'):'ON(=O)=O', hash('trans-but-2-enedioic'):'OC(=O)C=CC(O)=O',
            hash('but-2-enedioic'):'OC(=O)C=CC(O)=O',
            hash('fumaric'):'OC(=O)C=CC(O)=O', hash('phosphonic'):'OP(O)=O',
            hash('perchloric'):'O[Cl](=O)(=O)=O', hash('phosphoric'):'OP(O)(O)=O',
            hash('pyridine-2,6-dicarboxylic'):'OC(=O)c1cccc(n1)C(O)=O',
            hash('benzene-1,2,4,5-tetracarboxylic'):'OC(=O)c1cc(C(O)=O)c(cc1C(O)=O)C(O)=O
            ', hash('trifluoroacetatic'):'OC(=O)C(F)(F)F'}
56          acid_charged_dictionary = {hash('oxalic'):'OC(=O)C(=O)[O-]',
            hash('nitric'):'O=N(=O)[O-]',
            hash('trans-but-2-enedioic'):'OC(=O)C=CC(=O)[O-]',
            hash('but-2-enedioic'):'OC(=O)C=CC(=O)[O-]',
            hash('fumaric'):'OC(=O)C=CC(=O)[O-]', hash('phosphonic'):'OP(=O)[O-]',
            hash('perchloric'):'O=[Cl](=O)(=O)[O-]',
            hash('phosphoric'):'OP(O)(=O)[O-]',
            hash('pyridine-2,6-dicarboxylic'):'OC(=O)c1cccc(n1)C(=O)[O-]',
            hash('benzene-1,2,4,5-tetracarboxylic'):'OC(=O)c1cc(C(=O)[O-])c(cc1C(=O)[O-])
            C(O)=O', hash('trifluoroacetatic'):'FC(F)(F)C(=O)[O-]'}
57
58          partial_name = [hash('ketone'), hash('propanoate'), hash('carbonate'),
            hash('formate'), hash('iodine'), hash('tetrachloride'), hash('sulfoxide'),
            hash('dichloride'), hash('chloride'), hash('benzoate'), hash('disulfide'),
            hash('oxide')]
59          full_solvate_name_dictionary = {'methyl isobutyl ketone':'CC(C)CC(C)=O',
            'deuterium oxide':'O', 'ethyl propanoate':'CCOC(=O)CC', 'dimethyl
            carbonate':'COC(=O)OC', 'ethyl formate':'CCOC=O', '12]cycloparaphenylene
            iodine':'II', 'carbon tetrachloride':'ClC(Cl)(Cl)Cl', 'dimethyl
            sulfoxide':'CS(C)=O', 'methylene dichloride':'ClCCl', 'methylene
            chloride':'ClCCl', 'ethyl benzoate':'CCOC(=O)c1ccccc1', 'carbon
            disulfide':'S=C=S'}
60
61          #There are ten instances of two distinct molecule SMILES strings giving the
            same hash value
62          #They are accounted for here in this dictionary
63          #Any time SMILES strings are hashed, the dictionary is checked so the
            following ten SMILES strings are represented by the numbers 1-10 rather
            than repeat hash values
64          duplicate_smiles_dictionary =
            {'CC(C)(C)c1c(O)c(O)c(Cl)c2[NH+]3CCN(CC3)c12':1,
            'CC1=C(C(CC2=NS(=O)(=O)c3ccccc23)c2ccccc2C)C(=O)N(N1)c1ccccc1':2,
            'OC(=O)COc1cccc2ccccc12':3, 'Cc1c[nH+]cc(C)c1':4,
            'Brc1ccc(NC(=O)c2cccnc2)cc1':5, 'CSC(N)=NNC(C)=CC(=O)c1ccccc1':6,
```

```
            'CC1(CO1)C(O)(CBr)CBr':7, 'OCC12CCCC(C1)(C(O)=O)C1(CCSCC1)O2':8,
            'COc1ccc2CC3[NH2+]CCC3(C)c2c1':9,
            'CC(=O)C(P(c1ccccc1)c1ccccc1)=C(C)Nc1c(F)cccc1F':10}

solvated_hydrates_entry = []
solvated_hydrates_hash = []
solvated_hydrates_SMILES = []

entry1 = entry_reader1
for z in range(len(entry1)):
    solvated_hydrates_entry.append(entry1[z])
    solvated_hydrates_hash.append(hash(entry1[z].identifier))

#Each hydrate structure is checked for missing solvent SMILES strings
#The chemical name of each hydrate structure is checked for in each solvent dictionary
#Any match to the solvent dictionary checks the entry SMILES string for the corresponding solvent SMILES string
entry1 = entry_reader1
tag = 12
for a in range(len(entry1)):
    component_SMILES = []
    entry_SMILES = entry1[a].molecule.smiles.split('.')
    for b in range(len(entry_SMILES)):
        if entry_SMILES[b] not in duplicate_smiles_dictionary:
            component_SMILES.append(hash(entry_SMILES[b]))
        else:

            component_SMILES.append(duplicate_smiles_dictionary.get(entry_SMILES[b]))
    component_SMILES = list(set(component_SMILES))
    #Some hydrate structures require manual corrections to their SMILES string
    #Example is xylene, orientation of the methyl group cannot be determined from chemical name or formula when SMILES string is missing
    if entry1[a].identifier == 'BULVEL':
        if hash('[O-][n+]1ccccc1') not in component_SMILES:
            component_SMILES.append(hash('[O-][n+]1ccccc1'))
        if hash('[O]n1ccccc1') in component_SMILES:
            component_SMILES.remove(hash('[O]n1ccccc1'))
    elif entry1[a].identifier == 'CEHKOS':
        component_SMILES.append(hash('II'))
    elif entry1[a].identifier == 'GOTJEE' or entry1[a].identifier == 'HASVEF' or entry1[a].identifier == 'MTFBTZ10':
        component_SMILES.append(tag)
        tag += 1
    elif entry1[a].identifier == 'JUHJIF':
        if hash('BrBr') not in component_SMILES:
            component_SMILES.append(hash('BrBr'))
        if hash('[Br]') in component_SMILES:
            component_SMILES.remove(hash('[Br]'))
    elif entry1[a].identifier == 'LOKXUF':
        if hash('OC(=O)C(F)(F)F') not in component_SMILES:
            component_SMILES.append(hash('OC(=O)C(F)(F)F'))
        if hash('[O]C(=O)C(F)(F)F') in component_SMILES:
            component_SMILES.remove(hash('[O]C(=O)C(F)(F)F'))
    elif entry1[a].identifier == 'OJUNEO':
        if hash('ClCl') not in component_SMILES:
            component_SMILES.append(hash('ClCl'))
        if hash('[Cl]') in component_SMILES:
            component_SMILES.remove(hash('[Cl]'))
    elif entry1[a].identifier == 'ZILFILM':
        component_SMILES.append(hash('CCO'))
    else:
        if entry1[a].chemical_name != None:
            pieces = entry1[a].chemical_name.split(' ')
            formula_pieces = entry1[a].formula.split(',')
```

```
                    for c in range(len(pieces)):
                        if hash(pieces[c]) == hash('xylene'):
                            if entry1[a].identifier == 'QOWNEV':
                                if hash('Cc1ccccc1C') not in component_SMILES:
                                    component_SMILES.append(hash('Cc1ccccc1C'))
                            elif entry1[a].identifier == 'TEYCEF' or
                            entry1[a].identifier == 'TEYCEF01':
                                if hash('Cc1cccc(C)c1') not in component_SMILES:
                                    component_SMILES.append(hash('Cc1cccc(C)c1'))
                            elif entry1[a].identifier == 'MAMNAR':
                                if hash('Cc1ccc(C)cc1') not in component_SMILES:
                                    component_SMILES.append(hash('Cc1ccc(C)cc1'))
                        elif hash(pieces[c]) == hash('hydrochloride') or
                        hash(pieces[c]) == hash('bis(hydrochloride)'):
                            for z in range(len(formula_pieces)):
                                if formula_pieces[z] == 'Cl1 1-' or '(Cl1 1-)' in
                                formula_pieces[z]:
                                    if hash('[Cl-]') not in component_SMILES:
                                        component_SMILES.append(hash('[Cl-]'))
                                if formula_pieces[z] == 'H1 Cl1' or '(H1 Cl1)' in
                                formula_pieces[z]:
                                    if hash('Cl') not in component_SMILES:
                                        component_SMILES.append(hash('Cl'))
                        elif hash(pieces[c]) == hash('unidentified') or
                        hash(pieces[c]) == hash('unknown'):
                            component_SMILES.append(tag)
                            tag += 1
                        elif hash(pieces[c]) == hash('glycol'):
                            if hash(pieces[c-1]) in glycol_dictionary:
                                if hash(glycol_dictionary.get(hash(pieces[c-1])))
                                not in component_SMILES:

                                    component_SMILES.append(hash(glycol_dictionary.ge
                                    t(hash(pieces[c-1]))))
                        elif hash(pieces[c]) == hash('oxalate'):
                            if hash(pieces[c-1]) in oxalates_dictionary:
                                if hash(oxalates_dictionary.get(hash(pieces[c-1])))
                                not in component_SMILES:

                                    component_SMILES.append(hash(oxalates_dictionary.
                                    get(hash(pieces[c-1]))))
                        elif hash(pieces[c]) == hash('acetate'):
                            if hash(pieces[c-1]) in acetates_dictionary:
                                if hash(acetates_dictionary.get(hash(pieces[c-1])))
                                not in component_SMILES:

                                    component_SMILES.append(hash(acetates_dictionary.
                                    get(hash(pieces[c-1]))))
                        elif hash(pieces[c]) == hash('ether'):
                            if hash(pieces[c-1]) == hash('petroleum'):
                                component_SMILES.append(11)
                            elif pieces[c-2] + ' ' + pieces[c-1] + ' ' + pieces[c]
                            in ethers_dictionary:
                                if hash(ethers_dictionary.get(pieces[c-2] + ' ' +
                                pieces[c-1] + ' ' + pieces[c])) not in
                                component_SMILES:

                                    component_SMILES.append(hash(ethers_dictionary.ge
                                    t(pieces[c-2] + ' ' + pieces[c-1] + ' ' +
                                    pieces[c])))
                            elif hash(pieces[c-1]) in ethers_dictionary:
                                if hash(ethers_dictionary.get(hash(pieces[c-1])))
                                not in component_SMILES:

                                    component_SMILES.append(hash(ethers_dictionary.ge
                                    t(hash(pieces[c-1]))))
                        elif hash(pieces[c]) in partial_name:
```

```
                    if pieces[c] == 'ketone':
                        if pieces[c-2] + ' ' + pieces[c-1] + ' ' +
                        pieces[c] in full_solvate_name_dictionary:
                            if
                            hash(full_solvate_name_dictionary.get(pieces[c-2]
                             + ' ' + pieces[c-1] + ' ' + pieces[c])) not in
                            component_SMILES:

                                component_SMILES.append(hash(full_solvate_nam
                                e_dictionary.get(pieces[c-2] + ' ' +
                                pieces[c-1] + ' ' + pieces[c])))
                    else:
                        if pieces[c-1] + ' ' + pieces[c] in
                        full_solvate_name_dictionary:
                            if
                            hash(full_solvate_name_dictionary.get(pieces[c-1]
                             + ' ' + pieces[c])) not in component_SMILES:

                                component_SMILES.append(hash(full_solvate_nam
                                e_dictionary.get(pieces[c-1] + ' ' +
                                pieces[c])))
                elif hash(pieces[c]) == hash('acid'):
                    if hash(pieces[c-1]) in acids_dictionary:
                        if hash(acids_dictionary.get(hash(pieces[c-1])))
                        not in component_SMILES:

                            component_SMILES.append(hash(acids_dictionary.get
                            (hash(pieces[c-1]))))
                    elif hash(pieces[c-1]) in acid_neutral_dictionary:
                        if hash(pieces[c-1]) == hash('oxalic'):
                            for z in range(len(formula_pieces)):
                                if formula_pieces[z] == 'C2 H2 O4' or '(C2
                                H2 O4)' in formula_pieces[z]:
                                    if
                                    hash(acid_neutral_dictionary.get(hash(pie
                                    ces[c-1]))) not in component_SMILES:

                                        component_SMILES.append(hash(acid_neu
                                        tral_dictionary.get(hash(pieces[c-1])
                                        )))
                                if formula_pieces[z] == 'C2 H1 O4 1-' or
                                '(C2 H1 O4 1-)' in formula_pieces[z]:
                                    if
                                    hash(acid_charged_dictionary.get(hash(pie
                                    ces[c-1]))) not in component_SMILES:

                                        component_SMILES.append(hash(acid_cha
                                        rged_dictionary.get(hash(pieces[c-1])
                                        )))
                        elif hash(pieces[c-1]) == hash('nitric'):
                            for z in range(len(formula_pieces)):
                                if formula_pieces[z] == 'H1 N1 O3' or '(H1
                                N1 O3)' in formula_pieces[z]:
                                    if
                                    hash(acid_neutral_dictionary.get(hash(pie
                                    ces[c-1]))) not in component_SMILES:

                                        component_SMILES.append(hash(acid_neu
                                        tral_dictionary.get(hash(pieces[c-1])
                                        )))
                                if formula_pieces[z] == 'N1 O3 1-' or '(N1
                                O3 1-)' in formula_pieces[z]:
                                    if
                                    hash(acid_charged_dictionary.get(hash(pie
                                    ces[c-1]))) not in component_SMILES:

                                        component_SMILES.append(hash(acid_cha
```

```
                                rged_dictionary.get(hash(pieces[c-1])
                                )))
195                         elif hash(pieces[c-1]) ==
                            hash('trans-but-2-enedioic') or hash(pieces[c-1])
                            == hash('but-2-enedioic') or hash(pieces[c-1]) ==
                            hash('fumaric'):
196                             for z in range(len(formula_pieces)):
197                                 if formula_pieces[z] == 'C4 H4 O4' or '(C4
                                    H4 O4)' in formula_pieces[z]:
198                                     if
                                        hash(acid_neutral_dictionary.get(hash(pie
                                        ces[c-1]))) not in component_SMILES:
199
                                            component_SMILES.append(hash(acid_neu
                                            tral_dictionary.get(hash(pieces[c-1])
                                            )))
200                                 if formula_pieces[z] == 'C4 H3 O4 1-' or
                                    '(C4 H3 O4 1-)' in formula_pieces[z]:
201                                     if
                                        hash(acid_charged_dictionary.get(hash(pie
                                        ces[c-1]))) not in component_SMILES:
202
                                            component_SMILES.append(hash(acid_cha
                                            rged_dictionary.get(hash(pieces[c-1])
                                            )))
203                         elif hash(pieces[c-1]) == hash('phosphonic'):
204                             for z in range(len(formula_pieces)):
205                                 if formula_pieces[z] == 'H3 O3 P1' or '(H3
                                    O3 P1)' in formula_pieces[z]:
206                                     if
                                        hash(acid_neutral_dictionary.get(hash(pie
                                        ces[c-1]))) not in component_SMILES:
207
                                            component_SMILES.append(hash(acid_neu
                                            tral_dictionary.get(hash(pieces[c-1])
                                            )))
208                                 if formula_pieces[z] == 'H2 O3 P1 1-' or
                                    '(H2 O3 P1 1-)' in formula_pieces[z]:
209                                     if
                                        hash(acid_charged_dictionary.get(hash(pie
                                        ces[c-1]))) not in component_SMILES:
210
                                            component_SMILES.append(hash(acid_cha
                                            rged_dictionary.get(hash(pieces[c-1])
                                            )))
211                         elif hash(pieces[c-1]) == hash('perchloric'):
212                             for z in range(len(formula_pieces)):
213                                 if formula_pieces[z] == 'H1 Cl1 O4' or '(H1
                                    Cl1 O4)' in formula_pieces[z]:
214                                     if
                                        hash(acid_neutral_dictionary.get(hash(pie
                                        ces[c-1]))) not in component_SMILES:
215
                                            component_SMILES.append(hash(acid_neu
                                            tral_dictionary.get(hash(pieces[c-1])
                                            )))
216                                 if formula_pieces[z] == 'Cl1 O4 1-' or
                                    '(Cl1 O4 1-)' in formula_pieces[z]:
217                                     if
                                        hash(acid_charged_dictionary.get(hash(pie
                                        ces[c-1]))) not in component_SMILES:
218
                                            component_SMILES.append(hash(acid_cha
                                            rged_dictionary.get(hash(pieces[c-1])
                                            )))
219                         elif hash(pieces[c-1]) == hash('phosphoric'):
220                             for z in range(len(formula_pieces)):
```

```
                        if formula_pieces[z] == 'H3 O4 P1' or '(H3 O4 P1)' in formula_pieces[z]:
                            if hash(acid_neutral_dictionary.get(hash(pieces[c-1]))) not in component_SMILES:
                                component_SMILES.append(hash(acid_neutral_dictionary.get(hash(pieces[c-1]))))
                        if formula_pieces[z] == 'H2 O4 P1 1-' or '(H2 O4 P1 1-)' in formula_pieces[z]:
                            if hash(acid_charged_dictionary.get(hash(pieces[c-1]))) not in component_SMILES:
                                component_SMILES.append(hash(acid_charged_dictionary.get(hash(pieces[c-1]))))
                elif hash(pieces[c-1]) == hash('pyridine-2,6-dicarboxylic'):
                    for z in range(len(formula_pieces)):
                        if formula_pieces[z] == 'C7 H5 N1 O4' or '(C7 H5 N1 O4)' in formula_pieces[z]:
                            if hash(acid_neutral_dictionary.get(hash(pieces[c-1]))) not in component_SMILES:
                                component_SMILES.append(hash(acid_neutral_dictionary.get(hash(pieces[c-1]))))
                        if formula_pieces[z] == 'C7 H4 N1 O4 1-' or '(C7 H4 N1 O4 1-)' in formula_pieces[z]:
                            if hash(acid_charged_dictionary.get(hash(pieces[c-1]))) not in component_SMILES:
                                component_SMILES.append(hash(acid_charged_dictionary.get(hash(pieces[c-1]))))
                elif hash(pieces[c-1]) == hash('benzene-1,2,4,5-tetracarboxylic'):
                    for z in range(len(formula_pieces)):
                        if formula_pieces[z] == 'C10 H6 O8' or '(C10 H6 O8)' in formula_pieces[z]:
                            if hash(acid_neutral_dictionary.get(hash(pieces[c-1]))) not in component_SMILES:
                                component_SMILES.append(hash(acid_neutral_dictionary.get(hash(pieces[c-1]))))
                        if formula_pieces[z] == 'C10 H4 O8 2-' or '(C10 H4 O8 2-)' in formula_pieces[z]:
                            if hash(acid_charged_dictionary.get(hash(pieces[c-1]))) not in component_SMILES:
                                component_SMILES.append(hash(acid_charged_dictionary.get(hash(pieces[c-1]))))
                elif hash(pieces[c-1]) == hash('trifluoroacetatic'):
                    for z in range(len(formula_pieces)):
                        if formula_pieces[z] == 'C2 H1 F3 O2' or '(C2 H1 F3 O2)' in formula_pieces[z]:
                            if hash(acid_neutral_dictionary.get(hash(pieces[c-1]))) not in component_SMILES:
```

```
247                                                 component_SMILES.append(hash(acid_neu
                                                    tral_dictionary.get(hash(pieces[c-1])
                                                    )))
248                                             if formula_pieces[z] == 'C2 F3 O2 1-' or
                                                '(C2 F3 O2 1-)' in formula_pieces[z]:
249                                                 if
                                                    hash(acid_charged_dictionary.get(hash(pie
                                                    ces[c-1]))) not in component_SMILES:
250
                                                    component_SMILES.append(hash(acid_cha
                                                    rged_dictionary.get(hash(pieces[c-1])
                                                    )))
251                                 elif hash(pieces[c]) in deuterated_dictionary:
252                                     if hash(deuterated_dictionary.get(hash(pieces[c]))) not
                                        in component_SMILES:
253
                                         component_SMILES.append(hash(deuterated_dictionary.ge
                                         t(hash(pieces[c]))))
254                                     if hash(pieces[c]) == hash('dideutero-dichloromethane')
                                        or hash(pieces[c]) == hash('dideuterodichloromethane'):
255                                         if hash('Cl[C]Cl') in component_SMILES:
256                                             component_SMILES.remove(hash('Cl[C]Cl'))
257                                     elif hash(pieces[c]) == hash('perdeutero-toluene'):
258                                         if hash('[C]c1[c][c][c][c][c][c]1') in component_SMILES:
259
                                             component_SMILES.remove(hash('[C]c1[c][c][c][c][c
                                             ]1'))
260                                     elif hash(pieces[c]) == hash('deutero-ethanol'):
261                                         if hash('[C][C][O]') in component_SMILES:
262                                             component_SMILES.remove(hash('[C][C][O]'))
263                                     elif hash(pieces[c]) == hash('deuterochloroform') or
                                        hash(pieces[c]) == hash('deutero-chloroform'):
264                                         if hash('Cl[C](Cl)Cl') in component_SMILES:
265                                             component_SMILES.remove(hash('Cl[C](Cl)Cl'))
266                                     elif hash(pieces[c]) == hash('hexadeutero-benzene') or
                                        hash(pieces[c]) == hash('deuterobenzene') or
                                        hash(pieces[c]) == hash('hexadeuterobenzene'):
267                                         if hash('[c]1[c][c][c][c][c][c]1') in component_SMILES:
268
                                             component_SMILES.remove(hash('[c]1[c][c][c][c][c]
                                             1'))
269                                     elif hash(pieces[c]) == hash('deuteromethanol') or
                                        hash(pieces[c]) == hash('perdeuteromethanol'):
270                                         if hash('[C][O]') in component_SMILES:
271                                             component_SMILES.remove(hash('[C][O]'))
272                                 elif hash(pieces[c]) in solvents_dictionary:
273                                     if hash(solvents_dictionary.get(hash(pieces[c]))) not
                                        in component_SMILES:
274
                                         component_SMILES.append(hash(solvents_dictionary.get(
                                         hash(pieces[c]))))
275
276                             if hash('O') in component_SMILES:
277                                 component_SMILES.remove(hash('O'))
278                             if hash('[O]') in component_SMILES:
279                                 component_SMILES.remove(hash('[O]'))
280
281                             component_SMILES = sorted(component_SMILES)
282                             solvated_hydrates_SMILES.append(component_SMILES)
283
284
285                         print len(solvated_hydrates_SMILES)
286                         print len(solvated_hydrates_hash)
287                         print len(solvated_hydrates_entry)
288
289                         while len(solvated_hydrates_SMILES) != 0:
```

```
                print len(solvated_hydrates_SMILES)
                identifiers = []
                number = []
                identifiers.append(solvated_hydrates_hash[0])

                number.append(solvated_hydrates_entry[0].crystal.spacegroup_number_and_se
                tting[0])
                #The first hydrate structure entry SMILES string is compared against
                all the other hydrate structure entry SMILES strings
                for d in range(1,len(solvated_hydrates_SMILES)):
                    #Hydrate structures that match the entry SMILES string are put into
                    a list with the first hydrate structure
                    #The spacegroup number of all matches are recorded in another list
                    if solvated_hydrates_SMILES[0] == solvated_hydrates_SMILES[d]:
                        identifiers.append(solvated_hydrates_hash[d])

                        number.append(solvated_hydrates_entry[d].crystal.spacegroup_numbe
                        r_and_setting[0])

                if len(identifiers) != 1:
                    #The water stoichiometry of each match is determined the same way
                    it was for hydrate structures without entry SMILES strings
                    waters = []
                    for g in range(len(identifiers)):
                        pin = solvated_hydrates_hash.index(identifiers[g])
                        title = solvated_hydrates_entry[pin].formula
                        pieces = title.split(',')
                        water_molecule = []
                        for e in range(len(pieces)):
                            if '(H2 O1)' in pieces[e] or pieces[e] == 'H2 O1' or '(D2
                            O1)' in pieces[e] or pieces[e] == 'D2 O1':
                                water_molecule.append(pieces[e])
                        if water_molecule == [] or len(water_molecule) > 1:
                            waters.append('undefined')
                        elif water_molecule[0] == 'H2 O1' or water_molecule[0] == 'D2
                        O1':
                            waters.append(1)
                        else:
                            water = str(water_molecule[0])
                            end = water.index('(')
                            water = water[:end]
                            if 'x' in water:
                                waters.append('x')
                            elif 'n' in water:
                                waters.append('n')
                            elif water.isdigit() == False:
                                waters.append(Decimal(water))
                            else:
                                waters.append(int(water))
                    #Once again, any cases where the water stoichiometry or spacegroup
                    number match undergo further analysis
                    if any(waters.count(x) > 1 for x in waters) or all(x == number[0]
                    for x in number) or any(number.count(y) > 1 for y in number):
                        identifiers_to_remove = []
                        while len(identifiers) != len(identifiers_to_remove):
                            for h in range(len(identifiers)):
                                if identifiers[h] not in identifiers_to_remove:
                                    identifiers_to_remove.append(identifiers[h])
                                    if waters.count(waters[h]) > 1:
                                        match = 0
                                        for i in range(len(identifiers)):
                                            if identifiers[i] not in
                                            identifiers_to_remove:
                                                if waters[h] == waters[i]:
                                                    if number[h] == number[i]:
                                                        if match == 0:
                                                            pin1 =
```

```
                                                    solvated_hydrates_hash.index(
                                                    identifiers[h])
345
                                                    duplicates_writer.write(solva
                                                    ted_hydrates_entry[pin1])
346                                                 pin2 =
                                                    solvated_hydrates_hash.index(iden
                                                    tifiers[i])
347
                                                    duplicates_writer.write(solvated_
                                                    hydrates_entry[pin2])
348                                                 match += 1
349
                                                    identifiers_to_remove.append(iden
                                                    tifiers[i])
350                                         if match != 0:
351
                                                duplicates_writer.write(solvated_hydrates_ent
                                                ry[pin1])
352
353                 #The overall list of hydrates is updated to remove the set of
                    structures that was analyzed
354                 for j in range(len(identifiers)):
355                     pin = solvated_hydrates_hash.index(identifiers[j])
356                     solvated_hydrates_hash.remove(solvated_hydrates_hash[pin])
357                     solvated_hydrates_entry.remove(solvated_hydrates_entry[pin])
358                     solvated_hydrates_SMILES.remove(solvated_hydrates_SMILES[pin])
359
360
361             print 'solvated hydrates list is empty'
362
363
364 if __name__ == '__main__':
365     # This runs the script
366     r = Runner()
367     r.run()
368
```

```
#9 Find duplicate SMILES hydrate structures with packing similarity

from ccdc import io, search
import argparse
import os
import glob
import re
import itertools
from decimal import Decimal
from ccdc.crystal import PackingSimilarity
import time

similarity_engine = PackingSimilarity()
similarity_engine.settings.ignore_hydrogen_positions = True

#This script looks at the potential duplicates identified in the previous script
#This script uses the packing similarity tool which gets stuck at certain entries as identified in a later comment
filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\hydrates_smiles_all_potential_duplicates.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='hydrates_smiles_all_first_occurrence.gcd',
            help='output file [hydrates_smiles_all_first_occurrence.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')

        with io.EntryWriter(self.args.output) as lone_wolf_writer:
            with io.EntryWriter("hydrates_smiles_all_duplicates.gcd") as writer1:
                with io.EntryWriter("hydrates_smiles_all_manual_check.gcd") as writer2:

                    solvated_hydrates_hash = []
                    solvated_hydrates_entry = []

                    #Split input file at entries: 1362, 1365, 2329, 2332
                    #The following entries get stuck at analysis calculation: GUQBOJ, NISNAF
                    #Removing these potential duplicates of these entries from the input file for a manual check before running the script should prevent stalling
                    for a in range(2332, len(entry_reader1)):
                        solvated_hydrates_hash.append(hash(entry_reader1[a].identifier))
                        solvated_hydrates_entry.append(entry_reader1[a])

                    while len(solvated_hydrates_hash) != 0:
                        print len(solvated_hydrates_hash)
```

```
                    identifiers = []
                    identifiers.append(solvated_hydrates_hash[0])
                    #Once again the set of potential duplicates is found by
                    checking for the second occurrence of the starting identifier
                    for d in range(1,len(solvated_hydrates_hash)):
                        if identifiers[0] != solvated_hydrates_hash[d]:
                            identifiers.append(solvated_hydrates_hash[d])
                        else:
                            del solvated_hydrates_hash[d]
                            del solvated_hydrates_entry[d]
                            break
                    remove_identifiers = []
                    while len(identifiers) != len(remove_identifiers):
                        for s in range(len(identifiers)):
                            count = 0
                            if identifiers[s] not in remove_identifiers:
                                pin1 = solvated_hydrates_hash.index(identifiers[s])
                                reference = solvated_hydrates_entry[pin1].crystal
                                remove_identifiers.append(identifiers[s])
                                #Here each hydrate structure in the set of
                                potential duplicates is compared to subsequent
                                hydrate structures
                                #The script iterates through the list so cases with
                                polymorphs are isolated
                                #For example, if my list of matches has [A, B, C,
                                D] and A and C are duplicates and B and D are
                                duplicates
                                #When A is checked for duplicates it will match C,
                                A and C will then be removed from the list so it is
                                now [B, D]
                                #Then B will be checked for duplicates and D will
                                return as a match
                                for f in range(len(identifiers)):
                                    if identifiers[f] not in remove_identifiers:
                                        pin2 =
                                        solvated_hydrates_hash.index(identifiers[f])
                                        sample =
                                        solvated_hydrates_entry[pin2].crystal
                                        #The packing similarity between two hydrate
                                        structures is calculated
                                        analysis =
                                        similarity_engine.compare(reference, sample)
                                        #If the analysis returns None, the hydrate
                                        structures are written to a list for a
                                        manual check
                                        if analysis == None:
                                            if count == 0:

                                                writer2.write(solvated_hydrates_entry
                                                [pin1])
                                                count += 1

                                            writer2.write(solvated_hydrates_entry[pin
                                            2])
                                            remove_identifiers.append(identifiers[f])
                                        else:
                                            #If the packing similarity returns 1
                                            (15 out of 15 packing points are the
                                            same) then the two hydrate structures
                                            are considered duplicates
                                            if
                                            analysis.nmatched_molecules/analysis.pack
                                            ing_shell_size == 1:
                                                #The first hydrate structure in the
                                                set is written to a file that
                                                contains the hydrate structures to
                                                keep
```

```
103                                                     #The second hydrate structure in
                                                        the set is written to a file that
                                                        contains the hydrate structures to
                                                        remove
104
                                                        lone_wolf_writer.write(solvated_hydra
                                                        tes_entry[pin1])
105
                                                        writer1.write(solvated_hydrates_entry
                                                        [pin2])
106
                                                        remove_identifiers.append(identifiers
                                                        [f])
107                                         if count != 0:
108                                             writer2.write(solvated_hydrates_entry[pin1])
109                                 #The overall list of hydrate structures is updated to remove
                                    those already analyzed
110                                 for j in range(len(identifiers)):
111                                     pin = solvated_hydrates_hash.index(identifiers[j])
112                                     solvated_hydrates_hash.remove(solvated_hydrates_hash[pin])
113                                     solvated_hydrates_entry.remove(solvated_hydrates_entry[pin])
114
115                             print 'solvated hydrates list is empty'
116
117 if __name__ == '__main__':
118     # This runs the script
119     r = Runner()
120     r.run()
121
```

```python
#10 Find potential duplicate none waterless forms with formula match

from ccdc import io, search
import argparse
import os
import glob
import re
from decimal import Decimal

#This script is the same as the part1 script to find duplicates for hydrate structures without entry SMILES strings
#The only difference is that the water stoichiometry of each entry does not need to be determined
#In order for structures with the same formula to be considered potential duplicates there needs to be at least two structures with the same spacegroup number
filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\complete_None_waterless_forms.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\complete_smiles_waterless_forms.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output',
            default='waterless_forms_None_all_potential_duplicates.gcd',
            help='output file [waterless_forms_None_all_potential_duplicates.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')
        entry_reader2 = io.EntryReader(filepath2, format='identifiers')

        with io.EntryWriter(self.args.output) as new_lists_writer:

            None_formulas = []
            formulas = []
            positions = []
            numbers = []
            for a in range(len(entry_reader1)):
                component_formulas = entry_reader1[a].formula.split(',')
                pre_formulas = []
                for b in range(len(component_formulas)):
                    if '(' and ')' in component_formulas[b]:
                        start = component_formulas[b].index('(')
                        end = component_formulas[b].index(')')
                        component_formulas[b] = component_formulas[b][start+1:end]
                    elements = component_formulas[b].split(' ')
                    deuteriums = [i for i in elements if re.sub(r'[0-9]+', '', i) == 'D']
                    if deuteriums != []:
                        if len(deuteriums) != 1:
```

```
                            print entry_reader1[a].identifier
                        hydrogens = [i for i in elements if re.sub(r'[0-9]+', '', i) ==
                        'H']
                        if hydrogens == []:
                            spot = elements.index(deuteriums[0])
                            deuteriums[0] = deuteriums[0].replace('D', 'H')
                            elements[spot] = deuteriums[0]
                        else:
                            if len(hydrogens) != 1:
                                print entry_reader1[a].identifier
                            spot = elements.index(deuteriums[0])
                            spot2 = elements.index(hydrogens[0])
                            stoichiometry1 = re.sub(r'[A-Z]+', '', deuteriums[0])
                            stoichiometry2 = re.sub(r'[A-Z]+', '', hydrogens[0])
                            total = int(stoichiometry1) + int(stoichiometry2)
                            hydrogens[0] = hydrogens[0].replace(stoichiometry2,
                            str(total))
                            elements[spot2] = hydrogens[0]
                            elements.remove(elements[spot])
                        component_formulas[b] = ' '.join(elements)
                    pre_formulas.append(hash(component_formulas[b]))
                pre_formulas = list(set(pre_formulas))
                pre_formulas = sorted(pre_formulas)
                None_formulas.append(pre_formulas)
                formulas.append(pre_formulas)
                positions.append(a)
                if hash(entry_reader1[a].identifier) == hash('KECYBU15'):
                    numbers.append(87)
                else:

                    numbers.append(entry_reader1[a].crystal.spacegroup_number_and_setting
                    [0])

            for d in range(len(entry_reader2)):
                component_formulas = entry_reader2[d].formula.split(',')
                pre_formulas2 = []
                for e in range(len(component_formulas)):
                    if '(' and ')' in component_formulas[e]:
                        start = component_formulas[e].index('(')
                        end = component_formulas[e].index(')')
                        component_formulas[e] = component_formulas[e][start+1:end]
                    elements = component_formulas[e].split(' ')
                    deuteriums = [i for i in elements if re.sub(r'[0-9]+', '', i) == 'D']
                    if deuteriums != []:
                        if len(deuteriums) != 1:
                            print entry_reader2[d].identifier
                        hydrogens = [i for i in elements if re.sub(r'[0-9]+', '', i) ==
                        'H']
                        if hydrogens == []:
                            spot = elements.index(deuteriums[0])
                            deuteriums[0] = deuteriums[0].replace('D', 'H')
                            elements[spot] = deuteriums[0]
                        else:
                            if len(hydrogens) != 1:
                                print entry_reader2[d].identifier
                            spot = elements.index(deuteriums[0])
                            spot2 = elements.index(hydrogens[0])
                            stoichiometry1 = re.sub(r'[A-Z]+', '', deuteriums[0])
                            stoichiometry2 = re.sub(r'[A-Z]+', '', hydrogens[0])
                            total = int(stoichiometry1) + int(stoichiometry2)
                            hydrogens[0] = hydrogens[0].replace(stoichiometry2,
                            str(total))
                            elements[spot2] = hydrogens[0]
                            elements.remove(elements[spot])
                        component_formulas[e] = ' '.join(elements)
                    pre_formulas2.append(hash(component_formulas[e]))
                pre_formulas2 = list(set(pre_formulas2))
```

```
123                 pre_formulas2 = sorted(pre_formulas2)
124                 formulas.append(pre_formulas2)
125                 positions.append(d+5450)
126                 if hash(entry_reader2[d].identifier) == hash('MTYHFB03'):
127                     numbers.append(29)
128                 else:
129
                        numbers.append(entry_reader2[d].crystal.spacegroup_number_and_setting
                        [0])
130
131             while None_formulas != []:
132                 print len(None_formulas)
133                 formulas_to_remove = []
134                 positions_to_write = []
135                 numbers_to_compare = []
136                 formulas_to_remove.append(formulas[0])
137                 positions_to_write.append(positions[0])
138                 numbers_to_compare.append(numbers[positions[0]])
139                 for c in range(1, len(formulas)):
140                     if formulas[0] == formulas[c]:
141                         positions_to_write.append(positions[c])
142                         numbers_to_compare.append(numbers[positions[c]])
143
144                 if len(numbers_to_compare) != 1:
145                     if all(x == numbers_to_compare[0] for x in numbers_to_compare) or
                        any(numbers_to_compare.count(x) > 1 for x in numbers_to_compare):
146                         positions_to_remove = []
147                         while len(positions_to_write) != len(positions_to_remove):
148                             for j in range(len(positions_to_write)):
149                                 if positions_to_write[j] not in positions_to_remove:
150                                     positions_to_remove.append(positions_to_write[j])
151                                     if positions_to_write[j] < 5450:
152                                         match = 0
153                                         for k in range(len(positions_to_write)):
154                                             if positions_to_write[k] not in
                                                positions_to_remove:
155                                                 if numbers_to_compare[j] ==
                                                    numbers_to_compare[k]:
156                                                     if match == 0:
157
                                                            new_lists_writer.write(entry_read
                                                            er1[positions_to_write[j]])
158                                                     if positions_to_write[k] < 5450:
159
                                                            new_lists_writer.write(entry_read
                                                            er1[positions_to_write[k]])
160                                                     else:
161
                                                            new_lists_writer.write(entry_read
                                                            er2[positions_to_write[k]-5450])
162                                                     match += 1
163
                                                        positions_to_remove.append(positions_
                                                        to_write[k])
164                                         if match != 0:
165
                                                new_lists_writer.write(entry_reader1[position
                                                s_to_write[j]])
166
167                 None_formulas = [i for i in None_formulas if i not in formulas_to_remove]
168                 formulas = [i for i in formulas if i not in formulas_to_remove]
169                 positions = [j for j in positions if j not in positions_to_write]
170
171
172
173
174 if __name__ == '__main__':
```

```
175         r = Runner()
176         r.run()
177
```

```
#11 Find potential duplicate none waterless forms with same identifier prefix

from ccdc import io, search
import argparse
import os
import glob
import re
from decimal import Decimal

#This script is the same as the part2 script to find duplicates for hydrate structures without entry SMILES strings
filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_None_all_potential_duplicates.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='waterless_forms_None_all_manual_check.gcd',
            help='output file [waterless_forms_None_all_manual_check.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')

        with io.EntryWriter(self.args.output) as new_lists_writer:

            duplicate_identifier_dictionary = {'GADKOL':1, 'MECYEY':2, 'OBASAN':3, 'OMOMAE':4, 'OTONUI':5, 'QOJWAO':6, 'TZBFZO':7, 'VOBXOZ':8, 'VUFJAI':9, 'VUFJAJ':10, 'WEPQOA':11, 'XADFUD':12, 'YABREY':13, 'YULCOZ':14, 'ZOVFEV':15}

            solvated_waterless_hash = []
            solvated_waterless_entry = []
            for a in range(len(entry_reader1)):
                if entry_reader1[a].identifier in duplicate_identifier_dictionary:

                    solvated_waterless_hash.append(duplicate_identifier_dictionary.get(entry_reader1[a].identifier))
                else:
                    solvated_waterless_hash.append(hash(entry_reader1[a].identifier))
                solvated_waterless_entry.append(entry_reader1[a])

            while len(solvated_waterless_hash) != 0:
                print len(solvated_waterless_hash)
                identifiers = []
                identifiers.append(solvated_waterless_hash[0])
                for d in range(1,len(solvated_waterless_hash)):
                    if identifiers[0] != solvated_waterless_hash[d]:
                        identifiers.append(solvated_waterless_hash[d])
                    else:
                        del solvated_waterless_hash[d]
```

```
                        del solvated_waterless_entry[d]
                        break

                nomenclature = []
                for e in range(len(identifiers)):
                    pin = solvated_waterless_hash.index(identifiers[e])
                    nomenclature.append(solvated_waterless_entry[pin].identifier[:6])

                if all(x == x[0] for x in nomenclature):
                    pin1 = solvated_waterless_hash.index(identifiers[0])
                    new_lists_writer.write(solvated_waterless_entry[pin1])
                    for f in range(1, len(identifiers)):
                        pin2 = solvated_waterless_hash.index(identifiers[f])
                        new_lists_writer.write(solvated_waterless_entry[pin2])
                    new_lists_writer.write(solvated_waterless_entry[pin1])
                else:
                    remove_identifiers = []
                    while len(identifiers) != len(remove_identifiers):
                        for h in range(len(identifiers)):
                            if identifiers[h] not in remove_identifiers:
                                remove_identifiers.append(identifiers[h])
                                if nomenclature.count(nomenclature[h]) > 1:
                                    pin3 = solvated_waterless_hash.index(identifiers[h])

                                    new_lists_writer.write(solvated_waterless_entry[pin3]
                                    )
                                    for i in range(len(identifiers)):
                                        if identifiers[i] not in remove_identifiers:
                                            if nomenclature[h] == nomenclature[i]:
                                                remove_identifiers.append(identifiers[i])
                                                pin4 =
                                                solvated_waterless_hash.index(identifiers
                                                [i])

                                                new_lists_writer.write(solvated_waterless
                                                _entry[pin4])

                                    new_lists_writer.write(solvated_waterless_entry[pin3]
                                    )

                for j in range(len(identifiers)):
                    pin5 = solvated_waterless_hash.index(identifiers[j])
                    solvated_waterless_hash.remove(solvated_waterless_hash[pin5])
                    solvated_waterless_entry.remove(solvated_waterless_entry[pin5])


if __name__ == '__main__':
    r = Runner()
    r.run()
```

```
#12 Find potential duplicate SMILES waterless forms with formula match

from ccdc import io, search
import argparse
import os
import glob
import re

#This script is different from the part1 script to find duplicates for hydrate structures with entry SMILES strings
#The work that script does for hydrates had to be broken into two for anhydrous forms since there are far more anhydrous forms with entry SMILES strings than hydrates with SMILES strings
#This script carries out the first part which is checking the anhydrous forms for formula matches
#This is done the same way as in the part1 script to find duplicates for anhydrous forms without entry SMILES strings
#This was done due to the large amount of time required to check all the anhydrous forms in the solvent dictionary
#This way time won't be wasted looking in the solvent dictionary for structures that do not match the formula of any other anhydrous forms pulled from the CSD
#Structures that don't match the formula of any other structure cannot have a duplicate in the list obtained for this work
filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\complete_smiles_waterless_forms.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='waterless_forms_smiles_all_repeated_formula.gcd',
            help='output file [waterless_forms_smiles_all_repeated_formula.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')

        with io.EntryWriter(self.args.output) as duplicates_writer:

            duplicate_identifier_dictionary = {'GADKOL':1, 'MECYEY':2, 'OBASAN':3, 'OMOMAE':4, 'OTONUI':5, 'QOJWAO':6, 'TZBFZO':7, 'VOBXOZ':8, 'VUFJAI':9, 'VUFJAJ':10, 'WEPQOA':11, 'XADFUD':12, 'YABREY':13, 'YULCOZ':14, 'ZOVFEV':15}

            formulas = []
            positions = []
            numbers = []
            for a in range(len(entry_reader1)):
                print a
                component_formulas = entry_reader1[a].formula.split(',')
                pre_formulas = []
                for b in range(len(component_formulas)):
```

```
                    if '(' and ')' in component_formulas[b]:
                        start = component_formulas[b].index('(')
                        end = component_formulas[b].index(')')
                        component_formulas[b] = component_formulas[b][start+1:end]
                    elements = component_formulas[b].split(' ')
                    deuteriums = [i for i in elements if re.sub(r'[0-9]+', '', i) == 'D']
                    if deuteriums != []:
                        if len(deuteriums) != 1:
                            #These lines terminate the script and act as a safety for
                            cases that are not accounted for
                            #With such a large set of structures, extraneous cases are
                            hard to predict
                            #To save time, these lines are put in to catch extraneous
                            cases, that way when there are not any time is not wasted
                            accounting for them
                            print entry_reader1[a].identifier
                            print deuteriums[90]
                        hydrogens = [i for i in elements if re.sub(r'[0-9]+', '', i) ==
                        'H']
                        if hydrogens == []:
                            spot = elements.index(deuteriums[0])
                            deuteriums[0] = deuteriums[0].replace('D', 'H')
                            elements[spot] = deuteriums[0]
                        else:
                            if len(hydrogens) != 1:
                                print entry_reader1[a].identifier
                                print hydrogens[90]
                            spot = elements.index(deuteriums[0])
                            spot2 = elements.index(hydrogens[0])
                            stoichiometry1 = re.sub(r'[A-Z]+', '', deuteriums[0])
                            stoichiometry2 = re.sub(r'[A-Z]+', '', hydrogens[0])
                            total = int(stoichiometry1) + int(stoichiometry2)
                            hydrogens[0] = hydrogens[0].replace(stoichiometry2,
                            str(total))
                            elements[spot2] = hydrogens[0]
                            elements.remove(elements[spot])
                        component_formulas[b] = ' '.join(elements)
                    pre_formulas.append(hash(component_formulas[b]))
                pre_formulas = list(set(pre_formulas))
                pre_formulas = sorted(pre_formulas)
                formulas.append(pre_formulas)
                positions.append(a)
                if hash(entry_reader1[a].identifier) == hash('KECYBU15'):
                    numbers.append(87)
                elif hash(entry_reader1[a].identifier) == hash('MTYHFB03'):
                    numbers.append(29)
                else:

                    numbers.append(entry_reader1[a].crystal.spacegroup_number_and_setting
                    [0])

            while formulas != []:
                print len(formulas)
                formulas_to_remove = []
                positions_to_write = []
                numbers_to_compare = []
                formulas_to_remove.append(formulas[0])
                positions_to_write.append(positions[0])
                numbers_to_compare.append(numbers[positions[0]])
                for c in range(1, len(formulas)):
                    if formulas[0] == formulas[c]:
                        positions_to_write.append(positions[c])
                        numbers_to_compare.append(numbers[positions[c]])

                if len(numbers_to_compare) != 1:
                    if all(x == numbers_to_compare[0] for x in numbers_to_compare) or
                    any(numbers_to_compare.count(x) > 1 for x in numbers_to_compare):
```

```
                        positions_to_remove = []
                        for h in range(len(positions_to_write)):
                            times1 = numbers_to_compare.count(numbers_to_compare[h])
                            if times1 == 1:
                                positions_to_remove.append(positions_to_write[h])
                        positions_to_check = [z for z in positions_to_write if z not in
                        positions_to_remove]
                        if positions_to_check != []:
                            for i in range(len(positions_to_check)):

                                duplicates_writer.write(entry_reader1[positions_to_check[
                                i]])
                            duplicates_writer.write(entry_reader1[positions_to_check[0]])

                formulas = [i for i in formulas if i not in formulas_to_remove]
                positions = [j for j in positions if j not in positions_to_write]

            print 'waterless formulas list is empty'


if __name__ == '__main__':
    r = Runner()
    r.run()

```

```
#13 Find potential duplicate SMILES waterless forms with SMILES string match

from ccdc import io, search
import argparse
import os
import glob
import re
import itertools
from decimal import Decimal
from ccdc.crystal import PackingSimilarity

similarity_engine = PackingSimilarity()
similarity_engine.settings.distance_tolerance = 0.25
similarity_engine.settings.angle_tolerance = 25.

#This script carries out the second part the previous script
#This script is written the same as the part1 script to find duplicates for hydrate structures with entry SMILES strings aside from two differences
#One difference is that the water stoichiometry of each entry does not need to be determined
#In order for structures with the same formula to be considered potential duplicates there needs to be at least two structures with the same spacegroup number
#The other difference is that anhydrous forms known to match the formula of other anhydrous forms are being screened rather than all the anhydrous forms with entry SMILES strings
filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_smiles_all_repeated_formula.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output',
            default='waterless_forms_smiles_all_potential_duplicates.gcd',
            help='output file [waterless_forms_smiles_all_potential_duplicates.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')

        with io.EntryWriter(self.args.output) as duplicates_writer:

            solvents_dictionary = {hash('but-2-ene'):'CC=CC',
            hash('1,1,2-dichloroethane'):'ClCC(Cl)Cl',
            hash('4-aminopyridine'):'Nc1ccncc1', hash('N-methylurea'):'CNC(N)=O',
            hash('2-methylphenol'):'Cc1ccccc1O', hash('tetraglyme'):'COCCOCCOCCOCCOC',
            hash('isophorone'):'CC1=CC(=O)CC(C)(C)C1',
            hash('L-carvone'):'CC(=C)C1CC=C(C)C(=O)C1',
            hash('2-cyanopyridine'):'N#Cc1ccccn1', hash('biuret'):'NC(=O)NC(N)=O',
            hash('styrene'):'C=Cc1ccccc1', hash('carbondisulfide'):'S=C=S',
            hash('tricyanomethanide'):'N#C[C-](C#N)C#N',
```

```
hash('methyl-cyclohexane'):'CC1CCCCC1',
hash('pyridine-3-carboxamide'):'NC(=O)c1cccnc1',
hash('3,7-dimethyl-3,7-dihydro-1H-purine-2,6-dione'):'CN1C=NC2=C1C(=O)NC(=O)N
2C', hash('nitroethane'):'CCN(=O)=O',
hash('dimethylaminobenzaldehyde'):'CN(C)c1ccc(C=O)cc1',
hash('3-(1H-pyrrol-1-yl)propanenitrile'):'C1=CN(C=C1)CCC#N',
hash('trifluoromethanol'):'OC(F)(F)F', hash('methoxyethane'):'CCOC',
hash('methylanisole'):'COc1ccccc1C', hash('xylene'):'Cc1ccc(C)cc1',
hash('petrol'):'CCCCCC', hash('n-alkane'):'CCCCC',
hash('dichloromethanal'):'ClC(Cl)=O',
hash('1-fluoro-4-methylbenzene'):'Cc1ccc(F)cc1',
hash('1,2,3-trimethoxybenzene'):'COc1cccc(OC)c1OC',
hash('1,3,5-trimethoxybenzene'):'COc1cc(OC)cc(OC)c1',
hash('chloroform'):'C1C(Cl)Cl', hash('methanol'):'CO',
hash('hexane'):'CCCCCC', hash('ethanol'):'CCO',
hash('acetonitrile'):'CC#N', hash('toluene'):'Cc1ccccc1',
hash('tetrahydrofuran'):'C1CCOC1', hash('acetone'):'CC(C)=O',
hash('dichloromethane'):'ClCCl', hash('1,2-dichloroethane'):'ClCCCl',
hash('dimethylsulfoxide'):'CS(C)=O', hash('1,4-dioxane'):'C1COCCO1',
hash('1,2-dimethoxyethane'):'COCCOC', hash('o-xylene'):'Cc1ccccc1c',
hash('propanol'):'CCO', hash('dioxane'):'C1COCCO1',
hash('n-pentane'):'CCCCC', hash('4-cyanopyridine'):'N#Cc1ccncc1',
hash("4,4'-bipyridine"):'c1cc(ccn1)c1ccncc1', hash('isopropanol'):'CC(C)O',
hash('1-butanol'):'CCCCO', hash('nitromethane'):'CN(=O)=O',
hash('n-hexane'):'CCCCCC', hash('dimethylformamide'):'CN(C)C=O',
hash('N,N-dimethylformamide'):'CN(C)C=O', hash('cyclohexane'):'C1CCCCC1',
hash('hydroquinone'):'Oc1ccc(O)cc1', hash('benzene'):'c1ccccc1',
hash('octane'):'CCCCCCCC', hash('cyclohexanone'):'O=C1CCCCC1',
hash('isobutanol'):'CC(C)CO', hash('p-xylene'):'Cc1ccc(C)cc1',
hash('pentane'):'CCCCC', hash('pyridine'):'c1ccncc1',
hash('t-butanol'):'CC(C)(C)O', hash('m-xylene'):'Cc1cccc(C)c1',
hash('phenol'):'Oc1ccccc1', hash('o-cresol'):'Cc1ccccc1O',
hash('p-cresol'):'Cc1ccc(O)cc1',
hash('2,4,6-trimethylpyridine'):'Cc1cc(C)nc(C)c1',
hash('1-methylpyrrolidin-2-one'):'CN1CCCC1=O',
hash('1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8-hexadecafluoro-1,8-diiodooctane'):'FC(F
)(I)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)I',
hash('trifluoroethanol'):'OCC(F)(F)F', hash('dichloroethane'):'ClCCCl',
hash('1,1,2,2-tetrachloroethane'):'ClC(Cl)C(Cl)Cl',
hash('2-propanol'):'CC(C)O', hash('chlorobenzene'):'Clc1ccccc1',
hash('naphthalene'):'c1ccc2ccccc2c1', hash('propan-2-ol'):'CC(C)O',
hash('benzonitrile'):'N#Cc1ccccc1', hash('nitrobenzene'):'O=N(=O)c1ccccc1',
hash('dodecane'):'CCCCCCCCCCCC',
hash('1,2-dichlorobenzene'):'Clc1ccccc1Cl', hash('n-heptane'):'CCCCCCC',
hash('methanesulfonate'):'CS(=O)(=O)[O-]',
hash('3-methylbutan-1-ol'):'CC(C)CCO',
hash('1,1,2-trichloroethane'):'ClCC(Cl)Cl', hash('n-butane'):'CCCC',
hash('dichlorine'):'ClCl', hash('o-dichlorobenzene'):'Clc1ccccc1Cl',
hash('N,N-dimethylacetamide'):'CN(C)C(C)=O',
hash('4-methylaniline'):'Cc1ccc(N)cc1', hash('morpholine'):'C1COCCN1',
hash('1,3-dimethylimidazolidin-2-one'):'CN1CCN(C)C1=O)',
hash('1,3-dimethoxybenzene'):'COc1cccc(OC)c1', hash('n-propanol'):'CCCO',
hash('mesitylene'):'Cc1cc(C)cc(C)c1',
hash('acetylacetone'):'CC(=O)CC(C)=O', hash('diethylether'):'CCOCC',
hash('hydrazine'):'NN', hash('dimethylamine'):'CNC',
hash('1,2-difluorobenzene'):'Fc1ccccc1F', hash('cycloheptane'):'C1CCCCCC1',
hash('formamide'):'NC=O', hash('2-butanol'):'CCC(C)O',
hash('ethylacetate'):'CCOC(C)=O', hash('butan-1-ol'):'CCCCO',
hash('bromomethane'):'CBr', hash('piperazine'):'C1CNCCN1',
hash('n-nonane'):'CCCCCCCCC',
hash('1,2,4-trimethoxybenzene'):'COc1ccc(OC)c(OC)c1',
hash('1,2-diaminoethane'):'NCCN', hash('n-butanol'):'CCCCO',
hash('N-methylformamide'):'CNC=O', hash('pyrrolidine'):'C1CCNC1',
hash('actone'):'CC(C)=O', hash('n-octane'):'CCCCCCCC',
hash('(S)-2-methyl-1-butanol'):'CCC(C)CO',
hash('N,N-diethylformamide'):'CCN(CC)C=O',
hash('tetracyanoethylene'):'N#CC(C#N)=C(C#N)C#N',
```

```
            hash('2-methyl-1,3,5-trinitrobenzene'):'Cc1c(cc(cc1N(=O)=O)N(=O)=O)N(=O)=O',
            hash('furan'):'O1C=CC=C1', hash('2-methylpentane'):'CCCC(C)C',
            hash('isohexane'):'CCCC(C)C', hash('anisole'):'COc1ccccc1',
            hash('butanol'):'CCCCO', hash('propan-1-ol'):'CCCO',
            hash('dimethoxyethane'):'COCCOC', hash('cyclopentane'):'C1CCCC1',
            hash('methoxybenzene'):'COc1ccccc1', hash('toulene'):'Cc1ccccc1',
            hash('pyrene'):'c1cc2ccc3cccc4ccc(c1)c2c34', hash('butan-2-ol'):'CCC(C)O',
            hash('3-methylpyridine'):'Cc1cccnc1', hash('triethylamine'):'CCN(CC)CC',
            hash('dioxan'):'C1COCCO1', hash('2-methoxyethanol'):'COCCO',
            hash("4,4'-bipyridyl"):'c1cc(ccn1)c1ccncc1',
            hash('dimethylacetamide'):'CN(C)C(C)=O', hash('thiophene'):'S1C=CC=C1',
            hash('1,3-dioxolane'):'C1COCO1',
            hash('1-chloronaphthalene'):'Clc1cccc2ccccc12',
            hash('pentan-1-ol'):'CCCCCO', hash('hexan-1-ol'):'CCCCCCO',
            hash('2-phenylethanol'):'OCCc1ccccc1',
            hash('1-chloro-4-methylbenzene'):'Cc1ccc(Cl)cc1',
            hash('acetaldehyde'):'CC=O', hash('propiononitrile'):'CCC#N',
            hash('bromobenzene'):'Brc1ccccc1'}
53          deuterated_dictionary = {hash('dideutero-dichloromethane'):'ClCCl',
            hash('perdeutero-toluene'):'Cc1ccccc1', hash('octadeuterofuran'):'C1CCOC1',
            hash('deutero-ethanol'):'CCO', hash('deuterochloroform'):'C1C(Cl)Cl',
            hash('deutero-chloroform'):'C1C(Cl)Cl', hash('deuterobenzene'):'c1ccccc1',
            hash('deuteromethanol'):'CO', hash('hexadeutero-benzene'):'c1ccccc1',
            hash('perdeuteroacetone'):'CC(C)=O',
            hash('dideuterodichloromethane'):'ClCCl', hash('perdeuteromethanol'):'CO',
            hash('hexadeuterobenzene'):'c1ccccc1'}
54          acids_dictionary = {hash('trifluoroacetic'):'OC(=O)C(F)(F)F',
            hash('acetic'):'CC(O)=O', hash('formic'):'OC=O',
            hash('sulfuric'):'OS(O)(=O)=O', hash('maleic'):'OC(=O)C=CC(O)=O',
            hash('benzoic'):'OC(=O)c1ccccc1', hash('succinic'):'OC(=O)CCC(O)=O'}
55          ethers_dictionary = {'t-butyl methyl ether':'COC(C)(C)C',
            hash('di-isopropyl'):'CC(C)OC(C)C', hash('dipropyl'):'CCCOCCC',
            hash('diethyl'):'CCOCC', hash('diisopropyl'):'CC(C)OC(C)C',
            hash('isopropyl'):'CC(C)OC(C)C', hash('t-butyl'):'COC(C)(C)C',
            hash('acetic'):'CCOC(C)=O'}
56          acetates_dictionary = {hash('ethyl'):'CCOC(C)=O',
            hash('amyl'):'CCCCCOC(C)=O', hash('methyl'):'COC(C)=O',
            hash('butyl'):'CCCCOC(C)=O', hash('isobutyl'):'CC(C)COC(C)=O'}
57          glycol_dictionary = {hash('ethylene'):'OCCO'}
58          oxalates_dictionary = {hash('dipropyl'):'CCCOC(=O)C(=O)OCCC'}
59          acid_neutral_dictionary = {hash('oxalic'):'OC(=O)C(O)=O',
            hash('nitric'):'ON(=O)=O', hash('trans-but-2-enedioic'):'OC(=O)C=CC(O)=O',
            hash('but-2-enedioic'):'OC(=O)C=CC(O)=O',
            hash('fumaric'):'OC(=O)C=CC(O)=O', hash('phosphonic'):'OP(O)=O',
            hash('perchloric'):'O[Cl](=O)(=O)=O', hash('phosphoric'):'OP(O)(O)=O',
            hash('pyridine-2,6-dicarboxylic'):'OC(=O)c1cccc(n1)C(O)=O',
            hash('benzene-1,2,4,5-tetracarboxylic'):'OC(=O)c1cc(C(O)=O)c(cc1C(O)=O)C(O)=O
            ', hash('trifluoroacetatic'):'OC(=O)C(F)(F)F'}
60          acid_charged_dictionary = {hash('oxalic'):'OC(=O)C(=O)[O-]',
            hash('nitric'):'O=N(=O)[O-]',
            hash('trans-but-2-enedioic'):'OC(=O)C=CC(=O)[O-]',
            hash('but-2-enedioic'):'OC(=O)C=CC(=O)[O-]',
            hash('fumaric'):'OC(=O)C=CC(=O)[O-]', hash('phosphonic'):'OP(=O)[O-]',
            hash('perchloric'):'O=[Cl](=O)(=O)[O-]',
            hash('phosphoric'):'OP(O)(=O)[O-]',
            hash('pyridine-2,6-dicarboxylic'):'OC(=O)c1cccc(n1)C(=O)[O-]',
            hash('benzene-1,2,4,5-tetracarboxylic'):'OC(=O)c1cc(C(=O)[O-])c(cc1C(=O)[O-])
            C(O)=O', hash('trifluoroacetatic'):'FC(F)(F)C(=O)[O-]'}
61
62          partial_name = [hash('ketone'), hash('propanoate'), hash('carbonate'),
            hash('formate'), hash('iodine'), hash('tetrachloride'), hash('sulfoxide'),
            hash('dichloride'), hash('chloride'), hash('benzoate'), hash('disulfide'),
            hash('oxide')]
63          full_solvate_name_dictionary = {'methyl isobutyl ketone':'CC(C)CC(C)=O',
            'deuterium oxide':'O', 'ethyl propanoate':'CCOC(=O)CC', 'dimethyl
            carbonate':'COC(=O)OC', 'ethyl formate':'CCOC=O', '12]cycloparaphenylene
            iodine':'II', 'carbon tetrachloride':'ClC(Cl)(Cl)Cl', 'dimethyl
```

```
sulfoxide':'CS(C)=O', 'methylene dichloride':'ClCCl', 'methylene
chloride':'ClCCl', 'ethyl benzoate':'CCOC(=O)c1ccccc1', 'carbon
disulfide':'S=C=S'}

#The appearance of the same hash value for two distinct strings happens for
15 of the anhydrous form identifiers
#These identifiers are also in a dictionary that assigns them a number
between 1-15 rather than the same hash value as another structure identifier
duplicate_identifier_dictionary = {'GADKOL':1, 'MECYEY':2, 'OBASAN':3,
'OMOMAE':4, 'OTONUI':5, 'QOJWAO':6, 'TZBFZO':7, 'VOBXOZ':8, 'VUFJAI':9,
'VUFJAJ':10, 'WEPQOA':11, 'XADFUD':12, 'YABREY':13, 'YULCOZ':14, 'ZOVFEV':15}
duplicate_smiles_dictionary =
{'CC(C)(C)c1c(O)c(O)c(Cl)c2[NH+]3CCN(CC3)c12':1,
'CC1=C(C(CC2=NS(=O)(=O)c3ccccc23)c2ccccc2C)C(=O)N(N1)c1ccccc1':2,
'OC(=O)COc1cccc2ccccc12':3, 'Cc1c[nH+]cc(C)c1':4,
'Brc1ccc(NC(=O)c2cccnc2)cc1':5, 'CSC(N)=NNC(C)=CC(=O)c1ccccc1':6,
'CC1(CO1)C(O)(CBr)CBr':7, 'OCC12CCCC(C1)(C(O)=O)C1(CCSCC1)O2':8,
'COc1ccc2CC3[NH2+]CCC3(C)c2c1':9,
'CC(=O)C(P(c1ccccc1)c1ccccc1)=C(C)Nc1c(F)cccc1F':10}

solvated_waterless_entry = []
solvated_waterless_hash = []
solvated_waterless_SMILES = []

#Split input file at entries: 25020, 50017, 75024, 100016, 125022, and 142002
entry1 = entry_reader1
for z in range(142002, len(entry1)):
    solvated_waterless_entry.append(entry1[z])
    if entry1[z].identifier in duplicate_identifier_dictionary:

        solvated_waterless_hash.append(duplicate_identifier_dictionary.get(en
        try1[z].identifier))
    else:
        solvated_waterless_hash.append(hash(entry1[z].identifier))

while len(solvated_waterless_hash) != 0:
    print len(solvated_waterless_hash)
    entries1 = []
    hashes1 = []
    entries1.append(solvated_waterless_entry[0])
    hashes1.append(solvated_waterless_hash[0])
    for y in range(1, len(solvated_waterless_hash)):
        if solvated_waterless_hash[0] != solvated_waterless_hash[y]:
            entries1.append(solvated_waterless_entry[y])
            hashes1.append(solvated_waterless_hash[y])
        else:
            del solvated_waterless_hash[y]
            del solvated_waterless_entry[y]
            break

    tag = 12
    for a in range(len(entries1)):
        component_SMILES = []
        entry_SMILES = entries1[a].molecule.smiles.split('.')
        for b in range(len(entry_SMILES)):
            if entry_SMILES[b] not in duplicate_smiles_dictionary:
                component_SMILES.append(hash(entry_SMILES[b]))
            else:

                component_SMILES.append(duplicate_smiles_dictionary.get(entry
                _SMILES[b]))
        component_SMILES = list(set(component_SMILES))
        if entries1[a].identifier == 'BULVEL':
            if hash('[O-][n+]1ccccc1') not in component_SMILES:
                component_SMILES.append(hash('[O-][n+]1ccccc1'))
            if hash('[O]n1ccccc1') in component_SMILES:
                component_SMILES.remove(hash('[O]n1ccccc1'))
```

```
                    elif entries1[a].identifier == 'CEHKOS':
                        component_SMILES.append(hash('II'))
                    elif entries1[a].identifier == 'GOTJEE' or entries1[a].identifier
                    == 'HASVEF' or entries1[a].identifier == 'MTFBTZ10':
                        component_SMILES.append(tag)
                        tag += 1
                    elif entries1[a].identifier == 'JUHJIF':
                        if hash('BrBr') not in component_SMILES:
                            component_SMILES.append(hash('BrBr'))
                        if hash('[Br]') in component_SMILES:
                            component_SMILES.remove(hash('[Br]'))
                    elif entries1[a].identifier == 'LOKXUF':
                        if hash('OC(=O)C(F)(F)F') not in component_SMILES:
                            component_SMILES.append(hash('OC(=O)C(F)(F)F'))
                        if hash('[O]C(=O)C(F)(F)F') in component_SMILES:
                            component_SMILES.remove(hash('[O]C(=O)C(F)(F)F'))
                    elif entries1[a].identifier == 'OJUNEO':
                        if hash('ClCl') not in component_SMILES:
                            component_SMILES.append(hash('ClCl'))
                        if hash('[Cl]') in component_SMILES:
                            component_SMILES.remove(hash('[Cl]'))
                    elif entries1[a].identifier == 'ZILFILM':
                        component_SMILES.append(hash('CCO'))
                    else:
                        if entries1[a].chemical_name != None:
                            pieces = entries1[a].chemical_name.split(' ')
                            formula_pieces = entries1[a].formula.split(',')
                            for c in range(len(pieces)):
                                if hash(pieces[c]) == hash('xylene'):
                                    if entries1[a].identifier == 'QOWNEV':
                                        if hash('Cc1ccccc1C') not in component_SMILES:
                                            component_SMILES.append(hash('Cc1ccccc1C'))
                                    elif entries1[a].identifier == 'TEYCEF' or
                                    entries1[a].identifier == 'TEYCEF01':
                                        if hash('Cc1cccc(C)c1') not in component_SMILES:
                                            component_SMILES.append(hash('Cc1cccc(C)c1'))
                                    elif entries1[a].identifier == 'MAMNAR':
                                        if hash('Cc1ccc(C)cc1') not in component_SMILES:
                                            component_SMILES.append(hash('Cc1ccc(C)cc1'))
                                elif hash(pieces[c]) == hash('hydrochloride') or
                                hash(pieces[c]) == hash('bis(hydrochloride)'):
                                    for z in range(len(formula_pieces)):
                                        if formula_pieces[z] == 'Cl1 1-' or '(Cl1 1-)'
                                        in formula_pieces[z]:
                                            if hash('[Cl-]') not in component_SMILES:
                                                component_SMILES.append(hash('[Cl-]'))
                                        if formula_pieces[z] == 'H1 Cl1' or '(H1 Cl1)'
                                        in formula_pieces[z]:
                                            if hash('Cl') not in component_SMILES:
                                                component_SMILES.append(hash('Cl'))
                                elif hash(pieces[c]) == hash('unidentified') or
                                hash(pieces[c]) == hash('unknown'):
                                    component_SMILES.append(tag)
                                    tag += 1
                                elif hash(pieces[c]) == hash('glycol'):
                                    if hash(pieces[c-1]) in glycol_dictionary:
                                        if
                                        hash(glycol_dictionary.get(hash(pieces[c-1])))
                                        not in component_SMILES:

                                            component_SMILES.append(hash(glycol_dictionar
                                            y.get(hash(pieces[c-1]))))
                                elif hash(pieces[c]) == hash('oxalate'):
                                    if hash(pieces[c-1]) in oxalates_dictionary:
                                        if
                                        hash(oxalates_dictionary.get(hash(pieces[c-1])))
                                        not in component_SMILES:
```

```
168                                                 component_SMILES.append(hash(oxalates_diction
                                                    ary.get(hash(pieces[c-1]))))
169                                         elif hash(pieces[c]) == hash('acetate'):
170                                             if hash(pieces[c-1]) in acetates_dictionary:
171                                                 if
                                                    hash(acetates_dictionary.get(hash(pieces[c-1])))
                                                    not in component_SMILES:
172
                                                     component_SMILES.append(hash(acetates_diction
                                                     ary.get(hash(pieces[c-1]))))
173                                         elif hash(pieces[c]) == hash('ether'):
174                                             if hash(pieces[c-1]) == hash('petroleum'):
175                                                 component_SMILES.append(11)
176                                             elif pieces[c-2] + ' ' + pieces[c-1] + ' ' +
                                                pieces[c] in ethers_dictionary:
177                                                 if hash(ethers_dictionary.get(pieces[c-2] + ' '
                                                    + pieces[c-1] + ' ' + pieces[c])) not in
                                                    component_SMILES:
178
                                                     component_SMILES.append(hash(ethers_dictionar
                                                     y.get(pieces[c-2] + ' ' + pieces[c-1] + ' '
                                                     + pieces[c])))
179                                             elif hash(pieces[c-1]) in ethers_dictionary:
180                                                 if
                                                    hash(ethers_dictionary.get(hash(pieces[c-1])))
                                                    not in component_SMILES:
181
                                                     component_SMILES.append(hash(ethers_dictionar
                                                     y.get(hash(pieces[c-1]))))
182                                         elif hash(pieces[c]) in partial_name:
183                                             if pieces[c] == 'ketone':
184                                                 if pieces[c-2] + ' ' + pieces[c-1] + ' ' +
                                                    pieces[c] in full_solvate_name_dictionary:
185                                                     if
                                                        hash(full_solvate_name_dictionary.get(pieces[
                                                        c-2] + ' ' + pieces[c-1] + ' ' +
                                                        pieces[c])) not in component_SMILES:
186
                                                         component_SMILES.append(hash(full_solvate
                                                         _name_dictionary.get(pieces[c-2] + ' '
                                                         + pieces[c-1] + ' ' + pieces[c])))
187                                             else:
188                                                 if pieces[c-1] + ' ' + pieces[c] in
                                                    full_solvate_name_dictionary:
189                                                     if
                                                        hash(full_solvate_name_dictionary.get(pieces[
                                                        c-1] + ' ' + pieces[c])) not in
                                                        component_SMILES:
190
                                                         component_SMILES.append(hash(full_solvate
                                                         _name_dictionary.get(pieces[c-1] + ' '
                                                         + pieces[c])))
191                                         elif hash(pieces[c]) == hash('acid'):
192                                             if hash(pieces[c-1]) in acids_dictionary:
193                                                 if
                                                    hash(acids_dictionary.get(hash(pieces[c-1])))
                                                    not in component_SMILES:
194
                                                     component_SMILES.append(hash(acids_dictionary
                                                     .get(hash(pieces[c-1]))))
195                                             elif hash(pieces[c-1]) in acid_neutral_dictionary:
196                                                 if hash(pieces[c-1]) == hash('oxalic'):
197                                                     for z in range(len(formula_pieces)):
198                                                         if formula_pieces[z] == 'C2 H2 O4' or
                                                            '(C2 H2 O4)' in formula_pieces[z]:
199                                                             if
```

```
                                hash(acid_neutral_dictionary.get(hash
                                (pieces[c-1]))) not in
                                component_SMILES:
200
                                    component_SMILES.append(hash(acid
                                    _neutral_dictionary.get(hash(piec
                                    es[c-1]))))
201                             if formula_pieces[z] == 'C2 H1 O4 1-'
                                or '(C2 H1 O4 1-)' in formula_pieces[z]:
202                                 if
                                    hash(acid_charged_dictionary.get(hash
                                    (pieces[c-1]))) not in
                                    component_SMILES:
203
                                    component_SMILES.append(hash(acid
                                    _charged_dictionary.get(hash(piec
                                    es[c-1]))))
204                         elif hash(pieces[c-1]) == hash('nitric'):
205                             for z in range(len(formula_pieces)):
206                                 if formula_pieces[z] == 'H1 N1 O3' or
                                    '(H1 N1 O3)' in formula_pieces[z]:
207                                     if
                                        hash(acid_neutral_dictionary.get(hash
                                        (pieces[c-1]))) not in
                                        component_SMILES:
208
                                        component_SMILES.append(hash(acid
                                        _neutral_dictionary.get(hash(piec
                                        es[c-1]))))
209                                 if formula_pieces[z] == 'N1 O3 1-' or
                                    '(N1 O3 1-)' in formula_pieces[z]:
210                                     if
                                        hash(acid_charged_dictionary.get(hash
                                        (pieces[c-1]))) not in
                                        component_SMILES:
211
                                        component_SMILES.append(hash(acid
                                        _charged_dictionary.get(hash(piec
                                        es[c-1]))))
212                         elif hash(pieces[c-1]) ==
                            hash('trans-but-2-enedioic') or
                            hash(pieces[c-1]) == hash('but-2-enedioic') or
                            hash(pieces[c-1]) == hash('fumaric'):
213                             for z in range(len(formula_pieces)):
214                                 if formula_pieces[z] == 'C4 H4 O4' or
                                    '(C4 H4 O4)' in formula_pieces[z]:
215                                     if
                                        hash(acid_neutral_dictionary.get(hash
                                        (pieces[c-1]))) not in
                                        component_SMILES:
216
                                        component_SMILES.append(hash(acid
                                        _neutral_dictionary.get(hash(piec
                                        es[c-1]))))
217                                 if formula_pieces[z] == 'C4 H3 O4 1-'
                                    or '(C4 H3 O4 1-)' in formula_pieces[z]:
218                                     if
                                        hash(acid_charged_dictionary.get(hash
                                        (pieces[c-1]))) not in
                                        component_SMILES:
219
                                        component_SMILES.append(hash(acid
                                        _charged_dictionary.get(hash(piec
                                        es[c-1]))))
220                         elif hash(pieces[c-1]) == hash('phosphonic'):
221                             for z in range(len(formula_pieces)):
222                                 if formula_pieces[z] == 'H3 O3 P1' or
```

```
                    '(H3 O3 P1)' in formula_pieces[z]:
223                     if
                        hash(acid_neutral_dictionary.get(hash
                        (pieces[c-1]))) not in
                        component_SMILES:
224
                            component_SMILES.append(hash(acid
                            _neutral_dictionary.get(hash(piec
                            es[c-1]))))
225                 if formula_pieces[z] == 'H2 O3 P1 1-'
                    or '(H2 O3 P1 1-)' in formula_pieces[z]:
226                     if
                        hash(acid_charged_dictionary.get(hash
                        (pieces[c-1]))) not in
                        component_SMILES:
227
                            component_SMILES.append(hash(acid
                            _charged_dictionary.get(hash(piec
                            es[c-1]))))
228             elif hash(pieces[c-1]) == hash('perchloric'):
229                 for z in range(len(formula_pieces)):
230                     if formula_pieces[z] == 'H1 Cl1 O4' or
                        '(H1 Cl1 O4)' in formula_pieces[z]:
231                         if
                            hash(acid_neutral_dictionary.get(hash
                            (pieces[c-1]))) not in
                            component_SMILES:
232
                                component_SMILES.append(hash(acid
                                _neutral_dictionary.get(hash(piec
                                es[c-1]))))
233                     if formula_pieces[z] == 'Cl1 O4 1-' or
                        '(Cl1 O4 1-)' in formula_pieces[z]:
234                         if
                            hash(acid_charged_dictionary.get(hash
                            (pieces[c-1]))) not in
                            component_SMILES:
235
                                component_SMILES.append(hash(acid
                                _charged_dictionary.get(hash(piec
                                es[c-1]))))
236             elif hash(pieces[c-1]) == hash('phosphoric'):
237                 for z in range(len(formula_pieces)):
238                     if formula_pieces[z] == 'H3 O4 P1' or
                        '(H3 O4 P1)' in formula_pieces[z]:
239                         if
                            hash(acid_neutral_dictionary.get(hash
                            (pieces[c-1]))) not in
                            component_SMILES:
240
                                component_SMILES.append(hash(acid
                                _neutral_dictionary.get(hash(piec
                                es[c-1]))))
241                     if formula_pieces[z] == 'H2 O4 P1 1-'
                        or '(H2 O4 P1 1-)' in formula_pieces[z]:
242                         if
                            hash(acid_charged_dictionary.get(hash
                            (pieces[c-1]))) not in
                            component_SMILES:
243
                                component_SMILES.append(hash(acid
                                _charged_dictionary.get(hash(piec
                                es[c-1]))))
244             elif hash(pieces[c-1]) ==
                hash('pyridine-2,6-dicarboxylic'):
245                 for z in range(len(formula_pieces)):
246                     if formula_pieces[z] == 'C7 H5 N1 O4'
```

```
                                    or '(C7 H5 N1 O4)' in formula_pieces[z]:
247                                     if
                                        hash(acid_neutral_dictionary.get(hash
                                        (pieces[c-1]))) not in
                                        component_SMILES:
248
                                            component_SMILES.append(hash(acid
                                            _neutral_dictionary.get(hash(piec
                                            es[c-1]))))
249                                     if formula_pieces[z] == 'C7 H4 N1 O4
                                        1-' or '(C7 H4 N1 O4 1-)' in
                                        formula_pieces[z]:
250                                     if
                                        hash(acid_charged_dictionary.get(hash
                                        (pieces[c-1]))) not in
                                        component_SMILES:
251
                                            component_SMILES.append(hash(acid
                                            _charged_dictionary.get(hash(piec
                                            es[c-1]))))
252                             elif hash(pieces[c-1]) ==
                                hash('benzene-1,2,4,5-tetracarboxylic'):
253                                 for z in range(len(formula_pieces)):
254                                     if formula_pieces[z] == 'C10 H6 O8' or
                                        '(C10 H6 O8)' in formula_pieces[z]:
255                                     if
                                        hash(acid_neutral_dictionary.get(hash
                                        (pieces[c-1]))) not in
                                        component_SMILES:
256
                                            component_SMILES.append(hash(acid
                                            _neutral_dictionary.get(hash(piec
                                            es[c-1]))))
257                                     if formula_pieces[z] == 'C10 H4 O8 2-'
                                        or '(C10 H4 O8 2-)' in formula_pieces[z]:
258                                     if
                                        hash(acid_charged_dictionary.get(hash
                                        (pieces[c-1]))) not in
                                        component_SMILES:
259
                                            component_SMILES.append(hash(acid
                                            _charged_dictionary.get(hash(piec
                                            es[c-1]))))
260                             elif hash(pieces[c-1]) ==
                                hash('trifluoroacetatic'):
261                                 for z in range(len(formula_pieces)):
262                                     if formula_pieces[z] == 'C2 H1 F3 O2'
                                        or '(C2 H1 F3 O2)' in formula_pieces[z]:
263                                     if
                                        hash(acid_neutral_dictionary.get(hash
                                        (pieces[c-1]))) not in
                                        component_SMILES:
264
                                            component_SMILES.append(hash(acid
                                            _neutral_dictionary.get(hash(piec
                                            es[c-1]))))
265                                     if formula_pieces[z] == 'C2 F3 O2 1-'
                                        or '(C2 F3 O2 1-)' in formula_pieces[z]:
266                                     if
                                        hash(acid_charged_dictionary.get(hash
                                        (pieces[c-1]))) not in
                                        component_SMILES:
267
                                            component_SMILES.append(hash(acid
                                            _charged_dictionary.get(hash(piec
                                            es[c-1]))))
268                         elif hash(pieces[c]) in deuterated_dictionary:
```

```
                                if hash(deuterated_dictionary.get(hash(pieces[c])))
                                not in component_SMILES:

                                    component_SMILES.append(hash(deuterated_dictionar
                                    y.get(hash(pieces[c]))))
                                if hash(pieces[c]) ==
                                hash('dideutero-dichloromethane') or
                                hash(pieces[c]) == hash('dideuterodichloromethane'):
                                    if hash('Cl[C]Cl') in component_SMILES:
                                        component_SMILES.remove(hash('Cl[C]Cl'))
                                elif hash(pieces[c]) == hash('perdeutero-toluene'):
                                    if hash('[C]c1[c][c][c][c][c]1') in
                                    component_SMILES:

                                        component_SMILES.remove(hash('[C]c1[c][c][c][
                                        c][c]1'))
                                elif hash(pieces[c]) == hash('deutero-ethanol'):
                                    if hash('[C][C][O]') in component_SMILES:
                                        component_SMILES.remove(hash('[C][C][O]'))
                                elif hash(pieces[c]) == hash('deuterochloroform')
                                or hash(pieces[c]) == hash('deutero-chloroform'):
                                    if hash('Cl[C](Cl)Cl') in component_SMILES:
                                        component_SMILES.remove(hash('Cl[C](Cl)Cl'))
                                elif hash(pieces[c]) == hash('hexadeutero-benzene')
                                or hash(pieces[c]) == hash('deuterobenzene') or
                                hash(pieces[c]) == hash('hexadeuterobenzene'):
                                    if hash('[c]1[c][c][c][c][c]1') in
                                    component_SMILES:

                                        component_SMILES.remove(hash('[c]1[c][c][c][c
                                        ][c]1'))
                                elif hash(pieces[c]) == hash('deuteromethanol') or
                                hash(pieces[c]) == hash('perdeuteromethanol'):
                                    if hash('[C][O]') in component_SMILES:
                                        component_SMILES.remove(hash('[C][O]'))
                            elif hash(pieces[c]) in solvents_dictionary:
                                if hash(solvents_dictionary.get(hash(pieces[c])))
                                not in component_SMILES:

                                    component_SMILES.append(hash(solvents_dictionary.
                                    get(hash(pieces[c]))))

                    component_SMILES = sorted(component_SMILES)
                    solvated_waterless_SMILES.append(component_SMILES)

                while len(solvated_waterless_SMILES) != 0:
                    number = []
                    identifiers = []
                    identifiers.append(hashes1[0])
                    if hashes1[0] == hash('KECYBU15'):
                        number.append(87)
                    elif hashes1[0] == hash('MTYHFB03'):
                        number.append(29)
                    else:

                        number.append(entries1[0].crystal.spacegroup_number_and_setting[0
                        ])
                    for d in range(1,len(solvated_waterless_SMILES)):
                        if solvated_waterless_SMILES[0] == solvated_waterless_SMILES[d]:
                            identifiers.append(hashes1[d])
                            if hashes1[d] == hash('KECYBU15'):
                                number.append(87)
                            elif hashes1[d] == hash('MTYHFB03'):
                                number.append(29)
                            else:

                                number.append(entries1[d].crystal.spacegroup_number_and_s
```

```
                                    etting[0])
315
316                         if len(identifiers) != 1:
317                             if all(x == number[0] for x in number) or any(number.count(y) >
                                1 for y in number):
318                                 identifiers_to_remove = []
319                                 for h in range(len(identifiers)):
320                                     times1 = number.count(number[h])
321                                     if times1 == 1:
322                                         identifiers_to_remove.append(identifiers[h])
323                                 identifiers_to_check = [z for z in identifiers if z not in
                                    identifiers_to_remove]
324                                 if identifiers_to_check != []:
325                                     for i in range(len(identifiers_to_check)):
326                                         pin =
                                            solvated_waterless_hash.index(identifiers_to_check[i]
                                            )
327
                                            duplicates_writer.write(solvated_waterless_entry[pin]
                                            )
328                                     pin =
                                        solvated_waterless_hash.index(identifiers_to_check[0])
329                                     duplicates_writer.write(solvated_waterless_entry[pin])
330                             for j in range(len(identifiers)):
331                                 pin = solvated_waterless_hash.index(identifiers[j])
332                                 solvated_waterless_hash.remove(solvated_waterless_hash[pin])
333                                 pin2 = hashes1.index(identifiers[j])
334                                 hashes1.remove(hashes1[pin2])
335                                 entries1.remove(entries1[pin2])
336
                                    solvated_waterless_entry.remove(solvated_waterless_entry[pin]
                                    )
337
                                    solvated_waterless_SMILES.remove(solvated_waterless_SMILES[pi
                                    n])
338
339                         else:
340                             solvated_waterless_hash.remove(hashes1[0])
341                             hashes1.remove(hashes1[0])
342                             solvated_waterless_entry.remove(entries1[0])
343                             entries1.remove(entries1[0])
344                             solvated_waterless_SMILES.remove(solvated_waterless_SMILES[0])
345
346                 print 'solvated waterless list is empty'
347
348
349 if __name__ == '__main__':
350     # This runs the script
351     r = Runner()
352     r.run()
353
```

```python
#14 Find duplicate SMILES waterless forms with packing similarity

from ccdc import io, search
import argparse
import os
import glob
import re
import itertools
from decimal import Decimal
from ccdc.crystal import PackingSimilarity

similarity_engine = PackingSimilarity()
similarity_engine.settings.ignore_hydrogen_positions = True

#This script is written the same as the part2 script to find duplicates for hydrate structures with entry SMILES strings
filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_smiles_all_potential_duplicates.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='waterless_forms_smiles_all_first_occurrence.gcd',
            help='output file [waterless_forms_smiles_all_first_occurrence.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')

        with io.EntryWriter(self.args.output) as lone_wolf_writer:
            with io.EntryWriter("waterless_forms_smiles_all_duplicates.gcd") as writer1:
                with io.EntryWriter("waterless_forms_smiles_all_manual_check.gcd") as writer2:

                    duplicate_identifier_dictionary = {'GADKOL':1, 'MECYEY':2, 'OBASAN':3, 'OMOMAE':4, 'OTONUI':5, 'QOJWAO':6, 'TZBFZO':7, 'VOBXOZ':8, 'VUFJAI':9, 'VUFJAJ':10, 'WEPQOA':11, 'XADFUD':12, 'YABREY':13, 'YULCOZ':14, 'ZOVFEV':15}

                    solvated_waterless_hash = []
                    solvated_waterless_entry = []

                    #Split input file at entries: 3301, 3306, 8071, 8077, 13514, 13518, 17337, 17340, 25752, 25756, 26203, 26207, 28814, 28820, 30802, 30805, 33952, 33955, 35040, 35043, 37480, 37483, 38315, 38320, 40159, 40162, 40293, 40296, 40626, 40629, 42708, 42711, 43261, 43264, 43472, 43475, 44646, 44649, 45347, 45350, 45944, 45947, 47758, 47769, 48118, 48129, 48550, 48561, 49064, 49070
                    #The following entries get stuck at analysis calculation: HPHBNZ, AMAXOD, IFAWAP, QEHLUL, FALZEX, DAZPUS, EBIGUR, FEFTUI, HELXUR,
```

```
                    ICUVUZ, KOLWEN, LISZAQ, NOCYAH, NUGWIW, OLEGIV, REFLIX, SAXNUA,
                    SEYZII, TPCMOM, VEDFAO, WILNIO, CUXHOT, CUXLAJ, PUWYUD, PUXCOC,
                    ZEZJAS
56                  for a in range(49070, len(entry_reader1)):
57                      if entry_reader1[a].identifier in
                        duplicate_identifier_dictionary:
58
                            solvated_waterless_hash.append(duplicate_identifier_dictionar
                            y.get(entry_reader1[a].identifier))
59                      else:
60
                            solvated_waterless_hash.append(hash(entry_reader1[a].identifi
                            er))
61                      solvated_waterless_entry.append(entry_reader1[a])
62
63                  while len(solvated_waterless_hash) != 0:
64                      print len(solvated_waterless_hash)
65                      identifiers = []
66                      identifiers.append(solvated_waterless_hash[0])
67                      for d in range(1,len(solvated_waterless_hash)):
68                          if identifiers[0] != solvated_waterless_hash[d]:
69                              identifiers.append(solvated_waterless_hash[d])
70                          else:
71                              del solvated_waterless_hash[d]
72                              del solvated_waterless_entry[d]
73                              break
74
75                      remove_identifiers = []
76                      while len(identifiers) != len(remove_identifiers):
77                          for s in range(len(identifiers)):
78                              count = 0
79                              if identifiers[s] not in remove_identifiers:
80                                  pin1 = solvated_waterless_hash.index(identifiers[s])
81                                  reference = solvated_waterless_entry[pin1].crystal
82                                  remove_identifiers.append(identifiers[s])
83                                  for f in range(len(identifiers)):
84                                      if identifiers[f] not in remove_identifiers:
85                                          pin2 =
                                            solvated_waterless_hash.index(identifiers[f])
86                                          sample =
                                            solvated_waterless_entry[pin2].crystal
87                                          analysis =
                                            similarity_engine.compare(reference, sample)
88                                          if analysis == None:
89                                              if count == 0:
90
                                                    writer2.write(solvated_waterless_entr
                                                    y[pin1])
91                                                  count += 1
92
                                                writer2.write(solvated_waterless_entry[pi
                                                n2])
93                                              remove_identifiers.append(identifiers[f])
94                                          else:
95                                              if
                                                analysis.nmatched_molecules/analysis.pack
                                                ing_shell_size == 1:
96
                                                    lone_wolf_writer.write(solvated_water
                                                    less_entry[pin1])
97
                                                    writer1.write(solvated_waterless_entr
                                                    y[pin2])
98
                                                    remove_identifiers.append(identifiers
                                                    [f])
99                              if count != 0:
```

```
                                        writer2.write(solvated_waterless_entry[pin1])

                        for j in range(len(identifiers)):
                            pin = solvated_waterless_hash.index(identifiers[j])
                            solvated_waterless_hash.remove(solvated_waterless_hash[pin])

                            solvated_waterless_entry.remove(solvated_waterless_entry[pin]
                            )

                    print 'solvated waterless list is empty'


if __name__ == '__main__':
    # This runs the script
    r = Runner()
    r.run()

```

```python
#15 Find duplicates that fail packing similarity check

from ccdc import io, search
import argparse
import os
import glob
import re

#Potential duplicates that cannot be analyzed with packing similarity tool are accounted for here
#Structures with unit cell parameters that are within 1 angstorm (for cell lengths) and 1 degree (for cell angles) are considered probable duplicates
#It is possible that two structures with similar unit cell parameters are polymorphs and not duplicates
#Unfortunately one of the polymorphs in these cases will be lost here
#This method was the least manually intensive way to ensure that duplicate structures are removed

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\hydrates_None_all_manual_check.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\hydrates_smiles_all_manual_check.txt"
filepath3 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_None_all_manual_check.txt"
filepath4 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_smiles_all_manual_check.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='hydrates_None_entry_to_keep.gcd',
            help='output file [hydrates_None_entry_to_keep.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')
        entry_reader2 = io.EntryReader(filepath2, format='identifiers')
        entry_reader3 = io.EntryReader(filepath3, format='identifiers')
        entry_reader4 = io.EntryReader(filepath4, format='identifiers')

        with io.EntryWriter(self.args.output) as new_lists_writer:
            with io.EntryWriter("hydrates_None_duplicates_to_remove.gcd") as writer1:
                with io.EntryWriter("hydrates_smiles_entry_to_keep.gcd") as writer2:
                    with io.EntryWriter("hydrates_smiles_duplicates_to_remove.gcd") as writer3:
                        with io.EntryWriter("waterless_forms_None_entry_to_keep.gcd") as writer4:
                            with io.EntryWriter("waterless_forms_None_duplicates_to_remove.gcd") as writer5:
```

```
                    with
                    io.EntryWriter("waterless_forms_smiles_entry_to_keep.gcd"
                    ) as writer6:
                        with
                        io.EntryWriter("waterless_forms_smiles_duplicates_to_
                        remove.gcd") as writer7:

                            loop = 0
                            entry1 = entry_reader1
                            while loop < 4:
                                identifiers = []
                                entries = []
                                positions = []
                                lengths = []
                                angles = []
                                #The unit cell lengths and angles for each
                                structure are recorded in two separate lists
                                for a in range(len(entry1)):
                                    pre_lengths = []
                                    pre_angles = []

                                    identifiers.append(hash(entry1[a].identif
                                    ier))
                                    entries.append(entry1[a])
                                    positions.append(a)
                                    for b in range(3):

                                        pre_lengths.append(entry1[a].crystal.
                                        cell_lengths[b])

                                        pre_angles.append(entry1[a].crystal.c
                                        ell_angles[b])
                                    lengths.append(pre_lengths)
                                    angles.append(pre_angles)

                                while len(identifiers) != 0:
                                    print len(identifiers)
                                    print positions[0]
                                    compare_identifiers = []
                                    compare_lengths = []
                                    compare_angles = []

                                    compare_identifiers.append(identifiers[0]
                                    )
                                    compare_lengths.append(lengths[0])
                                    compare_angles.append(angles[0])
                                    #The input files are organized so that
                                    sets of potential duplicates are
                                    written as A, B, C, ArgumentParser
                                    #The second occurence of A indicates
                                    the end of one set and tells the script
                                    to compare A, B, and C
                                    for c in range(1, len(identifiers)):
                                        if identifiers[0] != identifiers[c]:

                                            compare_identifiers.append(identi
                                            fiers[c])

                                            compare_lengths.append(lengths[c]
                                            )
                                            compare_angles.append(angles[c])
                                        else:
                                            del identifiers[c]
                                            del entries[c]
                                            del positions[c]
                                            del lengths[c]
                                            del angles[c]
```

```
                                                    break
                                            identifiers_to_remove = []
                                            #The same method to identify which
                                            structures are duplicates used in the
                                            previous scripts is applied
                                            while len(compare_identifiers) !=
                                            len(identifiers_to_remove):
                                                for d in
                                                range(len(compare_identifiers)):
                                                    if compare_identifiers[d] not
                                                    in identifiers_to_remove:

                                                        identifiers_to_remove.append(
                                                        compare_identifiers[d])
                                                        for e in
                                                        range(len(compare_identifiers
                                                        )):
                                                            if
                                                            compare_identifiers[e]
                                                            not in
                                                            identifiers_to_remove:
                                                                count = 0
                                                                found = []
                                                                #Nonstandard
                                                                settings of the
                                                                unit cell can cause
                                                                one structure's
                                                                a-axis and alpha
                                                                angle to match
                                                                another structures
                                                                c-axis and gamma
                                                                angle
                                                                #Therefore, the
                                                                a-axis and alpha
                                                                angle of one
                                                                structure are
                                                                compared to each
                                                                set of axes and
                                                                angles
                                                                #i.e. the a-axis
                                                                and alpha angle of
                                                                structure A are
                                                                compared to the
                                                                a-axis and alpha
                                                                angle, b-axis and
                                                                beta angle, c-axis
                                                                and gamma angle of
                                                                structure B
                                                                for f in range(3):
                                                                    if
                                                                    abs(compare_lengt
                                                                    hs[d][0]-compare_
                                                                    lengths[e][f])
                                                                    < 1 and
                                                                    abs(compare_angle
                                                                    s[d][0]-compare_a
                                                                    ngles[e][f]) < 1:
                                                                        #If a match
    is found, the axis and angle from each structure that match are recorded as found
                                                                        count += 1

    found.append(f)
                                                                        break
                                                                if count == 1:
                                                                    for g in
                                                                    range(3):
                                                                        #Previously
```

```python
matched axes and angles will not be compared against new axes and angles
                                    #For example, if the a-axis and alpha angle of structure A match the c-axis and gamma angle of structure B
                                    #Then the b-axis and beta angle of structure A will only be compared to the a-axis and alpha angle and b-axis and beta angle of structure B
                                    if g not in found:
                                        if abs(compare_lengths[d][1]-compare_lengths[e][g]) < 1 and abs(compare_angles[d][1]-compare_angles[e][g]) < 1:
                                            count += 1
                                            found.append(g)
                                            break
                                if count == 2:
                                    for h in range(3):
                                        if h not in found:
                                            if abs(compare_lengths[d][2]-compare_lengths[e][h]) < 1 and abs(compare_angles[d][2]-compare_angles[e][h]) < 1:
                                                count += 1
                                                break
                                    #If all three axis and angle pairs are matched between the two structures, they are considered duplicates and written to two output files
                                    #One output file will have the first occurence of the structure, the other will have all the duplicate occurrences
                                    if count == 3:
                                        if loop == 0:
                                            new_lists_writer.write(entries[d])
                                            writer1.write(entries[e])
                                        if loop == 1:
                                            writer2.write(entries[d])
                                            writer3.write(entries[e])
                                        if loop == 2:
                                            writer4.write(entries[d])
                                            writer5.write(entries[e])
                                        if loop == 3:
                                            writer6.write(entries[d])
                                            writer7.write(entries[e])
                                        identifiers_to_remove.append(compare_identifiers[e])
        #The list of entries to analyze is updated and the next set is evaluated
        for y in range(len(compare_identifiers)):
            del identifiers[0]
```

```
                    del entries[0]
                    del positions[0]
                    del lengths[0]
                    del angles[0]

                loop += 1
                if loop == 1:
                    entry1 = entry_reader2
                if loop == 2:
                    entry1 = entry_reader3
                if loop == 3:
                    entry1 = entry_reader4









if __name__ == '__main__':
    r = Runner()
    r.run()

```

```python
#16 Remove overlapping duplicate structures

from ccdc import io, search
import argparse
import os
import glob
import re

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\hydrates_smiles_all_first_occurrence.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\hydrates_smiles_all_duplicates.txt"
filepath3 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\hydrates_None_entry_to_keep.txt"
filepath4 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\hydrates_None_duplicates_to_remove.txt"

filepath5 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_smiles_all_first_occurrence.txt"
filepath6 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_smiles_all_duplicates.txt"
filepath7 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_None_entry_to_keep.txt"
filepath8 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_None_duplicates_to_remove.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output',
            default='complete_hydrates_merged_all_first_occurrence.gcd',
            help='output file [complete_hydrates_merged_all_first_occurrence.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')
        entry_reader2 = io.EntryReader(filepath2, format='identifiers')
        entry_reader3 = io.EntryReader(filepath3, format='identifiers')
        entry_reader4 = io.EntryReader(filepath4, format='identifiers')
        entry_reader5 = io.EntryReader(filepath5, format='identifiers')
        entry_reader6 = io.EntryReader(filepath6, format='identifiers')
        entry_reader7 = io.EntryReader(filepath7, format='identifiers')
        entry_reader8 = io.EntryReader(filepath8, format='identifiers')

        with io.EntryWriter(self.args.output) as new_lists_writer:
            with io.EntryWriter("complete_hydrates_merged_all_duplicates.gcd") as
            writer1:
                with
                io.EntryWriter("complete_waterless_forms_merged_all_first_occurrence.gcd"
                ) as writer2:
```

```
                    with
                    io.EntryWriter("complete_waterless_forms_merged_all_duplicates.gcd")
                    as writer3:

                        duplicate_identifier_dictionary = {'GADKOL':1, 'MECYEY':2,
                        'OBASAN':3, 'OMOMAE':4, 'OTONUI':5, 'QOJWAO':6, 'TZBFZO':7,
                        'VOBXOZ':8, 'VUFJAI':9, 'VUFJAJ':10, 'WEPQOA':11, 'XADFUD':12,
                        'YABREY':13, 'YULCOZ':14, 'ZOVFEV':15}

                        loop = 0
                        entry1 = entry_reader1
                        entry2 = entry_reader2
                        entry3 = entry_reader3
                        entry4 = entry_reader4
                        while loop < 2:
                            #Since the first occurrence is based on the results from
                            the structures with entry SMILES strings
                            #All the first occurrence structures are written to the new
                            output file of first occurrences
                            first_occurrence = []
                            for a in range(len(entry1)):
                                if entry1[a].identifier in
                                duplicate_identifier_dictionary:

                                    first_occurrence.append(duplicate_identifier_dictiona
                                    ry.get(entry1[a].identifier))
                                else:
                                    first_occurrence.append(hash(entry1[a].identifier))
                                if loop == 0:
                                    new_lists_writer.write(entry1[a])
                                if loop == 1:
                                    writer2.write(entry1[a])

                            #The same is done for all the duplicate structures from the
                            structures with entry SMILES strings
                            duplicate = []
                            for b in range(len(entry2)):
                                if entry2[b].identifier in
                                duplicate_identifier_dictionary:

                                    duplicate.append(duplicate_identifier_dictionary.get(
                                    entry2[b].identifier))
                                else:
                                    duplicate.append(hash(entry2[b].identifier))
                                if loop == 0:
                                    writer1.write(entry2[b])
                                if loop == 1:
                                    writer3.write(entry2[b])

                            #First occurrence and duplicate results from structures
                            without entry SMILES strings are screened
                            for c in range(len(entry3)):
                                if entry3[c].identifier in
                                duplicate_identifier_dictionary:
                                    query1 =
                                    duplicate_identifier_dictionary.get(entry3[c].identif
                                    ier)
                                else:
                                    query1 = hash(entry3[c].identifier)
                                if entry4[c].identifier in
                                duplicate_identifier_dictionary:
                                    query2 =
                                    duplicate_identifier_dictionary.get(entry4[c].identif
                                    ier)
                                else:
                                    query2 = hash(entry4[c].identifier)
```

```
102                             #All structures from the entries to keep files are
                                without an entry SMILES string
103                             #Therefore they will not be present in the
                                first_occurrence and duplicate lists because these only
                                contain structures with entry SMILES strings
104                             #Only the structures in the duplicates to remove files
                                are compared to these lists
105                             #If the structure from the duplicates to remove file is
                                not in the first_occurrence and duplicate lists, the
                                structure is part of a new duplicate pair
106                             #The structures are written to the appropriate output
                                files and the pair is added to the first_occurrence and
                                duplicate lists
107                             if query2 not in first_occurrence and query2 not in
                                duplicate:
108                                 if loop == 0:
109                                     new_lists_writer.write(entry3[c])
110                                     writer1.write(entry4[c])
111                                 if loop == 1:
112                                     writer2.write(entry3[c])
113                                     writer3.write(entry4[c])
114                                 first_occurrence.append(query1)
115                                 duplicate.append(query2)
116                             else:
117                                 #If the structure from the duplicates to remove
                                    file is in the first_occurrence list, the pair
                                    needs to be re-written
118                                 #For example, A is in the entries to keep file and
                                    B is its partener in the duplicates to remove file
119                                 #The first occurrence list has B paired with C in
                                    the duplicates list
120                                 #Therefore the pair A and B needs to be swapped so
                                    B is the first occurrence and A is the duplicate
121                                 if query2 in first_occurrence:
122                                     #Here the duplicate to remove is written to the
                                        first occurrence output file and the entry to
                                        keep is written to the duplicate file
123                                     if loop == 0:
124                                         new_lists_writer.write(entry4[c])
125                                         writer1.write(entry3[c])
126                                     if loop == 1:
127                                         writer2.write(entry4[c])
128                                         writer3.write(entry3[c])
129                                     first_occurrence.append(query2)
130                                     duplicate.append(query1)
131                                 #If the structure from the duplicates to remove
                                    file is in the duplicates list, the entry to keep
                                    needs to paired with the appropriate first occurrence
132                                 #For example, A is in the entries to keep file and
                                    B is its partener in the duplicates to remove file
133                                 #The duplicates list has B paired with C in the
                                    first occurrence list
134                                 #This means A is also a duplicate of C
135                                 #An additional instance of C needs to be added to
                                    first_occurrence and A needs to be added to
                                    duplicates
136                                 elif query2 in duplicate:
137                                     #Here the location of B in duplicates is used
                                        to find where C is in first_occurrence
138                                     #Here the entry to keep is written to the
                                        duplicate output file and the first_occurrence
                                        (i.e. C) is written to the first_occurrence file
139                                     place = duplicate.index(query2)
140                                     if loop == 0:
141                                         new_lists_writer.write(entry1[place])
142                                         writer1.write(entry3[c])
143                                     if loop == 1:
```

```
                                        writer2.write(entry1[place])
                                        writer3.write(entry3[c])
                                    first_occurrence.append(first_occurrence[place])
                                    duplicate.append(query1)

                        loop += 1
                        if loop == 1:
                            entry1 = entry_reader5
                            entry2 = entry_reader6
                            entry3 = entry_reader7
                            entry4 = entry_reader8




if __name__ == '__main__':
    r = Runner()
    r.run()
```

```
#17 Find stoichiometrically distinct duplicate structures

from ccdc import io, search
import argparse
import os
import glob
import re

#This script finds duplicate matches that have different stoichiometries for the molecular components

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\hydrates_all_first_occurrence.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\hydrates_all_duplicates.txt"
filepath3 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_all_first_occurrence.txt"
filepath4 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_all_duplicates.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='check_formula_hydrate_entry.gcd',
            help='output file [check_formula_hydrate_entry.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')
        entry_reader2 = io.EntryReader(filepath2, format='identifiers')
        entry_reader3 = io.EntryReader(filepath3, format='identifiers')
        entry_reader4 = io.EntryReader(filepath4, format='identifiers')

        with io.EntryWriter(self.args.output) as new_lists_writer:
            with io.EntryWriter("check_formula_hydrate_duplicate.gcd") as writer1:
                with io.EntryWriter("check_formula_waterless_form_entry.gcd") as writer2:
                    with io.EntryWriter("check_formula_waterless_form_duplicate.gcd")
                    as writer3:

                    loop = 0
                    entry1 = entry_reader1
                    entry2 = entry_reader2
                    while loop < 2:
                        for a in range(len(entry1)):
                            #The formula of each structure in the match is used to compare the stoichiometry of each molecule in the structure
                            formulas1 = entry1[a].formula.split(',')
                            #Once again, any formulas containing deuterium are modified to contain the equivalent number of hydrogen atoms
                            for b in range(len(formulas1)):
```

```
                    if 'H' in formulas1[b] and 'D' in formulas1[b]:
                        place = formulas1[b].index('D')
                        place2 = formulas1[b].index('H')
                        #Structures with more than 9 or less than 10 hydrogen
                        or deuterium atoms are specified below so the script is
                        able to pull out the correct number of atoms in each case
                        #Here there are less than 10 deuterium atoms and less
                        than 10 hydrogen atoms
                        #The formula ends with the number of deuterium atoms in
                        this case
                        #In all other cases the number of hydrogen and
                        deuterium atoms are not at the end of the formula and
                        the location of a space can be used to determine the
                        place value (one or ten)
                        if len(str(formulas1[b])) == place+2:
                            if formulas1[b][place2+2] == ' ':
                                total = int(formulas1[b][place+1]) +
                                int(formulas1[b][place2+1])
                                formulas1[b] =
                                formulas1[b].replace(formulas1[b][place2:place2+2
                                ], str('H') + str(total))
                                formulas1[b] = formulas1[b][:place-1]
                            else:
                                print 'line 58'
                                print entry1[a].identifier
                        #Here there are less than 10 deuterium atoms and less
                        than 10 hydrogen atoms
                        elif formulas1[b][place2+2] == ' ' and
                        formulas1[b][place+2] == ' ':
                            total = int(formulas1[b][place+1]) +
                            int(formulas1[b][place2+1])
                            formulas1[b] =
                            formulas1[b].replace(formulas1[b][place2:place2+2],
                            str('H') + str(total))
                            formulas1[b] = formulas1[b][:place] +
                            formulas1[b][place+3:]
                        #Here there are less than 10 deuterium atoms and more
                        than 9 hydrogen atoms
                        elif entry1[a].identifier == 'ROBYEM01' or
                        entry1[a].identifier == 'ELOCIR' or
                        entry1[a].identifier == 'DOMQOL' or
                        entry1[a].identifier == 'VISZAZ' or
                        entry1[a].identifier == 'UNATAE':
                            total = int(formulas1[b][place+1]) +
                            int(formulas1[b][place2+1:place2+3])
                            formulas1[b] =
                            formulas1[b].replace(formulas1[b][place2:place2+3],
                            str('H') + str(total))
                            formulas1[b] = formulas1[b][:place] +
                            formulas1[b][place+3:]
                        #Here there are more than 9 deuterium atoms and more
                        than 9 hydrogen atoms
                        elif entry1[a].identifier == 'SIGHOF01':
                            total = int(formulas1[b][place+1:place+3]) +
                            int(formulas1[b][place2+1:place2+3])
                            formulas1[b] =
                            formulas1[b].replace(formulas1[b][place2:place2+3],
                            str('H') + str(total))
                            formulas1[b] = formulas1[b][:place] +
                            formulas1[b][place+3:]
                        #Here there are more than 9 deuterium atoms and less
                        than 10 hydrogen atoms
                        elif entry1[a].identifier == 'QQQAEJ01':
                            total = int(formulas1[b][place+1:place+3]) +
                            int(formulas1[b][place2+1])
                            formulas1[b] =
                            formulas1[b].replace(formulas1[b][place2:place2+2],
```

```
                                str('H') + str(total))
94                              formulas1[b] = formulas1[b][:place] +
                                formulas1[b][place+3:]
95                          else:
96                              print 'line 61'
97                              print entry1[a].identifier
98                      elif 'D' in formulas1[b]:
99                          formulas1[b] = re.sub(r'[D]', 'H', formulas1[b])
100                 formulas2 = entry2[a].formula.split(',')
101                 #The same procedure is carried out on the duplicate structures
102                 for c in range(len(formulas2)):
103                     if 'H' in formulas2[c] and 'D' in formulas2[c]:
104                         place = formulas2[c].index('D')
105                         place2 = formulas2[c].index('H')
106                         if len(str(formulas2[c])) == place+2:
107                             if formulas2[c][place2+2] == ' ':
108                                 total = int(formulas2[c][place+1]) +
                                    int(formulas2[c][place2+1])
109                                 formulas2[c] =
                                    formulas2[c].replace(formulas2[c][place2:place2+2
                                    ], str('H') + str(total))
110                             elif entry2[a].identifier == 'ANTMEU04' or
                                entry2[a].identifier == 'HATYUW01' or
                                entry2[a].identifier == 'GASGEM01':
111                                 total = int(formulas2[c][place+1]) +
                                    int(formulas2[c][place2+1:place2+3])
112                                 formulas2[c] =
                                    formulas2[c].replace(formulas2[c][place2:place2+3
                                    ], str('H') + str(total))
113                             else:
114                                 print 'line 78'
115                                 print entry2[a].identifier
116                             formulas2[c] = formulas2[c][:place-1]
117                         elif formulas2[c][place2+2] == ' ' and
                            formulas2[c][place+2] == ' ':
118                             total = int(formulas2[c][place+1]) +
                            int(formulas2[c][place2+1])
119                             formulas2[c] =
                            formulas2[c].replace(formulas2[c][place2:place2+2],
                            str('H') + str(total))
120                             formulas2[c] = formulas2[c][:place] +
                            formulas2[c][place+3:]
121                         elif entry2[a].identifier == 'DIBZAQ03' or
                            entry2[a].identifier == 'DIBZAQ04' or
                            entry2[a].identifier == 'ROBYEM02' or
                            entry2[a].identifier == 'REXFIM01' or
                            entry2[a].identifier == 'GUQXIB01' or
                            entry2[a].identifier == 'LUSLOA01' or
                            entry2[a].identifier == 'MAQWIM09' or
                            entry2[a].identifier == 'MAQWIM10' or
                            entry2[a].identifier == 'MAQWIM11' or
                            entry2[a].identifier == 'MAQWIM12' or
                            entry2[a].identifier == 'MAQWIM14' or
                            entry2[a].identifier == 'MAQWIM06' or
                            entry2[a].identifier == 'MAQWIM07' or
                            entry2[a].identifier == 'MAQWIM08' or
                            entry2[a].identifier == 'MAQWIM13' or
                            entry2[a].identifier == 'OROQET' or
                            entry2[a].identifier == 'ICUXUC01' or
                            entry2[a].identifier == 'NALWUS03' or
                            entry2[a].identifier == 'KOFCUE05' or
                            entry2[a].identifier == 'NEZRUF01' or
                            entry2[a].identifier == 'RUJCII01' or
                            entry2[a].identifier == 'UPIVIA01' or
                            entry2[a].identifier == 'VIWQUO' or
                            entry2[a].identifier == 'GESXOT01' or
                            entry2[a].identifier == 'MPRGOM01' or
```

```
                                    entry2[a].identifier == 'TUTBUG01' or
                                    entry2[a].identifier == 'KOFCUE04':
122                                     total = int(formulas2[c][place+1]) +
                                        int(formulas2[c][place2+1:place2+3])
123                                     formulas2[c] =
                                        formulas2[c].replace(formulas2[c][place2:place2+3],
                                        str('H') + str(total))
124                                     formulas2[c] = formulas2[c][:place] +
                                        formulas2[c][place+3:]
125                                 elif entry2[a].identifier == 'SIGHOF02' or
                                    entry2[a].identifier == 'LEPXOT01':
126                                     total = int(formulas2[c][place+1:place+3]) +
                                        int(formulas2[c][place2+1:place2+3])
127                                     formulas2[c] =
                                        formulas2[c].replace(formulas2[c][place2:place2+3],
                                        str('H') + str(total))
128                                     formulas2[c] = formulas2[c][:place] +
                                        formulas2[c][place+3:]
129                                 elif entry2[a].identifier == 'YUHHUE02' or
                                    entry2[a].identifier == 'NALWUS04' or
                                    entry2[a].identifier == 'NALWUS05' or
                                    entry2[a].identifier == 'NALWUS06' or
                                    entry2[a].identifier == 'NALWUS07':
130                                     total = int(formulas2[c][place+1:place+3]) +
                                        int(formulas2[c][place2+1])
131                                     formulas2[c] =
                                        formulas2[c].replace(formulas2[c][place2:place2+2],
                                        str('H') + str(total))
132                                     formulas2[c] = formulas2[c][:place] +
                                        formulas2[c][place+3:]
133                                 else:
134                                     print 'line 81'
135                                     print entry2[a].identifier
136                             elif 'D' in formulas2[c]:
137                                 formulas2[c] = re.sub(r'[D]', 'H', formulas2[c])
138                         #The formulas are sorted so the molecules will appear in the
                            same order for structures composed of the same molecules
139                         #If the formulas do not match, each structure is written to an
                            output file where they are flagged as having distinct formulas
140                         formulas1 = sorted(formulas1)
141                         formulas2 = sorted(formulas2)
142                         if formulas1 != formulas2:
143                             new_lists_writer.write(entry1[a])
144                             writer1.write(entry2[a])
145
146                     loop += 1
147                     if loop == 1:
148                         entry1 = entry_reader3
149                         entry2 = entry_reader4
150
151
152
153
154
155
156
157
158
159
160
161
162
163 if __name__ == '__main__':
164     r = Runner()
165     r.run()
166
```

```python
#18 Find distinct identifier prefix duplicate structures

from ccdc import io, search
import argparse
import os
import glob
import re

#This script finds duplicate matches that have a different set of the first six letters of the refcode
#Structures with the same first six letters in their refcode are multiple entries of the same structural components
#It is plausible that structures with distinct sets of the first six letters are composed of structural components that do not share the same stereochemistry
#Therefore, these matches were isolated to determine if there are any chiral centers that do not match up (R and S configurations)

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\hydrates_all_first_occurrence.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\hydrates_all_duplicates.txt"
filepath3 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_all_first_occurrence.txt"
filepath4 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_all_duplicates.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='check_chirality_hydrates_entry.gcd',
            help='output file [check_chirality_hydrates_entry.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')
        entry_reader2 = io.EntryReader(filepath2, format='identifiers')
        entry_reader3 = io.EntryReader(filepath3, format='identifiers')
        entry_reader4 = io.EntryReader(filepath4, format='identifiers')

        with io.EntryWriter(self.args.output) as new_lists_writer:
            with io.EntryWriter("check_chirality_hydrates_duplicate.gcd") as writer1:
                with io.EntryWriter("check_chirality_waterless_forms_entry.gcd") as writer2:
                    with io.EntryWriter("check_chirality_waterless_forms_duplicate.gcd") as writer3:

                        loop = 0
                        entry1 = entry_reader1
                        entry2 = entry_reader2
```

```
                        while loop < 2:
                            for a in range(len(entry1)):
                                #The first six letters of each identifer are compared
                                #Any cases where the first six letters do not match are
                                written to output file where they are flagged as having
                                distinct six letter identifiers
                                if entry1[a].identifier[:6] != entry2[a].identifier[:6]:
                                    if loop == 0:
                                        new_lists_writer.write(entry1[a])
                                        writer1.write(entry2[a])
                                    if loop == 1:
                                        writer2.write(entry1[a])
                                        writer3.write(entry2[a])
                            loop += 1
                            if loop == 1:
                                entry1 = entry_reader3
                                entry2 = entry_reader4
```

```
if __name__ == '__main__':
    r = Runner()
    r.run()
```

```
#19 Check chirality of duplicates with distinct identifier prefixes

from ccdc import io, search
import argparse
import os
import glob
import re

#This script determines which matches with different sets of the first six letters in their refcodes have distinct stereochemistry
#The hydrate structures and waterless form structures were evaulated with this script
#The input files and output file names were changed when switching from hydrate structures to waterless forms
#i.e. Every instance of the word "hydrate" was changed to "waterless form" in the filenames

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\check_chirality_hydrate_entry.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\check_chirality_hydrate_duplicate.txt"

#filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\check_chirality_waterless_forms_entry.txt"
#filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\check_chirality_waterless_forms_duplicate.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='same_chirality_hydrate_entry.gcd',
            help='output file [same_chirality_hydrate_entry.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')
        entry_reader2 = io.EntryReader(filepath2, format='identifiers')

        with io.EntryWriter(self.args.output) as chiral_writer:
            with io.EntryWriter("same_chirality_hydrate_duplicate.gcd") as writer1:
                with io.EntryWriter("distinct_chirality_hydrate_entry.gcd") as writer2:
                    with io.EntryWriter("distinct_chirality_hydrate_duplicate.gcd") as writer3:
                        with io.EntryWriter("achiral_names_one_missing_stereo_hydrate_entry.gcd") as writer4:
                            with io.EntryWriter("achiral_names_one_missing_stereo_hydrate_duplicate.gcd") as writer5:
                                with io.EntryWriter("one_chiral_name_one_missing_stereo_hydrat
```

```
                                e_entry.gcd") as writer6:
55                                  with
                                    io.EntryWriter("one_chiral_name_one_missing_stereo_hy
                                    drate_duplicate.gcd") as writer7:
56                                      with
                                        io.EntryWriter("chiral_names_check_hydrate_entry.
                                        gcd") as writer8:
57                                          with
                                            io.EntryWriter("chiral_names_check_hydrate_du
                                            plicate.gcd") as writer9:
58
59                                              duplicate_identifier_dictionary =
                                                {'GADKOL':1, 'MECYEY':2, 'OBASAN':3,
                                                'OMOMAE':4, 'OTONUI':5, 'QOJWAO':6,
                                                'TZBFZO':7, 'VOBXOZ':8, 'VUFJAI':9,
                                                'VUFJAJ':10, 'WEPQOA':11, 'XADFUD':12,
                                                'YABREY':13, 'YULCOZ':14, 'ZOVFEV':15}
60                                              #Each structure can output the formula
                                                of each molecule that is present in the
                                                structure
61                                              #In some cases the structure does not
                                                generate a list of molecule formulas
62                                              #These cases are recorded in the
                                                following lists and are treated in a
                                                separate section of the script from
                                                those that are able to generate a list
                                                of molecule formulas
63                                              #This only occurred for hydrate
                                                structures
64                                              hydrate_tags_empty_one_stereo_hydrates
                                                = [hash('BEXPOK'), hash('QQQCYA01'),
                                                hash('VOLZON'), hash('VOMPEU'),
                                                hash('WEGPED'), hash('AHLPRO'),
                                                hash('YUPBER')]
65
                                                hydrate_tags_empty_one_stereo_waterless_f
                                                orms = [hash('BEXPIE'), hash('NAMNSB'),
                                                hash('VOMNUI'), hash('VOMNUI'),
                                                hash('WEGPIH'), hash('HOPROL'),
                                                hash('VALINO01')]
66                                              hydrate_tags_empty_no_stereo_hydrates =
                                                [hash('ADPRTY'), hash('BEWWIK'),
                                                hash('BEWWIK'), hash('BUTAHC11'),
                                                hash('DUGTOQ'), hash('ETHYLO'),
                                                hash('ETHYLO'), hash('ETXDHY01'),
                                                hash('ETXDHY01'), hash('FEFNOT01'),
                                                hash('FEFNOT01'), hash('FEFNOT01'),
                                                hash('FEFNOT01'), hash('FEFNOT01'),
                                                hash('FEFNOT01'), hash('FEFNOT01'),
                                                hash('FEFNOT01'), hash('FEFNOT01'),
                                                hash('FEFNOT01'), hash('FEFNOT01'),
                                                hash('FEFNOT01'), hash('FEFNOT01'),
                                                hash('FEFNOT01'), hash('FEFNOT01'),
                                                hash('FEFNOT01'), hash('FEFNOT01'),
                                                hash('FEFNOT05'), hash('FEFNOT05'),
                                                hash('FEFNOT05'), hash('FEFNOT05'),
                                                hash('FEFNOT05'), hash('FEFNOT05'),
                                                hash('FEFNOT05'), hash('FEFNOT05'),
                                                hash('FEFNOT05'), hash('FEFNOT05'),
                                                hash('FEFNOT05'), hash('FEFNOT05'),
                                                hash('FEFNOT05'), hash('FEFNOT05'),
                                                hash('FEFNOT05'), hash('FEFNOT05'),
                                                hash('FEFNOT05'), hash('FEFNOT10'),
                                                hash('FEFNOT10'), hash('FEFNOT10'),
                                                hash('FEFNOT10'), hash('FEFNOT10'),
```

```
hash('FEFNOT10'), hash('FEFNOT10'),
hash('FEFNOT10'), hash('FEFNOT10'),
hash('FEFNOT10'), hash('FEFNOT10'),
hash('FEFNOT10'), hash('FEFNOT10'),
hash('FEFNOT10'), hash('FEFNOT10'),
hash('FEFNOT10'), hash('FEFNOT10'),
hash('FEFNOT10'), hash('FEFNOT10'),
hash('FEFNOT11'), hash('FEFNOT11'),
hash('FEFNOT11'), hash('FEFNOT11'),
hash('FEFNOT11'), hash('FEFNOT11'),
hash('FEFNOT11'), hash('FEFNOT11'),
hash('FEFNOT11'), hash('FEFNOT11'),
hash('FEFNOT11'), hash('FEFNOT11'),
hash('FEFNOT11'), hash('FEFNOT11'),
hash('FEFNOT11'), hash('FEFNOT11'),
hash('FEFNOT11'), hash('FEFNOT11'),
hash('FEFNOT11'), hash('FUSWOH04'),
hash('FUSWOH04'), hash('FUSWOH04'),
hash('FUSWOH04'), hash('FUSWOH04'),
hash('FUSWOH04'), hash('FUSWOH04'),
hash('FUSWOH04'), hash('IKALIR'),
hash('IKALIR'), hash('IKALIR01'),
hash('IKALIR01'), hash('IKALIR02'),
hash('IKALIR02'), hash('IKALIR03'),
hash('IKALIR03'), hash('IKAMEO'),
hash('IKAMEO'), hash('IKAMEO01'),
hash('IKAMEO01'), hash('IKAMOY'),
hash('IKAMOY'), hash('IKAMOY01'),
hash('IKAMOY01'), hash('IKAMOY02'),
hash('IKAMOY02'), hash('IKAMOY03'),
hash('IKAMOY03'), hash('IKAMOY04'),
hash('IKAMOY04'), hash('IKAMOY05'),
hash('IKAMOY05'), hash('IKAVOF'),
hash('IKAVOF'), hash('IKAVOF'),
hash('IKAVOF'), hash('IKAVOF'),
hash('IKAVOF'), hash('IKAVOF'),
hash('IKAVOF'), hash('IKAVOF'),
hash('JARBUA'), hash('KUYGOA'),
hash('KUYGUG'), hash('KUYGUG'),
hash('KUYGUG'), hash('KUYGUG'),
hash('KUYGUG'), hash('KUYGUG'),
hash('KUYHAN'), hash('KUYHAN'),
hash('KUYHAN'), hash('KUYHAN'),
hash('KUYHAN'), hash('KUYHAN'),
hash('KUYHAN'), hash('KUYHAN'),
hash('KUYHAN'), hash('KUYHAN'),
hash('KUYHER'), hash('KUYHER'),
hash('MUSDEL'), hash('MUSDOV'),
hash('NAHCIJ'), hash('NAHCIJ'),
hash('NAHCOP'), hash('NAHCOP'),
hash('NAHCOP'), hash('NAHCOP'),
hash('NAHCOP'), hash('NAHCOP'),
hash('NAHCOP'), hash('NAHCOP'),
hash('QQQCIY10'), hash('QQQCIY10'),
hash('QQQCIY10'), hash('QQQCIY10'),
hash('QQQCIY10'), hash('QQQCIY10'),
hash('QQQCIY10'), hash('RITMEN'),
hash('RITMEN'), hash('SOGUAN06'),
hash('XEBQAZ'), hash('ZZZPPI01'),
hash('ZZZWQA01'), hash('ZZZWQA01'),
hash('FEFNOT'), hash('FEFNOT'),
hash('FEFNOT'), hash('FEFNOT'),
hash('FEFNOT'), hash('FEFNOT'),
hash('FEFNOT'), hash('FEFNOT'),
hash('FEFNOT'), hash('FEFNOT'),
hash('FEFNOT'), hash('FEFNOT'),
hash('FEFNOT'), hash('FEFNOT'),
```

```
hash('CBMZPN02'), hash('CBMZPN03'),
hash('CBMZPN10'), hash('CBMZPN11'),
hash('CBMZPN12'), hash('CBMZPN13'),
hash('CBMZPN14'), hash('CBMZPN16'),
hash('CBMZPN17'), hash('CBMZPN18'),
hash('CBMZPN20'), hash('CBMZPN21'),
hash('CBMZPN22'), hash('CBMZPN23'),
hash('CBMZPN27'), hash('CBMZPN28'),
hash('CBMZPN29'), hash('CBMZPN30'),
hash('CBMZPN01'), hash('CBMZPN02'),
hash('CBMZPN03'), hash('CBMZPN10'),
hash('CBMZPN11'), hash('CBMZPN12'),
hash('CBMZPN13'), hash('CBMZPN14'),
hash('CBMZPN16'), hash('CBMZPN17'),
hash('CBMZPN18'), hash('CBMZPN20'),
hash('CBMZPN21'), hash('CBMZPN22'),
hash('CBMZPN23'), hash('CBMZPN27'),
hash('CBMZPN28'), hash('CBMZPN29'),
hash('CBMZPN30'), hash('CBMZPN01'),
hash('CBMZPN02'), hash('CBMZPN03'),
hash('CBMZPN10'), hash('CBMZPN11'),
hash('CBMZPN12'), hash('CBMZPN13'),
hash('CBMZPN14'), hash('CBMZPN16'),
hash('CBMZPN17'), hash('CBMZPN18'),
hash('CBMZPN20'), hash('CBMZPN21'),
hash('CBMZPN22'), hash('CBMZPN23'),
hash('CBMZPN27'), hash('CBMZPN28'),
hash('CBMZPN29'), hash('CBMZPN30'),
hash('EFUMAU'), hash('EFUMAU01'),
hash('EFUMAU02'), hash('EFUMAU03'),
hash('EFUMAU04'), hash('EFUMAU05'),
hash('EFUMAU06'), hash('EFUMAU07'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('ETHANE01'), hash('ETHANE04'),
hash('ETHANE05'), hash('ETHANE06'),
hash('ETHANE07'), hash('ETHANE08'),
hash('ETHANE09'), hash('ETHANE10'),
hash('ETHANE11'), hash('JARBOU'),
hash('ZZZITY01'), hash('KAMTAW'),
hash('KAMTAW01'), hash('KAMTAW02'),
hash('KAMTAW03'), hash('KAMTAW04'),
hash('KAMTAW05'), hash('KAMTAW06'),
hash('DCLBEN'), hash('DCLBEN01'),
hash('DCLBEN02'), hash('DCLBEN03'),
hash('DCLBEN04'), hash('DCLBEN05'),
hash('DCLBEN06'), hash('DCLBEN07'),
hash('DCLBEN11'), hash('CISTON01'),
hash('TAQYUG'), hash('DUHJIB'),
hash('DUHJIB'), hash('ACETYL02'),
hash('ACETYL03'), hash('JAYDUI'),
hash('JAYDUI01'), hash('JAYDUI02'),
hash('JAYDUI03'), hash('JAYDUI04'),
hash('JAYDUI05'), hash('JAYDUI06'),
hash('JAYDUI07'), hash('CEKGUU'),
hash('CEKGUU01'), hash('CEKGUU02'),
hash('CEKGUU03'), hash('CEKGUU04'),
```

```
                                    hash('DAYFUH'), hash('LUXYIM'),
                                    hash('LUYMAT'), hash('LUYMAT01'),
                                    hash('NIWFEE02'), hash('NIWFEE02'),
                                    hash('NIWFEE04'), hash('NIWFEE04'),
                                    hash('SABZOK'), hash('SOCTOT'),
                                    hash('THIOUR06'), hash('THIOUR06'),
                                    hash('THIOUR06'), hash('THIOUR06'),
                                    hash('PUSBOV')]
68
69                                  #This script was adapted from the one
                                    used to find hydrate-anhydrate pairs
                                    with distinct stereochemistry
70                                  #The hydrates list contains the
                                    information for the entry to keep
                                    (first occurrence)
71                                  #The waterless list contains the
                                    informaiton for the duplicate
72                                  entry1 = entry_reader1
73                                  hydrates = []
74                                  waterless = []
75                                  count1 = 0
76                                  count2 = 0
77                                  count3 = 0
78                                  #The formula for the entry will always
                                    be complete unless a solvent molecule
                                    is squeezed out
79                                  #The formula of a molecule in an entry
                                    can be incomplete or overdone
80                                  #An example would be a disordered
                                    structure where a hydrogen atom 3D
                                    coordinate is not reported (incomplete)
                                    or a carbon atom position has three
                                    possibilities and is listed three times
                                    (overdone)
81                                  #This part of the script compares the
                                    entry formula pieces to each molecule
                                    formula to find cases where they do not
                                    match
82                                  #Cases where one formula is deuterated
                                    and the other is not are considered a
                                    match
83                                  for a in range(len(entry1)):
84                                      print a
85                                      formula_error = 0
86                                      missing_components = 0
87                                      loop2 = 0
88                                      while loop2 < 2:
89                                          letter_formulas = []
90                                          formulas =
                                            entry1[a].formula.split(',')
91                                          for b in range(len(formulas)):
92                                              if '(' in formulas[b] and
                                                ')' in formulas[b]:
93                                                  if '(H2 O1)' not in
                                                    formulas[b] and '(D2
                                                    O1)' not in formulas[b]:
94                                                      pin =
                                                        formulas[b].index('('
                                                        )
95
                                                        letter_formulas.appen
                                                        d(formulas[b][pin+1:-
                                                        1])
96                                              else:
97                                                  if formulas[b] != 'H2
                                                    O1' and formulas[b] !=
                                                    'D2 O1':
```

```
                                                                letter_formulas.appen
                                                                d(formulas[b])
                                                    molecule1 = entry1[a].molecule
                                                    #Checked contains all the
                                                    molecule formulas that have
                                                    been matched to a formula in
                                                    the entry formula
                                                    #Tags contains the location of
                                                    the molecule formula that has
                                                    been found in the list of
                                                    molecule.components
                                                    checked = []
                                                    tags = []
                                                    #Error records the number of
                                                    cases where a molecule formula
                                                    does not match any entry formulas
                                                    #Count records the number of
                                                    molecule formulas that have
                                                    matched formulas in the entry
                                                    formula
                                                    error = 0
                                                    count = 0
                                                    #Molecule.components generates
                                                    the molecule formulas
                                                    for g in
                                                    range(len(molecule1.components)):
                                                        if
                                                        molecule1.components[g].formu
                                                        la not in checked and
                                                        molecule1.components[g].formu
                                                        la != 'H2 O1':
                                                            if
                                                            molecule1.components[g].f
                                                            ormula not in
                                                            letter_formulas:
                                                                if 'H' in
                                                                molecule1.components[
                                                                g].formula:
                                                                    if
                                                                    molecule1.compone
                                                                    nts[g].formula.re
                                                                    place('H', 'D')
                                                                    not in
                                                                    letter_formulas:

partial_deuterated = 0                                                                  for t in
range(len(formulas)):                                                                       if 'H'
in formulas[t] and 'D' in formulas[t]:

pieces = formulas[t].split(' ')                                                                 for
s in range(len(pieces)):
if 'H' in pieces[s]:
    tag1 = pieces[s].index('H')
    number1 = int(pieces[s][tag1+1:])
if 'D' in pieces[s]:
    tag2 = pieces[s].index('D')
    number2 = int(pieces[s][tag2+1:])
```

```
                    partial_deuterated += 1
                if partial_deuterated == 1:
                    pieces2 = molecule1.components[g].formula.split(' ')
                    for u in range(len(pieces2)):
                        if 'H' in pieces2[u]:
                            tag3 = pieces2[u].index('H')
                            number3 = int(pieces2[u][tag3+1:])
                            if number1+number2 == number3:
                                checked.append(molecule1.components[g].formula)
                                tags.append(g)
                                count += 1
                            else:
                                checked.append(molecule1.components[g].formula)
                                tags.append(g)
                                error += 1
                else:
                    checked.append(molecule1.components[g].formula)
                    tags.append(g)
                    error += 1
            else:
                checked.append(molecule1.components[g].formula)
                tags.append(g)
                count += 1
        else:
            checked.append(molecule1.components[g].formula)
            tags.append(g)
            error += 1
    else:
        checked.append(molecule1.components[g].formula)
        tags.append(g)
        count += 1
if loop2 == 0:
    hydrate_tags = tags
    hydrates_checked = checked
else:
    waterless_tags = tags
    waterless_checked = checked
if error != 0:
    formula_error += 1
```

```
164                                     #If the number of molecule
                                        formulas found does not match
                                        the total number of formulas in
                                        the entry formula, then one of
                                        the molecule formulas is
                                        missing from that entry
165                                     if count != len(letter_formulas):
166                                         missing_components += 1
167                                     loop2 += 1
168                                     entry1 = entry_reader2
169
170
171                                 entry2 = entry_reader1
172                                 entry3 = entry_reader2
173
174                                 #Any cases where there is an error
                                    or components are missing go
                                    through a secondary refinement
175                                 #If one structure (ex. entry) is
                                    missing a molecule, then that
                                    molecule is removed from the other
                                    structure (ex. duplicate) since
                                    they can now no longer be compared
176                                 if formula_error != 0 or
                                    missing_components != 0:
177                                     remove_hydrate_tags = []
178                                     for c in
                                        range(len(hydrate_tags)):
179                                         if
                                            entry2[a].molecule.components
                                            [hydrate_tags[c]].formula
                                            not in waterless_checked:
180
                                                remove_hydrate_tags.appen
                                                d(hydrate_tags[c])
181                                     for e in
                                        range(len(remove_hydrate_tags)):
182
                                            hydrate_tags.remove(remove_hy
                                            drate_tags[e])
183                                     remove_waterless_tags = []
184                                     for d in
                                        range(len(waterless_tags)):
185                                         if
                                            entry3[a].molecule.components
                                            [waterless_tags[d]].formula
                                            not in hydrates_checked:
186
                                                remove_waterless_tags.app
                                                end(waterless_tags[d])
187                                     for f in
                                        range(len(remove_waterless_tags))
                                        :
188
                                            waterless_tags.remove(remove_
                                            waterless_tags[f])
189
190                                 #If the length of molecules for
                                    each structure does not match, the
                                    identifiers of the two structures
                                    are printed for a manual inspection
191                                 if len(hydrate_tags) !=
                                    len(waterless_tags):
192                                     print entry2[a].identifier
193                                     print entry3[a].identifier
194
195                                 #There are several possibilities
```

```
                                                    that can come from this script
196                                                 #Structures that have chiral
                                                    centers may not have that
                                                    information recorded for the atoms
                                                    of the chiral molecule
197                                                 #Structures that have chiral
                                                    centers may not have any indication
                                                    in the chemical name
198                                                 #The chemical name is used as a
                                                    preliminary check before
                                                    investigating the atom chiral
                                                    configurations of each molecule
199                                                 #If both structures have the same
                                                    name and the name indicates the
                                                    chirality, the match is true
200                                                 #If the structures have distinct
                                                    names, both of which indicate
                                                    chirality, the match is false
201                                                 #If either or both names do not
                                                    indicate the chirality, then the
                                                    chiral configuration of the atoms
                                                    in the molecules are checked
202                                                 same_names = 0
203                                                 possibly_the_same_names = 0
204                                                 distinct_names = 0
205                                                 #Double R = both names indicate
                                                    chirality but are not the same
206                                                 #Double blank = same names with no
                                                    chirality indicated
207                                                 #RandBlank = one name has chirality
                                                    and the other does not
208                                                 DoubleR = 0
209                                                 DoubleBlank = 0
210                                                 RandBlank = 0
211                                                 #The chemical name of each
                                                    structure, along with any synonyms
                                                    are placed into a list
212                                                 #The list for each structure are
                                                    compared against each other to see
                                                    if any of the names are the same
213                                                 hydrate_titles =
                                                    list(entry2[a].synonyms)
214                                                 if entry2[a].identifier == 'WACHIR':
215                                                     hydrate_titles =
                                                     hydrate_titles[0].split(',')
216
                                                    hydrate_titles.append(entry2[a].chemi
                                                    cal_name)
217                                                 for u in range(len(hydrate_titles)):
218                                                     caps = [i for i in
                                                     hydrate_titles[u] if i.isupper()]
219                                                     caps = list(set(caps))
220                                                     for q in range(len(caps)):
221                                                         if caps[q] != 'D' and
                                                         caps[q] != 'L' and caps[q]
                                                         != 'R' and caps[q] != 'S'
                                                         and caps[q] != 'E' and
                                                         caps[q] != 'Z':
222                                                             hydrate_titles[u] =
                                                             re.sub(caps[q],
                                                             caps[q].lower(),
                                                             hydrate_titles[u])
223                                                         else:
224                                                             places = [j for j,
                                                             letter in
                                                             enumerate(hydrate_titles[
                                                             u]) if letter == caps[q]]
```

```
                    for k in
                    range(len(places)):
                        if places[k]+1 ==
                        len(hydrate_titles[u]
                        ):

                            hydrate_titles[u]
                             =
                            hydrate_titles[u]
                            .replace(hydrate_
                            titles[u][places[
                            k]],
                            hydrate_titles[u]
                            [places[k]].lower
                            ())
                        elif
                        hydrate_titles[u][pla
                        ces[k]+1] != ')'
                        and
                        hydrate_titles[u][pla
                        ces[k]+1] != '-'
                        and
                        hydrate_titles[u][pla
                        ces[k]+1] != ',':

                            hydrate_titles[u]
                             =
                            hydrate_titles[u]
                            .replace(hydrate_
                            titles[u][places[
                            k]],
                            hydrate_titles[u]
                            [places[k]].lower
                            ())
            hydrate_pieces =
            hydrate_titles[u].split(' ')
            hydrate_deuterated = [s for s
            in hydrate_pieces if 'deutero'
            in s]
            if len(hydrate_deuterated) == 1:
                clip =
                hydrate_deuterated[0].index('
                deutero')
                if 'deutero-' in
                hydrate_deuterated[0]:
                    hydrate_replacement =
                    [hydrate_deuterated[0][cl
                    ip+8:]]
                else:
                    hydrate_replacement =
                    [hydrate_deuterated[0][cl
                    ip+7:]]
                for t in
                range(len(hydrate_pieces)):
                    if hydrate_pieces[t] ==
                    hydrate_deuterated[0]:
                        hydrate_pieces[t] =
                        hydrate_replacement[0
                        ]
            water = [r for r in
            hydrate_pieces if 'hydrate' in
            r and 'perhydrate' not in r]
            D2O = 0
            if len(water) == 0:
                if 'deuterium oxide
                solvate' in
                hydrate_titles[u].lower()
```

```
                    or '(deuterium oxide)
                    solvate' in
                    hydrate_titles[u].lower():
245                     water = [r for r in
                        hydrate_pieces if
                        'deuterium' in r or
                        'oxide' in r or
                        'solvate' in r]
246                 elif '(dideuterium oxide)'
                    in hydrate_titles[u].lower():
247                     water = [r for r in
                        hydrate_pieces if
                        'dideuterium' in r or
                        'oxide' in r]
248                 elif 'deuterium oxide' in
                    hydrate_titles[u].lower():
249                     water = [r for r in
                        hydrate_pieces if
                        'deuterium' in r or
                        'oxide' in r]
250                 hydrate_pieces = [v for v in
                    hydrate_pieces if v not in water]
251                 hydrate_titles[u] = '
                    '.join(hydrate_pieces)
252             waterless_titles =
                list(entry3[a].synonyms)
253
                waterless_titles.append(entry3[a].che
                mical_name)
254             for u in
                range(len(waterless_titles)):
255                 caps2 = [i for i in
                    waterless_titles[u] if
                    i.isupper()]
256                 caps2 = list(set(caps2))
257                 for q in range(len(caps2)):
258                     if caps2[q] != 'D' and
                        caps2[q] != 'L' and
                        caps2[q] != 'R' and
                        caps2[q] != 'S' and
                        caps2[q] != 'E' and
                        caps2[q] != 'Z':
259                         waterless_titles[u] =
                            re.sub(caps2[q],
                            caps2[q].lower(),
                            waterless_titles[u])
260                     else:
261                         places2 = [j for j,
                            letter in
                            enumerate(waterless_title
                            s[u]) if letter ==
                            caps2[q]]
262                         for k in
                            range(len(places2)):
263                             if places2[k]+1 ==
                                len(waterless_titles[
                                u]):
264
                                    waterless_titles[
                                    u] =
                                    waterless_titles[
                                    u].replace(waterl
                                    ess_titles[u][pla
                                    ces2[k]],
                                    waterless_titles[
                                    u][places2[k]].lo
                                    wer())
```

```
                    elif
                    waterless_titles[u][p
                    laces2[k]+1] != ')'
                    and
                    waterless_titles[u][p
                    laces2[k]+1] != '-'
                    and
                    waterless_titles[u][p
                    laces2[k]+1] != ',':

                        waterless_titles[
                        u] =
                        waterless_titles[
                        u].replace(waterl
                        ess_titles[u][pla
                        ces2[k]],
                        waterless_titles[
                        u][places2[k]].lo
                        wer())
            waterless_pieces =
            waterless_titles[u].split(' ')
            waterless_deuterated = [s for s
            in waterless_pieces if
            'deutero' in s.lower()]
            if len(waterless_deuterated) ==
            1:
                clip =
                waterless_deuterated[0].lower
                ().index('deutero')
                if 'deutero-' in
                waterless_deuterated[0].lower
                ():
                    waterless_replacement =
                    [waterless_deuterated[0][
                    clip+8:]]
                else:
                    waterless_replacement =
                    [waterless_deuterated[0][
                    clip+7:]]
                for t in
                range(len(waterless_pieces)):
                    if waterless_pieces[t]
                    ==
                    waterless_deuterated[0]:
                        waterless_pieces[t]
                        =
                        waterless_replacement
                        [0]
            waterless_titles[u] = '
            '.join(waterless_pieces)
#Words that indicate chirality are
searched for in each chemical name
(ex. rac, D-, R-, etc.)
#Racemates are searched for first,
followed by enantiomers, then cases
where there is no chirality in
either name, and cases where there
is chirality in one name
if any('rac-' in x for x in
hydrate_titles) and any('rac-' in y
for y in waterless_titles) or
any('DL' in x for x in
hydrate_titles) and any('DL' in y
for y in waterless_titles) or
any('RS' in x for x in
hydrate_titles) and any('RS' in y
for y in waterless_titles) or
```

```
                                    any('RS' in x for x in
                                    hydrate_titles) and any('SR' in y
                                    for y in waterless_titles) or
                                    any('SR' in x for x in
                                    hydrate_titles) and any('RS' in y
                                    for y in waterless_titles) or
                                    any('SR' in x for x in
                                    hydrate_titles) and any('SR' in y
                                    for y in waterless_titles) or
                                    any('+-' in x for x in
                                    hydrate_titles) and any('+-' in y
                                    for y in waterless_titles):
282                                     same_names += 1
283                                     #If the two names are the same,
                                        the two structures are written
                                        to output files for matches
                                        with the same chirality
284                                     if bool(set(hydrate_titles) &
                                        set(waterless_titles)) == True:
285
                                            chiral_writer.write(entry2[a]
                                            )
286                                         writer1.write(entry3[a])
287                                     else:
288                                         possibly_the_same_names += 1
289                                         DoubleR += 1
290
291                                 elif any('D' in x for x in
                                    hydrate_titles) and any('D' in y
                                    for y in waterless_titles) or
                                    any('L' in x for x in
                                    hydrate_titles) and any('L' in y
                                    for y in waterless_titles) or
                                    any('R' in x for x in
                                    hydrate_titles) and any('R' in y
                                    for y in waterless_titles) or
                                    any('S' in x for x in
                                    hydrate_titles) and any('S' in y
                                    for y in waterless_titles) or
                                    any('(+)-' in x for x in
                                    hydrate_titles) and any('(+)-' in y
                                    for y in waterless_titles) or
                                    any('(-)-' in x for x in
                                    hydrate_titles) and any('(-)-' in y
                                    for y in waterless_titles) or
                                    any('cis-' in x for x in
                                    hydrate_titles) and any('cis-' in y
                                    for y in waterless_titles) or
                                    any('trans-' in x for x in
                                    hydrate_titles) and any('trans-' in
                                    y for y in waterless_titles) or
                                    any('meso-' in x for x in
                                    hydrate_titles) and any('meso-' in
                                    y for y in waterless_titles) or
                                    any('E' in x for x in
                                    hydrate_titles) and any('E' in y
                                    for y in waterless_titles) or
                                    any('Z' in x for x in
                                    hydrate_titles) and any('Z' in y
                                    for y in waterless_titles):
292                                     same_names += 1
293                                     if bool(set(hydrate_titles) &
                                        set(waterless_titles)) == True:
294
                                            chiral_writer.write(entry2[a]
                                            )
295                                         writer1.write(entry3[a])
```

```
                else:
                    possibly_the_same_names += 1
                    DoubleR += 1
            elif all('rac-' not in x for x in
            hydrate_titles) and all('rac-' not
            in y for y in waterless_titles) and
            all('+-' not in x for x in
            hydrate_titles) and all('+-' not in
            y for y in waterless_titles) and
            all('D' not in x for x in
            hydrate_titles) and all('D' not in
            y for y in waterless_titles) and
            all('L' not in x for x in
            hydrate_titles) and all('L' not in
            y for y in waterless_titles) and
            all('R' not in x for x in
            hydrate_titles) and all('R' not in
            y for y in waterless_titles) and
            all('S' not in x for x in
            hydrate_titles) and all('S' not in
            y for y in waterless_titles) and
            all('(+)-' not in x for x in
            hydrate_titles) and all('(+)-' not
            in y for y in waterless_titles) and
            all('(-)-' not in x for x in
            hydrate_titles) and all('(-)-' not
            in y for y in waterless_titles) and
            all('cis-' not in x for x in
            hydrate_titles) and all('cis-' not
            in y for y in waterless_titles) and
            all('trans-' not in x for x in
            hydrate_titles) and all('trans-'
            not in y for y in waterless_titles)
            and all('meso-' not in x for x in
            hydrate_titles) and all('meso-' not
            in y for y in waterless_titles) and
            all('E' not in x for x in
            hydrate_titles) and all('E' not in
            y for y in waterless_titles) and
            all('Z' not in x for x in
            hydrate_titles) and all('Z' not in
            y for y in waterless_titles):
                possibly_the_same_names += 1
                DoubleBlank += 1
            else:
                if any('D' in x for x in
                hydrate_titles) or any('L' in x
                for x in hydrate_titles) or
                any('R' in x for x in
                hydrate_titles) or any('S' in x
                for x in hydrate_titles) or
                any('(+)-' in x for x in
                hydrate_titles) or any('(-)-'
                in x for x in hydrate_titles)
                or any('cis-' in x for x in
                hydrate_titles) or any('trans-'
                in x for x in hydrate_titles)
                or any('E' in x for x in
                hydrate_titles) or any('Z' in x
                for x in hydrate_titles):
                    if all('rac-' not in y for
                    y in waterless_titles) and
                    all('+-' not in y for y in
                    waterless_titles) and
                    all('D' not in y for y in
                    waterless_titles) and
                    all('L' not in y for y in
```

```
                    waterless_titles) and
                    all('R' not in y for y in
                    waterless_titles) and
                    all('S' not in y for y in
                    waterless_titles) and
                    all('(+)-' not in y for y
                    in waterless_titles) and
                    all('(-)-' not in y for y
                    in waterless_titles) and
                    all('cis-' not in y for y
                    in waterless_titles) and
                    all('trans-' not in y for y
                    in waterless_titles) and
                    all('meso-' not in y for y
                    in waterless_titles) and
                    all('E' not in y for y in
                    waterless_titles) and
                    all('Z' not in y for y in
                    waterless_titles):
305                     possibly_the_same_names
                        += 1
306                     RandBlank += 1
307                 else:
308                     distinct_names += 1
309                     writer2.write(entry2[a])
310                     writer3.write(entry3[a])
311         elif any('D' in y for y in
            waterless_titles) or any('L' in
            y for y in waterless_titles) or
            any('R' in y for y in
            waterless_titles) or any('S' in
            y for y in waterless_titles) or
            any('(+)-' in y for y in
            waterless_titles) or any('(-)-'
            in y for y in waterless_titles)
            or any('cis-' in y for y in
            waterless_titles) or
            any('trans-' in y for y in
            waterless_titles) or any('E' in
            y for y in waterless_titles) or
            any('Z' in y for y in
            waterless_titles):
312             if all('rac-' not in x for
                x in hydrate_titles) and
                all('+-' not in x for x in
                hydrate_titles) and all('D'
                not in x for x in
                hydrate_titles) and all('L'
                not in x for x in
                hydrate_titles) and all('R'
                not in x for x in
                hydrate_titles) and all('S'
                not in x for x in
                hydrate_titles) and
                all('(+)-' not in x for x
                in hydrate_titles) and
                all('(-)-' not in x for x
                in hydrate_titles) and
                all('cis-' not in x for x
                in hydrate_titles) and
                all('trans-' not in x for x
                in hydrate_titles) and
                all('meso-' not in x for x
                in hydrate_titles) and
                all('E' not in x for x in
                hydrate_titles) and all('Z'
                not in x for x in
```

```
                                    hydrate_titles):
313                                     possibly_the_same_names
                                        += 1
314                                     RandBlank += 1
315                                 else:
316                                     distinct_names += 1
317                                     writer2.write(entry2[a])
318                                     writer3.write(entry3[a])
319                         #Structures with distinct
                            chirality in their names are
                            found here and are written to
                            output files for matches with
                            distinct chirality
320                         else:
321                             distinct_names += 1
322                             writer2.write(entry2[a])
323                             writer3.write(entry3[a])
324
325                 #This script finds the cases where
                    none of the molecule components
                    have formulas that match the entry
                    formula
326                 #The first part looks at structures
                    where there are no chiral
                    configurations of the molecules
                    either
327                 if possibly_the_same_names != 0 and
                    hydrate_tags == []:
328                     if hash(entry2[a].identifier)
                        in
                        hydrate_tags_empty_no_stereo_hydr
                        ates:
329                         indices = [i for i, x in
                            enumerate(hydrate_tags_empty_
                            no_stereo_hydrates) if x ==
                            hash(entry2[a].identifier)]
330                         if
                            any(hydrate_tags_empty_no_ste
                            reo_waterless_forms[j] ==
                            hash(entry3[a].identifier)
                            for j in indices) == True:
331                             #If the two structures
                                have names where both
                                structures indicate the
                                chirality but are not
                                the same, the
                                structures are written
                                to two output files to
                                be manually checked
332                             if DoubleR != 0:
333
                                    writer8.write(entry2[
                                    a])
334
                                    writer9.write(entry3[
                                    a])
335                             #Otherwise they are
                                written to two output
                                files as matches with
                                the same chirality
336                             else:
337
                                    chiral_writer.write(e
                                    ntry2[a])
338
                                    writer1.write(entry3[
                                    a])
```

```
339                                     else:
340                                         print
                                            entry2[a].identifier
341                                         print
                                            entry3[a].identifier
342                                 #In this case there is only one
                                    structure with chiral
                                    configurations of the atoms
343                                 elif hash(entry2[a].identifier)
                                    in
                                    hydrate_tags_empty_one_stereo_hyd
                                    rates and
                                    hydrate_tags_empty_one_stereo_wat
                                    erless_forms[hydrate_tags_empty_o
                                    ne_stereo_hydrates.index(hash(ent
                                    ry2[a].identifier))] ==
                                    hash(entry3[a].identifier):
344                                     #If the two structures have
                                        names where both structures
                                        indicate the chirality but
                                        are not the same, the
                                        structures are written to
                                        two output files to be
                                        manually checked
345                                     if DoubleR != 0:
346                                         writer8.write(entry2[a])
347                                         writer9.write(entry3[a])
348                                     else:
349                                         #Otherwise they are
                                            written to two output
                                            files as matches with
                                            the same chirality
350
                                            chiral_writer.write(entry
                                            2[a])
351                                         writer1.write(entry3[a])
352                                         #If both names have no
                                            chirality, the
                                            structures are written
                                            to additional output
                                            files for structures
                                            with achiral names and
                                            only one structure with
                                            a chiral configuration
353                                         if DoubleBlank != 0:
354
                                                writer4.write(entry2[
                                                a])
355
                                                writer5.write(entry3[
                                                a])
356                                         #If only one name has
                                            chirality, the
                                            structures are written
                                            to additional output
                                            files for structures
                                            with only one chiral
                                            name structure and only
                                            one structure with a
                                            chiral configuration
357                                         if RandBlank != 0:
358
                                                writer6.write(entry2[
                                                a])
359
                                                writer7.write(entry3[
                                                a])
```

```
                else:
                    print entry2[a].identifier
                    print entry3[a].identifier

            #This is where the non-exception (separate list at top) cases are analyzed
            if possibly_the_same_names != 0 and hydrate_tags != []:
                #For each molecule with a chiral atom, the chiral configuration is recorded
                hydrate_chirality = []
                for c in range(len(hydrate_tags)):
                    for e in range(len(entry2[a].molecule.components[hydrate_tags[c]].atoms)):
                        if entry2[a].molecule.components[hydrate_tags[c]].atoms[e].is_chiral == True:
                            hydrate_chirality.append(entry2[a].molecule.components[hydrate_tags[c]].atoms[e].chirality)
                waterless_chirality = []
                for d in range(len(waterless_tags)):
                    for f in range(len(entry3[a].molecule.components[waterless_tags[d]].atoms)):
                        if entry3[a].molecule.components[waterless_tags[d]].atoms[f].is_chiral == True:
                            waterless_chirality.append(entry3[a].molecule.components[waterless_tags[d]].atoms[f].chirality)

                #If one structure has no chiral configurations and the other structure does the name is used as a reference
                if hydrate_chirality == [] and waterless_chirality != [] or hydrate_chirality != [] and waterless_chirality == []:
                    if DoubleR != 0:
                        writer8.write(entry2[a])
                        writer9.write(entry3[a])
                    else:
                        chiral_writer.write(entry2[a])
                        writer1.write(entry3[a])
                        if DoubleBlank != 0:
                            writer4.write(entry2[
```

```
                        a])
388
                        writer5.write(entry3[
                        a])
389                 if RandBlank != 0:
390
                        writer6.write(entry2[
                        a])
391
                        writer7.write(entry3[
                        a])
392
393             if hydrate_chirality == [] and
                waterless_chirality == []:
394                 if DoubleR != 0:
395                     writer8.write(entry2[a])
396                     writer9.write(entry3[a])
397                 else:
398
                        chiral_writer.write(entry
                        2[a])
399                     writer1.write(entry3[a])
400
401             #If there are chiral
                configurations recorded for
                both structures, they are
                compared against each other
402             #This is done by finding the
                same molecule for each structure
403             #Once that is done, the atoms
                in that molecule are compared
                to find the same atom
404             #The element is first compared,
                then the bonds to that element
                are compared in a branching out
                mannner until only one atom
                remains that matches the target
                atom of the other structure
405             #If these matching atoms have
                the same chirality, the next
                atom in the list is checked
406             if hydrate_chirality != [] and
                waterless_chirality != []:
407                 #same molecules = have the
                    same chiral configuration
                    (R, S, Mixed, or none)
408                 #possibly the same
                    molecules = one has a
                    chiral configuration for at
                    least one atom that the
                    other has no chiral
                    configuration for
409                 #distinct molecules = at
                    least one atom has a
                    different chiral
                    configuration in the
                    molecule (R for one, S for
                    the other)
410                 same_molecules = 0
411                 possibly_the_same_molecules
                    = 0
412                 distinct_molecules = 0
413                 for c in
                    range(len(hydrate_tags)):
414                     for d in
                        range(len(waterless_tags)
                        ):
```

```
if entry2[a].molecule.components[hydrate_tags[c]].formula == entry3[a].molecule.components[waterless_tags[d]].formula:
    ordered_hydrate_chiral_centers = []
    ordered_waterless_chiral_centers = []
    atomic_symbols = []
    for e in range(len(entry2[a].molecule.components[hydrate_tags[c]].atoms)):
        if entry2[a].molecule.components[hydrate_tags[c]].atoms[e].atomic_symbol not in atomic_symbols:
            atomic_symbols.append(entry2[a].molecule.components[hydrate_tags[c]].atoms[e].atomic_symbol)
            same_elements = [x for x in entry2[a].molecule.components[hydrate_tags[c]].atoms if x.atomic_symbol == entry2[a].molecule.components[hydrate_tags[c]].atoms[e].atomic_symbol]
            chiral_centers = []
            for f in range(len(same_elements)):
                if len(same_elements[f].bonds) == 4:
                    string1 = str(same_elements[f].bonds)
                    string1 = string1.replace(str(same_elements[f]), '')
                    if ') )' in string1:
                        string1 = string1.replace(') )', '))')
                    if '  Atom' in string1:
                        string1 = string1.replace('  Atom', ' Atom')
                    string1 = re.sub(r'[0-9]+', '', string1)
                    capitals = [i for i in range(len(string1)) if string1[i].isupper() == True]
                    extra = [j for j in capitals if j-1 in capitals]
                    if extra != []:
                        if len(extra) == 1:
                            string2 = string1[:extra[0]]
                            omit_to = string1[extra[0]:].index(')')
                            string2 = string2 + string1[extra[0]+omit_to:]
                        else:
```

```
441
        for k in range(len(extra)):
442
            if k == 0:
443
                string2 = string1[:extra[k]]
444
                omit_to = string1[extra[k]:].index(')')
445
            elif k == len(extra)-1:
446
                string2 = string2 + string1[extra[k-1]+omit_to:extra[k]]
447
                omit_to = string1[extra[k]:].index(')')
448
                string2 = string2 + string1[extra[k]+omit_to:]
449
            else:
450
                string2 = string2 + string1[extra[k-1]+omit_to:extra[k]]
451
                omit_to = string1[extra[k]:].index(')')
452
    if string2.count('Atom(H)') <= 1 and string2.count('Atom(D)') == 0 or
string2.count('Atom(H)') == 0 and string2.count('Atom(D)') <= 1:
453
        chiral_centers.append(same_elements[f])
454
else:
455
    if string1.count('Atom(H)') <= 1 and string1.count('Atom(D)') == 0 or
string1.count('Atom(H)') == 0 and string1.count('Atom(D)') <= 1:
456
        chiral_centers.append(same_elements[f])
457
ordered_hydrate_chiral_centers = ordered_hydrate_chiral_centers + chiral_centers
458                                                                 start = 0
459
                                                                    waterless_matches
                                                                     = []
460                                                                 while
                                                                    len(ordered_hydra
                                                                    te_chiral_centers
                                                                    ) !=
                                                                    len(ordered_water
                                                                    less_chiral_cente
                                                                    rs):
461                                                                     if
waterless_matches == []:
462
symbol_matches = [x for x in entry3[a].molecule.components[waterless_tags[d]].atoms if
x.atomic_symbol == ordered_hydrate_chiral_centers[start].atomic_symbol]
463                                                                             for t
in range(len(symbol_matches)):
464                                                                                 if
len(symbol_matches[t].bonds) == 4:
465
string3 = str(symbol_matches[t].bonds)
466
string3 = string3.replace(str(symbol_matches[t]), '')
467
if ') )' in string3:
468
    string3 = string3.replace(') )', '))')
469
if '  Atom' in string3:
470
```

```
        string3 = string3.replace('  Atom', ' Atom')
471
    string3 = re.sub(r'[0-9]+', '', string3)
472
    capitals = [i for i in range(len(string3)) if string3[i].isupper() == True]
473
    extra = [j for j in capitals if j-1 in capitals]
474
    if extra != []:
475
        if len(extra) == 1:
476
            string4 = string3[:extra[0]]
477
            omit_to = string3[extra[0]:].index(')')
478
            string4 = string4 + string3[extra[0]+omit_to:]
479
        else:
480
            for k in range(len(extra)):
481
                if k == 0:
482
                    string4 = string3[:extra[k]]
483
                    omit_to = string3[extra[k]:].index(')')
484
                elif k == len(extra)-1:
485
                    string4 = string4 + string3[extra[k-1]+omit_to:extra[k]]
486
                    omit_to = string3[extra[k]:].index(')')
487
                    string4 = string4 + string3[extra[k]+omit_to:]
488
                else:
489
                    string4 = string4 + string3[extra[k-1]+omit_to:extra[k]]
490
                    omit_to = string3[extra[k]:].index(')')
491
        if string4.count('Atom(H)') <= 1 and string4.count('Atom(D)') == 0 or
    string4.count('Atom(H)') == 0 and string4.count('Atom(D)') <= 1:
492
            waterless_matches.append(symbol_matches[t])
493
    else:
494
        if string3.count('Atom(H)') <= 1 and string3.count('Atom(D)') == 0 or
    string3.count('Atom(H)') == 0 and string3.count('Atom(D)') <= 1:
495
            waterless_matches.append(symbol_matches[t])
496
    waterless_bonds = []
497
    updated_waterless_bonds = []
498                                                                    for f
    in range(len(waterless_matches)):
499
    pre_waterless_bonds = []
500                                                                        for
    y in range(len(waterless_matches[f].bonds)):
501
    string2 = str(waterless_matches[f].bonds[y])
502
    string2 = string2.replace(str(waterless_matches[f]), '')
```

```
503
    if ') )' in string2:
504
        string2 = string2.replace(') )', '))')
505
    if '  Atom' in string2:
506
        string2 = string2.replace('  Atom', ' Atom')
507
    string2 = re.sub(r'[0-9]+', '', string2)
508
    pre_waterless_bonds.append(string2)
509
    pre_waterless_bonds = sorted(pre_waterless_bonds)
510
    waterless_bonds.append(pre_waterless_bonds)
511
    updated_waterless_bonds.append(pre_waterless_bonds)
512                                                                    else:
513
    waterless_bonds = list(updated_waterless_bonds)
514                                                                    if
    len(waterless_matches) == 1:
515
    ordered_waterless_chiral_centers.append(waterless_matches[0])
516
    waterless_matches.remove(waterless_matches[0])
517                                                                        start
    += 1
518                                                                    else:
519
    hydrate_spider_atoms = []
520
    waterless_spider_atoms = []
521
    found_hydrate_atoms = []
522
    hydrate_bonds = []
523                                                                        for z
    in range(len(ordered_hydrate_chiral_centers[start].bonds)):
524
    string1 = str(ordered_hydrate_chiral_centers[start].bonds[z])
525
    string1 = string1.replace(str(ordered_hydrate_chiral_centers[start]), '')
526                                                                            if
    ') )' in string1:
527
    string1 = string1.replace(') )', '))')
528                                                                            if
    '  Atom' in string1:
529
    string1 = string1.replace('  Atom', ' Atom')
530
    string1 = re.sub(r'[0-9]+', '', string1)
531
    hydrate_bonds.append(string1)
532
    hydrate_bonds = sorted(hydrate_bonds)
533
    hydrate_neighbours = ordered_hydrate_chiral_centers[start].neighbours
534
    found_hydrate_atoms.append(ordered_hydrate_chiral_centers[start])
535
    numberless_hydrate_neighbours = []
536
    numberless_waterless_neighbours = []
537
```

```
    found_match = 0
538                                                                                 while
    found_match == 0:
539
    bond_matches = []
540
    atom_matches = []
541                                                                                 if
    numberless_hydrate_neighbours == []:
542
    for w in range(len(waterless_bonds)):
543
        if waterless_bonds[w] == hydrate_bonds:
544
            bond_matches.append(waterless_bonds[w])
545
            atom_matches.append(w)
546                                                                                 else:
547
    if hydrate_bonds == [] and all(x == [] for x in waterless_bonds):
548
        bond_matches.append(waterless_bonds[w])
549
        atom_matches.append(w)
550
    else:
551
        for w in range(len(numberless_waterless_neighbours)):
552
            off_the_table_waterless = []
553
            off_the_table_hydrate = []
554
            match = 0
555
            for x in range(len(numberless_waterless_neighbours[w])):
556
                if len(waterless_bonds[w]) == len(hydrate_bonds):
557
                    if waterless_neighbours[w][x] not in off_the_table_waterless:
558
                        go = 0
559
                        for u in range(len(numberless_hydrate_neighbours)):
560
                            if go == 0:
561
                                if hydrate_neighbours[u] not in off_the_table_hydrate:
562
                                    if numberless_waterless_neighbours[w][x] ==
    numberless_hydrate_neighbours[u]:
563
                                        waterless_to_compare = []
564
                                        hydrate_to_compare = []
565
                                        if x == 0:
566
                                            for t in range(waterless_spider_atoms[w][x]):
567

    waterless_to_compare.append(waterless_bonds[w][t])
568
                                        else:
569
                                            for t in range(waterless_spider_atoms[w][x-1],
    waterless_spider_atoms[w][x]):
```

```
570
                waterless_to_compare.append(waterless_bonds[w][t])
571
                                        if u == 0:
572
                                            for s in range(hydrate_spider_atoms[u]):
573
                                                hydrate_to_compare.append(hydrate_bonds[s])
574
                                        else:
575
                                            for s in range(hydrate_spider_atoms[u-1],
                hydrate_spider_atoms[u]):
576
                                                hydrate_to_compare.append(hydrate_bonds[s])
577
                                        if waterless_to_compare == hydrate_to_compare:
578
                off_the_table_waterless.append(waterless_neighbours[w][x])
579
                off_the_table_hydrate.append(hydrate_neighbours[u])
580
                                            match += 1
581
                                            go += 1
582
                    if match == len(numberless_hydrate_neighbours):
583
                        bond_matches.append(waterless_bonds[w])
584
                        atom_matches.append(w)
585
586
            #When the length of bond matches is one that means the target atom only has one match,
            making that matching atom its equivalent in the other structure
587                                                                                 if
            len(bond_matches) == 1:
588
            pin2 = waterless_bonds.index(bond_matches[0])
589
            ordered_waterless_chiral_centers.append(waterless_matches[pin2])
590
            waterless_matches.remove(waterless_matches[pin2])
591
            updated_waterless_bonds.remove(updated_waterless_bonds[pin2])
592
            start += 1
593
            found_match += 1
594                                                                                 else:
595
            if numberless_hydrate_neighbours != []:
596
                hydrate_neighbours = list(new_hydrate_neighbours)
597
                numberless_hydrate_neighbours = []
598
                hydrate_spider_atoms = []
599
            for g in range(len(hydrate_neighbours)):
600
                string1 = str(hydrate_neighbours[g])
601
```

```
        string2 = re.sub(r'[0-9]+', '', string1)
602
        numberless_hydrate_neighbours.append(string2)
603
    new_hydrate_neighbours = []
604
    hydrate_atom_branches = 0
605
    for j in range(len(hydrate_neighbours)):
606
        if hydrate_neighbours[j] not in found_hydrate_atoms:
607
            for k in range(len(hydrate_neighbours[j].neighbours)):
608
                if hydrate_neighbours[j].neighbours[k] not in found_hydrate_atoms:
609
                    new_hydrate_neighbours.append(hydrate_neighbours[j].neighbours[k])
610
                    hydrate_atom_branches += 1
611
        hydrate_spider_atoms.append(hydrate_atom_branches)
612
    hydrate_bonds = []
613
    for l in range(len(hydrate_neighbours)):
614
        pre_hydrate_bonds = []
615
        for z in range(len(hydrate_neighbours[l].bonds)):
616
            if all(str(x) not in str(hydrate_neighbours[l].bonds[z]) for x in
    found_hydrate_atoms):
617
                string1 = str(hydrate_neighbours[l].bonds[z])
618
                string1 = string1.replace(str(hydrate_neighbours[l]), '')
619
                if ') )' in string1:
620
                    string1 = string1.replace(') )', '))')
621
                if '  Atom' in string1:
622
                    string1 = string1.replace('  Atom', ' Atom')
623
                string1 = re.sub(r'[0-9]+', '', string1)
624
                pre_hydrate_bonds.append(string1)
625
        pre_hydrate_bonds = sorted(pre_hydrate_bonds)
626
        hydrate_bonds = hydrate_bonds + pre_hydrate_bonds
627
    if numberless_waterless_neighbours != []:
628
        waterless_neighbours = list(new_waterless_neighbours)
629
        waterless_neighbours_to_remove = []
630
        for y in range(len(waterless_neighbours)):
631
            if y not in atom_matches:
632
                waterless_neighbours_to_remove.append(waterless_neighbours[y])
633
        for z in range(len(waterless_neighbours_to_remove)):
634
```

```
            waterless_neighbours.remove(waterless_neighbours_to_remove[z])
635
        numberless_waterless_neighbours = []
636
        waterless_spider_atoms = []
637
    else:
638
        found_waterless_atoms = []
639
        waterless_neighbours = []
640
        for y in range(len(waterless_matches)):
641
            waterless_neighbours.append(waterless_matches[y].neighbours)
642
            if str(waterless_matches[y]) != 'Atom(H)':
643
                found_waterless_atoms.append([waterless_matches[y]])
644
    for f in range(len(waterless_neighbours)):
645
        pre_numberless_waterless_neighbours = []
646
        for e in range(len(waterless_neighbours[f])):
647
            string1 = str(waterless_neighbours[f][e])
648
            string2 = re.sub(r'[0-9]+', '', string1)
649
            pre_numberless_waterless_neighbours.append(string2)
650
        numberless_waterless_neighbours.append(pre_numberless_waterless_neighbours)
651
    new_waterless_neighbours = []
652
    for m in range(len(waterless_neighbours)):
653
        waterless_atom_branches = 0
654
        pre_new_waterless_neighbours = []
655
        pre_waterless_spider_atoms = []
656
        for n in range(len(waterless_neighbours[m])):
657
            if waterless_neighbours[m][n] not in found_waterless_atoms[m]:
658
                for o in range(len(waterless_neighbours[m][n].neighbours)):
659
                    if waterless_neighbours[m][n].neighbours[o] not in
    found_waterless_atoms[m]:
660

    pre_new_waterless_neighbours.append(waterless_neighbours[m][n].neighbours[o])
661
                        waterless_atom_branches += 1
662
            pre_waterless_spider_atoms.append(waterless_atom_branches)
663
        new_waterless_neighbours.append(pre_new_waterless_neighbours)
664
        waterless_spider_atoms.append(pre_waterless_spider_atoms)
665
    waterless_bonds = []
666
    for p in range(len(waterless_neighbours)):
```

```
667
        pre_waterless_bonds = []
668
        for q in range(len(waterless_neighbours[p])):
669
            extra_pre_waterless_bonds = []
670
            for z in range(len(waterless_neighbours[p][q].bonds)):
671
                if all(str(x) not in str(waterless_neighbours[p][q].bonds[z]) for x in
    found_waterless_atoms[p]):
672
                    string2 = str(waterless_neighbours[p][q].bonds[z])
673
                    string2 = string2.replace(str(waterless_neighbours[p][q]), '')
674
                    if ') )' in string2:
675
                        string2 = string2.replace(') )', '))')
676
                    if '  Atom' in string2:
677
                        string2 = string2.replace('  Atom', ' Atom')
678
                    string2 = re.sub(r'[0-9]+', '', string2)
679
                    extra_pre_waterless_bonds.append(string2)
680
            extra_pre_waterless_bonds = sorted(extra_pre_waterless_bonds)
681
            pre_waterless_bonds = pre_waterless_bonds + extra_pre_waterless_bonds
682
        waterless_bonds.append(pre_waterless_bonds)
683
    for j in range(len(hydrate_neighbours)):
684
        if str(hydrate_neighbours[j]) != 'Atom(H)':
685
            found_hydrate_atoms.append(hydrate_neighbours[j])
686
    for m in range(len(waterless_neighbours)):
687
        for n in range(len(waterless_neighbours[m])):
688
            if str(waterless_neighbours[m][n]) != 'Atom(H)':
689
                found_waterless_atoms[m].append(waterless_neighbours[m][n])
690
691
    if len(waterless_neighbours) == 0:
692
        ordered_waterless_chiral_centers.append(waterless_matches[0])
693
        waterless_matches.remove(waterless_matches[0])
694
        updated_waterless_bonds.remove(updated_waterless_bonds[0])
695
        start += 1
696
        found_match += 1
697
698                                                                    #Once the atoms
                                                                       have been
                                                                       matched
                                                                       (ordered) the
                                                                       chirality of
```

```
                                                                    each one is
                                                                    recorded
699
                                                                    hydrate_chirality
                                                                     = []
700
                                                                    waterless_chirali
                                                                    ty = []
701                                                                 for e in
                                                                    range(len(ordered
                                                                    _hydrate_chiral_c
                                                                    enters)):
702                                                                     if
    ordered_hydrate_chiral_centers[e].is_chiral == True:
703
    hydrate_chirality.append(ordered_hydrate_chiral_centers[e].chirality)
704                                                                     else:
705
    hydrate_chirality.append(' ')
706                                                                 for f in
                                                                    range(len(ordered
                                                                    _waterless_chiral
                                                                    _centers)):
707                                                                     if
    ordered_waterless_chiral_centers[f].is_chiral == True:
708
    waterless_chirality.append(ordered_waterless_chiral_centers[f].chirality)
709                                                                     else:
710
    waterless_chirality.append(' ')
711                                                                 if
                                                                    hydrate_chirality
                                                                     ==
                                                                    waterless_chirali
                                                                    ty:
712
    same_molecules += 1
713                                                                 elif [s for s
                                                                    in
                                                                    hydrate_chirality
                                                                     if s != ' ']
                                                                    == [] or [t for
                                                                    t in
                                                                    waterless_chirali
                                                                    ty if t != ' ']
                                                                    == []:
714
    possibly_the_same_molecules += 1
715                                                                 else:
716
    distinct_molecules += 1
717
718                                                         #No distinct molecules but
                                                            possibly the same molecules
                                                            relies on the name as a
                                                            secondary check as has been
                                                            seen before
719                                                         if distinct_molecules == 0:
720                                                             if
                                                                possibly_the_same_molecul
                                                                es == 0:
721                                                                 if DoubleR != 0:
722
                                                                    writer8.write(ent
                                                                    ry2[a])
723
                                                                    writer9.write(ent
```

```
                                                                    ry3[a])
724                                                             else:
725
                                                                    chiral_writer.wri
                                                                    te(entry2[a])
726
                                                                    writer1.write(ent
                                                                    ry3[a])
727                                                         #No distinct molecules
                                                            and no possibly the
                                                            same molecules also
                                                            relies on the name as a
                                                            secondary check as has
                                                            been seen before
728                                                         else:
729                                                             if DoubleR != 0:
730
                                                                    writer8.write(ent
                                                                    ry2[a])
731
                                                                    writer9.write(ent
                                                                    ry3[a])
732                                                             else:
733
                                                                    chiral_writer.wri
                                                                    te(entry2[a])
734
                                                                    writer1.write(ent
                                                                    ry3[a])
735                                                                 if DoubleBlank
                                                                    != 0:
736
    writer4.write(entry2[a])
737
    writer5.write(entry3[a])
738                                                                 if RandBlank !=
                                                                    0:
739
    writer6.write(entry2[a])
740
    writer7.write(entry3[a])
741
742                                                     else:
743                                                         #Any distinct molecules
                                                            implies that the
                                                            chirality of each
                                                            structure is distinct,
                                                            the structures are
                                                            written to output files
                                                            for matches with
                                                            distinct chirality
744                                                         writer2.write(entry2[a])
745                                                         writer3.write(entry3[a])
746
747                                                 print 'another one bites the dust'
748
749 if __name__ == '__main__':
750     # This runs the script
751     r = Runner()
752     r.run()
753
```

```
#20 Remove duplicates with distinct chirality from list of duplicates

from ccdc import io, search
import argparse
import os
import glob
import re

#This script was used interchangably to update the list of first occurrence and duplicate structures
#Filepath1 and filepath2 are always the list of first occurrence and duplicate structures in its current state
#Filepath3 and filepath4 are added to the list of first occurrence and duplicate structures (this was done after script #14)
#Filepath5 and filepath6 are removed from the list of first occurrence and duplicate structures (this was done after script #18)
#The same script was used for hydrate and waterless forms, substituting for each in the output filenames

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\hydrates_all_first_occurrence.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\hydrates_all_duplicates.txt"

#filepath3 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\hydrates_None_entry_to_keep.txt"
#filepath4 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\hydrates_None_duplicates_to_remove.txt"
#filepath3 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\hydrates_smiles_entry_to_keep.txt"
#filepath4 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\hydrates_smiles_duplicates_to_remove.txt"

#filepath5 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\check_formula_hydrate_entry.txt"
#filepath6 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\check_formula_hydrate_duplicate.txt"
filepath5 = "C:\Users\jenwe\Hydrates Manuscript\Step4 Find Pairs and Duplicates with Distinct Chirality\distinct_chirality_hydrate_entry.txt"
filepath6 = "C:\Users\jenwe\Hydrates Manuscript\Step4 Find Pairs and Duplicates with Distinct Chirality\distinct_chirality_hydrate_duplicate.txt"

#filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_all_first_occurrence.txt"
#filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_all_duplicates.txt"
#filepath3 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_None_entry_to_keep.txt"
#filepath4 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_None_duplicates_to_remove.txt"
#filepath3 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_smiles_entry_to_keep.txt"
#filepath4 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_smiles_duplicates_to_remove.txt"

#filepath5 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\check_formula_waterless_forms_entry.txt"
#filepath6 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\check_formula_waterless_forms_duplicate.txt"
#filepath5 = "C:\Users\jenwe\Hydrates Manuscript\Step4 Find Pairs and Duplicates with Distinct Chirality\distinct_chirality_waterless_forms_entry.txt"
#filepath6 = "C:\Users\jenwe\Hydrates Manuscript\Step4 Find Pairs and Duplicates with Distinct Chirality\distinct_chirality_waterless_forms_duplicate.txt"


class Runner(argparse.ArgumentParser):

```

```
43      def __init__(self):
44          super(self.__class__, self).__init__(description=__doc__)
45          self.add_argument(
46              '-i', '--input', default=filepath1,
47              help='input database filepath1'
48          )
49          self.add_argument(
50              '-o', '--output', default='hydrates_all_first_occurrence.gcd',
51              help='output file [hydrates_all_first_occurrence.gcd]'
52          )
53          self.add_argument(
54              '-m', '--maximum', default=0, type=int,
55              help='Maximum number of structures to find [all]'
56          )
57
58          args = self.parse_args()
59
60          self.args = args
61          self.settings = search.Search.Settings()
62          self.settings.max_hit_structures = self.args.maximum
63
64      def run(self):
65
66          entry_reader1 = io.EntryReader(filepath1, format='identifiers')
67          entry_reader2 = io.EntryReader(filepath2, format='identifiers')
68          #entry_reader3 = io.EntryReader(filepath3, format='identifiers')
69          #entry_reader4 = io.EntryReader(filepath4, format='identifiers')
70          entry_reader5 = io.EntryReader(filepath5, format='identifiers')
71          entry_reader6 = io.EntryReader(filepath6, format='identifiers')
72
73          with io.EntryWriter(self.args.output) as new_lists_writer:
74              with io.EntryWriter("hydrates_all_duplicates.gcd") as writer1:
75
76                  duplicate_identifier_dictionary = {'GADKOL':1, 'MECYEY':2, 'OBASAN':3,
                    'OMOMAE':4, 'OTONUI':5, 'QOJWAO':6, 'TZBFZO':7, 'VOBXOZ':8, 'VUFJAI':9,
                    'VUFJAJ':10, 'WEPQOA':11, 'XADFUD':12, 'YABREY':13, 'YULCOZ':14,
                    'ZOVFEV':15}
77
78                  #Create lists of first occurrences and duplicates that need to be removed
79                  first_occurrence = []
80                  for a in range(len(entry_reader5)):
81                      if entry_reader5[a].identifier in duplicate_identifier_dictionary:
82
                            first_occurrence.append(duplicate_identifier_dictionary.get(entry
                            _reader5[a].identifier))
83                      else:
84                          first_occurrence.append(hash(entry_reader5[a].identifier))
85
86                  duplicate = []
87                  for b in range(len(entry_reader6)):
88                      if entry_reader6[b].identifier in duplicate_identifier_dictionary:
89
                            duplicate.append(duplicate_identifier_dictionary.get(entry_reader
                            6[b].identifier))
90                      else:
91                          duplicate.append(hash(entry_reader6[b].identifier))
92
93                  #Check whether each match in the current list of first occurrences and
                    duplicates is on the list to be removed
94                  for a in range(len(entry_reader1)):
95                      skip = 0
96                      if entry_reader1[a].identifier in duplicate_identifier_dictionary:
97                          query =
                            duplicate_identifier_dictionary.get(entry_reader1[a].identifier)
98                      else:
99                          query = hash(entry_reader1[a].identifier)
100                     if entry_reader2[a].identifier in duplicate_identifier_dictionary:
```

```
                        query2 =
                        duplicate_identifier_dictionary.get(entry_reader2[a].identifier)
                    else:
                        query2 = hash(entry_reader2[a].identifier)
                    #If both structures are in the corresponding lists for removal
                    #Whether or not these two structures appear on the same line
                    dictates if they are removed
                    if query in first_occurrence and query2 in duplicate:
                        indice = [i for i, x in enumerate(first_occurrence) if x ==
                        query and duplicate[i] == query2]
                        if len(indice) == 1:
                            skip += 1

                    #The new output file updates the first occurrence and duplicate
                    list so it does not contain any matches that needed to be removed
                    if skip == 0:
                        new_lists_writer.write(entry_reader1[a])
                        writer1.write(entry_reader2[a])

                #Matches that need to be added are simply appended to the output files
                for the new list of first occurrences and duplicates
                #for c in range(len(entry_reader3)):
                    #new_lists_writer.write(entry_reader3[c])
                    #writer1.write(entry_reader4[c])









if __name__ == '__main__':
    r = Runner()
    r.run()

```

```python
#21 Find hydrates with waterless forms using SMILES string method

from ccdc import io, search
import argparse
import os
import glob
import re

#This script finds hydrate-anhydrate pairs from structures with an entry SMILES string

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\smiles_solvated_hydrates.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\smiles_solvated_waterless_forms.txt"
filepath3 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\smiles_hydrates.txt"
filepath4 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\smiles_waterless_forms.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='potential_hydrates_with_waterless_form.gcd',
            help='output file [potential_hydrates_with_waterless_form.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')
        entry_reader2 = io.EntryReader(filepath2, format='identifiers')
        entry_reader3 = io.EntryReader(filepath3, format='identifiers')
        entry_reader4 = io.EntryReader(filepath4, format='identifiers')

        total = 0
        count = 0

        with io.EntryWriter(self.args.output) as pairing_writer:
            with io.EntryWriter("potential_waterless_forms_with_hydrate.gcd") as writer1:
                with io.EntryWriter("potential_hydrates_without_waterless_form.gcd") as writer2:
                    with io.EntryWriter("potential_waterless_forms_without_hydrate.gcd") as writer3:

                        solvents_dictionary = {hash('but-2-ene'):'CC=CC',
                        hash('1,1,2-dichloroethane'):'ClCC(Cl)Cl',
                        hash('4-aminopyridine'):'Nc1ccncc1',
                        hash('N-methylurea'):'CNC(N)=O',
                        hash('2-methylphenol'):'Cc1ccccc1O',
                        hash('tetraglyme'):'COCCOCCOCCOCCOC',
                        hash('isophorone'):'CC1=CC(=O)CC(C)(C)C1',
```

```
hash('L-carvone'):'CC(=C)C1CC=C(C)C(=O)C1',
hash('2-cyanopyridine'):'N#Cc1ccccn1',
hash('biuret'):'NC(=O)NC(N)=O', hash('styrene'):'C=Cc1ccccc1',
hash('carbondisulfide'):'S=C=S',
hash('tricyanomethanide'):'N#C[C-](C#N)C#N',
hash('methyl-cyclohexane'):'CC1CCCCC1',
hash('pyridine-3-carboxamide'):'NC(=O)c1cccnc1',
hash('3,7-dimethyl-3,7-dihydro-1H-purine-2,6-dione'):'CN1C=NC2=C1
C(=O)NC(=O)N2C', hash('nitroethane'):'CCN(=O)=O',
hash('dimethylaminobenzaldehyde'):'CN(C)c1ccc(C=O)cc1',
hash('3-(1H-pyrrol-1-yl)propanenitrile'):'C1=CN(C=C1)CCC#N',
hash('trifluoromethanol'):'OC(F)(F)F',
hash('methoxyethane'):'CCOC',
hash('methylanisole'):'COc1ccccc1C',
hash('xylene'):'Cc1ccc(C)cc1', hash('petrol'):'CCCCCC',
hash('n-alkane'):'CCCCC', hash('dichloromethanal'):'ClC(Cl)=O',
hash('1-fluoro-4-methylbenzene'):'Cc1ccc(F)cc1',
hash('1,2,3-trimethoxybenzene'):'COc1cccc(OC)c1OC',
hash('1,3,5-trimethoxybenzene'):'COc1cc(OC)cc(OC)c1',
hash('chloroform'):'C1C(Cl)Cl', hash('methanol'):'CO',
hash('hexane'):'CCCCCC', hash('ethanol'):'CCO',
hash('acetonitrile'):'CC#N', hash('toluene'):'Cc1ccccc1',
hash('tetrahydrofuran'):'C1CCOC1', hash('acetone'):'CC(C)=O',
hash('dichloromethane'):'ClCCl',
hash('1,2-dichloroethane'):'ClCCCl',
hash('dimethylsulfoxide'):'CS(C)=O',
hash('1,4-dioxane'):'C1COCCO1',
hash('1,2-dimethoxyethane'):'COCCOC',
hash('o-xylene'):'Cc1ccccc1C', hash('propanol'):'CCO',
hash('dioxane'):'C1COCCO1', hash('n-pentane'):'CCCCC',
hash('4-cyanopyridine'):'N#Cc1ccncc1',
hash("4,4'-bipyridine"):'c1cc(ccn1)c1ccncc1',
hash('isopropanol'):'CC(C)O', hash('1-butanol'):'CCCCO',
hash('nitromethane'):'CN(=O)=O', hash('n-hexane'):'CCCCCC',
hash('dimethylformamide'):'CN(C)C=O',
hash('N,N-dimethylformamide'):'CN(C)C=O',
hash('cyclohexane'):'C1CCCCC1',
hash('hydroquinone'):'Oc1ccc(O)cc1',
hash('benzene'):'c1ccccc1', hash('octane'):'CCCCCCCC',
hash('cyclohexanone'):'O=C1CCCCC1',
hash('isobutanol'):'CC(C)CO', hash('p-xylene'):'Cc1ccc(C)cc1',
hash('pentane'):'CCCCC', hash('pyridine'):'c1ccncc1',
hash('t-butanol'):'CC(C)(C)O', hash('m-xylene'):'Cc1cccc(C)c1',
hash('phenol'):'Oc1ccccc1', hash('o-cresol'):'Cc1ccccc1O',
hash('p-cresol'):'Cc1ccc(O)cc1',
hash('2,4,6-trimethylpyridine'):'Cc1cc(C)nc(C)c1',
hash('1-methylpyrrolidin-2-one'):'CN1CCCC1=O',
hash('1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8-hexadecafluoro-1,8-diiodooc
tane'):'FC(F)(I)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)
I', hash('trifluoroethanol'):'OCC(F)(F)F',
hash('dichloroethane'):'ClCCCl',
hash('1,1,2,2-tetrachloroethane'):'ClC(Cl)C(Cl)Cl',
hash('2-propanol'):'CC(C)O',
hash('chlorobenzene'):'Clc1ccccc1',
hash('naphthalene'):'c1ccc2ccccc2c1',
hash('propan-2-ol'):'CC(C)O',
hash('benzonitrile'):'N#Cc1ccccc1',
hash('nitrobenzene'):'O=N(=O)c1ccccc1',
hash('dodecane'):'CCCCCCCCCCCC',
hash('1,2-dichlorobenzene'):'Clc1ccccc1Cl',
hash('n-heptane'):'CCCCCCC',
hash('methanesulfonate'):'CS(=O)(=O)[O-]',
hash('3-methylbutan-1-ol'):'CC(C)CCO',
hash('1,1,2-trichloroethane'):'ClCC(Cl)Cl',
hash('n-butane'):'CCCC', hash('dichlorine'):'ClCl',
hash('o-dichlorobenzene'):'Clc1ccccc1Cl',
hash('N,N-dimethylacetamide'):'CN(C)C(C)=O',
```

```
hash('diethylether'):'CCOCC', hash('hydrazine'):'NN',
hash('dimethylamine'):'CNC',
hash('1,2-difluorobenzene'):'Fc1ccccc1F',
hash('cycloheptane'):'C1CCCCCC1', hash('formamide'):'NC=O',
hash('2-butanol'):'CCC(C)O', hash('ethylacetate'):'CCOC(C)=O',
hash('butan-1-ol'):'CCCCO', hash('bromomethane'):'CBr',
hash('piperazine'):'C1CNCCN1', hash('n-nonane'):'CCCCCCCCC',
hash('1,2,4-trimethoxybenzene'):'COc1ccc(OC)c(OC)c1',
hash('1,2-diaminoethane'):'NCCN', hash('n-butanol'):'CCCCO',
hash('N-methylformamide'):'CNC=O',
hash('pyrrolidine'):'C1CCNC1', hash('actone'):'CC(C)=O',
hash('n-octane'):'CCCCCCCC',
hash('(S)-2-methyl-1-butanol'):'CCC(C)CO',
hash('N,N-diethylformamide'):'CCN(CC)C=O',
hash('tetracyanoethylene'):'N#CC(C#N)=C(C#N)C#N',
hash('fluorobenzene'):'Fc1ccccc1', hash('heptane'):'CCCCCCC',
hash('tetrachloroethane'):'ClC(Cl)C(Cl)Cl',
hash('methylcyclohexane'):'CC1CCCCC1',
hash('tetrachloromethane'):'ClC(Cl)(Cl)Cl',
hash('1-propanol'):'CCCO', hash('acetamide'):'CC(N)=O',
hash('propane'):'CCC', hash('diethylamine'):'CCNCC',
hash('hexafluorophosphate'):'F[P-](F)(F)(F)(F)F',
hash('hexafluoropropan-2-ol'):'OC(C(F)(F)F)C(F)(F)F',
hash('bis(hexafluorophosphate)'):'F[P-](F)(F)(F)(F)F',
hash('2-methyl-1,3,5-trinitrobenzene'):'Cc1c(cc(cc1N(=O)=O)N(=O)=
O)N(=O)=O', hash('furan'):'O1C=CC=C1',
hash('2-methylpentane'):'CCCC(C)C',
hash('isohexane'):'CCCC(C)C', hash('anisole'):'COc1ccccc1',
hash('butanol'):'CCCCO', hash('propan-1-ol'):'CCCO',
hash('dimethoxyethane'):'COCCOC',
hash('cyclopentane'):'C1CCCC1',
hash('methoxybenzene'):'COc1ccccc1',
hash('toulene'):'Cc1ccccc1',
hash('pyrene'):'c1cc2ccc3cccc4ccc(c1)c2c34',
hash('butan-2-ol'):'CCC(C)O',
hash('3-methylpyridine'):'Cc1cccnc1',
hash('triethylamine'):'CCN(CC)CC', hash('dioxan'):'C1COCCO1',
hash('2-methoxyethanol'):'COCCO',
hash("4,4'-bipyridyl"):'c1cc(ccn1)c1ccncc1',
hash('dimethylacetamide'):'CN(C)C(C)=O',
hash('thiophene'):'S1C=CC=C1', hash('1,3-dioxolane'):'C1COCO1',
hash('1-chloronaphthalene'):'Clc1cccc2ccccc12',
hash('pentan-1-ol'):'CCCCCO', hash('hexan-1-ol'):'CCCCCCO',
hash('2-phenylethanol'):'OCCc1ccccc1',
hash('1-chloro-4-methylbenzene'):'Cc1ccc(Cl)cc1',
hash('acetaldehyde'):'CC=O', hash('propiononitrile'):'CCC#N',
hash('bromobenzene'):'Brc1ccccc1'}
deuterated_dictionary =
{hash('dideutero-dichloromethane'):'ClCCl',
hash('perdeutero-toluene'):'Cc1ccccc1',
hash('octadeuterofuran'):'C1CCOC1',
hash('deutero-ethanol'):'CCO',
hash('deuterochloroform'):'C1C(Cl)Cl',
hash('deutero-chloroform'):'C1C(Cl)Cl',
hash('deuterobenzene'):'c1ccccc1',
hash('deuteromethanol'):'CO',
hash('hexadeutero-benzene'):'c1ccccc1',
hash('perdeuteroacetone'):'CC(C)=O',
hash('dideuterodichloromethane'):'ClCCl',
hash('perdeuteromethanol'):'CO',
hash('hexadeuterobenzene'):'c1ccccc1'}

acids_dictionary = {hash('trifluoroacetic'):'OC(=O)C(F)(F)F',
hash('acetic'):'CC(O)=O', hash('formic'):'OC=O',
hash('sulfuric'):'OS(O)(=O)=O',
hash('maleic'):'OC(=O)C=CC(O)=O',
hash('benzoic'):'OC(=O)c1ccccc1',
```

```
                    hash('succinic'):'OC(=O)CCC(O)=O'}
57                  ethers_dictionary = {'t-butyl methyl ether':'COC(C)(C)C',
                    hash('di-isopropyl'):'CC(C)OC(C)C', hash('dipropyl'):'CCCOCCC',
                    hash('diethyl'):'CCOCC', hash('diisopropyl'):'CC(C)OC(C)C',
                    hash('isopropyl'):'CC(C)OC(C)C', hash('t-butyl'):'COC(C)(C)C',
                    hash('acetic'):'CCOC(C)=O'}
58                  acetates_dictionary = {hash('ethyl'):'CCOC(C)=O',
                    hash('amyl'):'CCCCCOC(C)=O', hash('methyl'):'COC(C)=O',
                    hash('butyl'):'CCCCOC(C)=O', hash('isobutyl'):'CC(C)COC(C)=O'}
59                  glycol_dictionary = {hash('ethylene'):'OCCO'}
60                  oxalates_dictionary = {hash('dipropyl'):'CCCOC(=O)C(=O)OCCC'}
61                  acid_neutral_dictionary = {hash('oxalic'):'OC(=O)C(O)=O',
                    hash('nitric'):'ON(=O)=O',
                    hash('trans-but-2-enedioic'):'OC(=O)C=CC(O)=O',
                    hash('but-2-enedioic'):'OC(=O)C=CC(O)=O',
                    hash('fumaric'):'OC(=O)C=CC(O)=O',
                    hash('phosphonic'):'OP(O)=O',
                    hash('perchloric'):'O[Cl](=O)(=O)=O',
                    hash('phosphoric'):'OP(O)(O)=O',
                    hash('pyridine-2,6-dicarboxylic'):'OC(=O)c1cccc(n1)C(O)=O',
                    hash('benzene-1,2,4,5-tetracarboxylic'):'OC(=O)c1cc(C(O)=O)c(cc1C
                    (O)=O)C(O)=O', hash('trifluoroacetatic'):'OC(=O)C(F)(F)F'}
62                  acid_charged_dictionary = {hash('oxalic'):'OC(=O)C(=O)[O-]',
                    hash('nitric'):'O=N(=O)[O-]',
                    hash('trans-but-2-enedioic'):'OC(=O)C=CC(=O)[O-]',
                    hash('but-2-enedioic'):'OC(=O)C=CC(=O)[O-]',
                    hash('fumaric'):'OC(=O)C=CC(=O)[O-]',
                    hash('phosphonic'):'OP(=O)[O-]',
                    hash('perchloric'):'O=[Cl](=O)(=O)[O-]',
                    hash('phosphoric'):'OP(O)(=O)[O-]',
                    hash('pyridine-2,6-dicarboxylic'):'OC(=O)c1cccc(n1)C(=O)[O-]',
                    hash('benzene-1,2,4,5-tetracarboxylic'):'OC(=O)c1cc(C(=O)[O-])c(c
                    c1C(=O)[O-])C(O)=O',
                    hash('trifluoroacetatic'):'FC(F)(F)C(=O)[O-]'}
63
64                  partial_name = [hash('ketone'), hash('propanoate'),
                    hash('carbonate'), hash('formate'), hash('iodine'),
                    hash('tetrachloride'), hash('sulfoxide'), hash('dichloride'),
                    hash('chloride'), hash('benzoate'), hash('disulfide'),
                    hash('oxide')]
65                  full_solvate_name_dictionary = {'methyl isobutyl
                    ketone':'CC(C)CC(C)=O', 'deuterium oxide':'O', 'ethyl
                    propanoate':'CCOC(=O)CC', 'dimethyl carbonate':'COC(=O)OC',
                    'ethyl formate':'CCOC=O', '12]cycloparaphenylene iodine':'II',
                    'carbon tetrachloride':'ClC(Cl)(Cl)Cl', 'dimethyl
                    sulfoxide':'CS(C)=O', 'methylene dichloride':'ClCCl',
                    'methylene chloride':'ClCCl', 'ethyl
                    benzoate':'CCOC(=O)c1ccccc1', 'carbon disulfide':'S=C=S'}
66
67                  hydrates_SMILES = []
68                  waterless_forms_SMILES = []
69                  hydrates_formulas = []
70                  waterless_forms_formulas = []
71
72                  duplicate_identifier_dictionary = {'GADKOL':1, 'MECYEY':2,
                    'OBASAN':3, 'OMOMAE':4, 'OTONUI':5, 'QOJWAO':6, 'TZBFZO':7,
                    'VOBXOZ':8, 'VUFJAI':9, 'VUFJAJ':10, 'WEPQOA':11, 'XADFUD':12,
                    'YABREY':13, 'YULCOZ':14, 'ZOVFEV':15}
73                  duplicate_name_dictionary =
                    {"N,N'-bis((Diethoxyphosphinoyl)phenylmethyl)-1,4-diaminobenzene"
                    :1, 'bis(n-Hexyl)':2, 'Benzoylmethyl':3,
                    '(((4-methylphenyl)sulfonyl)amino)(phenyl)acetic':4,
                    '1,1-diphenylethyl':5, 'hemikis(1-naphthylamine)':6,
                    '(E)-1-(2-Nitrobenzylidene)-4-phenylthiosemicarbazide':7}
74                  duplicate_smiles_dictionary =
                    {'CC(C)(C)c1c(O)c(O)c(Cl)c2[NH+]3CCN(CC3)c12':1,
                    'CC1=C(C(CC2=NS(=O)(=O)c3ccccc23)c2ccccc2C)C(=O)N(N1)c1ccccc1':2,
```

```
                        'OC(=O)COc1cccc2ccccc12':3, 'Cc1c[nH+]cc(C)c1':4,
                        'Brc1ccc(NC(=O)c2cccnc2)cc1':5,
                        'CSC(N)=NNC(C)=CC(=O)c1ccccc1':6, 'CC1(CO1)C(O)(CBr)CBr':7,
                        'OCC12CCCC(C1)(C(O)=O)C1(CCSCC1)O2':8,
                        'COc1ccc2CC3[NH2+]CCC3(C)c2c1':9,
                        'CC(=O)C(P(c1ccccc1)c1ccccc1)=C(C)Nc1c(F)cccc1F':10}

loop = 0
entry1 = entry_reader1
tag = 12
#The solvent dictionaries are used to correct incomplete SMILES
strings for solvent molecules that are missing
while loop < 4:
    for a in range(len(entry1)):
        component_SMILES = []
        entry_SMILES = entry1[a].molecule.smiles.split('.')
        for b in range(len(entry_SMILES)):
            if entry_SMILES[b] not in
            duplicate_smiles_dictionary:
                component_SMILES.append(hash(entry_SMILES[b]))
            else:

                component_SMILES.append(duplicate_smiles_dictiona
                ry.get(entry_SMILES[b]))
        component_SMILES =
        list(set(component_SMILES))
        if entry1[a].identifier == 'BULVEL':
            if hash('[O-][n+]1ccccc1') not in component_SMILES:
                component_SMILES.append(hash('[O-][n+]1ccccc1'))
            if hash('[O]n1ccccc1') in component_SMILES:
                component_SMILES.remove(hash('[O]n1ccccc1'))
        elif entry1[a].identifier == 'CEHKOS':
            component_SMILES.append(hash('II'))
        elif entry1[a].identifier == 'GOTJEE' or
        entry1[a].identifier == 'HASVEF' or
        entry1[a].identifier == 'MTFBTZ10':
            component_SMILES.append(tag)
            tag += 1
        elif entry1[a].identifier == 'JUHJIF':
            if hash('BrBr') not in component_SMILES:
                component_SMILES.append(hash('BrBr'))
            if hash('[Br]') in component_SMILES:
                component_SMILES.remove(hash('[Br]'))
        elif entry1[a].identifier == 'LOKXUF':
            if hash('OC(=O)C(F)(F)F') not in component_SMILES:
                component_SMILES.append(hash('OC(=O)C(F)(F)F'))
            if hash('[O]C(=O)C(F)(F)F') in component_SMILES:
                component_SMILES.remove(hash('[O]C(=O)C(F)(F)F'))
        elif entry1[a].identifier == 'OJUNEO':
            if hash('ClCl') not in component_SMILES:
                component_SMILES.append(hash('ClCl'))
            if hash('[Cl]') in component_SMILES:
                component_SMILES.remove(hash('[Cl]'))
        elif entry1[a].identifier == 'ZILFILM':
            component_SMILES.append(hash('CCO'))
        else:
            if entry1[a].chemical_name != None:
                pieces = entry1[a].chemical_name.split(' ')
                formula_pieces = entry1[a].formula.split(',')
                for c in range(len(pieces)):
                    if hash(pieces[c]) == hash('xylene'):
                        if entry1[a].identifier == 'QOWNEV':
                            if hash('Cc1ccccc1C') not in
                            component_SMILES:

                                component_SMILES.append(hash('Cc1
                                ccccc1C'))
```

```
126                     elif entry1[a].identifier == 'TEYCEF'
                        or entry1[a].identifier ==
                        'TEYCEF01':
127                         if hash('Cc1cccc(C)c1') not in
                            component_SMILES:
128
                                component_SMILES.append(hash('Cc1
                                cccc(C)c1'))
129                     elif entry1[a].identifier == 'MAMNAR':
130                         if hash('Cc1ccc(C)cc1') not in
                            component_SMILES:
131
                                component_SMILES.append(hash('Cc1
                                ccc(C)cc1'))
132                 elif hash(pieces[c]) ==
                    hash('hydrochloride') or hash(pieces[c]) ==
                    hash('bis(hydrochloride)'):
133                     for z in range(len(formula_pieces)):
134                         if formula_pieces[z] == 'Cl1 1-' or
                            '(Cl1 1-)' in formula_pieces[z]:
135                             if hash('[Cl-]') not in
                                component_SMILES:
136
                                    component_SMILES.append(hash(
                                    '[Cl-]'))
137                         if formula_pieces[z] == 'H1 Cl1' or
                            '(H1 Cl1)' in formula_pieces[z]:
138                             if hash('Cl') not in
                                component_SMILES:
139
                                    component_SMILES.append(hash(
                                    'Cl'))
140                 elif hash(pieces[c]) ==
                    hash('unidentified') or hash(pieces[c]) ==
                    hash('unknown'):
141                     component_SMILES.append(tag)
142                     tag += 1
143                 elif hash(pieces[c]) == hash('glycol'):
144                     if hash(pieces[c-1]) in
                        glycol_dictionary:
145                         if
                            hash(glycol_dictionary.get(hash(piece
                            s[c-1]))) not in component_SMILES:
146
                                component_SMILES.append(hash(glyc
                                ol_dictionary.get(hash(pieces[c-1
                                ]))))
147                 elif hash(pieces[c]) == hash('oxalate'):
148                     if hash(pieces[c-1]) in
                        oxalates_dictionary:
149                         if
                            hash(oxalates_dictionary.get(hash(pie
                            ces[c-1]))) not in component_SMILES:
150
                                component_SMILES.append(hash(oxal
                                ates_dictionary.get(hash(pieces[c
                                -1]))))
151                 elif hash(pieces[c]) == hash('acetate'):
152                     if hash(pieces[c-1]) in
                        acetates_dictionary:
153                         if
                            hash(acetates_dictionary.get(hash(pie
                            ces[c-1]))) not in component_SMILES:
154
                                component_SMILES.append(hash(acet
                                ates_dictionary.get(hash(pieces[c
                                -1]))))
```

```
155                                 elif hash(pieces[c]) == hash('ether'):
156                                     if hash(pieces[c-1]) ==
                                        hash('petroleum'):
157                                         component_SMILES.append(11)
158                                     elif pieces[c-2] + ' ' + pieces[c-1] +
                                        ' ' + pieces[c] in ethers_dictionary:
159                                         if
                                            hash(ethers_dictionary.get(pieces[c-2
                                            ] + ' ' + pieces[c-1] + ' ' +
                                            pieces[c])) not in component_SMILES:
160
                                                component_SMILES.append(hash(ethe
                                                rs_dictionary.get(pieces[c-2] +
                                                ' ' + pieces[c-1] + ' ' +
                                                pieces[c])))
161                                     elif hash(pieces[c-1]) in
                                        ethers_dictionary:
162                                         if
                                            hash(ethers_dictionary.get(hash(piece
                                            s[c-1]))) not in component_SMILES:
163
                                                component_SMILES.append(hash(ethe
                                                rs_dictionary.get(hash(pieces[c-1
                                                ])))))
164                                 elif hash(pieces[c]) in partial_name:
165                                     if pieces[c] == 'ketone':
166                                         if pieces[c-2] + ' ' + pieces[c-1]
                                            + ' ' + pieces[c] in
                                            full_solvate_name_dictionary:
167                                             if
                                                hash(full_solvate_name_dictionary
                                                .get(pieces[c-2] + ' ' +
                                                pieces[c-1] + ' ' + pieces[c]))
                                                not in component_SMILES:
168
                                                    component_SMILES.append(hash(
                                                    full_solvate_name_dictionary.
                                                    get(pieces[c-2] + ' ' +
                                                    pieces[c-1] + ' ' +
                                                    pieces[c])))
169                                     else:
170                                         if pieces[c-1] + ' ' + pieces[c] in
                                            full_solvate_name_dictionary:
171                                             if
                                                hash(full_solvate_name_dictionary
                                                .get(pieces[c-1] + ' ' +
                                                pieces[c])) not in
                                                component_SMILES:
172
                                                    component_SMILES.append(hash(
                                                    full_solvate_name_dictionary.
                                                    get(pieces[c-1] + ' ' +
                                                    pieces[c])))
173                                 elif hash(pieces[c]) == hash('acid'):
174                                     if hash(pieces[c-1]) in acids_dictionary:
175                                         if
                                            hash(acids_dictionary.get(hash(pieces
                                            [c-1]))) not in component_SMILES:
176
                                                component_SMILES.append(hash(acid
                                                s_dictionary.get(hash(pieces[c-1]
                                                ))))
177                                     elif hash(pieces[c-1]) in
                                        acid_neutral_dictionary:
178                                         if hash(pieces[c-1]) ==
                                            hash('oxalic'):
179                                             for z in
```

```
                range(len(formula_pieces)):
180                 if formula_pieces[z] == 'C2
                    H2 O4' or '(C2 H2 O4)' in
                    formula_pieces[z]:
181                     if
                        hash(acid_neutral_diction
                        ary.get(hash(pieces[c-1])
                        )) not in
                        component_SMILES:
182
                            component_SMILES.appe
                            nd(hash(acid_neutral_
                            dictionary.get(hash(p
                            ieces[c-1])))))
183                 if formula_pieces[z] == 'C2
                    H1 O4 1-' or '(C2 H1 O4
                    1-)' in formula_pieces[z]:
184                     if
                        hash(acid_charged_diction
                        ary.get(hash(pieces[c-1])
                        )) not in
                        component_SMILES:
185
                            component_SMILES.appe
                            nd(hash(acid_charged_
                            dictionary.get(hash(p
                            ieces[c-1])))))
186             elif hash(pieces[c-1]) ==
                hash('nitric'):
187                 for z in
                    range(len(formula_pieces)):
188                     if formula_pieces[z] == 'H1
                        N1 O3' or '(H1 N1 O3)' in
                        formula_pieces[z]:
189                         if
                            hash(acid_neutral_diction
                            ary.get(hash(pieces[c-1])
                            )) not in
                            component_SMILES:
190
                                component_SMILES.appe
                                nd(hash(acid_neutral_
                                dictionary.get(hash(p
                                ieces[c-1])))))
191                     if formula_pieces[z] == 'N1
                        O3 1-' or '(N1 O3 1-)' in
                        formula_pieces[z]:
192                         if
                            hash(acid_charged_diction
                            ary.get(hash(pieces[c-1])
                            )) not in
                            component_SMILES:
193
                                component_SMILES.appe
                                nd(hash(acid_charged_
                                dictionary.get(hash(p
                                ieces[c-1])))))
194             elif hash(pieces[c-1]) ==
                hash('trans-but-2-enedioic') or
                hash(pieces[c-1]) ==
                hash('but-2-enedioic') or
                hash(pieces[c-1]) == hash('fumaric'):
195                 for z in
                    range(len(formula_pieces)):
196                     if formula_pieces[z] == 'C4
                        H4 O4' or '(C4 H4 O4)' in
                        formula_pieces[z]:
```

```
                                    if
                                    hash(acid_neutral_diction
                                    ary.get(hash(pieces[c-1])
                                    )) not in
                                    component_SMILES:
                                        component_SMILES.appe
                                        nd(hash(acid_neutral_
                                        dictionary.get(hash(p
                                        ieces[c-1])))))
                                if formula_pieces[z] == 'C4
                                H3 O4 1-' or '(C4 H3 O4
                                1-)' in formula_pieces[z]:
                                    if
                                    hash(acid_charged_diction
                                    ary.get(hash(pieces[c-1])
                                    )) not in
                                    component_SMILES:
                                        component_SMILES.appe
                                        nd(hash(acid_charged_
                                        dictionary.get(hash(p
                                        ieces[c-1])))))
                            elif hash(pieces[c-1]) ==
                            hash('phosphonic'):
                                for z in
                                range(len(formula_pieces)):
                                    if formula_pieces[z] == 'H3
                                    O3 P1' or '(H3 O3 P1)' in
                                    formula_pieces[z]:
                                        if
                                        hash(acid_neutral_diction
                                        ary.get(hash(pieces[c-1])
                                        )) not in
                                        component_SMILES:
                                            component_SMILES.appe
                                            nd(hash(acid_neutral_
                                            dictionary.get(hash(p
                                            ieces[c-1])))))
                                    if formula_pieces[z] == 'H2
                                    O3 P1 1-' or '(H2 O3 P1
                                    1-)' in formula_pieces[z]:
                                        if
                                        hash(acid_charged_diction
                                        ary.get(hash(pieces[c-1])
                                        )) not in
                                        component_SMILES:
                                            component_SMILES.appe
                                            nd(hash(acid_charged_
                                            dictionary.get(hash(p
                                            ieces[c-1])))))
                            elif hash(pieces[c-1]) ==
                            hash('perchloric'):
                                for z in
                                range(len(formula_pieces)):
                                    if formula_pieces[z] == 'H1
                                    Cl1 O4' or '(H1 Cl1 O4)' in
                                    formula_pieces[z]:
                                        if
                                        hash(acid_neutral_diction
                                        ary.get(hash(pieces[c-1])
                                        )) not in
                                        component_SMILES:
                                            component_SMILES.appe
```

```
                                    nd(hash(acid_neutral_
                                    dictionary.get(hash(p
                                    ieces[c-1])))))
215                             if formula_pieces[z] ==
                                'Cl1 O4 1-' or '(Cl1 O4
                                1-)' in formula_pieces[z]:
216                                 if
                                    hash(acid_charged_diction
                                    ary.get(hash(pieces[c-1])
                                    )) not in
                                    component_SMILES:
217
                                        component_SMILES.appe
                                        nd(hash(acid_charged_
                                        dictionary.get(hash(p
                                        ieces[c-1])))))
218                         elif hash(pieces[c-1]) ==
                            hash('phosphoric'):
219                             for z in
                                range(len(formula_pieces)):
220                                 if formula_pieces[z] == 'H3
                                    O4 P1' or '(H3 O4 P1)' in
                                    formula_pieces[z]:
221                                     if
                                        hash(acid_neutral_diction
                                        ary.get(hash(pieces[c-1])
                                        )) not in
                                        component_SMILES:
222
                                            component_SMILES.appe
                                            nd(hash(acid_neutral_
                                            dictionary.get(hash(p
                                            ieces[c-1])))))
223                                 if formula_pieces[z] == 'H2
                                    O4 P1 1-' or '(H2 O4 P1
                                    1-)' in formula_pieces[z]:
224                                     if
                                        hash(acid_charged_diction
                                        ary.get(hash(pieces[c-1])
                                        )) not in
                                        component_SMILES:
225
                                            component_SMILES.appe
                                            nd(hash(acid_charged_
                                            dictionary.get(hash(p
                                            ieces[c-1])))))
226                         elif hash(pieces[c-1]) ==
                            hash('pyridine-2,6-dicarboxylic'):
227                             for z in
                                range(len(formula_pieces)):
228                                 if formula_pieces[z] == 'C7
                                    H5 N1 O4' or '(C7 H5 N1
                                    O4)' in formula_pieces[z]:
229                                     if
                                        hash(acid_neutral_diction
                                        ary.get(hash(pieces[c-1])
                                        )) not in
                                        component_SMILES:
230
                                            component_SMILES.appe
                                            nd(hash(acid_neutral_
                                            dictionary.get(hash(p
                                            ieces[c-1])))))
231                                 if formula_pieces[z] == 'C7
                                    H4 N1 O4 1-' or '(C7 H4 N1
                                    O4 1-)' in formula_pieces[z]:
232                                     if
```

```
                                hash(acid_charged_diction
                                ary.get(hash(pieces[c-1])
                                )) not in
                                component_SMILES:
233
                                    component_SMILES.appe
                                    nd(hash(acid_charged_
                                    dictionary.get(hash(p
                                    ieces[c-1])))))
234                         elif hash(pieces[c-1]) ==
                            hash('benzene-1,2,4,5-tetracarboxylic
                            '):
235                             for z in
                                range(len(formula_pieces)):
236                                 if formula_pieces[z] ==
                                    'C10 H6 O8' or '(C10 H6
                                    O8)' in formula_pieces[z]:
237                                     if
                                        hash(acid_neutral_diction
                                        ary.get(hash(pieces[c-1])
                                        )) not in
                                        component_SMILES:
238
                                            component_SMILES.appe
                                            nd(hash(acid_neutral_
                                            dictionary.get(hash(p
                                            ieces[c-1])))))
239                                 if formula_pieces[z] ==
                                    'C10 H4 O8 2-' or '(C10 H4
                                    O8 2-)' in formula_pieces[z]:
240                                     if
                                        hash(acid_charged_diction
                                        ary.get(hash(pieces[c-1])
                                        )) not in
                                        component_SMILES:
241
                                            component_SMILES.appe
                                            nd(hash(acid_charged_
                                            dictionary.get(hash(p
                                            ieces[c-1])))))
242                         elif hash(pieces[c-1]) ==
                            hash('trifluoroacetatic'):
243                             for z in
                                range(len(formula_pieces)):
244                                 if formula_pieces[z] == 'C2
                                    H1 F3 O2' or '(C2 H1 F3
                                    O2)' in formula_pieces[z]:
245                                     if
                                        hash(acid_neutral_diction
                                        ary.get(hash(pieces[c-1])
                                        )) not in
                                        component_SMILES:
246
                                            component_SMILES.appe
                                            nd(hash(acid_neutral_
                                            dictionary.get(hash(p
                                            ieces[c-1])))))
247                                 if formula_pieces[z] == 'C2
                                    F3 O2 1-' or '(C2 F3 O2
                                    1-)' in formula_pieces[z]:
248                                     if
                                        hash(acid_charged_diction
                                        ary.get(hash(pieces[c-1])
                                        )) not in
                                        component_SMILES:
249
                                            component_SMILES.appe
```

```
                                                    nd(hash(acid_charged_
                                                    dictionary.get(hash(p
                                                    ieces[c-1])))):
250                                         elif hash(pieces[c]) in
                                            deuterated_dictionary:
251                                             if
                                                hash(deuterated_dictionary.get(hash(piece
                                                s[c]))) not in component_SMILES:
252
                                                    component_SMILES.append(hash(deuterat
                                                    ed_dictionary.get(hash(pieces[c]))))
253                                             if hash(pieces[c]) ==
                                                hash('dideutero-dichloromethane') or
                                                hash(pieces[c]) ==
                                                hash('dideuterodichloromethane'):
254                                                 if hash('Cl[C]Cl') in
                                                    component_SMILES:
255
                                                        component_SMILES.remove(hash('Cl[
                                                        C]Cl'))
256                                             elif hash(pieces[c]) ==
                                                hash('perdeutero-toluene'):
257                                                 if hash('[C]c1[c][c][c][c][c]1') in
                                                    component_SMILES:
258
                                                        component_SMILES.remove(hash('[C]
                                                        c1[c][c][c][c][c]1'))
259                                             elif hash(pieces[c]) ==
                                                hash('deutero-ethanol'):
260                                                 if hash('[C][C][O]') in
                                                    component_SMILES:
261
                                                        component_SMILES.remove(hash('[C]
                                                        [C][O]'))
262                                             elif hash(pieces[c]) ==
                                                hash('deuterochloroform') or
                                                hash(pieces[c]) ==
                                                hash('deutero-chloroform'):
263                                                 if hash('Cl[C](Cl)Cl') in
                                                    component_SMILES:
264
                                                        component_SMILES.remove(hash('Cl[
                                                        C](Cl)Cl'))
265                                             elif hash(pieces[c]) ==
                                                hash('hexadeutero-benzene') or
                                                hash(pieces[c]) ==
                                                hash('deuterobenzene') or
                                                hash(pieces[c]) ==
                                                hash('hexadeuterobenzene'):
266                                                 if hash('[c]1[c][c][c][c][c]1') in
                                                    component_SMILES:
267
                                                        component_SMILES.remove(hash('[c]
                                                        1[c][c][c][c][c]1'))
268                                             elif hash(pieces[c]) ==
                                                hash('deuteromethanol') or
                                                hash(pieces[c]) ==
                                                hash('perdeuteromethanol'):
269                                                 if hash('[C][O]') in
                                                    component_SMILES:
270
                                                        component_SMILES.remove(hash('[C]
                                                        [O]'))
271                                         elif hash(pieces[c]) in solvents_dictionary:
272                                             if
                                                hash(solvents_dictionary.get(hash(pieces[
                                                c]))) not in component_SMILES:
```

```
                                        component_SMILES.append(hash(solvents
                                        _dictionary.get(hash(pieces[c]))))

                    #The formula of each structure is also recorded
                    formulas = entry1[a].formula.split(',')
                    letter_formulas = []
                    for b in range(len(formulas)):
                        if 'D' in formulas[b] and 'H' in formulas[b]:
                            pieces = formulas[b].split(' ')
                            for c in range(len(pieces)):
                                if 'H' in pieces[c]:
                                    pin1 = pieces[c].index('H')
                                    piece1 = pieces[c]
                                    stoich1 = int(pieces[c][pin1+1:])
                                elif 'D' in pieces[c]:
                                    pin2 = pieces[c].index('D')
                                    piece2 = pieces[c]
                                    stoich2 = int(pieces[c][pin2+1:])
                            stoich = stoich1 + stoich2
                            pieces.remove(piece1)
                            pieces.remove(piece2)
                            pieces.append('H' + str(stoich))
                            pieces = sorted(pieces)
                            formulas[b] = ' '.join(pieces)
                        elif 'D' in formulas[b]:
                            formulas[b] = str(formulas[b]).replace('D', 'H')
                        if loop == 0 or loop == 2:
                            if '(' in formulas[b] and ')' in formulas[b]:
                                if '(H2 O1)' not in formulas[b]:
                                    pin = formulas[b].index('(')

                                    letter_formulas.append(formulas[b][pin+1:
                                    -1])
                                else:
                                    if formulas[b] != 'H2 O1':
                                        letter_formulas.append(formulas[b])
                            else:
                                if '(' in formulas[b] and ')' in formulas[b]:
                                    pin = formulas[b].index('(')
                                    letter_formulas.append(formulas[b][pin+1:-1])
                                else:
                                    letter_formulas.append(formulas[b])

                    if loop == 0 or loop == 2:
                        hydrates_formulas.append(letter_formulas)
                    else:
                        waterless_forms_formulas.append(letter_formulas)

                    if loop == 0 or loop == 2:
                        if hash('O') in component_SMILES:
                            component_SMILES.remove(hash('O'))
                        if hash('[O]') in component_SMILES:
                            component_SMILES.remove(hash('[O]'))
                        component_SMILES = sorted(component_SMILES)
                        hydrates_SMILES.append(component_SMILES)
                    if loop == 1 or loop == 3:
                        component_SMILES = sorted(component_SMILES)
                        waterless_forms_SMILES.append(component_SMILES)
                loop += 1
                if loop == 1:
                    entry1 = entry_reader2
                if loop == 2:
                    entry1 = entry_reader3
                if loop == 3:
                    entry1 = entry_reader4

```

```
                    waterless_forms_recorded = []

                    print 'made it to comparison'

                    for d in range(len(hydrates_SMILES)):
                        count = 0
                        for e in range(len(waterless_forms_SMILES)):
                            #If the SMILES string and formula of the hydrate
                            structure match those of the waterless form, the two
                            structures are written to output files as
                            hydrate-anhydrate pairs
                            if hydrates_SMILES[d] == waterless_forms_SMILES[e] and
                            hydrates_formulas[d] == waterless_forms_formulas[e]:
                                count += 1
                                #There are 28,388 structures in the solvated
                                waterless forms input file
                                #These numbers serve as place holders so the
                                correct structures are compared and written to
                                output files
                                if e < 28388:
                                    if entry_reader2[e].identifier not in
                                    duplicate_identifier_dictionary:
                                        #Each waterless form that matches a hydrate
                                        is recorded in a list

                                        waterless_forms_recorded.append(hash(entry_re
                                        ader2[e].identifier))
                                    else:

                                        waterless_forms_recorded.append(duplicate_ide
                                        ntifier_dictionary.get(entry_reader2[e].ident
                                        ifier))
                                else:
                                    if entry_reader4[e-28388].identifier not in
                                    duplicate_identifier_dictionary:

                                        waterless_forms_recorded.append(hash(entry_re
                                        ader4[e-28388].identifier))
                                    else:

                                        waterless_forms_recorded.append(duplicate_ide
                                        ntifier_dictionary.get(entry_reader4[e-28388]
                                        .identifier))
                                #There are 3,499 structures in the solvated
                                hydrates input file
                                if d < 3439 and e < 28388:
                                    pairing_writer.write(entry_reader1[d])
                                    writer1.write(entry_reader2[e])
                                elif d < 3439 and e >= 28388:
                                    pairing_writer.write(entry_reader1[d])
                                    writer1.write(entry_reader4[e-28388])
                                elif d >= 3439 and e < 28388:
                                    pairing_writer.write(entry_reader3[d-3439])
                                    writer1.write(entry_reader2[e])
                                elif d >= 3439 and e >= 28388:
                                    pairing_writer.write(entry_reader3[d-3439])
                                    writer1.write(entry_reader4[e-28388])
                        #Hydrates that do not match any waterless forms are written
                        as hydrates without known anhydrate forms
                        if count == 0:
                            if d < 3439:
                                writer2.write(entry_reader1[d])
                            else:
                                writer2.write(entry_reader3[d-3439])

                        print d
```

```
                        print 'made it to generating files with unpaired structures'

                        #Any waterless forms that did not match a hydrate structure are
                        written as anhydrous forms without a known hydrate form
                        for f in range(len(entry_reader2)):
                            if entry_reader2[f].identifier not in
                            duplicate_identifier_dictionary:
                                if hash(entry_reader2[f].identifier) not in
                                waterless_forms_recorded:
                                    writer3.write(entry_reader2[f])
                            else:
                                if
                                duplicate_identifier_dictionary.get(entry_reader2[f].iden
                                tifier) not in waterless_forms_recorded:
                                    writer3.write(entry_reader2[f])
                        for g in range(len(entry_reader4)):
                            if entry_reader4[g].identifier not in
                            duplicate_identifier_dictionary:
                                if hash(entry_reader4[g].identifier) not in
                                waterless_forms_recorded:
                                    writer3.write(entry_reader4[g])
                            else:
                                if
                                duplicate_identifier_dictionary.get(entry_reader4[g].iden
                                tifier) not in waterless_forms_recorded:
                                    writer3.write(entry_reader4[g])


    if __name__ == '__main__':
        # This runs the script
        r = Runner()
        r.run()
```

```python
#22 Find hydrates with waterless forms using None method

from ccdc import io, search
import argparse
import os
import glob
import re

#This script finds hydrates-anhydrate pairs where either one or both structure is without an entry SMILES string
#The only automated piece that is done by the script is finding all the hydrate structures without an entry SMILES string that match the formula of a waterless format
#It also finds all the waterless forms without an entry SMILES string that match the formula of a hydrate
#The output files from this script were manually screened to find true hydrate-anhydrate pairs

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\None_solvated_hydrates.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\None_solvated_waterless_forms.txt"
filepath3 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\None_hydrates.txt"
filepath4 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\None_waterless_forms.txt"
filepath5 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\smiles_solvated_hydrates.txt"
filepath6 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\smiles_solvated_waterless_forms.txt"
filepath7 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\smiles_hydrates.txt"
filepath8 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\smiles_waterless_forms.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='None_hydrates_with_formula_match.gcd',
            help='output file [None_hydrates_with_formula_match.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')
        entry_reader2 = io.EntryReader(filepath2, format='identifiers')
        entry_reader3 = io.EntryReader(filepath3, format='identifiers')
        entry_reader4 = io.EntryReader(filepath4, format='identifiers')
        entry_reader5 = io.EntryReader(filepath5, format='identifiers')
        entry_reader6 = io.EntryReader(filepath6, format='identifiers')
        entry_reader7 = io.EntryReader(filepath7, format='identifiers')
        entry_reader8 = io.EntryReader(filepath8, format='identifiers')
```

```
56
57          total = 0
58          count = 0
59
60          with io.EntryWriter(self.args.output) as pairing_writer:
61              with io.EntryWriter("waterless_forms_that_match_None_hydrate_formula.gcd")
                as writer1:
62                  with io.EntryWriter("None_waterless_forms_with_formula_match.gcd") as
                    writer2:
63                      with io.EntryWriter("hydrates_that_match_None_waterless_forms.gcd")
                        as writer3:
64
65                          None_hydrates_formulas = []
66                          None_waterless_forms_formulas = []
67                          hydrates_formulas = []
68                          waterless_forms_formulas = []
69
70                          loop = 0
71                          entry1 = entry_reader1
72                          #Generate a list of formulas for hydrates and waterless forms
                            with and without an entry SMILES string
73                          while loop < 8:
74                              for a in range(len(entry1)):
75                                  component_formulas = entry1[a].formula.split(',')
76                                  for b in range(len(component_formulas)):
77                                      if '(' and ')' in component_formulas[b]:
78                                          start = component_formulas[b].index('(')
79                                          end = component_formulas[b].index(')')
80                                          component_formulas[b] =
                                            component_formulas[b][start+1:end]
81                                      component_formulas[b] =
                                        hash(component_formulas[b])
82                                  if loop == 0 or loop == 2:
83                                      if hash('H2 O1') in component_formulas:
84                                          component_formulas.remove(hash('H2 O1'))
85                                      if hash('D2 O1') in component_formulas:
86                                          component_formulas.remove(hash('D2 O1'))
87                                      component_formulas = sorted(component_formulas)
88                                      None_hydrates_formulas.append(component_formulas)
89                                  if loop == 1 or loop == 3:
90                                      component_formulas = sorted(component_formulas)
91
92                                      None_waterless_forms_formulas.append(component_formul
                                        as)
93                                  if loop == 4 or loop == 6:
94                                      if hash('H2 O1') in component_formulas:
95                                          component_formulas.remove(hash('H2 O1'))
96                                      if hash('D2 O1') in component_formulas:
97                                          component_formulas.remove(hash('D2 O1'))
98                                      component_formulas = sorted(component_formulas)
99                                      hydrates_formulas.append(component_formulas)
100                                 if loop == 5 or loop == 7:
101                                     component_formulas = sorted(component_formulas)
102
                                        waterless_forms_formulas.append(component_formulas)
103                             loop += 1
104                             if loop == 1:
105                                 entry1 = entry_reader2
106                             if loop == 2:
107                                 entry1 = entry_reader3
108                             if loop == 3:
109                                 entry1 = entry_reader4
110                             if loop == 4:
111                                 entry1 = entry_reader5
112                             if loop == 5:
                                    entry1 = entry_reader6
```

```
                    if loop == 6:
                        entry1 = entry_reader7
                    if loop == 7:
                        entry1 = entry_reader8

                #If a hydrate formula matches a waterless form, the two structures are written to output files as potential hydrate-anhdyrate pairs
                #Hydrates without an entry SMILES string are first compared to waterless forms without an entry SMILES string
                for d in range(len(None_hydrates_formulas)):
                    for e in range(len(None_waterless_forms_formulas)):
                        if None_hydrates_formulas[d] ==
                        None_waterless_forms_formulas[e]:
                            #There are 192 solvated hydrates without an entry SMILES string
                            #There are 1,555 solvated waterless forms without an entry SMILES string
                            if d < 192 and e < 1555:
                                pairing_writer.write(entry_reader1[d])
                                writer1.write(entry_reader2[e])
                            elif d < 192 and e >= 1555:
                                pairing_writer.write(entry_reader1[d])
                                writer1.write(entry_reader4[e-1555])
                            elif d >= 192 and e < 1555:
                                pairing_writer.write(entry_reader3[d-192])
                                writer1.write(entry_reader2[e])
                            elif d >= 192 and e >= 1555:
                                pairing_writer.write(entry_reader3[d-192])
                                writer1.write(entry_reader4[e-1555])
                    #Hydrates without an entry SMILES string are then compared to waterless forms with an entry SMILES string
                    for f in range(len(waterless_forms_formulas)):
                        if None_hydrates_formulas[d] ==
                        waterless_forms_formulas[f]:
                            if d < 192 and f < 28388:
                                pairing_writer.write(entry_reader1[d])
                                writer1.write(entry_reader6[f])
                            elif d < 192 and f >= 28388:
                                pairing_writer.write(entry_reader1[d])
                                writer1.write(entry_reader8[f-28388])
                            elif d >= 192 and f < 28388:
                                pairing_writer.write(entry_reader3[d-192])
                                writer1.write(entry_reader6[f])
                            elif d >= 192 and f >= 28388:
                                pairing_writer.write(entry_reader3[d-192])
                                writer1.write(entry_reader8[f-28388])

                #Waterless forms without an entry SMILES string are compared to hydrates with an entry SMILES string
                for g in range(len(None_waterless_forms_formulas)):
                    for h in range(len(hydrates_formulas)):
                        if None_waterless_forms_formulas[g] ==
                        hydrates_formulas[h]:
                            if g < 1555 and h < 3439:
                                writer2.write(entry_reader2[g])
                                writer3.write(entry_reader5[h])
                            elif g < 1555 and h >= 3439:
                                writer2.write(entry_reader2[g])
                                writer3.write(entry_reader7[h-3439])
                            elif g >= 1555 and h < 3439:
                                writer2.write(entry_reader4[g-1555])
                                writer3.write(entry_reader5[h])
                            elif g >= 1555 and h >= 3439:
                                writer2.write(entry_reader4[g-1555])
                                writer3.write(entry_reader7[h-3439])

```

```
170
171
172 if __name__ == '__main__':
173     # This runs the script
174     r = Runner()
175     r.run()
176
```

```python
#23 Find hydrate-anhydrate pairs with distinct chirality

from ccdc import io, search
import argparse
import os
import glob
import re

#This script finds hydrate-anhydrate pairs with distinct chirality.
#This script is the same as the one used to find duplicate structures with distinct chirality
#The only difference is that water is not included in the molecule formula check against the entry formula for hydrate structures

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step5 Find Paired Hydrates and Waterless Forms\set1_potential_hydrates_with_waterless_form.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step5 Find Paired Hydrates and Waterless Forms\set1_potential_waterless_forms_with_hydrate.txt"
filepath3 = "C:\Users\jenwe\Hydrates Manuscript\Step5 Find Paired Hydrates and Waterless Forms\set2_potential_hydrates_with_waterless_form.txt"
filepath4 = "C:\Users\jenwe\Hydrates Manuscript\Step5 Find Paired Hydrates and Waterless Forms\set2_potential_waterless_forms_with_hydrate.txt"
filepath5 = "C:\Users\jenwe\Hydrates Manuscript\Step5 Find Paired Hydrates and Waterless Forms\set3_potential_hydrates_with_waterless_form.txt"
filepath6 = "C:\Users\jenwe\Hydrates Manuscript\Step5 Find Paired Hydrates and Waterless Forms\set3_potential_waterless_forms_with_hydrate.txt"
filepath7 = "C:\Users\jenwe\Hydrates Manuscript\Step5 Find Paired Hydrates and Waterless Forms\potential_None_hydrates_with_waterless_form.txt"
filepath8 = "C:\Users\jenwe\Hydrates Manuscript\Step5 Find Paired Hydrates and Waterless Forms\potential_None_waterless_forms_with_hydrate.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='same_chirality_hydrates_with_waterless_form.gcd',
            help='output file [same_chirality_hydrates_with_waterless_form.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')
        entry_reader2 = io.EntryReader(filepath2, format='identifiers')
        entry_reader3 = io.EntryReader(filepath3, format='identifiers')
        entry_reader4 = io.EntryReader(filepath4, format='identifiers')
        entry_reader5 = io.EntryReader(filepath5, format='identifiers')
        entry_reader6 = io.EntryReader(filepath6, format='identifiers')
        entry_reader7 = io.EntryReader(filepath7, format='identifiers')
        entry_reader8 = io.EntryReader(filepath8, format='identifiers')

        with io.EntryWriter(self.args.output) as chiral_writer:
            with io.EntryWriter("same_chirality_waterless_forms_with_hydrate.gcd") as
```

```
writer1:
    with
    io.EntryWriter("distinct_chirality_hydrates_with_enantiomer_waterless_for
    m.gcd") as writer2:
        with
        io.EntryWriter("distinct_chirality_waterless_forms_with_enantiomer_hy
        drate.gcd") as writer3:
            with
            io.EntryWriter("achiral_names_one_missing_stereo_hydrates_with_wa
            terless_form.gcd") as writer4:
                with
                io.EntryWriter("achiral_names_one_missing_stereo_waterless_fo
                rms_with_hydrate.gcd") as writer5:
                    with
                    io.EntryWriter("one_chiral_name_one_missing_stereo_hydrat
                    es_with_waterless_form.gcd") as writer6:
                        with
                        io.EntryWriter("one_chiral_name_one_missing_stereo_wa
                        terless_forms_with_hydrate.gcd") as writer7:
                            with
                            io.EntryWriter("chiral_names_check_hydrates.gcd")
                             as writer8:
                                with
                                io.EntryWriter("chiral_names_check_waterless_
                                forms.gcd") as writer9:

                                    duplicate_identifier_dictionary =
                                    {'GADKOL':1, 'MECYEY':2, 'OBASAN':3,
                                    'OMOMAE':4, 'OTONUI':5, 'QOJWAO':6,
                                    'TZBFZO':7, 'VOBXOZ':8, 'VUFJAI':9,
                                    'VUFJAJ':10, 'WEPQOA':11, 'XADFUD':12,
                                    'YABREY':13, 'YULCOZ':14, 'ZOVFEV':15}
                                    hydrate_tags_empty_one_stereo_hydrates
                                    = [hash('BEXPOK'), hash('QQQCYA01'),
                                    hash('VOLZON'), hash('VOMPEU'),
                                    hash('WEGPED'), hash('AHLPRO'),
                                    hash('YUPBER')]

                                    hydrate_tags_empty_one_stereo_waterless_f
                                    orms = [hash('BEXPIE'), hash('NAMNSB'),
                                    hash('VOMNUI'), hash('VOMNUI'),
                                    hash('WEGPIH'), hash('HOPROL'),
                                    hash('VALINO01')]
                                    hydrate_tags_empty_no_stereo_hydrates =
                                    [hash('ADPRTY'), hash('BEWWIK'),
                                    hash('BEWWIK'), hash('BUTAHC11'),
                                    hash('DUGTOQ'), hash('ETHYLO'),
                                    hash('ETHYLO'), hash('ETXDHY01'),
                                    hash('ETXDHY01'), hash('FEFNOT01'),
                                    hash('FEFNOT01'), hash('FEFNOT01'),
                                    hash('FEFNOT01'), hash('FEFNOT01'),
                                    hash('FEFNOT01'), hash('FEFNOT01'),
                                    hash('FEFNOT01'), hash('FEFNOT01'),
                                    hash('FEFNOT01'), hash('FEFNOT01'),
                                    hash('FEFNOT01'), hash('FEFNOT01'),
                                    hash('FEFNOT01'), hash('FEFNOT01'),
                                    hash('FEFNOT01'), hash('FEFNOT01'),
                                    hash('FEFNOT05'), hash('FEFNOT05'),
                                    hash('FEFNOT05'), hash('FEFNOT05'),
                                    hash('FEFNOT05'), hash('FEFNOT05'),
                                    hash('FEFNOT05'), hash('FEFNOT05'),
                                    hash('FEFNOT05'), hash('FEFNOT05'),
                                    hash('FEFNOT05'), hash('FEFNOT05'),
                                    hash('FEFNOT05'), hash('FEFNOT05'),
                                    hash('FEFNOT05'), hash('FEFNOT05'),
```

```
hash('FEFNOT05'), hash('FEFNOT10'),
hash('FEFNOT10'), hash('FEFNOT10'),
hash('FEFNOT10'), hash('FEFNOT10'),
hash('FEFNOT10'), hash('FEFNOT10'),
hash('FEFNOT10'), hash('FEFNOT10'),
hash('FEFNOT10'), hash('FEFNOT10'),
hash('FEFNOT10'), hash('FEFNOT10'),
hash('FEFNOT10'), hash('FEFNOT10'),
hash('FEFNOT10'), hash('FEFNOT10'),
hash('FEFNOT10'), hash('FEFNOT10'),
hash('FEFNOT11'), hash('FEFNOT11'),
hash('FEFNOT11'), hash('FEFNOT11'),
hash('FEFNOT11'), hash('FEFNOT11'),
hash('FEFNOT11'), hash('FEFNOT11'),
hash('FEFNOT11'), hash('FEFNOT11'),
hash('FEFNOT11'), hash('FEFNOT11'),
hash('FEFNOT11'), hash('FEFNOT11'),
hash('FEFNOT11'), hash('FEFNOT11'),
hash('FEFNOT11'), hash('FEFNOT11'),
hash('FEFNOT11'), hash('FUSWOH04'),
hash('FUSWOH04'), hash('FUSWOH04'),
hash('FUSWOH04'), hash('FUSWOH04'),
hash('FUSWOH04'), hash('FUSWOH04'),
hash('FUSWOH04'), hash('IKALIR'),
hash('IKALIR'), hash('IKALIR01'),
hash('IKALIR01'), hash('IKALIR02'),
hash('IKALIR02'), hash('IKALIR03'),
hash('IKALIR03'), hash('IKAMEO'),
hash('IKAMEO'), hash('IKAMEO01'),
hash('IKAMEO01'), hash('IKAMOY'),
hash('IKAMOY'), hash('IKAMOY01'),
hash('IKAMOY01'), hash('IKAMOY02'),
hash('IKAMOY02'), hash('IKAMOY03'),
hash('IKAMOY03'), hash('IKAMOY04'),
hash('IKAMOY04'), hash('IKAMOY05'),
hash('IKAMOY05'), hash('IKAVOF'),
hash('IKAVOF'), hash('IKAVOF'),
hash('IKAVOF'), hash('IKAVOF'),
hash('IKAVOF'), hash('IKAVOF'),
hash('IKAVOF'), hash('IKAVOF'),
hash('JARBUA'), hash('KUYGOA'),
hash('KUYGUG'), hash('KUYGUG'),
hash('KUYGUG'), hash('KUYGUG'),
hash('KUYGUG'), hash('KUYGUG'),
hash('KUYHAN'), hash('KUYHAN'),
hash('KUYHAN'), hash('KUYHAN'),
hash('KUYHAN'), hash('KUYHAN'),
hash('KUYHAN'), hash('KUYHAN'),
hash('KUYHAN'), hash('KUYHAN'),
hash('KUYHER'), hash('KUYHER'),
hash('MUSDEL'), hash('MUSDOV'),
hash('NAHCIJ'), hash('NAHCIJ'),
hash('NAHCOP'), hash('NAHCOP'),
hash('NAHCOP'), hash('NAHCOP'),
hash('NAHCOP'), hash('NAHCOP'),
hash('NAHCOP'), hash('NAHCOP'),
hash('QQQCIY10'), hash('QQQCIY10'),
hash('QQQCIY10'), hash('QQQCIY10'),
hash('QQQCIY10'), hash('QQQCIY10'),
hash('QQQCIY10'), hash('RITMEN'),
hash('RITMEN'), hash('SOGUAN06'),
hash('XEBQAZ'), hash('ZZZPPI01'),
hash('ZZZWQA01'), hash('ZZZWQA01'),
hash('FEFNOT'), hash('FEFNOT'),
hash('FEFNOT'), hash('FEFNOT'),
hash('FEFNOT'), hash('FEFNOT'),
hash('FEFNOT'), hash('FEFNOT'),
```

```
hash('CBMZPN16'), hash('CBMZPN17'),
hash('CBMZPN18'), hash('CBMZPN20'),
hash('CBMZPN21'), hash('CBMZPN22'),
hash('CBMZPN23'), hash('CBMZPN27'),
hash('CBMZPN28'), hash('CBMZPN29'),
hash('CBMZPN30'), hash('CBMZPN01'),
hash('CBMZPN02'), hash('CBMZPN03'),
hash('CBMZPN10'), hash('CBMZPN11'),
hash('CBMZPN12'), hash('CBMZPN13'),
hash('CBMZPN14'), hash('CBMZPN16'),
hash('CBMZPN17'), hash('CBMZPN18'),
hash('CBMZPN20'), hash('CBMZPN21'),
hash('CBMZPN22'), hash('CBMZPN23'),
hash('CBMZPN27'), hash('CBMZPN28'),
hash('CBMZPN29'), hash('CBMZPN30'),
hash('CBMZPN01'), hash('CBMZPN02'),
hash('CBMZPN03'), hash('CBMZPN10'),
hash('CBMZPN11'), hash('CBMZPN12'),
hash('CBMZPN13'), hash('CBMZPN14'),
hash('CBMZPN16'), hash('CBMZPN17'),
hash('CBMZPN18'), hash('CBMZPN20'),
hash('CBMZPN21'), hash('CBMZPN22'),
hash('CBMZPN23'), hash('CBMZPN27'),
hash('CBMZPN28'), hash('CBMZPN29'),
hash('CBMZPN30'), hash('CBMZPN01'),
hash('CBMZPN02'), hash('CBMZPN03'),
hash('CBMZPN10'), hash('CBMZPN11'),
hash('CBMZPN12'), hash('CBMZPN13'),
hash('CBMZPN14'), hash('CBMZPN16'),
hash('CBMZPN17'), hash('CBMZPN18'),
hash('CBMZPN20'), hash('CBMZPN21'),
hash('CBMZPN22'), hash('CBMZPN23'),
hash('CBMZPN27'), hash('CBMZPN28'),
hash('CBMZPN29'), hash('CBMZPN30'),
hash('EFUMAU'), hash('EFUMAU01'),
hash('EFUMAU02'), hash('EFUMAU03'),
hash('EFUMAU04'), hash('EFUMAU05'),
hash('EFUMAU06'), hash('EFUMAU07'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('IKAKOW'), hash('IKAKOW01'),
hash('ETHANE01'), hash('ETHANE04'),
hash('ETHANE05'), hash('ETHANE06'),
hash('ETHANE07'), hash('ETHANE08'),
hash('ETHANE09'), hash('ETHANE10'),
hash('ETHANE11'), hash('JARBOU'),
hash('ZZZITY01'), hash('KAMTAW'),
hash('KAMTAW01'), hash('KAMTAW02'),
hash('KAMTAW03'), hash('KAMTAW04'),
hash('KAMTAW05'), hash('KAMTAW06'),
hash('DCLBEN'), hash('DCLBEN01'),
hash('DCLBEN02'), hash('DCLBEN03'),
hash('DCLBEN04'), hash('DCLBEN05'),
hash('DCLBEN06'), hash('DCLBEN07'),
hash('DCLBEN11'), hash('CISTON01'),
hash('TAQYUG'), hash('DUHJIB'),
hash('DUHJIB'), hash('ACETYL02'),
hash('ACETYL03'), hash('JAYDUI'),
```

```
hash('BUNJAV01'), hash('CBMZPN01'),
hash('CBMZPN02'), hash('CBMZPN03'),
hash('CBMZPN10'), hash('CBMZPN11'),
hash('CBMZPN12'), hash('CBMZPN13'),
hash('CBMZPN14'), hash('CBMZPN16'),
hash('CBMZPN17'), hash('CBMZPN18'),
hash('CBMZPN20'), hash('CBMZPN21'),
hash('CBMZPN22'), hash('CBMZPN23'),
hash('CBMZPN27'), hash('CBMZPN28'),
hash('CBMZPN29'), hash('CBMZPN30'),
hash('PINCOL'), hash('PINCOL01'),
hash('PINCOL02'), hash('PINCOL03'),
hash('DAYFUH'), hash('LUXYIM'),
hash('LUYMAT'), hash('LUYMAT01'),
hash('NIWFEE02'), hash('NIWFEE02'),
hash('NIWFEE04'), hash('NIWFEE04'),
hash('SABZOK'), hash('SOCTOT'),
hash('THIOUR06'), hash('THIOUR06'),
hash('THIOUR06'), hash('THIOUR06'),
hash('PUSBOV')]

loop = 2
while loop < 4:
    if loop == 0:
        entry1 = entry_reader1
    if loop == 1:
        entry1 = entry_reader3
    if loop == 2:
        entry1 = entry_reader5
    if loop == 3:
        entry1 = entry_reader7
    hydrates = []
    waterless = []
    count1 = 0
    count2 = 0
    count3 = 0
    for a in range(len(entry1)):
        print a
        loop2 = 0
        if loop2 == 0:
            if loop == 0:
                entry1 = entry_reader1
            if loop == 1:
                entry1 = entry_reader3
            if loop == 2:
                entry1 = entry_reader5
            if loop == 3:
                entry1 = entry_reader7
        formula_error = 0
        missing_components = 0
        while loop2 < 2:
            letter_formulas = []
            formulas =
            entry1[a].formula.split(',')
            for b in
            range(len(formulas)):
                if '(' in formulas[b]
                and ')' in formulas[b]:
                    if '(H2 O1)' not in
                    formulas[b] and
                    '(D2 O1)' not in
                    formulas[b]:
                        pin =
                        formulas[b].index
                        ('(')

                        letter_formulas.a
```

```
                                                                    ppend(formulas[b]
                                                                    [pin+1:-1])
                                                            else:
                                                                if formulas[b] !=
                                                                'H2 O1' and
                                                                formulas[b] != 'D2
                                                                O1':
                                                                    letter_formulas.a
                                                                    ppend(formulas[b]
                                                                    )
                                                    molecule1 =
                                                    entry1[a].molecule
                                                    checked = []
                                                    tags = []
                                                    error = 0
                                                    count = 0
                                                    for g in
                                                    range(len(molecule1.component
                                                    s)):
                                                        if
                                                        molecule1.components[g].f
                                                        ormula not in checked
                                                        and
                                                        molecule1.components[g].f
                                                        ormula != 'H2 O1':
                                                            if
                                                            molecule1.components[
                                                            g].formula not in
                                                            letter_formulas:
                                                                if 'H' in
                                                                molecule1.compone
                                                                nts[g].formula:
                                                                    if
molecule1.components[g].formula.replace('H', 'D') not in letter_formulas:
                                                                        for t
in range(len(formulas)):
                                                                            if
'H' in formulas[t] and 'D' in formulas[t]:
                                                                                for s in range(len(pieces)):
                                                                                    if 'H' in pieces[s]:
                                                                                        tag1 = pieces[s].index('H')
                                                                                        number1 = int(pieces[s][tag1+1:])
                                                                                    if 'D' in pieces[s]:
                                                                                        tag2 = pieces[s].index('D')
                                                                                        number2 = int(pieces[s][tag2+1:])
                                                                                partial_deuterated += 1
                                                                            if
partial_deuterated == 1:
                                                                                pieces2 = molecule1.components[g].formula.split(' ')
                                                                                for
u in range(len(pieces2)):
                                                                                    if 'H' in pieces2[u]:
```

```
tag3 = pieces2[u].index('H')
number3 = int(pieces2[u][tag3+1:])
if number1+number2 == number3:
checked.append(molecule1.components[g].formula)
tags.append(g)
count += 1
else:
checked.append(molecule1.components[g].formula)
tags.append(g)
error += 1
else:
checked.append(molecule1.components[g].formula)
tags.append(g)
error += 1
else:
checked.append(molecule1.components[g].formula)
tags.append(g)
count += 1
else:
checked.append(molecule1.components[g].formula)
tags.append(g)
error += 1
else:
checked.append(molecule1.components[g].formula)
tags.append(g)
count += 1
if loop2 == 0:
    hydrate_tags = tags
    hydrates_checked = checked
else:
    waterless_tags = tags
    waterless_checked = checked
if error != 0:
    formula_error += 1
if count != len(letter_formulas):
    missing_components += 1
loop2 += 1
if loop == 0:
    entry1 = entry_reader2
if loop == 1:
```

```
                entry1 = entry_reader4
            if loop == 2:
                entry1 = entry_reader6
            if loop == 3:
                entry1 = entry_reader8

        if loop == 0:
            entry2 = entry_reader1
            entry3 = entry_reader2
        if loop == 1:
            entry2 = entry_reader3
            entry3 = entry_reader4
        if loop == 2:
            entry2 = entry_reader5
            entry3 = entry_reader6
        if loop == 3:
            entry2 = entry_reader7
            entry3 = entry_reader8

        if formula_error != 0 or
        missing_components != 0:
            remove_hydrate_tags = []
            for c in
            range(len(hydrate_tags)):
                if
                entry2[a].molecule.compon
                ents[hydrate_tags[c]].for
                mula not in
                waterless_checked:

                    remove_hydrate_tags.a
                    ppend(hydrate_tags[c]
                    )
            for e in
            range(len(remove_hydrate_tags
            )):

                hydrate_tags.remove(remov
                e_hydrate_tags[e])
            remove_waterless_tags = []
            for d in
            range(len(waterless_tags)):
                if
                entry3[a].molecule.compon
                ents[waterless_tags[d]].f
                ormula not in
                hydrates_checked:

                    remove_waterless_tags
                    .append(waterless_tag
                    s[d])
            for f in
            range(len(remove_waterless_ta
            gs)):

                waterless_tags.remove(rem
                ove_waterless_tags[f])

        if len(hydrate_tags) !=
        len(waterless_tags):
            print entry2[a].identifier
            print entry3[a].identifier
            print waterless_tags[90]

        same_names = 0
        possibly_the_same_names = 0
        distinct_names = 0
```

```
                                                    DoubleR = 0
                                                    DoubleBlank = 0
                                                    RandBlank = 0
                                                    hydrate_titles =
                                                    list(entry2[a].synonyms)
                                                    if entry2[a].identifier ==
                                                    'WACHIR':
                                                        hydrate_titles =
                                                        hydrate_titles[0].split(',')

                                                    hydrate_titles.append(entry2[a].c
                                                    hemical_name)
                                                    for u in
                                                    range(len(hydrate_titles)):
                                                        caps = [i for i in
                                                        hydrate_titles[u] if
                                                        i.isupper()]
                                                        caps = list(set(caps))
                                                        for q in range(len(caps)):
                                                            if caps[q] != 'D' and
                                                            caps[q] != 'L' and
                                                            caps[q] != 'R' and
                                                            caps[q] != 'S' and
                                                            caps[q] != 'E' and
                                                            caps[q] != 'Z':
                                                                hydrate_titles[u] =
                                                                re.sub(caps[q],
                                                                caps[q].lower(),
                                                                hydrate_titles[u])
                                                            else:
                                                                places = [j for j,
                                                                letter in
                                                                enumerate(hydrate_tit
                                                                les[u]) if letter
                                                                == caps[q]]
                                                                for k in
                                                                range(len(places)):
                                                                    if places[k]+1
                                                                    ==
                                                                    len(hydrate_title
                                                                    s[u]):

hydrate_titles[u] = hydrate_titles[u].replace(hydrate_titles[u][places[k]],
hydrate_titles[u][places[k]].lower())
                                                                    elif
                                                                    hydrate_titles[u]
                                                                    [places[k]+1]
                                                                    != ')' and
                                                                    hydrate_titles[u]
                                                                    [places[k]+1]
                                                                    != '-' and
                                                                    hydrate_titles[u]
                                                                    [places[k]+1]
                                                                    != ',':

hydrate_titles[u] = hydrate_titles[u].replace(hydrate_titles[u][places[k]],
hydrate_titles[u][places[k]].lower())
                                                    hydrate_pieces =
                                                    hydrate_titles[u].split(' ')
                                                    hydrate_deuterated = [s for
                                                    s in hydrate_pieces if
                                                    'deutero' in s]
                                                    if len(hydrate_deuterated)
                                                    == 1:
                                                        clip =
                                                        hydrate_deuterated[0].ind
                                                        ex('deutero')
```

```
                    if 'deutero-' in
                    hydrate_deuterated[0]:
                        hydrate_replacement
                        =
                        [hydrate_deuterated[0
                        ][clip+8:]]
                    else:
                        hydrate_replacement
                        =
                        [hydrate_deuterated[0
                        ][clip+7:]]
                    for t in
                    range(len(hydrate_pieces)
                    ):
                        if
                        hydrate_pieces[t]
                        ==
                        hydrate_deuterated[0]
                        :

                            hydrate_pieces[t]
                             =
                            hydrate_replaceme
                            nt[0]
                water = [r for r in
                hydrate_pieces if 'hydrate'
                in r and 'perhydrate' not
                in r]
                D2O = 0
                if len(water) == 0:
                    if 'deuterium oxide
                    solvate' in
                    hydrate_titles[u].lower()
                     or '(deuterium oxide)
                    solvate' in
                    hydrate_titles[u].lower()
                    :
                        water = [r for r in
                        hydrate_pieces if
                        'deuterium' in r or
                        'oxide' in r or
                        'solvate' in r]
                    elif '(dideuterium
                    oxide)' in
                    hydrate_titles[u].lower()
                    :
                        water = [r for r in
                        hydrate_pieces if
                        'dideuterium' in r
                        or 'oxide' in r]
                    elif 'deuterium oxide'
                    in
                    hydrate_titles[u].lower()
                    :
                        water = [r for r in
                        hydrate_pieces if
                        'deuterium' in r or
                        'oxide' in r]
                    hydrate_pieces = [v for v
                    in hydrate_pieces if v not
                    in water]
                    hydrate_titles[u] = '
                    '.join(hydrate_pieces)
            waterless_titles =
            list(entry3[a].synonyms)

            waterless_titles.append(entry3[a]
```

```
                                                    .chemical_name)
264                                                 for u in
                                                    range(len(waterless_titles)):
265                                                     caps2 = [i for i in
                                                        waterless_titles[u] if
                                                        i.isupper()]
266                                                     caps2 = list(set(caps2))
267                                                     for q in range(len(caps2)):
268                                                         if caps2[q] != 'D' and
                                                            caps2[q] != 'L' and
                                                            caps2[q] != 'R' and
                                                            caps2[q] != 'S' and
                                                            caps2[q] != 'E' and
                                                            caps2[q] != 'Z':
269                                                             waterless_titles[u]
                                                            = re.sub(caps2[q],
                                                            caps2[q].lower(),
                                                            waterless_titles[u])
270                                                         else:
271                                                             places2 = [j for j,
                                                            letter in
                                                            enumerate(waterless_t
                                                            itles[u]) if letter
                                                            == caps2[q]]
272                                                             for k in
                                                            range(len(places2)):
273                                                                 if places2[k]+1
                                                                ==
                                                                len(waterless_tit
                                                                les[u]):
274
    waterless_titles[u] = waterless_titles[u].replace(waterless_titles[u][places2[k]],
    waterless_titles[u][places2[k]].lower())
275                                                                 elif
                                                                waterless_titles[
                                                                u][places2[k]+1]
                                                                != ')' and
                                                                waterless_titles[
                                                                u][places2[k]+1]
                                                                != '-' and
                                                                waterless_titles[
                                                                u][places2[k]+1]
                                                                != ',':
276
    waterless_titles[u] = waterless_titles[u].replace(waterless_titles[u][places2[k]],
    waterless_titles[u][places2[k]].lower())
277                                                     waterless_pieces =
                                                        waterless_titles[u].split('
                                                        ')
278                                                     waterless_deuterated = [s
                                                        for s in waterless_pieces
                                                        if 'deutero' in s.lower()]
279                                                     if
                                                        len(waterless_deuterated)
                                                        == 1:
280                                                         clip =
                                                        waterless_deuterated[0].l
                                                        ower().index('deutero')
281                                                         if 'deutero-' in
                                                        waterless_deuterated[0].l
                                                        ower():
282
                                                            waterless_replacement
                                                             =
                                                            [waterless_deuterated
                                                            [0][clip+8:]]
283                                                         else:
```

```
                    waterless_replacement
                     =
                    [waterless_deuterated
                    [0][clip+7:]]
                for t in
                range(len(waterless_piece
                s)):
                    if
                    waterless_pieces[t]
                    ==
                    waterless_deuterated[
                    0]:

                        waterless_pieces[
                        t] =
                        waterless_replace
                        ment[0]
        waterless_titles[u] = '
        '.join(waterless_pieces)
if any('rac-' in x for x in
hydrate_titles) and any('rac-'
in y for y in waterless_titles)
or any('DL' in x for x in
hydrate_titles) and any('DL' in
y for y in waterless_titles) or
any('RS' in x for x in
hydrate_titles) and any('RS' in
y for y in waterless_titles) or
any('RS' in x for x in
hydrate_titles) and any('SR' in
y for y in waterless_titles) or
any('SR' in x for x in
hydrate_titles) and any('RS' in
y for y in waterless_titles) or
any('SR' in x for x in
hydrate_titles) and any('SR' in
y for y in waterless_titles) or
any('+-' in x for x in
hydrate_titles) and any('+-' in
y for y in waterless_titles):
    same_names += 1
    if bool(set(hydrate_titles)
    & set(waterless_titles)) ==
    True:

        chiral_writer.write(entry
        2[a])
        writer1.write(entry3[a])
    else:
        possibly_the_same_names
        += 1
        DoubleR += 1

elif any('D' in x for x in
hydrate_titles) and any('D' in
y for y in waterless_titles) or
any('L' in x for x in
hydrate_titles) and any('L' in
y for y in waterless_titles) or
any('R' in x for x in
hydrate_titles) and any('R' in
y for y in waterless_titles) or
any('S' in x for x in
hydrate_titles) and any('S' in
y for y in waterless_titles) or
any('(+)-' in x for x in
```

```
hydrate_titles) and any('(+)-'
in y for y in waterless_titles)
or any('(-)-' in x for x in
hydrate_titles) and any('(-)-'
in y for y in waterless_titles)
or any('cis-' in x for x in
hydrate_titles) and any('cis-'
in y for y in waterless_titles)
or any('trans-' in x for x in
hydrate_titles) and
any('trans-' in y for y in
waterless_titles) or
any('meso-' in x for x in
hydrate_titles) and any('meso-'
in y for y in waterless_titles)
or any('E' in x for x in
hydrate_titles) and any('E' in
y for y in waterless_titles) or
any('Z' in x for x in
hydrate_titles) and any('Z' in
y for y in waterless_titles):
    same_names += 1
    if bool(set(hydrate_titles)
    & set(waterless_titles)) ==
    True:

        chiral_writer.write(entry
        2[a])
        writer1.write(entry3[a])
    else:
        possibly_the_same_names
        += 1
        DoubleR += 1
elif all('rac-' not in x for x
in hydrate_titles) and
all('rac-' not in y for y in
waterless_titles) and all('+-'
not in x for x in
hydrate_titles) and all('+-'
not in y for y in
waterless_titles) and all('D'
not in x for x in
hydrate_titles) and all('D' not
in y for y in waterless_titles)
and all('L' not in x for x in
hydrate_titles) and all('L' not
in y for y in waterless_titles)
and all('R' not in x for x in
hydrate_titles) and all('R' not
in y for y in waterless_titles)
and all('S' not in x for x in
hydrate_titles) and all('S' not
in y for y in waterless_titles)
and all('(+)-' not in x for x
in hydrate_titles) and
all('(+)-' not in y for y in
waterless_titles) and
all('(-)-' not in x for x in
hydrate_titles) and all('(-)-'
not in y for y in
waterless_titles) and
all('cis-' not in x for x in
hydrate_titles) and all('cis-'
not in y for y in
waterless_titles) and
all('trans-' not in x for x in
hydrate_titles) and
```

```
all('trans-' not in y for y in
waterless_titles) and
all('meso-' not in x for x in
hydrate_titles) and all('meso-'
not in y for y in
waterless_titles) and all('E'
not in x for x in
hydrate_titles) and all('E' not
in y for y in waterless_titles)
and all('Z' not in x for x in
hydrate_titles) and all('Z' not
in y for y in waterless_titles):
    possibly_the_same_names += 1
    DoubleBlank += 1
else:
    if any('D' in x for x in
    hydrate_titles) or any('L'
    in x for x in
    hydrate_titles) or any('R'
    in x for x in
    hydrate_titles) or any('S'
    in x for x in
    hydrate_titles) or
    any('(+)-' in x for x in
    hydrate_titles) or
    any('(-)-' in x for x in
    hydrate_titles) or
    any('cis-' in x for x in
    hydrate_titles) or
    any('trans-' in x for x in
    hydrate_titles) or any('E'
    in x for x in
    hydrate_titles) or any('Z'
    in x for x in
    hydrate_titles):
        if all('rac-' not in y
        for y in
        waterless_titles) and
        all('+-' not in y for y
        in waterless_titles)
        and all('D' not in y
        for y in
        waterless_titles) and
        all('L' not in y for y
        in waterless_titles)
        and all('R' not in y
        for y in
        waterless_titles) and
        all('S' not in y for y
        in waterless_titles)
        and all('(+)-' not in y
        for y in
        waterless_titles) and
        all('(-)-' not in y for
        y in waterless_titles)
        and all('cis-' not in y
        for y in
        waterless_titles) and
        all('trans-' not in y
        for y in
        waterless_titles) and
        all('meso-' not in y
        for y in
        waterless_titles) and
        all('E' not in y for y
        in waterless_titles)
        and all('Z' not in y
```

```
                for y in
                waterless_titles):
312
                    possibly_the_same_nam
                    es += 1
313                 RandBlank += 1
314             else:
315                 distinct_names += 1
316
                    writer2.write(entry2[
                    a])
317
                    writer3.write(entry3[
                    a])
318         elif any('D' in y for y in
            waterless_titles) or
            any('L' in y for y in
            waterless_titles) or
            any('R' in y for y in
            waterless_titles) or
            any('S' in y for y in
            waterless_titles) or
            any('(+)-' in y for y in
            waterless_titles) or
            any('(-)-' in y for y in
            waterless_titles) or
            any('cis-' in y for y in
            waterless_titles) or
            any('trans-' in y for y in
            waterless_titles) or
            any('E' in y for y in
            waterless_titles) or
            any('Z' in y for y in
            waterless_titles):
319             if all('rac-' not in x
                for x in
                hydrate_titles) and
                all('+-' not in x for x
                in hydrate_titles) and
                all('D' not in x for x
                in hydrate_titles) and
                all('L' not in x for x
                in hydrate_titles) and
                all('R' not in x for x
                in hydrate_titles) and
                all('S' not in x for x
                in hydrate_titles) and
                all('(+)-' not in x for
                x in hydrate_titles)
                and all('(-)-' not in x
                for x in
                hydrate_titles) and
                all('cis-' not in x for
                x in hydrate_titles)
                and all('trans-' not in
                x for x in
                hydrate_titles) and
                all('meso-' not in x
                for x in
                hydrate_titles) and
                all('E' not in x for x
                in hydrate_titles) and
                all('Z' not in x for x
                in hydrate_titles):
320
                    possibly_the_same_nam
                    es += 1
```

```
                        RandBlank += 1
                    else:
                        distinct_names += 1

                        writer2.write(entry2[
                        a])

                        writer3.write(entry3[
                        a])
                else:
                    distinct_names += 1
                    writer2.write(entry2[a])
                    writer3.write(entry3[a])

            if possibly_the_same_names != 0
            and hydrate_tags == []:
                if
                hash(entry2[a].identifier)
                in
                hydrate_tags_empty_no_stereo_
                hydrates:
                    indices = [i for i, x
                    in
                    enumerate(hydrate_tags_em
                    pty_no_stereo_hydrates)
                    if x ==
                    hash(entry2[a].identifier
                    )]
                    if
                    any(hydrate_tags_empty_no
                    _stereo_waterless_forms[j
                    ] ==
                    hash(entry3[a].identifier
                    ) for j in indices) ==
                    True:
                        if DoubleR != 0:

                            writer8.write(ent
                            ry2[a])

                            writer9.write(ent
                            ry3[a])
                        else:

                            chiral_writer.wri
                            te(entry2[a])

                            writer1.write(ent
                            ry3[a])
                    else:
                        print
                        entry2[a].identifier
                        print
                        entry3[a].identifier
                        print
                        hydrate_tags[90]
                elif
                hash(entry2[a].identifier)
                in
                hydrate_tags_empty_one_stereo
                _hydrates and
                hydrate_tags_empty_one_stereo
                _waterless_forms[hydrate_tags
                _empty_one_stereo_hydrates.in
                dex(hash(entry2[a].identifier
                ))] ==
                hash(entry3[a].identifier):
```

```
                        if DoubleR != 0:
                            writer8.write(entry2[a])
                            writer9.write(entry3[a])
                        else:
                            chiral_writer.write(entry2[a])
                            writer1.write(entry3[a])
                            if DoubleBlank != 0:
                                writer4.write(entry2[a])
                                writer5.write(entry3[a])
                            if RandBlank != 0:
                                writer6.write(entry2[a])
                                writer7.write(entry3[a])
                    else:
                        print entry2[a].identifier
                        print entry3[a].identifier
                        print hydrate_tags[90]

                if possibly_the_same_names != 0 and hydrate_tags != []:
                    hydrate_chirality = []
                    for c in range(len(hydrate_tags)):
                        for e in range(len(entry2[a].molecule.components[hydrate_tags[c]].atoms)):
                            if entry2[a].molecule.components[hydrate_tags[c]].atoms[e].is_chiral == True:
                                hydrate_chirality.append(entry2[a].molecule.components[hydrate_tags[c]].atoms[e].chirality)
                    waterless_chirality = []
                    for d in range(len(waterless_tags)):
                        for f in range(len(entry3[a].molecule.components[waterless_tags[d]].atoms)):
                            if entry3[a].molecule.components[waterless_tags[d]].atoms[f].is_ch
```

```
                    iral == True:
373
                        waterless_chirali
                        ty.append(entry3[
                        a].molecule.compo
                        nents[waterless_t
                        ags[d]].atoms[f].
                        chirality)
374
375             if hydrate_chirality == []
                and waterless_chirality !=
                [] or hydrate_chirality !=
                [] and waterless_chirality
                == []:
376                 if DoubleR != 0:
377
                        writer8.write(entry2[
                        a])
378
                        writer9.write(entry3[
                        a])
379                 else:
380
                        chiral_writer.write(e
                        ntry2[a])
381
                        writer1.write(entry3[
                        a])
382                     if DoubleBlank != 0:
383
                            writer4.write(ent
                            ry2[a])
384
                            writer5.write(ent
                            ry3[a])
385                     if RandBlank != 0:
386
                            writer6.write(ent
                            ry2[a])
387
                            writer7.write(ent
                            ry3[a])
388
389             if hydrate_chirality == []
                and waterless_chirality ==
                []:
390                 if DoubleR != 0:
391
                        writer8.write(entry2[
                        a])
392
                        writer9.write(entry3[
                        a])
393                 else:
394
                        chiral_writer.write(e
                        ntry2[a])
395
                        writer1.write(entry3[
                        a])
396
397             if hydrate_chirality != []
                and waterless_chirality !=
                []:
398                 same_molecules = 0
399
                    possibly_the_same_molecul
```

```
                                                                es = 0
                                                                distinct_molecules = 0
                                                                for c in
                                                                range(len(hydrate_tags)):
                                                                    for d in
                                                                    range(len(waterless_t
                                                                    ags)):
                                                                        if
                                                                        entry2[a].molecul
                                                                        e.components[hydr
                                                                        ate_tags[c]].form
                                                                        ula ==
                                                                        entry3[a].molecul
                                                                        e.components[wate
                                                                        rless_tags[d]].fo
                                                                        rmula:

    ordered_hydrate_chiral_centers = []

    ordered_waterless_chiral_centers = []

    atomic_symbols = []
                                                                            for e in
    range(len(entry2[a].molecule.components[hydrate_tags[c]].atoms)):
                                                                                if
    entry2[a].molecule.components[hydrate_tags[c]].atoms[e].atomic_symbol not in
    atomic_symbols:

    atomic_symbols.append(entry2[a].molecule.components[hydrate_tags[c]].atoms[e].atomic_symb
    ol)

    same_elements = [x for x in entry2[a].molecule.components[hydrate_tags[c]].atoms if
    x.atomic_symbol == entry2[a].molecule.components[hydrate_tags[c]].atoms[e].atomic_symbol]

    chiral_centers = []
                                                                                for
    f in range(len(same_elements)):

    if len(same_elements[f].bonds) == 4:

        string1 = str(same_elements[f].bonds)

        string1 = string1.replace(str(same_elements[f]), '')

        if ') )' in string1:

            string1 = string1.replace(') )', '))')

        if '  Atom' in string1:

            string1 = string1.replace('  Atom', ' Atom')

        string1 = re.sub(r'[0-9]+', '', string1)

        capitals = [i for i in range(len(string1)) if string1[i].isupper() == True]

        extra = [j for j in capitals if j-1 in capitals]

        if extra != []:

            if len(extra) == 1:

                string2 = string1[:extra[0]]

                omit_to = string1[extra[0]:].index(')')

                string2 = string2 + string1[extra[0]+omit_to:]
```

```
        else:
            for k in range(len(extra)):
                if k == 0:
                    string2 = string1[:extra[k]]
                    omit_to = string1[extra[k]:].index(')')
                elif k == len(extra)-1:
                    string2 = string2 + string1[extra[k-1]+omit_to:extra[k]]
                    omit_to = string1[extra[k]:].index(')')
                    string2 = string2 + string1[extra[k]+omit_to:]
                else:
                    string2 = string2 + string1[extra[k-1]+omit_to:extra[k]]
                    omit_to = string1[extra[k]:].index(')')
        if string2.count('Atom(H)') <= 1 and string2.count('Atom(D)') == 0 or
string2.count('Atom(H)') == 0 and string2.count('Atom(D)') <= 1:
            chiral_centers.append(same_elements[f])
    else:
        if string1.count('Atom(H)') <= 1 and string1.count('Atom(D)') == 0 or
string1.count('Atom(H)') == 0 and string1.count('Atom(D)') <= 1:
            chiral_centers.append(same_elements[f])
ordered_hydrate_chiral_centers = ordered_hydrate_chiral_centers + chiral_centers
                                                                    start = 0
waterless_matches = []
                                                                    while
len(ordered_hydrate_chiral_centers) != len(ordered_waterless_chiral_centers):
                                                                        if
waterless_matches == []:
symbol_matches = [x for x in entry3[a].molecule.components[waterless_tags[d]].atoms if
x.atomic_symbol == ordered_hydrate_chiral_centers[start].atomic_symbol]
                                                                    for
t in range(len(symbol_matches)):
if len(symbol_matches[t].bonds) == 4:
    string3 = str(symbol_matches[t].bonds)
    string3 = string3.replace(str(symbol_matches[t]), '')
    if ') )' in string3:
        string3 = string3.replace(') )', '))')
    if '  Atom' in string3:
        string3 = string3.replace('  Atom', ' Atom')
    string3 = re.sub(r'[0-9]+', '', string3)
```

```
        capitals = [i for i in range(len(string3)) if string3[i].isupper() == True]
461
        extra = [j for j in capitals if j-1 in capitals]
462
        if extra != []:
463
            if len(extra) == 1:
464
                string4 = string3[:extra[0]]
465
                omit_to = string3[extra[0]:].index(')')
466
                string4 = string4 + string3[extra[0]+omit_to:]
467
            else:
468
                for k in range(len(extra)):
469
                    if k == 0:
470
                        string4 = string3[:extra[k]]
471
                        omit_to = string3[extra[k]:].index(')')
472
                    elif k == len(extra)-1:
473
                        string4 = string4 + string3[extra[k-1]+omit_to:extra[k]]
474
                        omit_to = string3[extra[k]:].index(')')
475
                        string4 = string4 + string3[extra[k]+omit_to:]
476
                    else:
477
                        string4 = string4 + string3[extra[k-1]+omit_to:extra[k]]
478
                        omit_to = string3[extra[k]:].index(')')
479
            if string4.count('Atom(H)') <= 1 and string4.count('Atom(D)') == 0 or
    string4.count('Atom(H)') == 0 and string4.count('Atom(D)') <= 1:
480
                waterless_matches.append(symbol_matches[t])
481
        else:
482
            if string3.count('Atom(H)') <= 1 and string3.count('Atom(D)') == 0 or
    string3.count('Atom(H)') == 0 and string3.count('Atom(D)') <= 1:
483
                waterless_matches.append(symbol_matches[t])
484
    waterless_bonds = []
485
    updated_waterless_bonds = []
486                                                                    for
    f in range(len(waterless_matches)):
487
    pre_waterless_bonds = []
488
    for y in range(len(waterless_matches[f].bonds)):
489
        string2 = str(waterless_matches[f].bonds[y])
490
        string2 = string2.replace(str(waterless_matches[f]), '')
491
        if ') )' in string2:
492
            string2 = string2.replace(') )', '))')
```

```
493
        if '  Atom' in string2:
494
            string2 = string2.replace('  Atom', ' Atom')
495
        string2 = re.sub(r'[0-9]+', '', string2)
496
        pre_waterless_bonds.append(string2)
497
    pre_waterless_bonds = sorted(pre_waterless_bonds)
498
    waterless_bonds.append(pre_waterless_bonds)
499
    updated_waterless_bonds.append(pre_waterless_bonds)
500                                                                    else:
501
    waterless_bonds = list(updated_waterless_bonds)
502                                                                    if
    len(waterless_matches) == 1:
503
    ordered_waterless_chiral_centers.append(waterless_matches[0])
504
    waterless_matches.remove(waterless_matches[0])
505
    start += 1
506                                                                    else:
507
    hydrate_spider_atoms = []
508
    waterless_spider_atoms = []
509
    found_hydrate_atoms = []
510
    hydrate_bonds = []
511                                                                        for
    z in range(len(ordered_hydrate_chiral_centers[start].bonds)):
512
    string1 = str(ordered_hydrate_chiral_centers[start].bonds[z])
513
    string1 = string1.replace(str(ordered_hydrate_chiral_centers[start]), '')
514
    if ') )' in string1:
515
        string1 = string1.replace(') )', '))')
516
    if '  Atom' in string1:
517
        string1 = string1.replace('  Atom', ' Atom')
518
    string1 = re.sub(r'[0-9]+', '', string1)
519
    hydrate_bonds.append(string1)
520
    hydrate_bonds = sorted(hydrate_bonds)
521
    hydrate_neighbours = ordered_hydrate_chiral_centers[start].neighbours
522
    found_hydrate_atoms.append(ordered_hydrate_chiral_centers[start])
523
    numberless_hydrate_neighbours = []
524
    numberless_waterless_neighbours = []
525
    found_match = 0
526
    while found_match == 0:
527
```

```
bond_matches = []
atom_matches = []
if numberless_hydrate_neighbours == []:
    for w in range(len(waterless_bonds)):
        if waterless_bonds[w] == hydrate_bonds:
            bond_matches.append(waterless_bonds[w])
            atom_matches.append(w)
else:
    if hydrate_bonds == [] and all(x == [] for x in waterless_bonds):
        bond_matches.append(waterless_bonds[w])
        atom_matches.append(w)
    else:
        for w in range(len(numberless_waterless_neighbours)):
            off_the_table_waterless = []
            off_the_table_hydrate = []
            match = 0
            for x in range(len(numberless_waterless_neighbours[w])):
                if len(waterless_bonds[w]) == len(hydrate_bonds):
                    if waterless_neighbours[w][x] not in off_the_table_waterless:
                        go = 0
                        for u in range(len(numberless_hydrate_neighbours)):
                            if go == 0:
                                if hydrate_neighbours[u] not in off_the_table_hydrate:
                                    if numberless_waterless_neighbours[w][x] ==
numberless_hydrate_neighbours[u]:
                                        waterless_to_compare = []
                                        hydrate_to_compare = []
                                        if x == 0:
                                            for t in range(waterless_spider_atoms[w][x]):

waterless_to_compare.append(waterless_bonds[w][t])
                                        else:
                                            for t in
range(waterless_spider_atoms[w][x-1], waterless_spider_atoms[w][x]):

waterless_to_compare.append(waterless_bonds[w][t])
```

```
                                        if u == 0:
                                            for s in range(hydrate_spider_atoms[u]):
hydrate_to_compare.append(hydrate_bonds[s])
                                        else:
                                            for s in range(hydrate_spider_atoms[u-1],
hydrate_spider_atoms[u]):
hydrate_to_compare.append(hydrate_bonds[s])
                                        if waterless_to_compare == hydrate_to_compare:
off_the_table_waterless.append(waterless_neighbours[w][x])
off_the_table_hydrate.append(hydrate_neighbours[u])
                                            match += 1
                                            go += 1
                if match == len(numberless_hydrate_neighbours):
                    bond_matches.append(waterless_bonds[w])
                    atom_matches.append(w)

if len(bond_matches) == 1:
    pin2 = waterless_bonds.index(bond_matches[0])
    ordered_waterless_chiral_centers.append(waterless_matches[pin2])
    waterless_matches.remove(waterless_matches[pin2])
    updated_waterless_bonds.remove(updated_waterless_bonds[pin2])
    start += 1
    found_match += 1
else:
    if numberless_hydrate_neighbours != []:
        hydrate_neighbours = list(new_hydrate_neighbours)
        numberless_hydrate_neighbours = []
        hydrate_spider_atoms = []
    for g in range(len(hydrate_neighbours)):
        string1 = str(hydrate_neighbours[g])
        string2 = re.sub(r'[0-9]+', '', string1)
        numberless_hydrate_neighbours.append(string2)
```

```
    new_hydrate_neighbours = []
    hydrate_atom_branches = 0
    for j in range(len(hydrate_neighbours)):
        if hydrate_neighbours[j] not in found_hydrate_atoms:
            for k in range(len(hydrate_neighbours[j].neighbours)):
                if hydrate_neighbours[j].neighbours[k] not in found_hydrate_atoms:
                    new_hydrate_neighbours.append(hydrate_neighbours[j].neighbours[k])
                    hydrate_atom_branches += 1
        hydrate_spider_atoms.append(hydrate_atom_branches)
    hydrate_bonds = []
    for l in range(len(hydrate_neighbours)):
        pre_hydrate_bonds = []
        for z in range(len(hydrate_neighbours[l].bonds)):
            if all(str(x) not in str(hydrate_neighbours[l].bonds[z]) for x in
found_hydrate_atoms):
                string1 = str(hydrate_neighbours[l].bonds[z])
                string1 = string1.replace(str(hydrate_neighbours[l]), '')
                if ') )' in string1:
                    string1 = string1.replace(') )', '))')
                if '  Atom' in string1:
                    string1 = string1.replace('  Atom', ' Atom')
                string1 = re.sub(r'[0-9]+', '', string1)
                pre_hydrate_bonds.append(string1)
        pre_hydrate_bonds = sorted(pre_hydrate_bonds)
        hydrate_bonds = hydrate_bonds + pre_hydrate_bonds
    if numberless_waterless_neighbours != []:
        waterless_neighbours = list(new_waterless_neighbours)
        waterless_neighbours_to_remove = []
        for y in range(len(waterless_neighbours)):
            if y not in atom_matches:
                waterless_neighbours_to_remove.append(waterless_neighbours[y])
        for z in range(len(waterless_neighbours_to_remove)):
            waterless_neighbours.remove(waterless_neighbours_to_remove[z])
        numberless_waterless_neighbours = []
```

```
623
            waterless_spider_atoms = []
624
        else:
625
            found_waterless_atoms = []
626
            waterless_neighbours = []
627
            for y in range(len(waterless_matches)):
628
                waterless_neighbours.append(waterless_matches[y].neighbours)
629
                if str(waterless_matches[y]) != 'Atom(H)':
630
                    found_waterless_atoms.append([waterless_matches[y]])
631
        for f in range(len(waterless_neighbours)):
632
            pre_numberless_waterless_neighbours = []
633
            for e in range(len(waterless_neighbours[f])):
634
                string1 = str(waterless_neighbours[f][e])
635
                string2 = re.sub(r'[0-9]+', '', string1)
636
                pre_numberless_waterless_neighbours.append(string2)
637
            numberless_waterless_neighbours.append(pre_numberless_waterless_neighbours)
638
        new_waterless_neighbours = []
639
        for m in range(len(waterless_neighbours)):
640
            waterless_atom_branches = 0
641
            pre_new_waterless_neighbours = []
642
            pre_waterless_spider_atoms = []
643
            for n in range(len(waterless_neighbours[m])):
644
                if waterless_neighbours[m][n] not in found_waterless_atoms[m]:
645
                    for o in range(len(waterless_neighbours[m][n].neighbours)):
646
                        if waterless_neighbours[m][n].neighbours[o] not in
    found_waterless_atoms[m]:
647

    pre_new_waterless_neighbours.append(waterless_neighbours[m][n].neighbours[o])
648
                            waterless_atom_branches += 1
649
                pre_waterless_spider_atoms.append(waterless_atom_branches)
650
            new_waterless_neighbours.append(pre_new_waterless_neighbours)
651
            waterless_spider_atoms.append(pre_waterless_spider_atoms)
652
        waterless_bonds = []
653
        for p in range(len(waterless_neighbours)):
654
            pre_waterless_bonds = []
655
```

```
656             for q in range(len(waterless_neighbours[p])):
657                 extra_pre_waterless_bonds = []
658                 for z in range(len(waterless_neighbours[p][q].bonds)):
                        if all(str(x) not in str(waterless_neighbours[p][q].bonds[z]) for x in
    found_waterless_atoms[p]):
659                         string2 = str(waterless_neighbours[p][q].bonds[z])
660                         string2 = string2.replace(str(waterless_neighbours[p][q]), '')
661                         if ') )' in string2:
662                             string2 = string2.replace(') )', '))')
663                         if '  Atom' in string2:
664                             string2 = string2.replace('  Atom', ' Atom')
665                         string2 = re.sub(r'[0-9]+', '', string2)
666                         extra_pre_waterless_bonds.append(string2)
667                 extra_pre_waterless_bonds = sorted(extra_pre_waterless_bonds)
668                 pre_waterless_bonds = pre_waterless_bonds + extra_pre_waterless_bonds
669             waterless_bonds.append(pre_waterless_bonds)
670         for j in range(len(hydrate_neighbours)):
671             if str(hydrate_neighbours[j]) != 'Atom(H)':
672                 found_hydrate_atoms.append(hydrate_neighbours[j])
673         for m in range(len(waterless_neighbours)):
674             for n in range(len(waterless_neighbours[m])):
675                 if str(waterless_neighbours[m][n]) != 'Atom(H)':
676                     found_waterless_atoms[m].append(waterless_neighbours[m][n])
677
678         if len(waterless_neighbours) == 0:
679             ordered_waterless_chiral_centers.append(waterless_matches[0])
680             waterless_matches.remove(waterless_matches[0])
681             updated_waterless_bonds.remove(updated_waterless_bonds[0])
682             start += 1
683             found_match += 1
684
685     hydrate_chirality = []
686     waterless_chirality = []
687                                                                 for e in
    range(len(ordered_hydrate_chiral_centers)):
688                                                                     if
```

```
    ordered_hydrate_chiral_centers[e].is_chiral == True:
689
    hydrate_chirality.append(ordered_hydrate_chiral_centers[e].chirality)
690                                                                 else:
691
    hydrate_chirality.append(' ')
692                                                             for f in
    range(len(ordered_waterless_chiral_centers)):
693                                                                 if
    ordered_waterless_chiral_centers[f].is_chiral == True:
694
    waterless_chirality.append(ordered_waterless_chiral_centers[f].chirality)
695                                                                 else:
696
    waterless_chirality.append(' ')
697                                                             if
    hydrate_chirality == waterless_chirality:
698
    same_molecules += 1
699                                                             elif [s for
    s in hydrate_chirality if s != ' '] == [] or [t for t in waterless_chirality if t != '
    '] == []:
700
    possibly_the_same_molecules += 1
701                                                             else:
702
    distinct_molecules += 1
703
704                                                         if distinct_molecules
                                                        == 0:
705                                                             if
                                                            possibly_the_same_mol
                                                            ecules == 0:
706                                                                 if DoubleR != 0:
707
    writer8.write(entry2[a])
708
    writer9.write(entry3[a])
709                                                                 else:
710
    chiral_writer.write(entry2[a])
711
    writer1.write(entry3[a])
712                                                         else:
713                                                             if DoubleR != 0:
714
    writer8.write(entry2[a])
715
    writer9.write(entry3[a])
716                                                             else:
717
    chiral_writer.write(entry2[a])
718
    writer1.write(entry3[a])
719                                                             if
    DoubleBlank != 0:
720
    writer4.write(entry2[a])
721
    writer5.write(entry3[a])
722                                                             if
    RandBlank != 0:
723
    writer6.write(entry2[a])
724
    writer7.write(entry3[a])
```

```
                                                    else:
                                                        writer2.write(entry2[a])
                                                        writer3.write(entry3[a])


                                                loop += 1
                                                print 'another loop bites the dust'

if __name__ == '__main__':
    # This runs the script
    r = Runner()
    r.run()
```

```python
#24 Create the three main classes

from ccdc import io, search
import argparse
import os
import glob
import re

#This script creates the three main classes of structures: (1) hydrate-anhydrate pairs, (2) hydrates without a known anhydrous form, and (3) anhydrous forms without a known hydrate-anhydrate
#It also places hydrates and waterless forms from hydrate-anhydrate pairs that have distinct chirality in classes 2 and 3 if there are no true hydrate-anhydrate pairs remaining for those structures

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\None_solvated_hydrates.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\None_solvated_waterless_forms.txt"
filepath3 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\None_hydrates.txt"
filepath4 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\None_waterless_forms.txt"
filepath5 = "C:\Users\jenwe\Hydrates Manuscript\Step5 Find Paired Hydrates and Waterless Forms\potential_None_hydrates_with_waterless_form.txt"
filepath6 = "C:\Users\jenwe\Hydrates Manuscript\Step5 Find Paired Hydrates and Waterless Forms\potential_None_waterless_forms_with_hydrate.txt"
filepath7 = "C:\Users\jenwe\Hydrates Manuscript\Step5 Find Paired Hydrates and Waterless Forms\potential_hydrates_with_waterless_form.txt"
filepath8 = "C:\Users\jenwe\Hydrates Manuscript\Step5 Find Paired Hydrates and Waterless Forms\potential_waterless_forms_with_hydrate.txt"
filepath9 = "C:\Users\jenwe\Hydrates Manuscript\Step5 Find Paired Hydrates and Waterless Forms\potential_hydrates_without_waterless_form.txt"
filepath10 = "C:\Users\jenwe\Hydrates Manuscript\Step5 Find Paired Hydrates and Waterless Forms\potential_waterless_forms_without_hydrate.txt"
filepath11 = "C:\Users\jenwe\Hydrates Manuscript\Step4 Find Pairs and Duplicates with Distinct Chirality\complete_distinct_chirality_hydrates_in_pairs.txt"
filepath12 = "C:\Users\jenwe\Hydrates Manuscript\Step4 Find Pairs and Duplicates with Distinct Chirality\complete_distinct_chirality_waterless_forms_in_pairs.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='class1_hydrates_with_waterless_forms.gcd',
            help='output file [class1_hydrates_with_waterless_forms.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')
        entry_reader2 = io.EntryReader(filepath2, format='identifiers')
```

```
        entry_reader3 = io.EntryReader(filepath3, format='identifiers')
        entry_reader4 = io.EntryReader(filepath4, format='identifiers')
        entry_reader5 = io.EntryReader(filepath5, format='identifiers')
        entry_reader6 = io.EntryReader(filepath6, format='identifiers')
        entry_reader7 = io.EntryReader(filepath7, format='identifiers')
        entry_reader8 = io.EntryReader(filepath8, format='identifiers')
        entry_reader9 = io.EntryReader(filepath9, format='identifiers')
        entry_reader10 = io.EntryReader(filepath10, format='identifiers')
        entry_reader11 = io.EntryReader(filepath11, format='identifiers')
        entry_reader12 = io.EntryReader(filepath12, format='identifiers')

        with io.EntryWriter(self.args.output) as check_writer:
            with io.EntryWriter("class1_waterless_forms_with_hydrate.gcd") as writer1:
                with
                io.EntryWriter("class2_hydrates_with_no_reported_waterless_form.gcd")
                as writer2:
                    with
                    io.EntryWriter("class3_waterless_forms_with_no_reported_hydrate.gcd")
                     as writer3:

                        duplicate_hash_dictionary = {'GADKOL':1, 'MECYEY':2,
                        'OBASAN':3, 'OMOMAE':4, 'OTONUI':5, 'QOJWAO':6, 'TZBFZO':7,
                        'VOBXOZ':8, 'VUFJAI':9, 'VUFJAJ':10, 'WEPQOA':11, 'XADFUD':12,
                        'YABREY':13, 'YULCOZ':14, 'ZOVFEV':15}

                        #True hydrate-anhydrate pairs are placed in checked lists
                        #False hydrate-anhydrate pairs are placed in false lists
                        checked_HW = []
                        checked_WH = []
                        false_HW = []
                        false_WH = []

                        loop = 0
                        entry1 = entry_reader1
                        list1 = []
                        list = list1
                        #A list of structures from each input file is created
                        while loop < 12:
                            for a in range(len(entry1)):
                                if entry1[a].identifier not in duplicate_hash_dictionary:
                                    list.append(hash(entry1[a].identifier))
                                else:

                                    list.append(duplicate_hash_dictionary.get(entry1[a].i
                                    dentifier))
                            loop += 1
                            if loop == 1:
                                entry1 = entry_reader2
                                list2 = []
                                list = list2
                            if loop == 2:
                                entry1 = entry_reader3
                                list3 = []
                                list = list3
                            if loop == 3:
                                entry1 = entry_reader4
                                list4 = []
                                list = list4
                            if loop == 4:
                                entry1 = entry_reader5
                                list5 = []
                                list = list5
                            if loop == 5:
                                entry1 = entry_reader6
                                list6 = []
                                list = list6
                            if loop == 6:
```

```
                    entry1 = entry_reader7
                    list7 = []
                    list = list7
                if loop == 7:
                    entry1 = entry_reader8
                    list8 = []
                    list = list8
                if loop == 8:
                    entry1 = entry_reader9
                    list9 = []
                    list = list9
                if loop == 9:
                    entry1 = entry_reader10
                    list10 = []
                    list = list10
                if loop == 10:
                    entry1 = entry_reader11
                    list11 = []
                    list = list11
                if loop == 11:
                    entry1 = entry_reader12
                    list12 = []
                    list = list12

            #Hydrates from the potential hydrate-anhydrate pairs list are
            checked for in the distinct chirality list
            for b in range(len(list7)):
                count = 0
                for c in range(len(list11)):
                    #If the hydrate and corresponding anhydrate are one of
                    the pairs in the distinct chirality list they are
                    appended to the false lists
                    if list11[c] == list7[b] and list12[c] == list8[b]:
                        if list7[b] not in false_HW:
                            false_HW.append(list7[b])
                        if list8[b] not in false_WH:
                            false_WH.append(list8[b])
                        count += 1
                #Otherwise, the two structures are written to class 1
                output files
                if count == 0:
                    check_writer.write(entry_reader7[b])
                    if list7[b] not in checked_HW:
                        checked_HW.append(list7[b])
                    writer1.write(entry_reader8[b])
                    if list8[b] not in checked_WH:
                        checked_WH.append(list8[b])

            #Hydrates-anhydrate pairs found using the None method are
            appended to the lists for class 1
            for d in range(len(list5)):
                check_writer.write(entry_reader5[d])
                if list5[d] not in checked_HW:
                    checked_HW.append(list5[d])
                writer1.write(entry_reader6[d])
                if list6[d] not in checked_WH:
                    checked_WH.append(list6[d])

            #Hydrates in the false list that are not in class 1 are written
            to the class 2 output file
            for z in range(len(false_HW)):
                if false_HW[z] not in checked_HW:
                    pin = list7.index(false_HW[z])
                    writer2.write(entry_reader7[pin])

            #Waterless forms in the false list that are not in class 1 are
            written to the class 3 output file
```

```
                        for y in range(len(false_WH)):
                            if false_WH[y] not in checked_WH:
                                pin2 = list8.index(false_WH[y])
                                writer3.write(entry_reader8[pin2])

                        #Hydrates that are not in class 1 and not in the false list are written to the class 2 output file
                        for f in range(len(list1)):
                            if list1[f] not in checked_HW and list1[f] not in false_HW:
                                writer2.write(entry_reader1[f])
                                false_HW.append(list1[f])

                        #Waterless forms that are not in class 1 and not in the false list are written to the class 3 output file
                        for g in range(len(list2)):
                            if list2[g] not in checked_WH and list2[g] not in false_WH:
                                writer3.write(entry_reader2[g])
                                false_WH.append(list2[g])

                        #Hydrates that are not in class 1 and not in the false list are written to the class 2 output file
                        for h in range(len(list3)):
                            if list3[h] not in checked_HW and list3[h] not in false_HW:
                                writer2.write(entry_reader3[h])
                                false_HW.append(list3[h])

                        #Waterless forms that are not in class 1 and not in the false list are written to the class 3 output file
                        for i in range(len(list4)):
                            if list4[i] not in checked_WH and list4[i] not in false_WH:
                                writer3.write(entry_reader4[i])
                                false_WH.append(list4[i])

                        #Hydrates that are not in class 1 and not in the false list are written to the class 2 output file
                        for j in range(len(list9)):
                            if list9[j] not in checked_HW and list9[j] not in false_HW:
                                writer2.write(entry_reader9[j])
                                false_HW.append(list9[j])

                        #Waterless forms that are not in class 1 and not in the false list are written to the class 3 output file
                        for k in range(len(list10)):
                            if list10[k] not in checked_WH and list10[k] not in false_WH:
                                writer3.write(entry_reader10[k])
                                false_WH.append(list10[k])









    if __name__ == '__main__':
        # This runs the script
        r = Runner()
        r.run()
```

```
#25 Find stoichiometrically distinct hydrate-anhydrate pairs

from ccdc import io, search
import argparse
import os
import glob
import re

#This script finds hydrate-anhydrate pairs that have distinct stoichiometry
#This script is the same as the one used to find duplicate matches with distinct stoichiometry

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step6 Change Composition of Three Classes\class1_hydrates_with_waterless_forms.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step6 Change Composition of Three Classes\class1_waterless_forms_with_hydrate.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='unique_stoichiometry_hydrates.gcd',
            help='output file [unique_stoichiometry_hydrates.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')
        entry_reader2 = io.EntryReader(filepath2, format='identifiers')

        total = 0
        count = 0

        with io.EntryWriter(self.args.output) as sample_writer:
            with io.EntryWriter("unique_stoichiometry_waterless_forms.gcd") as writer1:

                entry1 = entry_reader1
                entry2 = entry_reader2
                for a in range(len(entry_reader1)):
                    formulas1 = entry1[a].formula.split(',')
                    water = []
                    for b in range(len(formulas1)):
                        if '(H2 O1)' in formulas1[b] or '(D2 O1)' in formulas1[b] or
                        formulas1[b] == 'H2 O1' or formulas1[b] == 'D2 O1':
                            water.append(formulas1[b])
                        elif 'H' in formulas1[b] and 'D' in formulas1[b]:
                            place = formulas1[b].index('D')
                            place2 = formulas1[b].index('H')
                            if len(str(formulas1[b])) == place+2:
                                if formulas1[b][place2+2] == ' ':
                                    total = int(formulas1[b][place+1]) +
                                    int(formulas1[b][place2+1])
```

```
                                formulas1[b] =
                                formulas1[b].replace(formulas1[b][place2:place2+2],
                                str('H') + str(total))
                                formulas1[b] = formulas1[b][:place-1]
                            else:
                                print 'line 58'
                                print entry_reader1[a].identifier
                        elif formulas1[b][place2+2] == ' ' and
                        formulas1[b][place+2] == ' ':
                            total = int(formulas1[b][place+1]) +
                            int(formulas1[b][place2+1])
                            formulas1[b] =
                            formulas1[b].replace(formulas1[b][place2:place2+2],
                            str('H') + str(total))
                            formulas1[b] = formulas1[b][:place] +
                            formulas1[b][place+3:]
                        elif entry1[a].identifier == 'ROBYEM01' or
                        entry1[a].identifier == 'ELOCIR' or entry1[a].identifier ==
                        'DOMQOL' or entry1[a].identifier == 'VISZAZ' or
                        entry1[a].identifier == 'UNATAE':
                            total = int(formulas1[b][place+1]) +
                            int(formulas1[b][place2+1:place2+3])
                            formulas1[b] =
                            formulas1[b].replace(formulas1[b][place2:place2+3],
                            str('H') + str(total))
                            formulas1[b] = formulas1[b][:place] +
                            formulas1[b][place+3:]
                        elif entry1[a].identifier == 'SIGHOF01':
                            total = int(formulas1[b][place+1:place+3]) +
                            int(formulas1[b][place2+1:place2+3])
                            formulas1[b] =
                            formulas1[b].replace(formulas1[b][place2:place2+3],
                            str('H') + str(total))
                            formulas1[b] = formulas1[b][:place] +
                            formulas1[b][place+3:]
                        elif entry1[a].identifier == 'PEMCOA' or
                        entry1[a].identifier == 'QQQAEJ01':
                            total = int(formulas1[b][place+1:place+3]) +
                            int(formulas1[b][place2+1])
                            formulas1[b] =
                            formulas1[b].replace(formulas1[b][place2:place2+2],
                            str('H') + str(total))
                            formulas1[b] = formulas1[b][:place] +
                            formulas1[b][place+3:]
                        else:
                            print 'line 61'
                            print entry1[a].identifier
                    elif 'D' in formulas1[b]:
                        formulas1[b] = re.sub(r'[D]', 'H', formulas1[b])
                formulas1 = [x for x in formulas1 if x not in water]
                formulas2 = entry2[a].formula.split(',')
                for c in range(len(formulas2)):
                    if 'H' in formulas2[c] and 'D' in formulas2[c]:
                        place = formulas2[c].index('D')
                        place2 = formulas2[c].index('H')
                        if len(str(formulas2[c])) == place+2:
                            if formulas2[c][place2+2] == ' ':
                                total = int(formulas2[c][place+1]) +
                                int(formulas2[c][place2+1])
                                formulas2[c] =
                                formulas2[c].replace(formulas2[c][place2:place2+2],
                                str('H') + str(total))
                            elif entry2[a].identifier == 'ANTMEU04' or
                            entry2[a].identifier == 'HATYUW01' or
                            entry2[a].identifier == 'GASGEM01':
                                total = int(formulas2[c][place+1]) +
                                int(formulas2[c][place2+1:place2+3])
```

```
101                                 formulas2[c] =
                                    formulas2[c].replace(formulas2[c][place2:place2+3],
                                    str('H') + str(total))
102                             else:
103                                 print 'line 78'
104                                 print entry_reader2[a].identifier
105                             formulas2[c] = formulas2[c][:place-1]
106                         elif formulas2[c][place2+2] == ' ' and
                            formulas2[c][place+2] == ' ':
107                             total = int(formulas2[c][place+1]) +
                                int(formulas2[c][place2+1])
108                             formulas2[c] =
                                formulas2[c].replace(formulas2[c][place2:place2+2],
                                str('H') + str(total))
109                             formulas2[c] = formulas2[c][:place] +
                                formulas2[c][place+3:]
110                         elif entry2[a].identifier == 'DIBZAQ03' or
                            entry2[a].identifier == 'DIBZAQ04' or entry2[a].identifier
                            == 'ROBYEM02' or entry2[a].identifier == 'REXFIM01' or
                            entry2[a].identifier == 'GUQXIB01' or entry2[a].identifier
                            == 'LUSLOA01' or entry2[a].identifier == 'MAQWIM09' or
                            entry2[a].identifier == 'MAQWIM10' or entry2[a].identifier
                            == 'MAQWIM11' or entry2[a].identifier == 'MAQWIM12' or
                            entry2[a].identifier == 'MAQWIM14' or entry2[a].identifier
                            == 'MAQWIM06' or entry2[a].identifier == 'MAQWIM07' or
                            entry2[a].identifier == 'MAQWIM08' or entry2[a].identifier
                            == 'MAQWIM13' or entry2[a].identifier == 'OROQET' or
                            entry2[a].identifier == 'ICUXUC01' or entry2[a].identifier
                            == 'NALWUS03' or entry2[a].identifier == 'KOFCUE05' or
                            entry2[a].identifier == 'NEZRUF01' or entry2[a].identifier
                            == 'RUJCII01' or entry2[a].identifier == 'UPIVIA01' or
                            entry2[a].identifier == 'VIWQUO' or entry2[a].identifier ==
                            'GESXOT01' or entry2[a].identifier == 'MPRGOM01' or
                            entry2[a].identifier == 'TUTBUG01' or entry2[a].identifier
                            == 'KOFCUE04':
111                             total = int(formulas2[c][place+1]) +
                                int(formulas2[c][place2+1:place2+3])
112                             formulas2[c] =
                                formulas2[c].replace(formulas2[c][place2:place2+3],
                                str('H') + str(total))
113                             formulas2[c] = formulas2[c][:place] +
                                formulas2[c][place+3:]
114                         elif entry2[a].identifier == 'SIGHOF02' or
                            entry2[a].identifier == 'LEPXOT01':
115                             total = int(formulas2[c][place+1:place+3]) +
                                int(formulas2[c][place2+1:place2+3])
116                             formulas2[c] =
                                formulas2[c].replace(formulas2[c][place2:place2+3],
                                str('H') + str(total))
117                             formulas2[c] = formulas2[c][:place] +
                                formulas2[c][place+3:]
118                         elif entry2[a].identifier == 'YUHHUE02' or
                            entry2[a].identifier == 'NALWUS04' or entry2[a].identifier
                            == 'NALWUS05' or entry2[a].identifier == 'NALWUS06' or
                            entry2[a].identifier == 'NALWUS07':
119                             total = int(formulas2[c][place+1:place+3]) +
                                int(formulas2[c][place2+1])
120                             formulas2[c] =
                                formulas2[c].replace(formulas2[c][place2:place2+2],
                                str('H') + str(total))
121                             formulas2[c] = formulas2[c][:place] +
                                formulas2[c][place+3:]
122                         else:
123                             print 'line 81'
124                             print entry2[a].identifier
125                     elif 'D' in formulas2[c]:
126                         formulas2[c] = re.sub(r'[D]', 'H', formulas2[c])
```

```
                    formulas1 = sorted(formulas1)
                    formulas2 = sorted(formulas2)
                    if len(formulas1) != 1 and len(formulas2) != 1:
                        if formulas1 != formulas2:
                            sample_writer.write(entry1[a])
                            writer1.write(entry2[a])



if __name__ == '__main__':
    # This runs the script
    r = Runner()
    r.run()

```

```
#26 Determine numbers for flowchart

from ccdc import io, search
import argparse
import os
import glob
import re

#This script refines the three classes that were generated previously to contain structures without metals, duplicates, or mismatched stoichiometry
#Duplicates and metals can be removed before finding hydrate-anhydrate pairs as shown in the flowchart of the paper
#Or they can be removed from everything simultaneously at this point which is what was done in this work

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\metals_hydrates.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\metals_waterless_forms.txt"
filepath3 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\hydrates_all_duplicates.txt"
filepath4 = "C:\Users\jenwe\Hydrates Manuscript\Step3 Find Metal and Duplicate Entries\waterless_forms_all_duplicates.txt"
filepath5 = "C:\Users\jenwe\Hydrates Manuscript\Step6 Change Composition of Three Classes\class1_hydrates_with_waterless_forms.txt"
filepath6 = "C:\Users\jenwe\Hydrates Manuscript\Step6 Change Composition of Three Classes\class1_waterless_forms_with_hydrate.txt"
filepath7 = "C:\Users\jenwe\Hydrates Manuscript\Step6 Change Composition of Three Classes\class2_hydrates_with_no_reported_waterless_form.txt"
filepath8 = "C:\Users\jenwe\Hydrates Manuscript\Step6 Change Composition of Three Classes\class3_waterless_forms_with_no_reported_hydrate.txt"
filepath9 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\complete_None_hydrates.txt"
filepath10 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\complete_None_waterless_forms.txt"
filepath11 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\complete_smiles_hydrates.txt"
filepath12 = "C:\Users\jenwe\Hydrates Manuscript\Step2 Find Hydrates and Waterless Forms\complete_smiles_waterless_forms.txt"
filepath13 = "C:\Users\jenwe\Hydrates Manuscript\Step5 Find Paired Hydrates and Waterless Forms\potential_hydrates_with_waterless_form.txt"
filepath14 = "C:\Users\jenwe\Hydrates Manuscript\Step5 Find Paired Hydrates and Waterless Forms\potential_waterless_forms_with_hydrate.txt"
filepath15 = "C:\Users\jenwe\Hydrates Manuscript\Step6 Change Composition of Three Classes\unique_stoichiometry_hydrates.txt"
filepath16 = "C:\Users\jenwe\Hydrates Manuscript\Step6 Change Composition of Three Classes\unique_stoichiometry_waterless_forms.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='chart_class1_hydrates_with_waterless_forms.gcd',
            help='output file [chart_class1_hydrates_with_waterless_forms.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

```

```
        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')
        entry_reader2 = io.EntryReader(filepath2, format='identifiers')
        entry_reader3 = io.EntryReader(filepath3, format='identifiers')
        entry_reader4 = io.EntryReader(filepath4, format='identifiers')
        entry_reader5 = io.EntryReader(filepath5, format='identifiers')
        entry_reader6 = io.EntryReader(filepath6, format='identifiers')
        entry_reader7 = io.EntryReader(filepath7, format='identifiers')
        entry_reader8 = io.EntryReader(filepath8, format='identifiers')
        entry_reader9 = io.EntryReader(filepath9, format='identifiers')
        entry_reader10 = io.EntryReader(filepath10, format='identifiers')
        entry_reader11 = io.EntryReader(filepath11, format='identifiers')
        entry_reader12 = io.EntryReader(filepath12, format='identifiers')
        entry_reader13 = io.EntryReader(filepath13, format='identifiers')
        entry_reader14 = io.EntryReader(filepath14, format='identifiers')
        entry_reader15 = io.EntryReader(filepath15, format='identifiers')
        entry_reader16 = io.EntryReader(filepath16, format='identifiers')

        total = 0
        count = 0

        with io.EntryWriter(self.args.output) as sample_writer:
            with io.EntryWriter("chart_class1_waterless_forms_with_hydrate.gcd") as
            writer1:
                with
                io.EntryWriter("chart_class2_hydrates_without_waterless_forms.gcd") as
                writer2:
                    with
                    io.EntryWriter("chart_class3_waterless_forms_without_hydrate.gcd")
                    as writer3:
                        with io.EntryWriter("chart_complete_None_hydrates.gcd") as
                        writer4:
                            with
                            io.EntryWriter("chart_complete_None_waterless_forms.gcd")
                            as writer5:
                                with
                                io.EntryWriter("chart_complete_smiles_hydrates.gcd") as
                                writer6:
                                    with
                                    io.EntryWriter("chart_complete_smiles_waterless_forms
                                    .gcd") as writer7:
                                        with
                                        io.EntryWriter("chart_potential_pairs_hydrates.gc
                                        d") as writer8:
                                            with
                                            io.EntryWriter("chart_potential_pairs_waterle
                                            ss_forms.gcd") as writer9:

                                                duplicate_identifier_dictionary =
                                                {'GADKOL':1, 'MECYEY':2, 'OBASAN':3,
                                                'OMOMAE':4, 'OTONUI':5, 'QOJWAO':6,
                                                'TZBFZO':7, 'VOBXOZ':8, 'VUFJAI':9,
                                                'VUFJAJ':10, 'WEPQOA':11, 'XADFUD':12,
                                                'YABREY':13, 'YULCOZ':14, 'ZOVFEV':15}

                                                loop = 0
                                                entry1 = entry_reader1
                                                entry2 = entry_reader2
                                                dont_want_hydrates = []
                                                dont_want_waterless_forms = []
                                                #All the hydrate and waterless form
                                                structures with metals are placed into
```

```
                                    lists with a "dont_want" label
94                                  #All the hydrate and waterless form
                                    structures that are duplicates are
                                    placed in the lists with a "dont_want"
                                    label
95                                  while loop < 2:
96                                      for a in range(len(entry1)):
97
                                            dont_want_hydrates.append(hash(en
                                            try1[a].identifier))
98                                      for b in range(len(entry2)):
99                                          if entry2[b].identifier in
                                            duplicate_identifier_dictionary:
100
                                                dont_want_waterless_forms.app
                                                end(duplicate_identifier_dict
                                                ionary.get(entry2[b].identifi
                                                er))
101                                         else:
102
                                                dont_want_waterless_forms.app
                                                end(hash(entry2[b].identifier
                                                ))
103                                     loop += 1
104                                     if loop == 1:
105                                         entry1 = entry_reader3
106                                         entry2 = entry_reader4
107
108                                 print 'lists of metals and duplicates
                                    complete'
109                                 print len(dont_want_hydrates)
110                                 print len(dont_want_waterless_forms)
111
112                                 #Hydrate and waterless forms that do
                                    not have the same stoichiometry are
                                    placed into two lists
113                                 hydrates_not_pair = []
114                                 for c in range(len(entry_reader15)):
115
                                        hydrates_not_pair.append(hash(entry_r
                                        eader15[c].identifier))
116
117                                 waterless_forms_not_pair = []
118                                 for d in range(len(entry_reader16)):
119                                     if entry_reader16[d].identifier not
                                        in duplicate_identifier_dictionary:
120
                                            waterless_forms_not_pair.append(h
                                            ash(entry_reader16[d].identifier)
                                            )
121                                     else:
122
                                            waterless_forms_not_pair.append(d
                                            uplicate_identifier_dictionary.ge
                                            t(entry_reader16[d].identifier))
123
124                                 print 'list of stoichiometrically
                                    distinct complete'
125                                 print len(hydrates_not_pair)
126                                 print len(waterless_forms_not_pair)
127


128                                 loop1 = 0
129                                 entry3 = entry_reader5
130                                 entry4 = entry_reader6
131                                 unpaired_hydrates = []
132                                 unpaired_waterless_forms = []
```

```
paired_hydrates = []
paired_waterless_forms = []
partner_hydrates = []
partner_waterless_forms = []
#Structures with metals and that are duplicates are removed from hydrate and waterless form lists in the flowchart
while loop1 < 5:
    if loop1 == 1 or loop1 == 2 or loop1 == 3:
        for e in range(len(entry3)):
            if hash(entry3[e].identifier) not in dont_want_hydrates:
                if loop1 == 1:

                    writer2.write(entry3[e])

                    unpaired_hydrates.append(hash(entry3[e].identifier))
                if loop1 == 2:

                    writer4.write(entry3[e])
                if loop1 == 3:

                    writer6.write(entry3[e])
        for f in range(len(entry4)):
            if entry4[f].identifier not in duplicate_identifier_dictionary:
                if hash(entry4[f].identifier) not in dont_want_waterless_forms:
                    if loop1 == 1:

                        writer3.write(entry4[f])

                        unpaired_waterless_forms.append(hash(entry4[f].identifier))
                    if loop1 == 2:

                        writer5.write(entry4[f])
                    if loop1 == 3:

                        writer7.write(entry4[f])
            else:
                if duplicate_identifier_dictionary.get(entry4[f].identifier) not in dont_want_waterless_forms:
                    if loop1 == 1:

                        writer3.write(ent
```

```
                                                                    ry4[f])
163
                                                                    unpaired_waterles
                                                                    s_forms.append(du
                                                                    plicate_identifie
                                                                    r_dictionary.get(
                                                                    entry4[f].identif
                                                                    ier))
164                                                             if loop1 == 2:
165
                                                                    writer5.write(ent
                                                                    ry4[f])
166                                                             if loop1 == 3:
167
                                                                    writer7.write(ent
                                                                    ry4[f])
168                                                     #For lists of hydrate-anhydrate
                                                        pairs, pairs that have distinct
                                                        stoichiometry are removed from the
                                                        pairs lists
169                                                     if loop1 == 0 or loop1 == 4:
170                                                         for g in range(len(entry3)):
171                                                             if entry4[g].identifier not
                                                                in
                                                                duplicate_identifier_dictiona
                                                                ry:
172                                                                 count = 0
173                                                                 if
                                                                    hash(entry3[g].identifier
                                                                    ) in hydrates_not_pair:
174                                                                     indices = [i for i,
                                                                        x in
                                                                        enumerate(hydrates_no
                                                                        t_pair) if x ==
                                                                        hash(entry3[g].identi
                                                                        fier)]
175                                                                     for h in
                                                                        range(len(indices)):
176                                                                         if
                                                                            hash(entry4[g].id
                                                                            entifier) ==
                                                                            waterless_forms_n
                                                                            ot_pair[indices[h
                                                                            ]]:
177                                                                             count += 1
178                                                                 if count != 0:
179                                                                     if loop1 == 0:
180
    partner_hydrates.append(entry3[g])
181
    partner_waterless_forms.append(entry4[g])
182                                                             if count == 0:
183                                                                 if
                                                                    hash(entry3[g].identi
                                                                    fier) not in
                                                                    dont_want_hydrates
                                                                    and
                                                                    hash(entry4[g].identi
                                                                    fier) not in
                                                                    dont_want_waterless_f
                                                                    orms:
184                                                                     if loop1 == 0:
185
    sample_writer.write(entry3[g])
186
    writer1.write(entry4[g])
187
```

```
                paired_hydrates.append(hash(entry3[g].identifier))
188
                paired_waterless_forms.append(hash(entry4[g].identifier))
189                                                                 if loop1 == 4:
190
                writer8.write(entry3[g])
191
                writer9.write(entry4[g])
192                                                             if loop1 == 0:
193                                                                 if
                                                                    hash(entry3[g].id
                                                                    entifier) not
                                                                    in
                                                                    dont_want_hydrate
                                                                    s and
                                                                    hash(entry4[g].id
                                                                    entifier) in
                                                                    dont_want_waterle
                                                                    ss_forms:
194
                partner_hydrates.append(entry3[g])
195                                                                 if
                                                                    hash(entry3[g].id
                                                                    entifier) in
                                                                    dont_want_hydrate
                                                                    s and
                                                                    hash(entry4[g].id
                                                                    entifier) not
                                                                    in
                                                                    dont_want_waterle
                                                                    ss_forms:
196
                partner_waterless_forms.append(entry4[g])
197                                                         else:
198                                                             count = 0
199                                                             if
                                                                hash(entry3[g].identifier
                                                                ) in hydrates_not_pair:
200                                                                 indices = [i for i,
                                                                    x in
                                                                    enumerate(hydrates_no
                                                                    t_pair) if x ==
                                                                    hash(entry3[g].identi
                                                                    fier)]
201                                                                 for h in
                                                                    range(len(indices)):
202                                                                     if
                                                                        duplicate_identif
                                                                        ier_dictionary.ge
                                                                        t(entry4[g].ident
                                                                        ifier) ==
                                                                        waterless_forms_n
                                                                        ot_pair[indices[h
                                                                        ]]:
203                                                                         count += 1
204                                                             if count != 0:
205                                                                 if loop1 == 0:
206
                partner_hydrates.append(entry3[g])
207
                partner_waterless_forms.append(entry4[g])
208                                                             if count == 0:
209                                                                 if
                                                                    hash(entry3[g].identi
                                                                    fier) not in
                                                                    dont_want_hydrates
                                                                    and
```

```
                                                                duplicate_identifier_
                                                                dictionary.get(entry4
                                                                [g].identifier) not
                                                                in
                                                                dont_want_waterless_f
                                                                orms:
210                                                                 if loop1 == 0:
211
    sample_writer.write(entry3[g])
212
    writer1.write(entry4[g])
213
    paired_hydrates.append(hash(entry3[g].identifier))
214
    paired_waterless_forms.append(duplicate_identifier_dictionary.get(entry4[g].identifier))
215                                                                 if loop1 == 4:
216
    writer8.write(entry3[g])
217
    writer9.write(entry4[g])
218                                                             if loop1 == 0:
219                                                                 if
                                                                    hash(entry3[g].id
                                                                    entifier) not
                                                                    in
                                                                    dont_want_hydrate
                                                                    s and
                                                                    duplicate_identif
                                                                    ier_dictionary.ge
                                                                    t(entry4[g].ident
                                                                    ifier) in
                                                                    dont_want_waterle
                                                                    ss_forms:
220
    partner_hydrates.append(entry3[g])
221                                                                 if
                                                                    hash(entry3[g].id
                                                                    entifier) in
                                                                    dont_want_hydrate
                                                                    s and
                                                                    duplicate_identif
                                                                    ier_dictionary.ge
                                                                    t(entry4[g].ident
                                                                    ifier) not in
                                                                    dont_want_waterle
                                                                    ss_forms:
222
    partner_waterless_forms.append(entry4[g])
223                                                 loop1 += 1
224                                                 if loop1 == 1:
225                                                     entry3 = entry_reader7
226                                                     entry4 = entry_reader8
227                                                 if loop1 == 2:
228                                                     entry3 = entry_reader9
229                                                     entry4 = entry_reader10
230                                                 if loop1 == 3:
231                                                     entry3 = entry_reader11
232                                                     entry4 = entry_reader12
233                                                 if loop1 == 4:
234                                                     entry3 = entry_reader13
235                                                     entry4 = entry_reader14
236
237                                             print 'first phase of new list
                                                generation complete'
238
239                                             #Here hydrates and waterless forms that
                                                were part of a pair that did not share
```

```
                                        the same stoichiometry are checked for
                                        a second appearance in the list of true
                                        hydrate-anhydrate pairs
240                                     #If these structures do not appear in
                                        the true hydrate-anhydrate pairs list,
                                        then they are written to the true class
                                        2 (hydrate) and class 3 (anhydrate)
                                        lists for structures without a known
                                        partner
241                                     partner_hydrates2 =
                                        list(set(partner_hydrates))
242                                     partner_waterless_forms2 =
                                        list(set(partner_waterless_forms))
243
244                                     for i in range(len(partner_hydrates2)):
245                                         if
                                        hash(partner_hydrates2[i].identifier)
                                         not in paired_hydrates and
                                        hash(partner_hydrates2[i].identifier)
                                         not in unpaired_hydrates:
246
                                                writer2.write(partner_hydrates2[i
                                                ])
247
248                                     for j in
                                        range(len(partner_waterless_forms2)):
249                                         if
                                        partner_waterless_forms2[j].identifie
                                        r not in
                                        duplicate_identifier_dictionary:
250                                             if
                                                hash(partner_waterless_forms2[j].
                                                identifier) not in
                                                paired_waterless_forms and
                                                hash(partner_waterless_forms2[j].
                                                identifier) not in
                                                unpaired_waterless_forms:
251
                                                    writer3.write(partner_waterle
                                                    ss_forms2[j])
252                                         else:
253                                             if
                                                duplicate_identifier_dictionary.g
                                                et(partner_waterless_forms2[j].id
                                                entifier) not in
                                                paired_waterless_forms and
                                                duplicate_identifier_dictionary.g
                                                et(partner_waterless_forms2[j].id
                                                entifier) not in
                                                unpaired_waterless_forms:
254
                                                    writer3.write(partner_waterle
                                                    ss_forms2[j])
255
256                                     print 'second phase of new list
                                        generation complete'
257
258
259
260
261 if __name__ == '__main__':
262     # This runs the script
263     r = Runner()
264     r.run()
265
```

#27 List of hydrate-anhydrate pairs

	Hydrates	Anhydrous Forms
1.	ACEQOQ	ACEQUW
2.	AVEPUN	AVEPEX
3.	BULMEA02	TFACET
4.	CYTOSM04	CYTSIN
5.	CYTOSM04	CYTSIN02
6.	DOGGUD	DOGHAK
7.	ELEVOG	UNOGOT
8.	KAPZUY	JAWXUB
9.	KAQBEL	WEMHOO
10.	KEMBIO	XOHVEX
11.	LUHHUS	LUHHOM
12.	NESKED	NESJUS
13.	NUMVEW01	JEGMIR
14.	OXACDH27	OXALAC02
15.	OXACDH27	OXALAC04
16.	OXACDH28	OXALAC02
17.	OXACDH28	OXALAC04
18.	PECYII	PECYEE
19.	RAVBUL	NAFMEM
20.	SEFGIW	NAFMEM
21.	TANCES	TANDET
22.	TAWJEH	MUBHOG
23.	TUDDAZ	TUDBOL
24.	TUDDED	TUDBUR
25.	VUZYIA	CELSOD
26.	WUVLOP	GEVYOY
27.	XEHPIJ	ESTILE10
28.	BIUHYD01	ZZZLQC01
29.	ABAZUB	ABEDUJ
30.	ABAZUB01	ABEDUJ
31.	ABEBIV	ABEDUJ
32.	ABEBOB	ABEDUJ
33.	ABEBUH	ABEDUJ
34.	ABIMUW	ABINAD
35.	ABIPEJ	ABINUX
36.	ABOQOZ	ABOQIT
37.	ACCAAH	ACEDAC01
38.	ACCAAH	ACEDAC10
39.	ACCAAH	ACEDAC11
40.	ACCAAH	ACEDAC15
41.	ACCAAH	ACEDAC16
42.	ACCAAH	ACEDAC17
43.	ACCAAH	ACEDAC18
44.	ACCAAH	ACEDAC19
45.	ACCAAH	ACEDAC20
46.	ACCAAH	HEWCOE
47.	ACXMAC	CEMHEK
48.	ADAMAU	NORJUA
49.	AFEVOX	HISHEX
50.	AFICAV	AFICEZ
51.	AFUTUS	MODNUP
52.	AGEKEE	AGEKII
53.	AGEQIO	TALVOU
54.	AGOKUD	AGOKOX
55.	AHEREK	JAYPUU
56.	AHEREK	JAYPUU01
57.	AHEREK	JAYPUU03
58.	AHIMUA	AHIMOU
59.	AHOXLH	MOYHAJ
60.	AHOXLH02	MOYHAJ
61.	AJISIW	AJISES
62.	AJUPAW	VILRIS
63.	AJUQEB	RABTIX
64.	AJUWIL	TESZIA
65.	AKATEN	FIDSAN
66.	AKIHIN	AKIHAF
67.	AKIJEK	NAFZEC
68.	ALABEU	DIKSUM
69.	ALUPAA	GAKQUF
70.	AMBZCL	PEYNOZ
71.	AMCHCA	AMMCHC10
72.	AMEQAK	AMEPOX
73.	AMEVET	AMEVAP
74.	AMPCIH01	AMCILL
75.	AMTETZ	EJIQEU
76.	AMTETZ	EJIQEU02
77.	ANDOON05	ANDOON
78.	ANIPET	ANIPIX
79.	ANIPUJ	ANIQAQ
80.	ANORIG	TAFJUH
81.	ANSFON	DAPSUO03
82.	ANSFON	DAPSUO15
83.	APICAE	TATBEX01
84.	AQAROZ	HEGLOV
85.	AQASAM	HEGMAI
86.	AQASEQ	HEGMEM
87.	AQASIU	HEGLUB
88.	AQOMAU	UNOGIN
89.	AQOMAU	UNOGIN02
90.	AQOMAU01	UNOGIN
91.	AQOMAU01	UNOGIN02
92.	AQOMEY	UNOGIN
93.	AQOMEY	UNOGIN02
94.	AQOMEY01	UNOGIN
95.	AQOMEY01	UNOGIN02
96.	ARGHCL10	LARGIN02
97.	ARGIND	TAQBIY
98.	ARIGAK	HUPWEV

99.	ASAXUO	ASAYAV
100.	ASIMOG	ASIMIA
101.	ASIMOG	ASIMIA01
102.	ASPARM	VIKKEG
103.	ATOYIR	LOLYUG
104.	ATOYIR	LOLYUG01
105.	ATZDZM10	ATZTHD10
106.	AVEMUK	KUXPUP
107.	AVEMUK	KUXPUP01
108.	AVEQAU	AVENUL
109.	AVINEZ	AVINID
110.	AVOKIG	AVOJUR
111.	AVOKIG	AVOJUR02
112.	AWUSEQ	AWUSAM
113.	AWUWAR	WAYBAB
114.	AXEGUH	AXEGOB
115.	AXEXIM	AXEXAE
116.	AXEXIM	AXEXAE01
117.	AXEXOS	AXEXAE
118.	AXEXOS	AXEXAE01
119.	AXOBEW	AXOBAS
120.	AXUGEG	AXUGIK
121.	AXUJIM	QIMZOB
122.	AXUJIM	QIMZOB01
123.	AXUJUY	QIMZOB
124.	AXUJUY	QIMZOB01
125.	AYAXOP	AYAXUV
126.	AYILAW	TEZVAW
127.	AYIRUX	ZEDDOB
128.	AYOGIE	IHOQUS
129.	AYOGIE	IHOQUS01
130.	AYUBEC	HICYEZ
131.	AZTHPN	CIPWUT
132.	AZULAJ	SEKNOM
133.	BADGAR	QOQFAF
134.	BAFDUI	BAFFAQ
135.	BAFFIY	BAFFOE
136.	BAFHUL10	BAFJAT10
137.	BAFPAZ	CALDEY
138.	BAFXOV	BAFXIP
139.	BAGJEY10	AZTHYM10
140.	BAHMAB	DOLBIR
141.	BAHMAB	DOLBIR07
142.	BAHMAB	DOLBIR08
143.	BAHMAB	DOLBIR14
144.	BAHMAB	DOLBIR43
145.	BAHMAB	GLYCIN35
146.	BAHMAB	GLYCIN57
147.	BAHMAB	GLYCIN67
148.	BAKJUS	POAZHN
149.	BANAPQ10	BANAQP10
150.	BAPJIL	BAPJEH
151.	BARBAD	BARBAC
152.	BARBAD	BARBAC02
153.	BARBAD02	BARBAC
154.	BARBAD02	BARBAC02
155.	BCDEXD03	WEWTOJ
156.	BCDEXD04	WEWTOJ
157.	BCDEXD05	WEWTOJ
158.	BCLMSN	WOYPAB
159.	BECXEO	TEGHAQ
160.	BEGVAL	CEXQEB
161.	BEKBOJ01	MEYGII01
162.	BENTAC	TANRIK
163.	BEQWID	LEWBOE
164.	BESLUG	BEGDIB01
165.	BEVNUM	DUYSUL
166.	BEVXUX	HOQLAB
167.	BEXMEX	WANJEB
168.	BICMEF	ZASQOZ
169.	BICQUB	FADHEZ
170.	BICQUB01	FADHEZ
171.	BIHNAH	BOGSEW
172.	BIHWAR	UFOLEJ
173.	BIJCOO	GAQQEV
174.	BIJDON03	WUYNUA
175.	BIJDON04	WUYNUA
176.	BIJDON05	WUYNUA
177.	BIJVIZ	BIJSAO
178.	BIKPIV	GENQIB
179.	BIKPIV	GENQIB01
180.	BIMYEB	BIMYAX
181.	BIMYEB	BIMYAX01
182.	BIMYEB	BIMYAX02
183.	BINWEC	PARBAC
184.	BIRMEU	MEBQEQ
185.	BIRMEU	MEBQEQ01
186.	BIRMEU03	MEBQEQ
187.	BIRMEU03	MEBQEQ01
188.	BIUHYD	ZZZLQC01
189.	BIYSOR	BIYSIL
190.	BIZWOY	BIZWIS
191.	BNITRA10	VECGIV
192.	BOBMEM	BOCBAY
193.	BOCNIQ	TICYOT
194.	BOLDOX	LDOPAS03
195.	BOLGAK	ACANLC10
196.	BOLLIZ	RUYFEX01
197.	BOPQAY	COFDUW10
198.	BOPQAY01	COFDUW10
199.	BOQNOM	BOQNIG
200.	BOXGAY	ISINOP
201.	BOYFOK03	ZULQAY
202.	BOYFOK04	ZULQAY

203.	BOZYUM	TIMZAQ
204.	BOZYUM	TIMZAQ01
205.	BRADOM	BIPADO
206.	BUDNEU	TOBHEZ
207.	BUDNEU01	TOBHEZ
208.	BUDPUM	FUHJOI
209.	BUDTUP	BUDTOJ
210.	BUDXII	LABZAP
211.	BUFTAZ	GUFREG
212.	BUGJES01	JEYDEW
213.	BUHSOM	DUPBOG
214.	BULMAW10	TFACET
215.	BUVSEQ01	WEWTOJ
216.	CADVIM	EHIWEZ
217.	CADVUY	YUYMOU
218.	CAFINE	NIWFEE03
219.	CAFINE	NIWFEE05
220.	CAKFAV	BUBTIB
221.	CAPWUL	HEYJOK
222.	CARNID	YUYPAJ
223.	CARNID	YUYPAJ01
224.	CASSEU	CUNRIO
225.	CAVREX	FEDLEG
226.	CAYPIC	HUYWAB
227.	CAZKUK	PATTOU
228.	CECQEJ	CECYIV
229.	CECQIN	CECPIM
230.	CEDLIJ	YUTCEV
231.	CEDLIJ	YUTCEV01
232.	CEDLIJ	YUTCEV02
233.	CEDLIJ	YUTCEV03
234.	CEFMAD	ALOMUK
235.	CEHTAK10	MECWIC
236.	CEHTAK10	MECWIC03
237.	CEJGIK	CEJGAC
238.	CEJGIK	CEJGAC01
239.	CELXEY	ACACAK
240.	CELYEZ	CELYAV
241.	CEMQUG	CEMQOA
242.	CEMQUG	CEMQOA01
243.	CESVII	CESWIJ
244.	CETZUY	PAKMAR
245.	CEYQOO	CEYQII
246.	CIGDON	CIGDIH
247.	CIJMEQ	BOXQUA
248.	CIJYUQ	FEPJOB
249.	CIJYUQ	FEPJOB01
250.	CIKDOQ	MAJRIZ
251.	CIKDOQ	MAJRIZ01
252.	CIKJAH	ZASQOZ
253.	CIMGUA	CIMETD
254.	CIMGUA	CIMETD01
255.	CIMGUA	CIMETD04
256.	CIRDIS	CIRDOY
257.	CIRFEQ	CIPZIM
258.	CITARC	CITRAC10
259.	CIWPIJ	CIWTUZ
260.	CIWQAC	CIWPEF
261.	CIYPUX	KOMTUC
262.	CIZGUN01	RABTIX
263.	CIZQAD	MENMIB
264.	CIZQAD	MENMIB01
265.	CIZQAD	MENMIB02
266.	CLANDH	CLANAC
267.	CLQUON01	HOJLOI
268.	COGQEW	CUCVUS
269.	COKCOV	COKCIP
270.	COMPAW	COMPEA
271.	COMPAW	COMPEA01
272.	CONYIO	VETVOG01
273.	COTVOX	COTVIR
274.	COTZIV	VETVOG01
275.	COVMEF	BARWUM
276.	COVMEF	BARWUM01
277.	COVPIN	UHITOV
278.	COWYAP	COWXUI
279.	COXPAH	YEPJOT
280.	COYFUS	COYFOM
281.	COYGAZ	COYFOM
282.	COYGED	COYFOM
283.	COYGIH	COYFOM
284.	COZVIY	UWIJUH
285.	CREATH	JOHJIB
286.	CREATH	JOHJIB01
287.	CREATH	JOHJIB02
288.	CUJKID	IJUVAL
289.	CUJREG	CUHVUY
290.	CUWKAG	CUWJUZ
291.	CUYWAU	CUYQUI
292.	CUYWAU	CUYQUI01
293.	CYSTAC	CYSTEA
294.	CYSTIN10	CYSTBR
295.	DAFNOO	RIZGEM
296.	DAFRAG	DAFXAM
297.	DAHXOZ	TPCYPO
298.	DAJWOA	DAJWUG
299.	DAMMIP	DAMNAI
300.	DAPRAQ	GAHBAV
301.	DAZLEW	POPDOO
302.	DEDBUJ	OCHTET
303.	DEDBUJ	OCHTET01
304.	DEDBUJ	OCHTET03
305.	DEDNIM	DEDMUX
306.	DEDNIM	DEDMUX01

307.	DEDNIM	DEDMUX02
308.	DEHBUP	NDNHCL10
309.	DEPDEH	DEPDAD
310.	DETHHC20	LIHJAQ
311.	DETHHC20	LIHJAQ01
312.	DEVXEK	DALGON
313.	DEVXEK	DALGON01
314.	DEVXEK	DALGON02
315.	DEVXEK	DALGON03
316.	DEYREF	DOPNED
317.	DGLSLM10	QQQFGD02
318.	DIFPEO	TAJPAW
319.	DIGXOJ	CASNEQ
320.	DILFAF	FIFGOQ
321.	DILFAF	FIFGOQ01
322.	DILNAP	DILNET
323.	DIPFAJ01	JEYDEW
324.	DIPFAJ10	JEYDEW
325.	DIPICA10	AFEBUI
326.	DIPMUK	NILYAI
327.	DITFUJ	DITFOD
328.	DIWNON	PIDJES
329.	DLPROM01	QANRUT
330.	DMAPYC	JUBKAS
331.	DOBLOV	CUVHAC01
332.	DOBLOV	CUVHAC02
333.	DOFTAT	DOFSUM
334.	DOFTAT	DOFSUM01
335.	DOHXII	DOHXAA
336.	DOLRUR	JEDXAR
337.	DOPLIF	DOPLOL
338.	DOPLIF	DOPLOL02
339.	DOPLIF	DOPLOL03
340.	DOPLIF	DOPLOL05
341.	DOXVUH	BOWWIT
342.	DOZYUO	DUBMAQ
343.	DRPRDL	GIXXOB
344.	DRPRDL	GIXXOB01
345.	DUBMEU	DUBMAQ
346.	DUCXEG	TICFUG
347.	DUDGAL	BEWKUM
348.	DUHKEW	BURLOP
349.	DUJPAB	BENPRL
350.	DUJPAB	BENPRL01
351.	DUJPAB	BENPRL02
352.	DUKTOS	CAPTAZ
353.	DUKWIQ	MEBQEQ
354.	DUKWIQ	MEBQEQ01
355.	DULKAX	OMEJUN
356.	DUNDAT	UGOVIX
357.	DUNDAT	UGOVIX01
358.	DUSHIJ	VUCXEX
359.	DUSQOY	POFNOO
360.	DUTLEJ10	JAXGUK
361.	DUZHIP	YERVAS
362.	EACLTH10	HIVROT
363.	EACLTH10	HIVROT01
364.	ECEFOJ	YEZLUN
365.	ECORAS	ECOQUL
366.	EDAJEA	MEQQEE
367.	EDIDUQ	QIMZOB
368.	EDIDUQ	QIMZOB01
369.	EDOKOX	RIZFEL
370.	EDUHIU	EDUGOZ
371.	EFASOV	AFEDIY
372.	EFIBUR	EFICAY
373.	EFIDUS01	UDAYUT
374.	EFIDUS01	UDAYUT01
375.	EFIDUS01	UDAYUT02
376.	EFIDUS02	UDAYUT
377.	EFIDUS02	UDAYUT01
378.	EFIDUS02	UDAYUT02
379.	EFIDUS03	UDAYUT
380.	EFIDUS03	UDAYUT01
381.	EFIDUS03	UDAYUT02
382.	EFIFOO01	KETXIR
383.	EFOBIL	EFOBEH
384.	EFOCOS	EFOCIM
385.	EGOBAF	EGUHIZ
386.	EGUTUW	EDUGUF
387.	EJEVIA	RAKSIG
388.	EJUPUU	DORBES
389.	ELASUG	ELATAN
390.	ELASUG	ELATAN01
391.	ELOJUM	ELOJOG
392.	ELOJUM	ELOJOG01
393.	ELOKIB	ELOKAT
394.	ELOKIB	ELOKEX
395.	ENEBUW	UGIVAI
396.	ENEBUW	UGIVAI01
397.	ENECAD	UGIVAI
398.	ENECAD	UGIVAI01
399.	ENODUH	UHITOV
400.	ENODUH01	UHITOV
401.	ENUJED	HOFQID
402.	EPHEDH04	EPHEDR01
403.	EPOCET	FEGCOM
404.	EQOTAG	FEPGUE
405.	EQOTUA	RUHTUL02
406.	EQOVEM	RUHTOF02
407.	EQOVOW	RUHTIZ02
408.	EQOWAJ	WOQREB
409.	EQOWAJ	WOQRIF
410.	ERUHUV	VIFSAF

411. ETAMCM HIVROT
412. ETAMCM HIVROT01
413. ETOHOM AWUNAI
414. EVAGIT EPACAB
415. EVAVIH BBENAN
416. EVIXIQ RIVKAJ
417. EVODOI EVODUO
418. EVUQAO BOBVIY
419. EVUQAO BOBVIY04
420. EVUQAO BOBVIY05
421. EVUQAO BOBVIY06
422. EVUQAO BOBVIY07
423. EVUQAO BOBVIY08
424. EVUQAO BOBVIY09
425. EVUQAO BOBVIY13
426. EVUQAO BOBVIY14
427. EVUQAO BOBVIY15
428. EVUQAO BOBVIY16
429. EVUQAO01 BOBVIY
430. EVUQAO01 BOBVIY04
431. EVUQAO01 BOBVIY05
432. EVUQAO01 BOBVIY06
433. EVUQAO01 BOBVIY07
434. EVUQAO01 BOBVIY08
435. EVUQAO01 BOBVIY09
436. EVUQAO01 BOBVIY13
437. EVUQAO01 BOBVIY14
438. EVUQAO01 BOBVIY15
439. EVUQAO01 BOBVIY16
440. EWAMIZ EWAMAR
441. EWAMIZ EWAMAR01
442. EWAQEZ EWAQAV
443. EWAYAE EWAYEI
444. EWIVOX VOKWAT
445. EYELAX ANISAC
446. EYELAX ANISAC03
447. EYELAX ANISAC04
448. EYIKUU FIFGOQ
449. EYIKUU FIFGOQ01
450. EYOGIJ YULJUM
451. EYUQAS CARPEQ
452. EZOBUQ EZOCAX
453. FADYIV FADYER
454. FAFPEJ YIXRON
455. FAJSOY FAJSIS
456. FAMMOY FAMPOB
457. FAYVUW FAYVOQ
458. FAYWUX HEPOPH01
459. FAYWUX HEPOPH02
460. FEDBUL FEDBOF
461. FEDNEI FEDNIM
462. FEFBEX NAFZEC
463. FEFNOT02 CBMZPN01
464. FEFNOT02 CBMZPN03
465. FEFNOT02 CBMZPN11
466. FEFNOT02 CBMZPN12
467. FEFNOT02 CBMZPN16
468. FEFNOT02 CBMZPN20
469. FEFRIS PUBMUU
470. FEFRIS PUBMUU01
471. FEFRIS PUBMUU02
472. FEFRIS PUBMUU23
473. FEFYEW FEFYAS
474. FEHMAJ IZIVUK
475. FEJBAA JOFWIM
476. FEJBAA JOFWIM01
477. FEJBAA JOFWIM03
478. FELCIJ FELCOP
479. FELVIE TEDYOT
480. FELWEB TEDYOT
481. FEPBAF FENZUV
482. FEPFOX WOQBAF
483. FEPFOX WOQBAF02
484. FETXEI IKASUK
485. FEWMID OJOQOV01
486. FEZHAV WICSUX
487. FIBTAM FIBTEQ
488. FICZAU BEJNEM
489. FIFFUV ACPRET03
490. FIFWAV FIFVOI
491. FIFWEZ FIFVOI
492. FILBEH ROMMEM
493. FIMVOM FESRIE
494. FITYAI SAGXOP
495. FITYAI SAGXOP01
496. FITYAI SAGXOP03
497. FIZVUF DUMRAE
498. FIZVUF DUMRAF
499. FOMPUT FOMPON
500. FONBUG TUWYOB
501. FONHEW BTCOAC
502. FOSLUW ZOVQEI
503. FOSPUZ UZIROM
504. FOWMUA HIMCAJ
505. FOWTOB HUCVEH
506. FOYTOC YEHROS
507. FURGAA01 JICLAI
508. FURGAA04 JICLAI
509. FUSWIB EFUMAU
510. FUSWIB EFUMAU03
511. FUSWIB EFUMAU06
512. FUSWOH EFUMAU
513. FUSWOH EFUMAU03
514. FUSWOH EFUMAU06

515.	GADYUI	GADXUH
516.	GAFVIS	VUXBAR
517.	GAFVOY	VUXBAR
518.	GAHJIL	GAHJEH
519.	GAJMIN	GAJMOT
520.	GANCUV	GANDAC
521.	GAQYON	ALETUG
522.	GAQYON	ALETUG01
523.	GASNIY	GASNEU
524.	GAVWUW	GAWSED
525.	GAXTII	USIWEZ
526.	GEDYUM	IJAQIV
527.	GEDYUM	IJAQIV01
528.	GEDYUM	IJAQIV02
529.	GEHRUH	DAMGEF
530.	GEHXAV	GEHWUO
531.	GEJQAO	URICAC
532.	GEMBEF	POSVEA
533.	GEMLAN	GEMKUG
534.	GEQXUX	GEQXOR
535.	GEQXUX	GEQXOR01
536.	GERWUX	GERXEI
537.	GERXAE	GERXEI
538.	GESCIS	GESBEN
539.	GESCOY	VIDMAX
540.	GESCOY	VIDMAX02
541.	GESHET	GESGAO
542.	GESKUK	GESKOE
543.	GETLUN	TIZVEE
544.	GETLUN	TIZVEE01
545.	GEVBIT	CASQUJ
546.	GEVNED	GEVNON
547.	GEVNIH	GEVNON
548.	GEXXAI	HOJLOI
549.	GEYVIS	GEYTIQ
550.	GICXOF	EQAHEL
551.	GIHJOX	DAZXOS
552.	GIMSIF	GIMSOL
553.	GITVAI	PUJZEA
554.	GIVCEU	GIVCAQ
555.	GIVYAL	GIVXUE
556.	GIXDAS	ETDIAM01
557.	GIXDAS	ETDIAM15
558.	GIXDAS	ETDIAM18
559.	GIXDEW	HEXMDA
560.	GIXDIA	QATVUC
561.	GIXDOG	ROKZOG
562.	GIXDUM	QATWAJ
563.	GIZMAE	GIXXOB
564.	GIZMAE	GIXXOB01
565.	GLUCMH	GLUCSA
566.	GLUCMH	GLUCSA20
567.	GOBMAN	YUPHUO
568.	GOFWOP	QQQAUJ03
569.	GOFWOP	QQQAUJ04
570.	GOFWOP	QQQAUJ07
571.	GOFWUV	QQQAUJ03
572.	GOFWUV	QQQAUJ04
573.	GOFWUV	QQQAUJ07
574.	GORFIC	ROBXEN
575.	GOWCAW	GOWBOJ
576.	GUBLAS	RIFBAL
577.	GUBLAS	RIFBAL01
578.	GUCNOJ	GUCNUP
579.	GUHLEC	FOWFOM
580.	GULTEM	DECGPY10
581.	GULTOW	LAKVOI
582.	GUMJIJ	NAGVAS
583.	GUSHAD01	SLFNMB01
584.	GUSHAD01	SLFNMB02
585.	GUSHAD01	SLFNMB05
586.	GUSHAD01	SLFNMB06
587.	GUXZEG	RABTIX
588.	GUYGUD01	GUYGOX03
589.	HABJUP	PAQUCL
590.	HABJUP01	PAQUCL
591.	HAFNUX	HAFNOR
592.	HAJWAR	HAJWIZ
593.	HAKDUT	HAKDON
594.	HATKET	ROMMEM
595.	HATQAX	MEQQOO
596.	HAXBUD	AZAXEG
597.	HAXBUD	LABJON01
598.	HAXBUD01	AZAXEG
599.	HAXBUD01	LABJON01
600.	HAXQAA	BADVIL10
601.	HBARBT	ALXANM01
602.	HEBZEW	HEBZAS
603.	HEBZEW	HEBZAS01
604.	HECMUX	VUNYUY10
605.	HECMUX	VUNYUY11
606.	HECNAE	VUNZEJ10
607.	HEPDUE	NOTEST01
608.	HEPNAR	ZUTMAC
609.	HESWIM	HEQMUM
610.	HESYAF	HESXUY
611.	HETHAQ	HETGUJ
612.	HEZWAK	WAGLUM
613.	HIFVEY	TUFGEG
614.	HIKGOZ	BILCUW
615.	HIKGOZ	BILCUW01
616.	HIKNIY	BEURID10
617.	HIKNIY	LYFURA
618.	HILXEG	BORSEG

619.	HILXEG	BORSEG01
620.	HIMDOY	HIMDAK
621.	HINFAN	VAGYEI
622.	HIPKAS	ALITOL
623.	HIPKAS	GLUCIT
624.	HIPKAS	GLUCIT02
625.	HIPKAS	GLUCIT03
626.	HIRLUQ	PEXXEW
627.	HIRZUF	UNOGIN
628.	HIRZUF	UNOGIN02
629.	HIYJII	HIYJEE
630.	HODHUE	MIYDII
631.	HODRUO	JELSEY
632.	HOFNAR	RUZWUE
633.	HOGCOX	HOGCIR
634.	HOMPRO10	QAMVAB
635.	HONXUD	REVZAT
636.	HOQHAW	XERBEB
637.	HOSHEC	NANRAV
638.	HOSHIG	NANRAV
639.	HOTYIZ	GIDRAN
640.	HOTYUL	GIDRAN
641.	HOWPEP	WAMFAS
642.	HOWPEP	WAMFAS01
643.	HOWPEP	WAMFAS02
644.	HOWPEP	WAMFAS04
645.	HOXGEG	HOXGIK
646.	HUCLIA	SAGMEU
647.	HUCSUV	CURFUR
648.	HUFLUQ	XAHHOG
649.	HUHWEM	TIJLUT
650.	HULKUU	AVURAK
651.	HUMJEE	COTZAN06
652.	HUMJEE	HXACAN
653.	HUMJEE	HXACAN29
654.	HUMJEE	HXACAN39
655.	HURMUC	HURMOW
656.	HUSVEX	HUSVIB
657.	HUSVEX	HUSVIB01
658.	HUVGUC	HUVHAJ
659.	HXBIUR10	HBIURT10
660.	HXMTHH	HXMTAM
661.	IBALOP	UBACOS
662.	IBOVED	IBOHIT
663.	IBUJOH	EWAYOS
664.	IBUXIN	UJEMEB
665.	ICEGAZ	SITCEE
666.	ICOSUR	AJISES01
667.	IDIWOI	SAZBIF
668.	IDUCUF	IDUDAM
669.	IDUYEM	MUYSUW
670.	IFAXOE	FAKDUS
671.	IFILIT	IFILAL
672.	IFILIT	IFILAL01
673.	IFILIT	IFILAL02
674.	IFIZIG	VENLUW
675.	IFIZIG04	VENLUW
676.	IFURIL	YOMDUA
677.	IGALIN	VACHAK
678.	IGATUH	HACVAL
679.	IHAQEO	IHAPOX
680.	IHAQEO	IHAPOX01
681.	IHAQEO	IHAPOX02
682.	IHATAN	XIPYIF
683.	IHATAN	XIPYIF01
684.	IHEGOT	IHEGIN
685.	IHOTUW	COKBIN
686.	IHUQEJ	IHURUA
687.	IJEQET	LASPRT
688.	IJESIZ	IJESAR
689.	IJESIZ01	IJESAR
690.	IJILET	XOQGIV
691.	IJISEB	IJISAX
692.	IJUKAA	SUZTAJ
693.	IKALAJ	IKAKOW
694.	IKEDIM	SAGYAC
695.	IKEXOM	VOLZEC
696.	IKEXOM	VOLZEC01
697.	IKUROV	IKURUB
698.	ILEJUG	PUBMUU
699.	ILEJUG	PUBMUU01
700.	ILEJUG	PUBMUU02
701.	ILEJUG	PUBMUU23
702.	ILINEW	GOLMIF
703.	IMEGIR	NELPUP
704.	IMOPIJ	IMOPEF
705.	INEFEM	ROMMEM
706.	INOSND01	INOSIN10
707.	INOSND01	INOSIN11
708.	IPEHUH	ITESIK
709.	IQUICM	IQUINC
710.	IRELEX	YEGGIA
711.	IRERUU	IRETAC
712.	IRIZAL	FARRUM
713.	IRIZAL	FARRUM03
714.	ISEKEW	RUYFEX01
715.	ISUSUK	UQUDOB
716.	IVEGIA	UZOVAH03
717.	IVOSAP	IVOQUH
718.	IVUQIZ	IVUQOF
719.	IVUQIZ	IVUQOF01
720.	IVUQIZ	IVUQOF02
721.	IVUQIZ	IVUQOF03
722.	IVUQIZ	IVUQOF04

723.	IVUSAU	IVURUN
724.	IWUDUZ	IFOXIM
725.	IXISAL02	IXIRUE02
726.	IYUXOQ	LUYQAX
727.	IZAPUU	YASTUI
728.	JAGXOE	HIGTEZ
729.	JARGUH	SENJEC
730.	JASSAY	VEVJOW
731.	JATDUD	JATFAL
732.	JATJAP	JATHUH
733.	JAYBOD	MOKBAR
734.	JAYBOD	MOKBAR01
735.	JEBJIM	JEBJOS
736.	JEBREO	CAQZOL
737.	JEDKIN	GASNEU
738.	JEDKIN01	GASNEU
739.	JEDKIN02	GASNEU
740.	JEDTOB	TPEPHO
741.	JEDTOB	TPEPHO01
742.	JEDTOB	TPEPHO07
743.	JEDTOB	TPEPHO12
744.	JEDTOB01	TPEPHO
745.	JEDTOB01	TPEPHO01
746.	JEDTOB01	TPEPHO07
747.	JEDTOB01	TPEPHO12
748.	JEDTOB02	TPEPHO
749.	JEDTOB02	TPEPHO01
750.	JEDTOB02	TPEPHO07
751.	JEDTOB02	TPEPHO12
752.	JEHRAP	KAHLEK
753.	JEHRAP	KAHLEK01
754.	JEJXAX	YIRZOP
755.	JEKGIP	JEKGEL
756.	JEMROL	KULBIE
757.	JEPJOD	BOBZAC
758.	JEPJOD	BOBZAC02
759.	JERRUU	JERROO
760.	JETJEY	VEYSEY
761.	JEVQUX	TOYSUX
762.	JEXZER	YADSOL
763.	JEYZIX	BOMDUC
764.	JIDPUH	WETFAD
765.	JIDPUH	WETFAD01
766.	JIWPIP	JIWPEL
767.	JIXCOI	VIDMAX
768.	JIXCOI	VIDMAX02
769.	JIYWET	JOWWIB
770.	JOGMUP	LOFFET
771.	JOGMUP01	LOFFET
772.	JONNIL	JONNEH
773.	JULBOH	HEZZER
774.	JUXYIM	HEQMUM
775.	KADBAU	BIPTID01
776.	KAKHEL	KAKHIP
777.	KALZIG	SUNKAO01
778.	KAMSEZ	KAMROI
779.	KAMSEZ	KAMROI01
780.	KAMSEZ01	KAMROI
781.	KAMSEZ01	KAMROI01
782.	KAPPUP	KAPPOJ
783.	KARGUI	KARFIV
784.	KASHAP	KASGIW
785.	KATSEG	KATSAC
786.	KAVFUL	KAVGAS
787.	KEFHUA	RAKSAZ
788.	KEKHOY	KEKHIS
789.	KEQYOX	KEQYIR
790.	KEXZAQ	KEXZEU
791.	KEXZIY	KEXZEU
792.	KEXZUK	KEXZEU
793.	KEYBAT	KEXZEU
794.	KEYBOH	KEXZEU
795.	KEYTOX	KEYSOW
796.	KICCOO	WEMWEQ
797.	KIFQEW	CEKHAC
798.	KIGMET	KIGMAP
799.	KIJFOB	ALKINA
800.	KIJFOB	ALKINA01
801.	KIKCIR	OTOLUC
802.	KIMDER	KIMDAN
803.	KIPPUU	TEKBUI
804.	KITMIL	AMUQOQ
805.	KITMIL	AMUQOQ01
806.	KOFCAL	GAPMAN
807.	KOGPIG	UXONAY
808.	KOJZUE	LICKUG
809.	KOMNIL	XINVIA
810.	KONTIQ	IJUMEG
811.	KONTIQ	IJUMEG04
812.	KONTIQ	IJUMEG06
813.	KONTIQ01	IJUMEG
814.	KONTIQ01	IJUMEG04
815.	KONTIQ01	IJUMEG06
816.	KONTIQ03	IJUMEG
817.	KONTIQ03	IJUMEG04
818.	KONTIQ03	IJUMEG06
819.	KONTIQ04	IJUMEG
820.	KONTIQ04	IJUMEG04
821.	KONTIQ04	IJUMEG06
822.	KONTIQ06	IJUMEG
823.	KONTIQ06	IJUMEG04
824.	KONTIQ06	IJUMEG06
825.	KOPDOK	KOPDAW
826.	KOPDOK	KOPDAW01

827.	KOPDOK	KOPDAW02
828.	KOPDOK	KOPDAW03
829.	KOPZEW	KOPZAS
830.	KUCSIM	KUCSAE
831.	KUCSIM	KUCSAE01
832.	KUGKAZ	EVESIJ04
833.	KUQTAS	PENDAM
834.	KUSYON	SEBBEG
835.	KUTSAU	DODAMB
836.	KUVBEI	ELLAGC
837.	KUVYUW	AJEYAQ
838.	KUVYUW	AJEYAQ01
839.	KUVYUW	AJEYAQ02
840.	KUVYUW	AJEYAQ03
841.	KUVYUW	AJEYAQ04
842.	KUZKOG	NODTIJ
843.	LACFON	LACGII
844.	LACFON01	LACGII
845.	LACTOS01	EYOCUQ
846.	LACTOS01	EYOCUQ01
847.	LACTOS03	EYOCUQ
848.	LACTOS03	EYOCUQ01
849.	LAFDIH	FIFVOI
850.	LAHQAO	SIYZIK
851.	LAJGUA	PIWXEY
852.	LAKTUM	GULTIQ
853.	LAPVEE	LAPVAA
854.	LAPWOQ	MUNCOO
855.	LAQSON	NAVSUY
856.	LAQSON	NAVSUY02
857.	LAQSON01	NAVSUY
858.	LAQSON01	NAVSUY02
859.	LARWIN	ZORTAB
860.	LASYOV	LASYUB
861.	LASZEM	LASZAI
862.	LATBOZ	DAVPEW
863.	LATPIH	UQUDOB
864.	LAXDIA	WACGEO
865.	LAYPEI	LAYPAE
866.	LCYSCC	CYSTCL
867.	LCYSCC	CYSTCL01
868.	LEBJUX	VATSAK
869.	LEBJUX	VATSAK01
870.	LEBKEI	VATSAK
871.	LEBKEI	VATSAK01
872.	LEBLEK	LEBLAG
873.	LEFSUM	BODVAS
874.	LEGZII	QAMSOO
875.	LEJLIX	TEGFIW
876.	LEJYOP	GOKNOK
877.	LETGAV	LETFUO
878.	LEXPUA	PUHCEB
879.	LEXPUA01	PUHCEB
880.	LEYKAE	PEWYUO
881.	LEYKAE	PEWYUO01
882.	LEYKAE	PEWYUO02
883.	LEYZOG	IFUPAB
884.	LEZJEF	LEZJAB
885.	LEZJEF	LEZJAB01
886.	LEZKAC	FIQFER
887.	LIBPUK	LIBPOE
888.	LIBWUQ	LUYWOR
889.	LIDNOE	LIDNUK
890.	LIFNOE	BISMEV
891.	LIFNOE	BISMEV01
892.	LIFNOE	BISMEV03
893.	LIFNOE	BISMEV04
894.	LIFNOE	BISMEV07
895.	LIFNOE	BISMEV14
896.	LIGXAC	RUKHAG
897.	LIGXAC	RUKHAG01
898.	LIKKUN	LIKLAU
899.	LIPZOA	NINSOS
900.	LIZWOI	CIRNEY
901.	LOBSOL	COTXUG
902.	LOBSOL01	COTXUG
903.	LOCCAI	YIVGIV
904.	LOCCAI	YIVGIV01
905.	LOCCAI	YIVGIV02
906.	LOCNEX	FILGEM
907.	LOCNEX	FILGEM01
908.	LOCNEX	FILGEM03
909.	LOFKOG	REWRUH
910.	LOFTEH	LOFTIL
911.	LOJJIE	ZEXLUJ
912.	LOJJIE	ZEXLUJ02
913.	LOLDEW	LOLDAS
914.	LONGUR	NAVGAT
915.	LONGUR	NAVGAT01
916.	LONKIK	PUBHOK
917.	LOPTOB	KUJDEA
918.	LORTUI	ZIFQEM
919.	LOXSUC10	YIKNUD
920.	LOZREZ	PUBTIQ
921.	LOZROJ	LOZRID
922.	LSERMH10	LSERIN01
923.	LSERMH10	LSERIN16
924.	LSERMH10	LSERIN21
925.	LSERMH10	LSERIN50
926.	LSERMH16	LSERIN01
927.	LSERMH16	LSERIN16
928.	LSERMH16	LSERIN21
929.	LSERMH16	LSERIN50
930.	LUDXOX	LUDYEO

931.	LUJGEE	LUJFUT
932.	LUKTER	LUKTAN
933.	LUKXIA	BZPPOS
934.	LUPAND	LPNECL
935.	LYSCLH	DLLYSC10
936.	MAFSUI	ALITOL
937.	MAFSUI	DMANTL
938.	MAFSUI	DMANTL01
939.	MAFSUI	DMANTL10
940.	MAFSUI	GALACT
941.	MAHDOP	LACJEE
942.	MAHKOX	GEPNOG
943.	MAHKOX	OHIVAE
944.	MAJQIY	AHIHAC
945.	MAQYOS	XIJMOS
946.	MARLUN	MARLOH
947.	MAVPUX	MAWDUM
948.	MAWMEF	MAWMIJ
949.	MAWMEF	MAWMIJ01
950.	MAWMEF	MAWMIJ03
951.	MCYTIM10	RADKOX01
952.	MEBQUG	MEBQEQ
953.	MEBQUG	MEBQEQ01
954.	MECLOV	LIFTIF
955.	MECXOJ	MECXID
956.	MECYUQ	FOYSUI
957.	MEDMAL	MEDLUE
958.	MEDMAL	MEDLUE01
959.	MEKPAW	MEKNOI
960.	MELFUF	MELFIT
961.	MELFUF	MELFIT01
962.	MELFUF	MELFIT02
963.	MELFUF	MELFIT04
964.	MELFUF	MELFIT05
965.	MELFUF	MELFIT06
966.	MELFUF	MELFIT07
967.	MELFUF	MELFIT09
968.	MELFUF	MELFIT19
969.	MELFUF01	MELFIT
970.	MELFUF01	MELFIT01
971.	MELFUF01	MELFIT02
972.	MELFUF01	MELFIT04
973.	MELFUF01	MELFIT05
974.	MELFUF01	MELFIT06
975.	MELFUF01	MELFIT07
976.	MELFUF01	MELFIT09
977.	MELFUF01	MELFIT19
978.	MENHIX	BELPAL
979.	MEPYRZ	MPYRAZ
980.	MEXJOO	MEXJII
981.	MEYRAJ	MEYQUC
982.	MHEXDO10	DMHXDL
983.	MICFEM	MICFAI
984.	MIFVEC	RIBWAB
985.	MIQXUH	JACYAQ
986.	MISMOR	MISMIL
987.	MISRIQ	MISREM
988.	MITCDH01	MITOMC
989.	MIVNIO	TAQGEY
990.	MIWJAE	YIZCOZ
991.	MIXTOD	HOFGAK
992.	MIYJUZ	CUBBIL
993.	MIZHAF	RABDAV
994.	MODMAV	MODLUO
995.	MOHCUI	MOPYEV
996.	MOJZUI	TOZKEA
997.	MOKRUA	XEBYUA
998.	MOKXIU	CIRDOY
999.	MOPSER	LERKUQ
1000.	MOPZOH	GAVVEF
1001.	MORPHC	EFASAH
1002.	MORPHM	MORPIN01
1003.	MOSPAM	VAXHEI
1004.	MOSWIB	FOSRAI
1005.	MOVTIB	EHOWIH
1006.	MOVTIB	EHOWIH02
1007.	MOVTIB	EHOWIH03
1008.	MOVTIB	EHOWIH04
1009.	MOVTIB	EHOWIH05
1010.	MOVTIB02	EHOWIH
1011.	MOVTIB02	EHOWIH02
1012.	MOVTIB02	EHOWIH03
1013.	MOVTIB02	EHOWIH04
1014.	MOVTIB02	EHOWIH05
1015.	MOZWOO	MOSXOI
1016.	MPIPCA	MPIPAN
1017.	MSULIM	POMDAW
1018.	MSULIM	POMDAW01
1019.	MUBPOP	NEQVOV
1020.	MUBPOP	NEQVOV01
1021.	MUDDUK	DLSERN
1022.	MUDDUK	DLSERN19
1023.	MUFNEG	IDIMAJ
1024.	MUKYOI	BOBHOP
1025.	MUSCOU	DUHJIB
1026.	MUSCUA	DUHJIB
1027.	MUWLIA	VULGUF
1028.	MUYTOQ	HTRYPT10
1029.	MYTOLD	EFURIH
1030.	MYTOLD	EFURIH02
1031.	MYTOLD	EPINOS
1032.	MYTOLD	FOPKOK
1033.	MYTOLD	IFAKAC
1034.	MYTOLD	MUINOS

1035.	MYTOLD	MYINOL
1036.	MYTOLD	MYINOL01
1037.	MYTOLD	MYINOL03
1038.	MYTOLD	QIKZAN
1039.	MYTOLD	QIKZAN01
1040.	MYTOLD	QIKZAN02
1041.	MYTOLD	QIKZUH
1042.	MYTOLD	YEPNOW
1043.	MYTOLD	ZIVVIL
1044.	NABWET	NABWAP
1045.	NAJYIH	NEFFUA
1046.	NATPEE	NEDXAX
1047.	NATPEE01	NEDXAX
1048.	NAZDAV	NAZCUO
1049.	NBARBT	NBARBA
1050.	NEBFIM	NEBFOS
1051.	NEFCUY	NEFCEI
1052.	NEFTEY	MUJKEH
1053.	NESVOZ	AFAZEM
1054.	NESVOZ	AFAZEM01
1055.	NEXSIU	THIOUR
1056.	NEXSIU	THIOUR19
1057.	NEXSUG	THIOUR
1058.	NEXSUG	THIOUR19
1059.	NEXXAP	SUYBAR
1060.	NIDPEY	NIDJIW
1061.	NIDPEY01	NIDJIW
1062.	NIFHAL	NIFGUE
1063.	NIGZEJ	POVFIP
1064.	NIHPIF	KEMHAL
1065.	NIHPIF	KEMHAL02
1066.	NIMRUX	FLUBIP01
1067.	NIMRUX	FLUBIP02
1068.	NIMRUX	YACZIO
1069.	NINSIM	NINSEI
1070.	NISHUS	TOAZOC
1071.	NOCNOK	NOCPAY
1072.	NOGKID	NOGKEZ
1073.	NOJWAK	NOJVUD
1074.	NOMDUP	NITPOL
1075.	NOMDUP	NITPOL01
1076.	NOQZID	EVAZAD
1077.	NOTKEO	NUVFEQ
1078.	NOVTID	NOVTEZ
1079.	NUBFAR	NUBDUJ
1080.	NUJLOU	NUJLIO
1081.	NUMFUY	CIZQUX
1082.	NUMFUY	CIZQUX01
1083.	NURAMH	NIMFOE
1084.	NURAMH	NIMFOE01
1085.	NURAMH	NIMFOE02
1086.	NURAMH01	NIMFOE
1087.	NURAMH01	NIMFOE01
1088.	NURAMH01	NIMFOE02
1089.	NURJEP	NURHOX
1090.	NURJEP01	NURHOX
1091.	NUTWII	REZRAP
1092.	NUYCUG	NUYCOA
1093.	OBEQAN	HEBFUR
1094.	OBEQAN	HEBFUR01
1095.	OBEQAN01	HEBFUR
1096.	OBEQAN01	HEBFUR01
1097.	OBILOA	ABEWAG
1098.	OCADEE	YAWZAZ
1099.	OCANIQ	VURWAH
1100.	OCANIQ	VURWAH03
1101.	OCAZEX	MOKYER
1102.	OCAZEX	MOKYER01
1103.	OCEGIO	IGARAK
1104.	OCESEW	CAXMOD
1105.	OCESEW	CAXMOD01
1106.	ODOBAK	KETXIR
1107.	ODUSAJ	NOWFEM
1108.	OFEJOA	OFEJUG
1109.	OFEJOA	OFEJUG01
1110.	OFIREC	OFIQUR
1111.	OFIREC	OFIQUR01
1112.	OHAVUQ	DUXSUK
1113.	OHAVUQ	DUXSUK04
1114.	OJOGEA	YINFEI
1115.	OKATIF	LAKJUC
1116.	OKEJIZ	OKEJEV
1117.	OKEMAT	WUYPOW
1118.	OKEMAT	WUYPOW01
1119.	OKESAY	LEBKUZ
1120.	OREHOK	ITIZOA
1121.	OROXAV	BUCRIA
1122.	OSEXIU	OSEXEQ
1123.	OSIGEE	GADBUL
1124.	OSIVAO	OSITUG
1125.	OVAYUH	OVAYOB
1126.	OVOMOC	OVOMIW
1127.	OXACDH09	OXALAC02
1128.	OXACDH09	OXALAC04
1129.	OXAYUJ	QUKHIP
1130.	OXAYUJ	QUKHIP01
1131.	OXAYUJ	QUKHIP02
1132.	OXUHEW02	OXUHAS02
1133.	OXUHIA02	PEXXEW
1134.	OYETET	TIJPAF
1135.	OZOWOR	URIRET
1136.	PABPAL	PABNAJ
1137.	PACHEH	LAKFEJ
1138.	PAFBEG	ABIPIN

1139.	PAFBEG	ETEXIK
1140.	PAFBEG	ETEXIK02
1141.	PAFBEG	ETEXIK06
1142.	PAFBEG	KAHVUL
1143.	PAGYUS	ALXANM01
1144.	PAHNAP	WENROX
1145.	PAJDOU	HICRUI
1146.	PAKOJM	PAYKOJ
1147.	PANKEV	VAGVAA
1148.	PANYLB	ANPYAB
1149.	PAPNIG	PAPNUS
1150.	PAPNIG01	PAPNUS
1151.	PASXAK	PASWUD
1152.	PASXIS	PASWUD
1153.	PASXIS02	PASWUD
1154.	PASXIS03	PASWUD
1155.	PAXQAH	GAVKOG
1156.	PAYWOD	VANXIS
1157.	PAZGII	PAZGUU
1158.	PECXIE	PECXEA
1159.	PEDNAQ	WASPEL
1160.	PEFGOX	PEFGIR
1161.	PEFGOX	PEFGIR01
1162.	PEFGOX	PEFGIR03
1163.	PEFGOX01	PEFGIR
1164.	PEFGOX01	PEFGIR01
1165.	PEFGOX01	PEFGIR03
1166.	PEGWUU	PEGWII
1167.	PEGWUU	PEGWII01
1168.	PEJREC	PEJRIG
1169.	PEKJUM	PEKJEW
1170.	PEKJUM	PEKJEW01
1171.	PEKJUM01	PEKJEW
1172.	PEKJUM01	PEKJEW01
1173.	PEKQUT	PEKQON
1174.	PEKRIG	PEKREC
1175.	PELYOW	PELYUC
1176.	PEMCOA	UDOMUW
1177.	PEMPIH	VEQHAB
1178.	PEMTEI	PEMTAE
1179.	PEPCIW	XODYOE
1180.	PERCAR	POFYEP
1181.	PERKEF	PERKIJ
1182.	PERKOP	PERKUV
1183.	PESHAX	QOQNEP
1184.	PETRIS	AFEBUI
1185.	PEWYOI	PEWYUO
1186.	PEWYOI	PEWYUO01
1187.	PEWYOI	PEWYUO02
1188.	PEWYOI01	PEWYUO
1189.	PEWYOI01	PEWYUO01
1190.	PEWYOI01	PEWYUO02
1191.	PEYSOD	CBMZPN01
1192.	PEYSOD	CBMZPN03
1193.	PEYSOD	CBMZPN11
1194.	PEYSOD	CBMZPN12
1195.	PEYSOD	CBMZPN16
1196.	PEYSOD	CBMZPN20
1197.	PEZBEB	WUVKON
1198.	PEZVOF	SODVUC
1199.	PHBARM	PHBARB
1200.	PHBARM	PHBARB05
1201.	PHBARM	PHBARB10
1202.	PHBARM	PHBARB11
1203.	PHBARM	PHBARB12
1204.	PHBZAC	JOZZIH
1205.	PHBZAC	JOZZIH01
1206.	PHGLOH	PHGLOL
1207.	PHOLCL	CUZDIK
1208.	PIGFET	PIGFAP
1209.	PIKKAV	VAWDEC
1210.	PILBUJ	PILBOD
1211.	PILSIO	KIXWEV
1212.	PINOLH01	PINCOL
1213.	PINOLH01	PINCOL01
1214.	PIPACA	PIPACB
1215.	PIPERH	ITIZOA
1216.	PIPTUF	PIPTOZ
1217.	PIRRAL	PIRREP
1218.	POBRON	WEWTOJ
1219.	PODPEE	RACCEE
1220.	POSTAS	POSTEW
1221.	POSTAS	POSTEW01
1222.	POTPET	GLYGLY
1223.	POTPET	GLYGLY01
1224.	POVSEZ	POVRUO
1225.	PROAMH10	QOBHEW
1226.	PROAMH11	QOBHEW
1227.	PUBMII	PUBMUU
1228.	PUBMII	PUBMUU01
1229.	PUBMII	PUBMUU02
1230.	PUBMII	PUBMUU23
1231.	PUBMII01	PUBMUU12
1232.	PUDYUI	PUDYOC
1233.	PUDYUI	PUDYOC01
1234.	PUFGUU	PUFJUX
1235.	PUFJIL	PUFJUX
1236.	PUHPOX	BOBHOP
1237.	PUKFAE	PUKBEE
1238.	PUVMAU	SAQJEZ
1239.	PUYSEI	EDEXOA
1240.	PUZGAT	VETVOG01
1241.	PUZGAT01	VETVOG01
1242.	PYMDSD	PYMSUL10

1243. PYMDSD PYMSUL11
1244. PYMELL10 KEFGUA
1245. PYMELL12 KEFGUA
1246. PYRTHA10 PYRDNA01
1247. PYRTHA10 PYRDNA02
1248. PYZDCX IYAWAG
1249. QABCII QABCEE
1250. QAMBIR IVEVIO
1251. QAMMUP QAMNIE
1252. QANBOW UJAWUZ
1253. QANVAF IJUTAJ
1254. QANVAF IJUTAJ01
1255. QANVAF IJUTAJ02
1256. QARVUB QARVOV
1257. QATJUR RUGPUF
1258. QATNEF QUWYIR
1259. QEHSEB IKOBAN
1260. QEHTUT RUDYIZ
1261. QEHWOQ XAQXAQ
1262. QEKDUI SAJFIT
1263. QEMGEV XEBTEF
1264. QEMLOK QEKWEJ
1265. QERHOL KINKOH
1266. QERHOL QIFPEC
1267. QEVPEP VATSAK
1268. QEVPEP VATSAK01
1269. QEVPEP01 VATSAK
1270. QEVPEP01 VATSAK01
1271. QEXRIV WASNAF
1272. QEYNEQ QEYNAM
1273. QIDCIP WIQMUE
1274. QIFKAT QIFJAS
1275. QIFKAT QIFJAS01
1276. QIFKAT QIFJAS02
1277. QIHSIJ CAWKEQ
1278. QIJZUE RORQEU
1279. QILHOK APANIQ
1280. QIMKOM QIMKIG
1281. QIMKOM QIMKIG02
1282. QIMKOM QIMKIG03
1283. QIMKOM02 QIMKIG
1284. QIMKOM02 QIMKIG02
1285. QIMKOM02 QIMKIG03
1286. QIMXOB QIMXER
1287. QIMXUH QIMXER
1288. QIQLUX QIQLIL
1289. QIQLUX QIQLIL01
1290. QIVTUK ZZZEEU01
1291. QIVTUK ZZZEEU08
1292. QIYKEO ZULQAY
1293. QOBJEW QOBJAS
1294. QOBJOH MAWMIJ
1295. QOBJOH MAWMIJ01
1296. QOBJOH MAWMIJ03
1297. QOHYAO TETDAM01
1298. QOHYAO TETDAM02
1299. QOHYAO01 TETDAM01
1300. QOHYAO01 TETDAM02
1301. QOHYES TETDAM01
1302. QOHYES TETDAM02
1303. QOQXAV FAFWIS
1304. QOQXAV FAFWIS01
1305. QOQXAV FAFWIS02
1306. QOSQIY GEMJIU
1307. QOZKOG KIXFIH
1308. QOZSAZ LAGJOU
1309. QUBREL PAKNOG
1310. QUFCUQ QUFCOK
1311. QUINCX QUINCB10
1312. QUKHUB QUKHIP
1313. QUKHUB QUKHIP01
1314. QUKHUB QUKHIP02
1315. QUNHAJ QUNGUC
1316. QUNHAJ QUNGUC01
1317. QURTIH HACTPH10
1318. QURTIH HACTPH12
1319. QURYEI QURYAE
1320. QUWCIV QUWBOA
1321. QUWXOW RUKHAG
1322. RABBUN NAGVUM
1323. RABCAU NAGWAT
1324. RABCEY NAGWAT
1325. RABJOP DUHJIB
1326. RAFHUE LABQUD
1327. RALDEN XEHKOK
1328. RAPJIE UMIQEO
1329. RAPRUW AZSTBB02
1330. RAQCOC TEXNEP
1331. RASBEV DUHJIB
1332. RASBIZ DUHJIB
1333. RATBOD RATBIX
1334. RAVFEA HQOXDO
1335. RAVFOK OCUSOU
1336. RAVFUQ OCUSOU
1337. RAVGEB PECYON
1338. RAVQAI RAVPUB
1339. RAWBIA RAWBEW
1340. RAZWEX RAZWIB
1341. REGRAW HOXOCD
1342. REGREA HOXOCD
1343. REGRIE HOXOCD
1344. RELBAM REKZUD
1345. RENQIN RENQEJ
1346. RENQIN01 RENQEJ

1347.	REQWEP	REQWAL
1348.	REVCOL	SEVPEP01
1349.	REVGUV	REVGOP
1350.	REVMIP	GAYJAT
1351.	REVQUF	REVRAM
1352.	RIGLAW	RIGLEA
1353.	RIGMIE	SEQVAN
1354.	RIKBIW	RIKBOC
1355.	RINXUH	RINWUG
1356.	RIPFON	RIPFIH
1357.	RIPVAP	RIPTUH
1358.	RIRBID	CTVHVH
1359.	RIRBID	CTVHVH01
1360.	RIRBID	VOKBIG
1361.	RIVJEM	RIVJAI
1362.	RIYYEE	WOVXIO
1363.	RIYYEE03	WOVXIO
1364.	ROFFEY	RIZFAI
1365.	ROFNUW	ZOTMOM
1366.	ROFXOA	KUTPOF
1367.	ROHBEV	QUCKOP
1368.	ROHQOW	OXUXIQ
1369.	ROJVIX	ROJVET
1370.	ROKSOZ	QOPYOK
1371.	ROLDAY	ROLCUR
1372.	ROLVOF	DIXFAR
1373.	ROMDUT	VETVOG01
1374.	ROPHUB	ROPJEN
1375.	ROPJAJ	ROPJEN
1376.	RUCFOL	RUCFUR
1377.	RUKGUZ	RUKHAG
1378.	RUVPUT	RUVPON
1379.	RUWGEV	PROLIN
1380.	RUWGEV	PROLIN04
1381.	RUWKEZ	RUWKAV
1382.	RUYGUO	MADTAP
1383.	SABZUQ	SABZEA
1384.	SAJRUQ	SAJQUP
1385.	SAKYOS10	SAKYUY10
1386.	SAMKIB	IWODUT
1387.	SAMSEH	SAMVOU
1388.	SAMSEH	SAMVOU02
1389.	SAMVUA	SAMVOU
1390.	SAMVUA	SAMVOU02
1391.	SAMVUA01	SAMVOU
1392.	SAMVUA01	SAMVOU02
1393.	SANACM	AFAZEM
1394.	SANACM	AFAZEM01
1395.	SANACM01	AFAZEM
1396.	SANACM01	AFAZEM01
1397.	SAQQUZ	SAQQOT
1398.	SAQYAM	PUDXES
1399.	SAQYAM	PUDXES02
1400.	SAQYAM	PUDXES04
1401.	SARJIF	FEDPUA
1402.	SASYOD	SASXOC
1403.	SATSEL	DURDAV
1404.	SATSEL	DURDAV01
1405.	SAXDUR	PINCOL
1406.	SAXDUR	PINCOL01
1407.	SAXQAL	SAXPOY
1408.	SAXQAL	SAXPOY01
1409.	SAYPIT	FADHUP
1410.	SAYPOZ	FADHOJ
1411.	SAYQEP	SILTOW
1412.	SAYQEP	SILTOW01
1413.	SAYQEP	SILTOW11
1414.	SAZQOA	PINCOL
1415.	SAZQOA	PINCOL01
1416.	SAZXAS	QQQACY01
1417.	SAZXAS01	QQQACY01
1418.	SECGIQ	SECGOW
1419.	SEDCAG	IVASON
1420.	SEHKEW	TARTAC
1421.	SEHKEW	TARTAC02
1422.	SEHKEW	TARTAC24
1423.	SENPAF	SENNUX
1424.	SENRUZ	BUDMOD
1425.	SEPDAS	WEDSOO
1426.	SEQBOF	HARJOC
1427.	SEQBOF	HARJOC01
1428.	SEWBIH	GUWWUS
1429.	SEYDIJ	OQUQEX
1430.	SEYDIJ01	OQUQEX
1431.	SEYDIJ02	OQUQEX
1432.	SEYDIM	SEYDOS
1433.	SIDCOY	RIZGEM
1434.	SIGBUG	TELCEU01
1435.	SILFAX	SILDUP
1436.	SILPEL	SILHON
1437.	SIMPOT	OJOCIB
1438.	SITRIY	SITREU
1439.	SLBZAC01	RASYOZ
1440.	SOBBUG	EYOJUZ
1441.	SOCMOO	ABIPIN
1442.	SOCMOO	ETEXIK
1443.	SOCMOO	ETEXIK02
1444.	SOCMOO	ETEXIK06
1445.	SOCMOO	KAHVUL
1446.	SOGUAN03	ZZZAYP03
1447.	SOGUAN20	ZZZAYP03
1448.	SOKPAL	SOKPEP
1449.	SOMLAI	NEQLOM
1450.	SOQNIW	VIWRAV

1451.	SOQNOC	VITBEG
1452.	SOWSEC	QATWAJ
1453.	SOWSIG	QATWUD
1454.	SOWSUS	ETDIAM01
1455.	SOWSUS	ETDIAM15
1456.	SOWSUS	ETDIAM18
1457.	SOWTIH	HEXMDA
1458.	SOYDAM	SOYCUF
1459.	SUGBIF	YADSOL
1460.	SUHHEI	OCUSOU
1461.	SULAMH10	SULAMD
1462.	SULAMH10	SULAMD01
1463.	SULAMH10	SULAMD02
1464.	SULAMH10	SULAMD11
1465.	SULSUX	GOBVEA
1466.	SULSUX	GOBVEA01
1467.	SUMMOC	SUMMIW
1468.	SUNGUD	SUNGOX
1469.	SUPKET	SUPKIX
1470.	SURRIG	SURREC
1471.	SUYVAJ	LITJED
1472.	TABQIY	TABPUJ
1473.	TAJRAX	QQQACY01
1474.	TAJRAY	TAJQUR
1475.	TAKJIB	TAJSUV
1476.	TAKJIB	TAKJUN
1477.	TAKJIB	TAKKAU
1478.	TAMQUW	OSILAF
1479.	TAMQUW	OSILAF01
1480.	TAPYEP	WIRNIU
1481.	TAQCOE	AQIYEG
1482.	TARTDL	TARTAC02
1483.	TARTMM	TARTAM
1484.	TARTMM01	TARTAM
1485.	TATBIB	TATBAT
1486.	TAZMOW	EFURIH
1487.	TAZMOW	EFURIH02
1488.	TAZMOW	EPINOS
1489.	TAZMOW	IFAKAC
1490.	TAZMOW	MUINOS
1491.	TAZMOW	MYINOL
1492.	TAZMOW	MYINOL01
1493.	TAZMOW	QIKZAN
1494.	TAZMOW	QIKZAN01
1495.	TAZMOW	QIKZAN02
1496.	TAZMOW	YEPNOW
1497.	TAZMOW	ZIVVIL
1498.	TECLAP	IKAKOW
1499.	TEDFAL	CEDKUT
1500.	TEGXEI	YERTOE
1501.	TEHREE	TEHRII
1502.	TEJGEX	TEJGAT
1503.	TEJMIE	LETBER
1504.	TEJPEE	BOKTIF
1505.	TEJTUX	XAZNES
1506.	TESTOM	TESTON10
1507.	TESTOM01	TESTON10
1508.	TEVFAB	CEXQEB
1509.	TEVFEG	TEVFIK
1510.	TEVFEG	TEVFIK01
1511.	TEVVOG	ETANIY
1512.	TEVVOG	ETANIY01
1513.	TEXQOC	LOFFET
1514.	TEZMER	JACXUJ
1515.	TFMSAD	TFMSUL
1516.	TFMSPH	TFMSUL
1517.	TFMSTH	TFMSUL
1518.	THEOPH	BAPLOT01
1519.	THEOPH	BAPLOT02
1520.	THEOPH	BAPLOT04
1521.	THEOPH	DUWXEA
1522.	THEOPH01	BAPLOT01
1523.	THEOPH01	BAPLOT02
1524.	THEOPH01	BAPLOT04
1525.	THEOPH01	DUWXEA
1526.	THEOPH02	BAPLOT01
1527.	THEOPH02	BAPLOT02
1528.	THEOPH02	BAPLOT04
1529.	THEOPH02	DUWXEA
1530.	THEOPH03	BAPLOT01
1531.	THEOPH03	BAPLOT02
1532.	THEOPH03	BAPLOT04
1533.	THEOPH03	DUWXEA
1534.	THEOPH04	BAPLOT01
1535.	THEOPH04	BAPLOT02
1536.	THEOPH04	BAPLOT04
1537.	THEOPH04	DUWXEA
1538.	THIAMC	UNEXOA
1539.	THIAMC13	UNEXOA
1540.	THIMCH10	GEYXOX
1541.	THIRDN10	BELZEX
1542.	THYMMH	THYMIN
1543.	THYMMH	THYMIN02
1544.	TIDPOL	JAYZEO
1545.	TIKBIY	TIKBEU
1546.	TIKBIY	TIKBEU01
1547.	TIRLOX	UYEHAI
1548.	TIRMIS	QAHQOI
1549.	TISJAH	PABHIJ
1550.	TITGOT	LEJVED
1551.	TIZNEW	TIZNIA
1552.	TLALAN10	ZZZIFQ01
1553.	TMAMSU02	NUNSIY
1554.	TMAMSU02	NUNSIY01

1555.	TOHGIG	OMIXUD
1556.	TOKYIC	BEQWUS
1557.	TOKYIC	NEDMIS
1558.	TOKYIC	NEDMIS03
1559.	TOLCAY	IBUMAT
1560.	TOLCAY01	IBUMAT
1561.	TOPBUX	TOPBOR
1562.	TOQFAI	TOQFEM
1563.	TORSEZ	OBAZOJ
1564.	TOSBOU	TOSBIO
1565.	TOSVAA	HUFXAJ
1566.	TOSVAA	HUFXAJ01
1567.	TREHAL01	DEKYEX01
1568.	TREHAL02	DEKYEX01
1569.	TREHAL03	DEKYEX01
1570.	TREHAL10	DEKYEX01
1571.	TREHAL11	DEKYEX
1572.	TREHAL11	DEKYEX01
1573.	TREHAL12	DEKYEX
1574.	TREHAL12	DEKYEX01
1575.	TREHAL13	DEKYEX
1576.	TREHAL14	DEKYEX
1577.	TRIMES10	BTCOAC
1578.	TRMHXD	DMHXDM
1579.	TUBWAQ	ROZHEU
1580.	TUFSOD	TUFSIX
1581.	TUQYUZ	TUQYOT
1582.	TUQZAG	TUQYOT
1583.	TUXLOP	VALQAD
1584.	TUZFIF	TUZFOL
1585.	TUZFIF01	TUZFOL
1586.	TYRAMH	SENJEC
1587.	UBAFEK	SALWEK
1588.	UBEGEP	KUVLAP
1589.	UCEHUG	WAKBAM
1590.	UCOJIG	MAJRIZ
1591.	UCOJIG	MAJRIZ01
1592.	UCUVET	CUHNEY10
1593.	UCUVET	CUHNEZ
1594.	UDAJEP	GUFMAV
1595.	UDENOH	USIMUF
1596.	UFETUX	UFEVAF
1597.	UHAHUG	ZOKYON
1598.	UHELAV	UHEKOI
1599.	UHIZES	BENPRL
1600.	UHIZES	BENPRL01
1601.	UHIZES	BENPRL02
1602.	UHURUL	KAPNAQ
1603.	UJIJEC	UJIJAY
1604.	UJIPIN	GEXKID
1605.	UJIPOT	GEXKID
1606.	UJIPUZ	GEXKID
1607.	UJIXAO	SURREC
1608.	UJOQUF	KAMPIY
1609.	UKIZUK	UKOBEC
1610.	UKIZUK01	UKOBEC
1611.	ULEKUS	DEYSAE
1612.	ULEQOR	ULEQIL
1613.	ULICOJ	NUZSIM
1614.	ULICOJ	NUZSIM02
1615.	ULIQAJ	ZEFTAI
1616.	ULOBAA	ULOBOO
1617.	UMIPIR	XOKVEA
1618.	UNEBIY	HEWTAF
1619.	UNEWUG	UNEWIU
1620.	UNEWUG	UNEWIU01
1621.	UNEXAN	UNEWIU
1622.	UNEXAN	UNEWIU01
1623.	UNUKAQ	SIHLOL
1624.	UPIZAW	KIZCUT
1625.	UPUKOH	CUFFUG
1626.	UPUKUN	CUFFUG
1627.	UPUNEA	QERCEW
1628.	UPUNEA	QERCEW01
1629.	UQIFOR	HIBXOF
1630.	UQIFOR	HIBXOF01
1631.	URUCER	URUCAN
1632.	UTOHOB	UQESAL
1633.	UVOHET	GIVJUQ
1634.	UVOHET	GIVJUQ02
1635.	UVOJUL	UVOKAS
1636.	UZOVIP	UZOVAH03
1637.	VABKOA	TICFUG
1638.	VACBOT	VACBUZ
1639.	VACBOT	VACBUZ01
1640.	VADVEC	DURDAV
1641.	VADVEC	DURDAV01
1642.	VAJWUZ	ZEGBOE
1643.	VALHCL10	VALEHC10
1644.	VALHCL11	VALEHC10
1645.	VALMOL	BERYAZ
1646.	VALMOL	BERYAZ01
1647.	VAMCAO	VAMBUH
1648.	VATKAD	TIRMIR
1649.	VAWWIB	VAWWEX
1650.	VAWWOH	VAWWEX
1651.	VAWWUN	VAWWEX
1652.	VAXDUT	DEMYEZ
1653.	VAXDUT	DEMYEZ01
1654.	VAXDUT	DEMYEZ03
1655.	VAXFAB	DEMYEZ
1656.	VAXFAB	DEMYEZ01
1657.	VAXFAB	DEMYEZ03
1658.	VAYBOO	VABVAX

1659. VEBREZ GEYRAD
1660. VEFNAY VEFNEC
1661. VEFPUR ZOYMOP06
1662. VEFPUR ZOYMOP07
1663. VEFPUR01 ZOYMOP06
1664. VEFPUR01 ZOYMOP07
1665. VEHZAK ETIWEJ
1666. VEJDAQ VEJCUJ
1667. VELQUZ AVULEI
1668. VELREM QUKHIP
1669. VELREM QUKHIP01
1670. VELREM QUKHIP02
1671. VELYUH TAKRON
1672. VERQOB VERQER
1673. VERUCH VERUCE
1674. VEXRUL VEXREV
1675. VICSIM NAVSUY
1676. VICSIM NAVSUY02
1677. VIMRIS KUZDIS
1678. VIPCUS ZOKSUN
1679. VIPKUA YADSOL
1680. VIVQIB OQUPAS
1681. VIVQOH OQUPAS
1682. VIVQOH01 OQUPAS
1683. VOBVUE VOBWAL
1684. VOBWEP VOBWOZ
1685. VODJII HUJBUK
1686. VOHQAM THEOPI01
1687. VOHWIZ RITDUU
1688. VOJRIV PAZADO10
1689. VOKNOA MORPLN01
1690. VOLWEA VOLWUQ
1691. VOYSIL CEXQUR
1692. VOYSOR CEXQUR
1693. VULGOZ VULGUF
1694. VUMDEN VUMDAJ
1695. VUNFEQ VUNFIU
1696. VURTIM VURSUX
1697. VUXREM VUXRIQ
1698. VUXREM VUXRIQ01
1699. WADROK WACNOF
1700. WAFNAT COTZAN06
1701. WAFNAT HXACAN
1702. WAFNAT HXACAN29
1703. WAFNAT HXACAN39
1704. WAFPAV WAFNUN
1705. WAGLOG WAGLUM
1706. WAKCAO WAKCES
1707. WATDUR WATFAZ
1708. WAYDEH WAYDAD
1709. WECGUI WECGIW
1710. WECGUI WECGIW01
1711. WECGUI WECGIW02
1712. WECHAP WECGIW
1713. WECHAP WECGIW01
1714. WECHAP WECGIW02
1715. WEGPAA YOGZAV
1716. WEPMOU AQEMIT
1717. WEPMOU AQEMIT02
1718. WIBQOP XAYCAB
1719. WIBWOW KIGKUH
1720. WIBWOW KIGKUH01
1721. WIBWOW KIGKUH02
1722. WICKUR JACDAV
1723. WIFCIY WIFCEU
1724. WIGXUG ICIBOL
1725. WIJNEI GESTOQ
1726. WIKDAV MUMBEL
1727. WIKDAV MUMBEL04
1728. WINKUA NETRUZ
1729. WINKUA NOBVEF
1730. WIRTIA NUTSUQ
1731. WITHIQ RUYFEX01
1732. WIVGEO RUHWUM
1733. WIVGEO02 RUHWUM
1734. WIWBIP WIVZUY
1735. WIZMUO SOLMOW
1736. WOBNEG RUJQOD
1737. WOBQAG DIPGIS10
1738. WOBQAG DIPGIS12
1739. WOBQAG DIPGIS13
1740. WOCREL SEPNUX
1741. WODKOQ WODKIK
1742. WODVAN PHTHAC
1743. WODZAS WODZEW
1744. WOGGIK XOTWUY
1745. WOMBAB WOLZUS
1746. WOVYEL HIQWEJ
1747. WOVYEL01 HIQWEJ
1748. WOZPAE MECWIC
1749. WOZPAE MECWIC03
1750. WUDHOU IVEVIO
1751. WUTVIQ RENCUI
1752. WUWJAA UNEXOA
1753. WUWREM WUWRIQ
1754. WUWROW ATPRCL01
1755. WUXPUC WUXQAJ
1756. WUZDIG WUZDEC
1757. XADXAE XADWUX
1758. XAHGAR KOVRUK
1759. XAHWOS GORKUV
1760. XAHWOS GORKUV01
1761. XAHXAF LEFHEL
1762. XAJTIM XAJSUX

1763.	XANLOO	WIGMUV
1764.	XAQREN	XAQRAJ
1765.	XASLOS	NIXZAX
1766.	XAVQEQ	IYUQIC
1767.	XAYGEJ	VETVOG01
1768.	XEBBAI	PAQLUP
1769.	XEGPUX	EXAXEG
1770.	XEGQEI	EWUZED
1771.	XEJLON	XEJLUT
1772.	XENTES	SEZREU
1773.	XENTES	SEZREU01
1774.	XEPNUD	XEPNIR
1775.	XEPREQ	ADEZER
1776.	XESPUH	XESQAO
1777.	XEVRUO	XEVROI
1778.	XEVRUO	XEVSAV
1779.	XEYYOQ	PIHLAU
1780.	XEYZUX	CEXXIO
1781.	XEZHER	XEZHOB
1782.	XIBGIZ	XIBGEV
1783.	XICPIK	XICPOQ
1784.	XICRAE	OFUYIZ
1785.	XIKPUF	XIKQAM
1786.	XIKPUF	XIKQAM01
1787.	XIPCUU	LONDOJ
1788.	XIRKUG	ZOBCOK
1789.	XIVHOA	ZIVKAQ
1790.	XIYSAA	ALAHCL
1791.	XIZPEB	XIZNUP
1792.	XIZPEB01	XIZNUP
1793.	XODYIY	XODYOE
1794.	XOFSOA	BOFTAT
1795.	XOHDAB	XOHCUU
1796.	XOHFEH	XOHFAD
1797.	XOJJAJ	XOJHUB
1798.	XOJMOA	XOJMIU
1799.	XOJMUG	XOJMIU
1800.	XOJPUJ	XOJQEU
1801.	XOKKIT	GIXXOB
1802.	XOKKIT	GIXXOB01
1803.	XOKTEY	XOKTAU
1804.	XOKVIE	XOKVEA
1805.	XOMBEH	XOMCAE
1806.	XOMWOL	COTZAN06
1807.	XOMWOL	HXACAN
1808.	XOMWOL	HXACAN29
1809.	XOMWOL	HXACAN39
1810.	XOSHUI	JAYFOF
1811.	XOXRAE	XOXQUX
1812.	XOYXUF	HUZKUI
1813.	XOZSUB	XOZTEM
1814.	XOZSUB	XOZTIQ
1815.	XOZTAI	XOZTEM
1816.	XOZTAI	XOZTIQ
1817.	XURHOH	IDIMAJ
1818.	XUSGUO	XUSGOI
1819.	XUVLOP	EFEMUX01
1820.	XUVPAG	PAHBUW
1821.	XUVROW	KIDHEN
1822.	XYFUAD10	BRADOS
1823.	YABHOA	BITCEM11
1824.	YABHOA	BITCEM13
1825.	YADZAH	CBMZPN01
1826.	YADZAH	CBMZPN03
1827.	YADZAH	CBMZPN11
1828.	YADZAH	CBMZPN12
1829.	YADZAH	CBMZPN16
1830.	YADZAH	CBMZPN20
1831.	YAFCOY	IXEWIR
1832.	YAFCOY	IXEWIR01
1833.	YAFCOY	IXEWIR02
1834.	YAFCOY	IXEWIR03
1835.	YAFCOY	IXEWIR04
1836.	YAFHUL	ECAQAD01
1837.	YAGFEU	SAQJAW
1838.	YAKWAJ	BISMEV
1839.	YAKWAJ	BISMEV01
1840.	YAKWAJ	BISMEV03
1841.	YAKWAJ	BISMEV04
1842.	YAKWAJ	BISMEV07
1843.	YAKWAJ	BISMEV14
1844.	YANCUM	YAKVAI
1845.	YANHIH	KOBBAG
1846.	YAPMEJ	UHITOV
1847.	YASHIL	YASGOQ
1848.	YASHIL	YASGOQ01
1849.	YASHIL	YASGOQ02
1850.	YASHIL	YASGOQ03
1851.	YECVUX	QOBGUL
1852.	YECVUX	QOBHAS
1853.	YECWAE	LIHJAQ
1854.	YECWAE	LIHJAQ01
1855.	YECWEI	QOBHIA
1856.	YEFBER	YASRIU
1857.	YEFLOL	BOJDOU
1858.	YEJNEG	QAMXUY
1859.	YEJNEG	QAMXUY01
1860.	YEKHIH	YEKHON
1861.	YESTOG	LEVSAJ
1862.	YETPIW	YETPES
1863.	YIJBID	YIJBEZ
1864.	YILGAB	XIPJOY
1865.	YILKUZ	SEBROG
1866.	YIMZAV	HOXOCD

1867. YINFAC ZZZIVG02
1868. YIQFEK OGEKOB
1869. YIXGUJ YIXHAQ
1870. YIXSUU YIXSOO
1871. YIZQOO PHTHAC
1872. YODRUF BOLBOT01
1873. YODYUL ROMMEM
1874. YOTKEY GAPSIB
1875. YOWRAF QONSET
1876. YOYPIL EGORIB
1877. YOYZIX MAJRIZ
1878. YOYZIX MAJRIZ01
1879. YUDFAF COWSIR
1880. YUDFAF COWSIR01
1881. YUFGIR YUFGAJ
1882. YUFGIR YUFGAJ01
1883. YUJNUM01 CUYCEF
1884. YUJNUM01 LICWOM
1885. YUJNUM01 LICWOM01
1886. YUJPAU01 CUYCEF
1887. YUJPAU01 LICWOM
1888. YUJPAU01 LICWOM01
1889. YUKVAD WOLXOK03
1890. YULFUH YULGAO
1891. YUPBER VALINO
1892. YUVVAO YUWYEW
1893. YUVVAO YUWYEW01
1894. YUXGUV ZZZEEU01
1895. YUXGUV ZZZEEU08
1896. YUYROZ YUYRIT
1897. YUYTIV PICACC
1898. YUZTET QIMKIG
1899. YUZTET QIMKIG02
1900. YUZTET QIMKIG03
1901. ZACSOO JALYAZ
1902. ZAGTUZ ZAGZOZ
1903. ZAHJIB CYURAC03
1904. ZAHQAA PAQUCL
1905. ZARLEM XAPTIU
1906. ZATTEV ZATTIZ
1907. ZEBWUA ZEBWOU
1908. ZEFMII ZEFMUU
1909. ZEFSUB ZEFTAI
1910. ZEKPEM ZEKPOW
1911. ZEKPEM ZEKPOW01
1912. ZEKPIQ ZEKPOW
1913. ZEKPIQ ZEKPOW01
1914. ZELTUI ZELVAQ
1915. ZESQUL ZESRAS
1916. ZEXYOQ MEWSUE
1917. ZICPAF FIBYEW
1918. ZIFQOW ZIFQEM
1919. ZIPWOM GAMBUT01
1920. ZIPWOM GAMBUT04
1921. ZIVVOR EFURIH
1922. ZIVVOR EFURIH02
1923. ZIVVOR EPINOS
1924. ZIVVOR FOPKOK
1925. ZIVVOR IFAKAC
1926. ZIVVOR MUINOS
1927. ZIVVOR MYINOL
1928. ZIVVOR MYINOL01
1929. ZIVVOR MYINOL03
1930. ZIVVOR QIKZAN
1931. ZIVVOR QIKZAN01
1932. ZIVVOR QIKZAN02
1933. ZIVVOR QIKZUH
1934. ZIVVOR YEPNOW
1935. ZIVVOR ZIVVIL
1936. ZIZYAI TITLIS
1937. ZOQHEU KIWTEP
1938. ZOYMUV ZOYMOP
1939. ZOYMUV ZOYMOP01
1940. ZOYMUV ZOYMOP02
1941. ZUDBUV PAPCOA
1942. ZUNHIB RAMGOB
1943. ZUNHIB RAMGOB02
1944. ZUNHOH RAMGOB
1945. ZUNHOH RAMGOB02
1946. ZUNHOH01 RAMGOB
1947. ZUNHOH01 RAMGOB02
1948. ZZZAMS03 OPENAN
1949. ZZZAMS06 OPENAN
1950. ZZZPPI02 URICAC
1951. ZZZPRW01 MAJRIZ
1952. ZZZPRW01 MAJRIZ01
1953. ZZZRLO01 ACRDIN
1954. ZZZRLO01 ACRDIN01
1955. ZZZRLO01 ACRDIN05
1956. ZZZRLO01 ACRDIN06
1957. ZZZRLO01 ACRDIN08
1958. ZZZSBA01 HUYBUY
1959. ZZZSBA03 HUYBUY
1960. ZZZSQK02 BEVBAF
1961. AEDTAH EDTAXX01
1962. AEDTAH EDTAXX02
1963. AMPALX AHPHAL
1964. DAWGOX KETXIR
1965. ETANOE ETANIY
1966. ETANOE ETANIY01
1967. ETAPIA ETANIY
1968. ETAPIA ETANIY01
1969. FINFEN SUNFIS
1970. NEZMAJ NEZTAQ

1971.	PASXIS01	PASWUD
1972.	PAXQAH01	GAVKOG
1973.	QESFID	LOBDUB
1974.	SAKDUF	FUDGOA
1975.	SAYQEP01	SILTOW
1976.	SAYQEP01	SILTOW01
1977.	SAYQEP01	SILTOW11
1978.	TIZVUT	TIZWAA
1979.	TIZVUT	TIZWAA01
1980.	UGUFOS	JERPIG
1981.	VAHKIY	VAHKIY10
1982.	YECWIM	QOBHIA
1983.	SAGSUR	BODWAS
1984.	YUFCAE	EFEZAR
1985.	ADPRTY	ADPRTR
1986.	BABYOV	BABYUB
1987.	BEXPOK	BEXPIE
1988.	BUTAHC11	DUHJIB
1989.	DUGTOQ	DUHJIB
1990.	ETHYLO	DUFBOV10
1991.	ETXDHY01	DUFBOV10
1992.	FEFNOT01	CBMZPN01
1993.	FEFNOT01	CBMZPN03
1994.	FEFNOT01	CBMZPN11
1995.	FEFNOT01	CBMZPN12
1996.	FEFNOT01	CBMZPN16
1997.	FEFNOT01	CBMZPN20
1998.	FUSWOH04	EFUMAU
1999.	FUSWOH04	EFUMAU03
2000.	FUSWOH04	EFUMAU06
2001.	GISWOW	UFADUC
2002.	IKALIR	IKAKOW
2003.	IKALIR01	IKAKOW
2004.	IKAMEO	IKAKOW
2005.	IKAMOY	IKAKOW
2006.	IKAMOY01	IKAKOW
2007.	IKAVOF	ETHANE01
2008.	IKAVOF	ETHANE04
2009.	JARBUA	JARBOU
2010.	KUYGOA	ZZZITY01
2011.	KUYGUG	KAMTAW
2012.	KUYGUG	KAMTAW03
2013.	KUYHAN	DCLBEN
2014.	KUYHAN	DCLBEN02
2015.	KUYHAN	DCLBEN03
2016.	KUYHER	CISTON01
2017.	LAFLOV	KAYSOT
2018.	MUSDEL	DUHJIB
2019.	MUSDOV	DUHJIB
2020.	NAHCIJ	ACETYL02
2021.	NAHCOP	JAYDUI
2022.	QANGES	RABTIX
2023.	QESVOA	QESVUG
2024.	QIMXOB01	QIMXER
2025.	QQQCIY10	CEKGUU
2026.	QQQCIY10	CEKGUU02
2027.	QQQCIY10	CEKGUU06
2028.	RITMEN	CIVXIO01
2029.	ROPRAR	ROPQUK
2030.	VOMPEU	VOMNUI
2031.	WEGPED	WEGPIH
2032.	XEBQAZ	CARPEQ
2033.	ZZZPPI01	URICAC
2034.	ZZZWQA01	BUNJAV
2035.	CUXYOK	CUXZAX
2036.	FABMOM	ESEZIM
2037.	HCPZHO	HCPZBO
2038.	KEMDIQ	KEMDEM
2039.	KEMDIQ	KEMDEM01
2040.	NIJLEX	NEFPET
2041.	PINOLH	PINCOL
2042.	PINOLH	PINCOL01
2043.	QAFKAK	DAYFUH
2044.	QQQBKD01	PYRGAL02
2045.	RESMIL	DADPDS
2046.	SOWVAB	QOBHUM
2047.	SOWVOP	QOBHEW
2048.	TOLQUG	NOVGUA
2049.	XOZSIP	RUCDOJ
2050.	LUXYAE	LUXYIM
2051.	YIHJOP	ETGUAN
2052.	AHEREK	JAYPUU05
2053.	NENMEZ	LUYMAT
2054.	CAFINE	NIWFEE02
2055.	CAFINE	NIWFEE04
2056.	YINDUU	SABZOK
2057.	JOSKUZ	SALBUT01
2058.	BIMGIP	SOCTOT
2059.	NEXSIU	THIOUR06
2060.	NEXSUG	THIOUR06
2061.	OCAZEZ	UBOTIQ
2062.	PUSBIP	PUSBOV
2063.	QEVLAE	VUKVAY
2064.	COSNUT	COSNON

```
#28 Determine water molecule stoichiometry

from __future__ import division
from ccdc import io, search
import argparse
import os
import glob
from decimal import*

#This script counts the number of water molecules in each hydrate entry

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step6 Change Composition of Three Classes\chart_class1_hydrates_with_waterless_forms.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step6 Change Composition of Three Classes\chart_class2_hydrates_without_waterless_forms.txt"

class Runner(argparse.ArgumentParser):
    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='waters.gcd',
            help='output file [waters.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')
        entry_reader2 = io.EntryReader(filepath2, format='identifiers')

        with io.EntryWriter(self.args.output) as waters_writer:

            duplicate_identifier_dictionary = {'GADKOL':1, 'MECYEY':2, 'OBASAN':3, 'OMOMAE':4, 'OTONUI':5, 'QOJWAO':6, 'TZBFZO':7, 'VOBXOZ':8, 'VUFJAI':9, 'VUFJAJ':10, 'WEPQOA':11, 'XADFUD':12, 'YABREY':13, 'YULCOZ':14, 'ZOVFEV':15}

            loop = 0
            total_monohydrate = 0
            total_dihydrate = 0
            total_trihydrate = 0
            total_tetrahydrate = 0
            total_pentahydrate = 0
            total_hexahydrate = 0
            total_heptahydrate = 0
            total_octahydrate = 0
            total_nonahydrate = 0
            total_decahydrate = 0
            total_more_than_ten = 0
            total_hemihydrate = 0
            total_less_than_one = 0
            total_more_than_one = 0
            total_undefined = 0
            grand_total = 0
            entry1 = entry_reader1
```

```
64              while loop < 2:
65                  monohydrate = 0
66                  dihydrate = 0
67                  trihydrate = 0
68                  tetrahydrate = 0
69                  pentahydrate = 0
70                  hexahydrate = 0
71                  heptahydrate = 0
72                  octahydrate = 0
73                  nonahydrate = 0
74                  decahydrate = 0
75                  more_than_ten = 0
76                  hemihydrate = 0
77                  less_than_one = 0
78                  more_than_one = 0
79                  undefined = 0
80                  sub_total = 0
81                  #An already found list is used so hydrates in hydrate-anhydrate pairs
                    that have multiple anhydrous forms will not be counted multiple times
82                  already_found = []
83                  for a in range(len(entry1)):
84                      sub_total += 1
85                      grand_total += 1
86                      skip = 0
87                      if loop == 0:
88                          if entry1[a].identifier in duplicate_identifier_dictionary:
89                              if
                                duplicate_identifier_dictionary.get(entry1[a].identifier)
                                not in already_found:
90
                                    already_found.append(duplicate_identifier_dictionary.get(
                                    entry1[a].identifier))
91                              else:
92                                  skip += 1
93                          else:
94                              if hash(entry1[a].identifier) not in already_found:
95                                  already_found.append(hash(entry1[a].identifier))
96                              else:
97                                  skip += 1
98                      #Some structures require a manual specification of the number of
                        water molecules
99                      if entry1[a].identifier == 'BULMEA03' or entry1[a].identifier ==
                        'LOYXAA':
100                         monohydrate += 1
101                         total_monohydrate += 1
102                     if entry1[a].identifier == 'HEQQOJ':
103                         less_than_one += 1
104                         total_less_than_one += 1
105                     if entry1[a].identifier == 'KIJFAN':
106                         undefined += 1
107                         total_undefined += 1
108                     if skip == 0:
109                         #The formula of each entry is searched for water
110                         #The integer preceding the formula for water will be either an
                            integral or non-integral number
111                         #The number of water molecules were sorted up to 10, then any
                            structure with more than 10 waters was one category
112                         #Non-integral waters was restricted to 0.5 (hemihydrates), less
                            than 1, more than 1, and undefined (ex. x(H2 O1) or n(H2 O1))
113                         title = entry1[a].formula
114                         pieces = title.split(',')
115                         water = []
116                         for b in range(len(pieces)):
117                             if '(H2 O1)' in pieces[b] or pieces[b] == 'H2 O1' or '(D2
                                O1)' in pieces[b] or pieces[b] == 'D2 O1':
118                                 water.append(pieces[b])
119                         if len(water) == 1:
```

```
                        if '(' not in water[0]:
                            monohydrate += 1
                            total_monohydrate += 1
                        else:
                            pin = water[0].find('(')
                            water[0] = water[0][:pin]
                            if 'x' in water[0] or 'n' in water[0]:
                                undefined += 1
                                total_undefined += 1
                            elif '.' in water[0]:
                                water[0] = Decimal(water[0])
                                if water[0] == 0.5:
                                    hemihydrate += 1
                                    total_hemihydrate += 1
                                elif water[0] < 1:
                                    less_than_one += 1
                                    total_less_than_one += 1
                                else:
                                    more_than_one += 1
                                    total_more_than_one += 1
                            else:
                                water[0] = int(water[0])
                                if water[0] == 2:
                                    dihydrate += 1
                                    total_dihydrate += 1
                                elif water[0] == 3:
                                    trihydrate += 1
                                    total_trihydrate += 1
                                elif water[0] == 4:
                                    tetrahydrate += 1
                                    total_tetrahydrate += 1
                                elif water[0] == 5:
                                    pentahydrate += 1
                                    total_pentahydrate += 1
                                elif water[0] == 6:
                                    hexahydrate += 1
                                    total_hexahydrate += 1
                                elif water[0] == 7:
                                    heptahydrate += 1
                                    total_heptahydrate += 1
                                elif water[0] == 8:
                                    octahydrate += 1
                                    total_octahydrate += 1
                                elif water[0] == 9:
                                    nonahydrate += 1
                                    total_nonahydrate += 1
                                elif water[0] == 10:
                                    decahydrate += 1
                                    total_decahydrate += 1
                                else:
                                    more_than_ten += 1
                                    total_more_than_ten += 1

                #The number of structures in each water category are printed followed
                by the percentage from the total number of structures in the input file
                print 'start of one round'
                print monohydrate
                print monohydrate/sub_total
                print dihydrate
                print dihydrate/sub_total
                print trihydrate
                print trihydrate/sub_total
                print tetrahydrate
                print tetrahydrate/sub_total
                print pentahydrate
                print pentahydrate/sub_total
                print hexahydrate
```

```
                print hexahydrate/sub_total
                print heptahydrate
                print heptahydrate/sub_total
                print octahydrate
                print octahydrate/sub_total
                print nonahydrate
                print nonahydrate/sub_total
                print decahydrate
                print decahydrate/sub_total
                print more_than_ten
                print more_than_ten/sub_total
                print hemihydrate
                print hemihydrate/sub_total
                print less_than_one
                print less_than_one/sub_total
                print more_than_one
                print more_than_one/sub_total
                print undefined
                print undefined/sub_total
                print 'end of one round'

                loop += 1
                if loop == 1:
                    entry1 = entry_reader2

            #Numbers for the grand total are also recorded, these are determined by
            adding the numbers for class 1 and class 2 hydrates together
            print 'start of one round'
            print total_monohydrate
            print total_monohydrate/grand_total
            print total_dihydrate
            print total_dihydrate/grand_total
            print total_trihydrate
            print total_trihydrate/grand_total
            print total_tetrahydrate
            print total_tetrahydrate/grand_total
            print total_pentahydrate
            print total_pentahydrate/grand_total
            print total_hexahydrate
            print total_hexahydrate/grand_total
            print total_heptahydrate
            print total_heptahydrate/grand_total
            print total_octahydrate
            print total_octahydrate/grand_total
            print total_nonahydrate
            print total_nonahydrate/grand_total
            print total_decahydrate
            print total_decahydrate/grand_total
            print total_more_than_ten
            print total_more_than_ten/grand_total
            print total_hemihydrate
            print total_hemihydrate/grand_total
            print total_less_than_one
            print total_less_than_one/grand_total
            print total_more_than_one
            print total_more_than_one/grand_total
            print total_undefined
            print total_undefined/grand_total
            print 'end of one round'

if __name__ == '__main__':
    r = Runner()
    r.run()

```

```
#29 Find pairs with 1 and 2+ components and 2000 unpaired subsets

from ccdc import io, search
import argparse
import os
import glob
from decimal import*

#This script separates structures from each class into those with 1 organic component and 2+ organic components

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step6 Change Composition of Three Classes\chart_class1_hydrates_with_waterless_forms.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step6 Change Composition of Three Classes\chart_class1_waterless_forms_with_hydrate.txt"
filepath3 = "C:\Users\jenwe\Hydrates Manuscript\Step6 Change Composition of Three Classes\chart_class2_hydrates_without_waterless_forms.txt"
filepath4 = "C:\Users\jenwe\Hydrates Manuscript\Step6 Change Composition of Three Classes\chart_class3_waterless_forms_without_hydrate.txt"

class Runner(argparse.ArgumentParser):
    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='one_organic_hydrates_with_waterless_forms.gcd',
            help='output file [one_organic_hydrates_with_waterless_forms.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')
        entry_reader2 = io.EntryReader(filepath2, format='identifiers')
        entry_reader3 = io.EntryReader(filepath3, format='identifiers')
        entry_reader4 = io.EntryReader(filepath4, format='identifiers')

        with io.EntryWriter(self.args.output) as waters_writer:
            with
            io.EntryWriter("more_than_one_organic_hydrates_with_waterless_forms.gcd")
            as writer1:
                with io.EntryWriter("one_organic_waterless_forms_with_hydrates.gcd") as
                writer2:
                    with
                    io.EntryWriter("more_than_one_organic_waterless_forms_with_hydrates.g
                    cd") as writer3:
                        with
                        io.EntryWriter("one_organic_hydrates_without_waterless_form.gcd")
                         as writer4:
                            with
                            io.EntryWriter("more_than_one_organic_hydrates_without_waterl
                            ess_form.gcd") as writer5:
                                with
                                io.EntryWriter("one_organic_waterless_forms_without_hydra
                                te.gcd") as writer6:
```

```
52                                  with
                                    io.EntryWriter("more_than_one_organic_waterless_forms
                                    _without_hydrate.gcd") as writer7:
53                                      with
                                        io.EntryWriter("subset_one_organic_hydrates_witho
                                        ut_waterless_form.gcd") as writer8:
54                                          with
                                            io.EntryWriter("subset_more_than_one_organic_
                                            hydrates_without_waterless_form.gcd") as
                                            writer9:
55                                              with
                                                io.EntryWriter("subset_one_organic_waterl
                                                ess_forms_without_hydrate.gcd") as
                                                writer10:
56                                                  with
                                                    io.EntryWriter("subset_more_than_one_
                                                    organic_waterless_forms_without_hydra
                                                    te.gcd") as writer11:
57
58                                                      for a in
                                                        range(len(entry_reader1)):
59                                                          #The entry formula is used
                                                            to determine the number of
                                                            components in each structure
60                                                          #Water does not count
                                                            toward the number of
                                                            components for hydrate
                                                            structures
61                                                          formulas =
                                                            entry_reader1[a].formula.spli
                                                            t(',')
62                                                          waters = []
63                                                          for b in
                                                            range(len(formulas)):
64                                                              if '(H2 O1)' in
                                                                formulas[b] or '(D2
                                                                O1)' in formulas[b] or
                                                                formulas[b] == 'H2 O1'
                                                                or formulas[b] == 'D2
                                                                O1':
65
                                                                    waters.append(formula
                                                                    s[b])
66                                                          formulas = [x for x in
                                                            formulas if x not in waters]
67                                                          if len(formulas) == 1:
68
                                                                waters_writer.write(entry
                                                                _reader1[a])
69
                                                                writer2.write(entry_reade
                                                                r2[a])
70                                                          else:
71
                                                                writer1.write(entry_reade
                                                                r1[a])
72
                                                                writer3.write(entry_reade
                                                                r2[a])
73
74                                                      #Subsets of class 2 and class 3
                                                        are generated in this script
75                                                      #The first 2,000 structures
                                                        with 1 component and no
                                                        disorder were written to an
                                                        output file
76                                                      #The first 2,000 structures
```

```
                                    with 2+ components and no
                                    disorder were written to an
                                    output file
77                                  count = 0
78                                  count1 = 0
79                                  for c in
                                    range(len(entry_reader3)):
80                                      formulas2 =
                                        entry_reader3[c].formula.spli
                                        t(',')
81                                      waters2 = []
82                                      for d in
                                        range(len(formulas2)):
83                                          if '(H2 O1)' in
                                            formulas2[d] or '(D2
                                            O1)' in formulas2[d] or
                                            formulas2[d] == 'H2 O1'
                                            or formulas2[d] == 'D2
                                            O1':
84
                                                waters2.append(formul
                                                as2[d])
85                                      formulas2 = [x for x in
                                        formulas2 if x not in
                                        waters2]
86                                      if len(formulas2) == 1:
87
                                            writer4.write(entry_reade
                                            r3[c])
88                                          if count < 2000 and
                                            entry_reader3[c].crystal.
                                            has_disorder == False:
89
                                                writer8.write(entry_r
                                                eader3[c])
90                                              count += 1
91                                      else:
92
                                            writer5.write(entry_reade
                                            r3[c])
93                                          if count1 < 2000 and
                                            entry_reader3[c].crystal.
                                            has_disorder == False:
94
                                                writer9.write(entry_r
                                                eader3[c])
95                                              count1 += 1
96
97                                  count2 = 0
98                                  count3 = 0
99                                  for e in
                                    range(len(entry_reader4)):
100                                     formulas3 =
                                        entry_reader4[e].formula.spli
                                        t(',')
101                                     if len(formulas3) == 1:
102
                                            writer6.write(entry_reade
                                            r4[e])
103                                         if count2 < 2000 and
                                            entry_reader4[e].crystal.
                                            has_disorder == False:
104
                                                writer10.write(entry_
                                                reader4[e])
105                                             count2 += 1
106                                     else:
```

```
                                                                writer7.write(entry_reade
                                                                r4[e])
                                                                if count3 < 2000 and
                                                                entry_reader4[e].crystal.
                                                                has_disorder == False:

                                                                    writer11.write(entry_
                                                                    reader4[e])
                                                                    count3 += 1


if __name__ == '__main__':
    r = Runner()
    r.run()
```

```
#30 Determine crystal system symmetry for hydrate-anhydrate pairs

from ccdc import io, search
import argparse
import os
import glob

#This script records the crystal system symmetry of hydrate and waterless form structures in the hydrate-anhydrate pairs list

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three Classes\one_organic_hydrates_with_waterless_forms.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three Classes\more_than_one_organic_hydrates_with_waterless_forms.txt"
filepath3 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three Classes\one_organic_waterless_forms_with_hydrates.txt"
filepath4 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three Classes\more_than_one_organic_waterless_forms_with_hydrates.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='systems_HA_subsets.gcd',
            help='output file [systems_HA_subsets.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        crystal_reader1 = io.CrystalReader(filepath1, format='identifiers')
        crystal_reader2 = io.CrystalReader(filepath2, format='identifiers')
        crystal_reader3 = io.CrystalReader(filepath3, format='identifiers')
        crystal_reader4 = io.CrystalReader(filepath4, format='identifiers')

        with io.EntryWriter(self.args.output) as system_writer:

            #This dictionary assigns values to the different crystal system symmetries
            #Higher numbers are given to crystal systems with higher degrees of symmetry
            system_dictionary = {'triclinic':1, 'monoclinic':2, 'orthorhombic':3, 'tetragonal':4, 'trigonal':5, 'rhombohedral':5, 'hexagonal':6, 'cubic':7}

            loop = 0
            crystal1 = crystal_reader1
            crystal2 = crystal_reader3
            while loop < 2:
                #The relationship between the degree of symmetry between the hydrate and waterless form crystal system are compared first
                same_system = 0
                hydrate_higher = 0
                waterless_higher = 0
                #Lower crystal system symmetries (triclinic, monoclinic, and orthorhombic) were further evaluated
```

```
                both_triclinic = 0
                #Htri_Wmono = hydrate is triclinic and waterless form is monoclinic
                Htri_Wmono = 0
                Htri_Wortho = 0
                both_monoclinic = 0
                Hmono_Wortho = 0
                both_orthorhombic = 0
                Hmono_Wtri = 0
                Hortho_Wtri = 0
                Hortho_Wmono = 0

                for b in range(len(crystal1)):
                    if system_dictionary.get(crystal1[b].crystal_system) ==
                    system_dictionary.get(crystal2[b].crystal_system):
                        same_system += 1
                    if system_dictionary.get(crystal1[b].crystal_system) >
                    system_dictionary.get(crystal2[b].crystal_system):
                        hydrate_higher += 1
                    if system_dictionary.get(crystal1[b].crystal_system) <
                    system_dictionary.get(crystal2[b].crystal_system):
                        waterless_higher += 1
                    if system_dictionary.get(crystal1[b].crystal_system) == 1:
                        if system_dictionary.get(crystal2[b].crystal_system) == 1:
                            both_triclinic += 1
                        elif system_dictionary.get(crystal2[b].crystal_system) == 2:
                            Htri_Wmono += 1
                        elif system_dictionary.get(crystal2[b].crystal_system) == 3:
                            Htri_Wortho += 1
                    elif system_dictionary.get(crystal1[b].crystal_system) == 2:
                        if system_dictionary.get(crystal2[b].crystal_system) == 2:
                            both_monoclinic += 1
                        elif system_dictionary.get(crystal2[b].crystal_system) == 3:
                            Hmono_Wortho += 1
                    elif system_dictionary.get(crystal1[b].crystal_system) == 3:
                        if system_dictionary.get(crystal2[b].crystal_system) == 3:
                            both_orthorhombic += 1
                    if system_dictionary.get(crystal2[b].crystal_system) == 1:
                        if system_dictionary.get(crystal1[b].crystal_system) == 2:
                            Hmono_Wtri += 1
                        elif system_dictionary.get(crystal1[b].crystal_system) == 3:
                            Hortho_Wtri += 1
                    elif system_dictionary.get(crystal2[b].crystal_system) == 2:
                        if system_dictionary.get(crystal1[b].crystal_system) == 3:
                            Hortho_Wmono += 1

                #The number of structures in each category is printed
                print 'start of one round'
                print same_system
                print hydrate_higher
                print waterless_higher
                print both_triclinic
                print Htri_Wmono
                print Htri_Wortho
                print both_monoclinic
                print Hmono_Wortho
                print both_orthorhombic
                print Hmono_Wtri
                print Hortho_Wtri
                print Hortho_Wmono
                print 'end of one round'

                loop += 1
                crystal1 = crystal_reader2
                crystal2 = crystal_reader4
```

```
124 if __name__ == '__main__':
125     # This runs the script
126     r = Runner()
127     r.run()
128
129
130
```

```
#31 Determine crystal system symmetry for different classes

from __future__ import division
from ccdc import io, search
import argparse
import os
import glob

#This script determines the percentage of structures for each class (divided into 1 component and 2+ components) with the seven different crystal system symmetries

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three Classes\one_organic_hydrates_with_waterless_forms.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three Classes\more_than_one_organic_hydrates_with_waterless_forms.txt"
filepath3 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three Classes\one_organic_waterless_forms_with_hydrates.txt"
filepath4 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three Classes\more_than_one_organic_waterless_forms_with_hydrates.txt"
filepath5 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three Classes\one_organic_hydrates_without_waterless_form.txt"
filepath6 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three Classes\more_than_one_organic_hydrates_without_waterless_form.txt"
filepath7 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three Classes\one_organic_waterless_forms_without_hydrate.txt"
filepath8 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three Classes\more_than_one_organic_waterless_forms_without_hydrate.txt"
filepath9 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three Classes\hydrates.txt"
filepath10 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three Classes\entries_in_working_data_set.txt"
filepath11 = "C:\Users\jenwe\Hydrates Manuscript\Step6 Change Composition of Three Classes\chart_class1_hydrates_with_waterless_forms.txt"
filepath12 = "C:\Users\jenwe\Hydrates Manuscript\Step6 Change Composition of Three Classes\chart_class1_waterless_forms_with_hydrate.txt"
filepath13 = "C:\Users\jenwe\Hydrates Manuscript\Step6 Change Composition of Three Classes\chart_class2_hydrates_without_waterless_forms.txt"
filepath14 = "C:\Users\jenwe\Hydrates Manuscript\Step6 Change Composition of Three Classes\chart_class3_waterless_forms_without_hydrate.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath7,
            help='input database filepath7'
        )
        self.add_argument(
            '-o', '--output', default='systems_four_subsets.gcd',
            help='output file [systems_four_subsets.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        crystal_reader1 = io.CrystalReader(filepath1, format='identifiers')
        crystal_reader2 = io.CrystalReader(filepath2, format='identifiers')
```

```
        crystal_reader3 = io.CrystalReader(filepath3, format='identifiers')
        crystal_reader4 = io.CrystalReader(filepath4, format='identifiers')
        crystal_reader5 = io.CrystalReader(filepath5, format='identifiers')
        crystal_reader6 = io.CrystalReader(filepath6, format='identifiers')
        crystal_reader7 = io.CrystalReader(filepath7, format='identifiers')
        crystal_reader8 = io.CrystalReader(filepath8, format='identifiers')
        crystal_reader9 = io.CrystalReader(filepath9, format='identifiers')
        crystal_reader10 = io.CrystalReader(filepath10, format='identifiers')
        crystal_reader11 = io.CrystalReader(filepath11, format='identifiers')
        crystal_reader12 = io.CrystalReader(filepath12, format='identifiers')
        crystal_reader13 = io.CrystalReader(filepath13, format='identifiers')
        crystal_reader14 = io.CrystalReader(filepath14, format='identifiers')

        with io.EntryWriter(self.args.output) as system_writer:

            duplicate_identifier_dictionary = {'GADKOL':1, 'MECYEY':2, 'OBASAN':3,
            'OMOMAE':4, 'OTONUI':5, 'QOJWAO':6, 'TZBFZO':7, 'VOBXOZ':8, 'VUFJAI':9,
            'VUFJAJ':10, 'WEPQOA':11, 'XADFUD':12, 'YABREY':13, 'YULCOZ':14, 'ZOVFEV':15}

            loop = 0
            crystal1 = crystal_reader1
            while loop < 14:
                triclinic = 0
                monoclinic = 0
                orthorhombic = 0
                tetragonal = 0
                trigonal = 0
                hexagonal = 0
                cubic = 0
                already_found = []
                for a in range(len(crystal1)):
                    skip = 0
                    if loop == 0 or loop == 1 or loop == 2 or loop == 3 or loop == 10
                    or loop == 11:
                        if crystal1[a].identifier in duplicate_identifier_dictionary:
                            if
                            duplicate_identifier_dictionary.get(crystal1[a].identifier)
                            not in already_found:

                                already_found.append(duplicate_identifier_dictionary.get(
                                crystal1[a].identifier))
                            else:
                                skip += 1
                        else:
                            if hash(crystal1[a].identifier) not in already_found:
                                already_found.append(hash(crystal1[a].identifier))
                            else:
                                skip += 1
                    if skip == 0:
                        system = crystal1[a].crystal_system
                        if system == 'triclinic':
                            triclinic +=1
                        elif system == 'monoclinic':
                            monoclinic +=1
                        elif system == 'orthorhombic':
                            orthorhombic +=1
                        elif system == 'tetragonal':
                            tetragonal +=1
                        elif system == 'trigonal' or system == 'rhombohedral':
                            trigonal +=1
                        elif system == 'hexagonal':
                            hexagonal +=1
                        else:
                            cubic +=1

                total = triclinic + monoclinic + orthorhombic + tetragonal + trigonal +
                hexagonal + cubic
```

```
112
113                 #The percentage of structures for each class is printed
114                 print 'start of one round'
115                 print triclinic/total
116                 print monoclinic/total
117                 print orthorhombic/total
118                 print tetragonal/total
119                 print trigonal/total
120                 print hexagonal/total
121                 print cubic/total
122                 print 'end of one round'
123
124                 loop += 1
125                 if loop == 1:
126                     crystal1 = crystal_reader2
127                 if loop == 2:
128                     crystal1 = crystal_reader3
129                 if loop == 3:
130                     crystal1 = crystal_reader4
131                 if loop == 4:
132                     crystal1 = crystal_reader5
133                 if loop == 5:
134                     crystal1 = crystal_reader6
135                 if loop == 6:
136                     crystal1 = crystal_reader7
137                 if loop == 7:
138                     crystal1 = crystal_reader8
139                 if loop == 8:
140                     crystal1 = crystal_reader9
141                 if loop == 9:
142                     crystal1 = crystal_reader10
143                 if loop == 10:
144                     crystal1 = crystal_reader11
145                 if loop == 11:
146                     crystal1 = crystal_reader12
147                 if loop == 12:
148                     crystal1 = crystal_reader13
149                 if loop == 13:
150                     crystal1 = crystal_reader14
151
152
153
154 if __name__ == '__main__':
155     # This runs the script
156     r = Runner()
157     r.run()
158
159
160
```

```
#32 Find hydrate-anhydrate pairs determined at the same temperature

from ccdc import io, search
import argparse
import os
import glob
from decimal import Decimal

#This script finds hydrate and waterless form structures in hydrate-anhydrate pairs
that were collected at the same temperature

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three
Classes\one_organic_hydrates_with_waterless_forms.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three
Classes\more_than_one_organic_hydrates_with_waterless_forms.txt"
filepath3 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three
Classes\one_organic_waterless_forms_with_hydrates.txt"
filepath4 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three
Classes\more_than_one_organic_waterless_forms_with_hydrates.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output',
            default='same_temp_one_organic_hydrates_with_waterless_forms.gcd',
            help='output file [same_temp_one_organic_hydrates_with_waterless_forms.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        entry_reader1 = io.EntryReader(filepath1, format='identifiers')
        entry_reader2 = io.EntryReader(filepath2, format='identifiers')
        entry_reader3 = io.EntryReader(filepath3, format='identifiers')
        entry_reader4 = io.EntryReader(filepath4, format='identifiers')

        with io.EntryWriter(self.args.output) as same_temp_writer:
            with
            io.EntryWriter("same_temp_one_organic_waterless_forms_with_hydrates.gcd")
            as writer1:
                with
                io.EntryWriter("same_temp_more_than_one_organic_hydrates_with_waterless_f
                orms.gcd") as writer2:
                    with
                    io.EntryWriter("same_temp_more_than_one_organic_waterless_forms_with_
                    hydrates.gcd") as writer3:

                        loop = 0
                        entry4 = entry_reader1
                        entry5 = entry_reader3
                        while loop < 2:
                            for b in range(len(entry4)):
```

```
                        #Some structures do not report a temperature in Python
                        and were discarded to prevent another manual sort
                        #For cases where there is a reported temperature, both
                        the unit and number were recorded
                        if entry4[b].temperature != None and
                        entry5[b].temperature != None:
                            loop1 = 0
                            if loop == 0:
                                entry1 = entry_reader1
                            if loop == 1:
                                entry1 = entry_reader2
                            while loop1 < 2:
                                temp_text = entry1[b].temperature.split(' ')
                                if len(temp_text) == 4 and temp_text[3] == 'K':
                                    if temp_text[1].isdigit() == False:
                                        structure_temp = Decimal(temp_text[1])
                                    else:
                                        structure_temp = int(temp_text[1])
                                    structure_unit = 'K'
                                elif len(temp_text) == 3 and temp_text[2] == 'K':
                                    if temp_text[1].isdigit() == False:
                                        structure_temp = Decimal(temp_text[1])
                                    else:
                                        structure_temp = int(temp_text[1])
                                    structure_unit = 'K'
                                elif len(temp_text) == 3 and temp_text[2] ==
                                'deg.C':
                                    if temp_text[1].isdigit() == False:
                                        structure_temp = Decimal(temp_text[1])
                                    else:
                                        structure_temp = int(temp_text[1])
                                    structure_unit = 'deg.C'
                                elif len(temp_text) == 3 and 'K' in temp_text[2]:
                                    if temp_text[2].isdigit() == False:
                                        start = temp_text[2].index('K')
                                        structure_temp =
                                        Decimal(temp_text[2][:start])
                                    else:
                                        start = temp_text[2].index('K')
                                        structure_temp =
                                        int(temp_text[2][:start])
                                    structure_unit = 'K'
                                elif len(temp_text) == 2 and temp_text[1] == 'K':
                                    if temp_text[0].isdigit() == False:
                                        structure_temp = Decimal(temp_text[0])
                                    else:
                                        structure_temp = int(temp_text[0])
                                    structure_unit = 'K'
                                elif len(temp_text) == 2 and temp_text[1] ==
                                'deg.C':
                                    if temp_text[0].isdigit() == False:
                                        structure_temp = Decimal(temp_text[0])
                                    else:
                                        structure_temp = int(temp_text[0])
                                    structure_unit = 'deg.C'
                                elif len(temp_text) == 2 and 'K' in temp_text[1]:
                                    if temp_text[1].isdigit() == False:
                                        start = temp_text[1].index('K')
                                        structure_temp =
                                        Decimal(temp_text[1][:start])
                                    else:
                                        start = temp_text[1].index('K')
                                        structure_temp =
                                        int(temp_text[1][:start])
                                    structure_unit = 'K'
                                elif len(temp_text) == 2 and 'deg.C' in
                                temp_text[1]:
```

```
                                        if temp_text[1].isdigit() == False:
                                            start = temp_text[1].index('deg.C')
                                            structure_temp =
                                            Decimal(temp_text[1][:start])
                                        else:
                                            start = temp_text[1].index('deg.C')
                                            structure_temp =
                                            int(temp_text[1][:start])
                                        structure_unit = 'deg.C'
                                    if loop1 == 0:
                                        hydrate_temp = structure_temp
                                        hydrate_unit = structure_unit
                                    if loop1 == 1:
                                        waterless_temp = structure_temp
                                        waterless_unit = structure_unit
                                    loop1 += 1
                                    if loop1 == 1 and loop == 0:
                                        entry1 = entry_reader3
                                    if loop1 == 1 and loop == 1:
                                        entry1 = entry_reader4

                                #If the temperature for each structure is within 5
                                degrees of each other, the structures are considered to
                                be taken at roughly the same temperature and are
                                written to output files
                                    if hydrate_unit == waterless_unit and
                                    abs(hydrate_temp-waterless_temp) < 5:
                                        if loop == 0:
                                            same_temp_writer.write(entry4[b])
                                            writer1.write(entry5[b])
                                        else:
                                            writer2.write(entry4[b])
                                            writer3.write(entry5[b])
                                    elif hydrate_unit == 'K' and waterless_unit ==
                                    'deg.C' and abs(hydrate_temp-(waterless_temp+273))
                                    < 5:
                                        if loop == 0:
                                            same_temp_writer.write(entry4[b])
                                            writer1.write(entry5[b])
                                        else:
                                            writer2.write(entry4[b])
                                            writer3.write(entry5[b])
                                    elif hydrate_unit == 'deg.C' and waterless_unit ==
                                    'K' and abs((hydrate_temp+273)-waterless_temp) < 5:
                                        if loop == 0:
                                            same_temp_writer.write(entry4[b])
                                            writer1.write(entry5[b])
                                        else:
                                            writer2.write(entry4[b])
                                            writer3.write(entry5[b])

                            loop += 1
                            entry4 = entry_reader2
                            entry5 = entry_reader4


if __name__ == '__main__':
    r = Runner()
    r.run()
```

```
#33 Determine the packing fraction of hydrate-anhydrate pairs at the same temperature

from ccdc import io, search
import argparse
import os
import glob
import numpy
from numpy import median

#This script determines the difference in packing fraction between hydrate and
waterless form structures collected at the same temperature

filepath1 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three
Classes\same_temp_one_organic_hydrates_with_waterless_forms.txt"
filepath2 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three
Classes\same_temp_more_than_one_organic_hydrates_with_waterless_forms.txt"
filepath3 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three
Classes\same_temp_one_organic_waterless_forms_with_hydrates.txt"
filepath4 = "C:\Users\jenwe\Hydrates Manuscript\Step7 Analyze Structures in Three
Classes\same_temp_more_than_one_organic_waterless_forms_with_hydrates.txt"

class Runner(argparse.ArgumentParser):

    def __init__(self):
        super(self.__class__, self).__init__(description=__doc__)
        self.add_argument(
            '-i', '--input', default=filepath1,
            help='input database filepath1'
        )
        self.add_argument(
            '-o', '--output', default='packing_fraction_outside_5percent_hydrate.gcd',
            help='output file [packing_fraction_outside_5percent_hydrate.gcd]'
        )
        self.add_argument(
            '-m', '--maximum', default=0, type=int,
            help='Maximum number of structures to find [all]'
        )

        args = self.parse_args()

        self.args = args
        self.settings = search.Search.Settings()
        self.settings.max_hit_structures = self.args.maximum

    def run(self):

        crystal_reader1 = io.CrystalReader(filepath1, format='identifiers')
        crystal_reader2 = io.CrystalReader(filepath2, format='identifiers')
        crystal_reader3 = io.CrystalReader(filepath3, format='identifiers')
        crystal_reader4 = io.CrystalReader(filepath4, format='identifiers')

        with io.EntryWriter(self.args.output) as packing_fraction_writer:
            with io.EntryWriter("packing_fraction_outside_5percent_waterless_form.gcd")
            as writer1:

                loop = 0
                crystal1 = crystal_reader1
                crystal2 = crystal_reader3
                #Hydrate-anhydrate pairs whose packing fractions are within 5% of each
                other were counted
                #The packing fraction erence was recorded at 0.5% intervals
                #Pairs where the hydrate has a higher packing fraction are counted
                separate from those where the waterless form has a higher packing
                fraction
                while loop < 2:
                    Hydrate_half_percent = 0
                    Hydrate_one_percent = 0
```

```
                    Hydrate_onehalf_percent = 0
                    Hydrate_two_percent = 0
                    Hydrate_twohalf_percent = 0
                    Hydrate_three_percent = 0
                    Hydrate_threehalf_percent = 0
                    Hydrate_four_percent = 0
                    Hydrate_fourhalf_percent = 0
                    Hydrate_five_percent = 0
                    Waterless_half_percent = 0
                    Waterless_one_percent = 0
                    Waterless_onehalf_percent = 0
                    Waterless_two_percent = 0
                    Waterless_twohalf_percent = 0
                    Waterless_three_percent = 0
                    Waterless_threehalf_percent = 0
                    Waterless_four_percent = 0
                    Waterless_fourhalf_percent = 0
                    Waterless_five_percent = 0
                    #Any pairs where one or both structure has disorder are omitted
                    from this analysis
                    for b in range(len(crystal1)):
                        if crystal1[b].has_disorder == False and
                        crystal2[b].has_disorder == False:
                            crystal1[b].molecule.normalise_hydrogens()
                            hydrate_fraction = crystal1[b].packing_coefficient
                            crystal2[b].molecule.normalise_hydrogens()
                            waterless_fraction = crystal2[b].packing_coefficient
                            if hydrate_fraction - waterless_fraction > 0:
                                if hydrate_fraction - waterless_fraction < 0.005:
                                    Hydrate_half_percent += 1
                                elif hydrate_fraction - waterless_fraction >= 0.005 and
                                hydrate_fraction - waterless_fraction < 0.01:
                                    Hydrate_one_percent += 1
                                elif hydrate_fraction - waterless_fraction >= 0.01 and
                                hydrate_fraction - waterless_fraction < 0.015:
                                    Hydrate_onehalf_percent += 1
                                elif hydrate_fraction - waterless_fraction >= 0.015 and
                                hydrate_fraction - waterless_fraction < 0.02:
                                    Hydrate_two_percent += 1
                                elif hydrate_fraction - waterless_fraction >= 0.02 and
                                hydrate_fraction - waterless_fraction < 0.025:
                                    Hydrate_twohalf_percent += 1
                                elif hydrate_fraction - waterless_fraction >= 0.025 and
                                hydrate_fraction - waterless_fraction < 0.03:
                                    Hydrate_three_percent += 1
                                elif hydrate_fraction - waterless_fraction >= 0.03 and
                                hydrate_fraction - waterless_fraction < 0.035:
                                    Hydrate_threehalf_percent += 1
                                elif hydrate_fraction - waterless_fraction >= 0.035 and
                                hydrate_fraction - waterless_fraction < 0.04:
                                    Hydrate_four_percent += 1
                                elif hydrate_fraction - waterless_fraction >= 0.04 and
                                hydrate_fraction - waterless_fraction < 0.045:
                                    Hydrate_fourhalf_percent += 1
                                elif hydrate_fraction - waterless_fraction >= 0.045 and
                                hydrate_fraction - waterless_fraction < 0.05:
                                    Hydrate_five_percent += 1
                                else:
                                    packing_fraction_writer.write(crystal1[b])
                                    writer1.write(crystal2[b])
                            else:
                                if waterless_fraction - hydrate_fraction < 0.005:
                                    Waterless_half_percent += 1
                                elif waterless_fraction - hydrate_fraction >= 0.005 and
                                waterless_fraction - hydrate_fraction < 0.01:
                                    Waterless_one_percent += 1
                                elif waterless_fraction - hydrate_fraction >= 0.01 and
```

```
                        waterless_fraction - hydrate_fraction < 0.015:
114                         Waterless_onehalf_percent += 1
115                     elif waterless_fraction - hydrate_fraction >= 0.015 and
                        waterless_fraction - hydrate_fraction < 0.02:
116                         Waterless_two_percent += 1
117                     elif waterless_fraction - hydrate_fraction >= 0.02 and
                        waterless_fraction - hydrate_fraction < 0.025:
118                         Waterless_twohalf_percent += 1
119                     elif waterless_fraction - hydrate_fraction >= 0.025 and
                        waterless_fraction - hydrate_fraction < 0.03:
120                         Waterless_three_percent += 1
121                     elif waterless_fraction - hydrate_fraction >= 0.03 and
                        waterless_fraction - hydrate_fraction < 0.035:
122                         Waterless_threehalf_percent += 1
123                     elif waterless_fraction - hydrate_fraction >= 0.035 and
                        waterless_fraction - hydrate_fraction < 0.04:
124                         Waterless_four_percent += 1
125                     elif waterless_fraction - hydrate_fraction >= 0.04 and
                        waterless_fraction - hydrate_fraction < 0.045:
126                         Waterless_fourhalf_percent += 1
127                     elif waterless_fraction - hydrate_fraction >= 0.045 and
                        waterless_fraction - hydrate_fraction < 0.05:
128                         Waterless_five_percent += 1
129                     else:
130                         packing_fraction_writer.write(crystal1[b])
131                         writer1.write(crystal2[b])
132
133                 #The number of structures in each difference category is printed
134                 print 'start of one round'
135                 print "Hydrate fraction higher by 0.5% for " +
                    str(Hydrate_half_percent) + " pairs"
136                 print "Hydrate fraction higher by 1.0% for " +
                    str(Hydrate_one_percent) + " pairs"
137                 print "Hydrate fraction higher by 1.5% for " +
                    str(Hydrate_onehalf_percent) + " pairs"
138                 print "Hydrate fraction higher by 2.0% for " +
                    str(Hydrate_two_percent) + " pairs"
139                 print "Hydrate fraction higher by 2.5% for " +
                    str(Hydrate_twohalf_percent) + " pairs"
140                 print "Hydrate fraction higher by 3.0% for " +
                    str(Hydrate_three_percent) + " pairs"
141                 print "Hydrate fraction higher by 3.5% for " +
                    str(Hydrate_threehalf_percent) + " pairs"
142                 print "Hydrate fraction higher by 4.0% for " +
                    str(Hydrate_four_percent) + " pairs"
143                 print "Hydrate fraction higher by 4.5% for " +
                    str(Hydrate_fourhalf_percent) + " pairs"
144                 print "Hydrate fraction higher by 5.0% for " +
                    str(Hydrate_five_percent) + " pairs"
145                 print "Waterless fraction higher by 0.5% for " +
                    str(Waterless_half_percent) + " pairs"
146                 print "Waterless fraction higher by 1.0% for " +
                    str(Waterless_one_percent) + " pairs"
147                 print "Waterless fraction higher by 1.5% for " +
                    str(Waterless_onehalf_percent) + " pairs"
148                 print "Waterless fraction higher by 2.0% for " +
                    str(Waterless_two_percent) + " pairs"
149                 print "Waterless fraction higher by 2.5% for " +
                    str(Waterless_twohalf_percent) + " pairs"
150                 print "Waterless fraction higher by 3.0% for " +
                    str(Waterless_three_percent) + " pairs"
151                 print "Waterless fraction higher by 3.5% for " +
                    str(Waterless_threehalf_percent) + " pairs"
152                 print "Waterless fraction higher by 4.0% for " +
                    str(Waterless_four_percent) + " pairs"
153                 print "Waterless fraction higher by 4.5% for " +
                    str(Waterless_fourhalf_percent) + " pairs"
```

```
                    print "Waterless fraction higher by 5.0% for " +
                    str(Waterless_five_percent) + " pairs"
                    print 'end of one round'

                    loop += 1
                    if loop == 1:
                        crystal1 = crystal_reader2
                        crystal2 = crystal_reader4




if __name__ == '__main__':
    # This runs the script
    r = Runner()
    r.run()
```