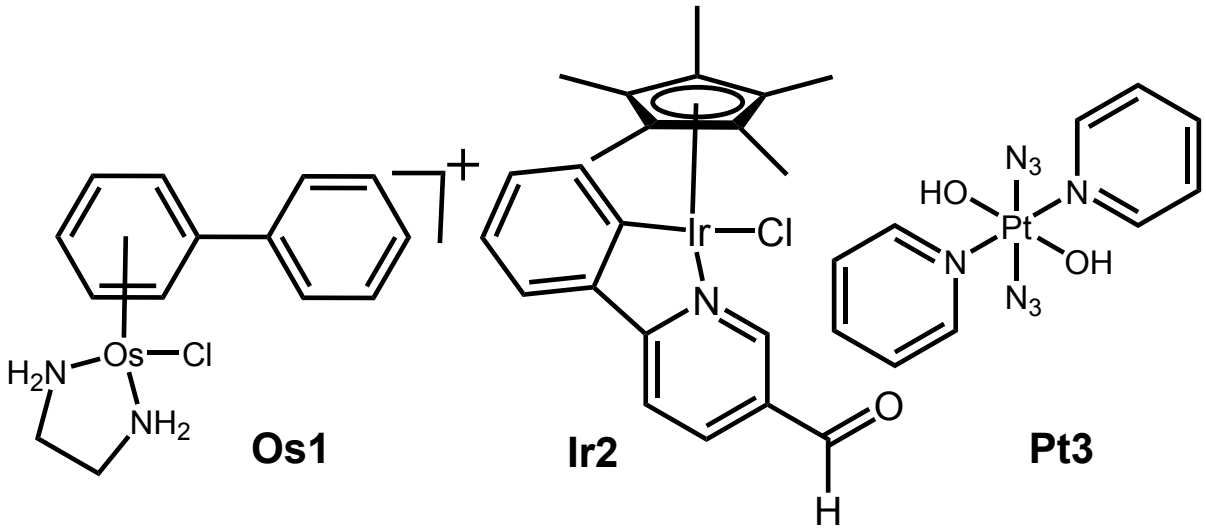
Electronic Supplementary Material (ESI) for Analyst.

Automated Assignment of Metal-containing Peptides in Proteomic LC-MS and MS/MS Datasets

Christopher A. Wootton,[a] Yuko P. Y. Lam, [a] Matthew Willetts,[b] Maria A. van Agthoven, [a] Mark P. Barrow, [a] Peter J. Sadler, [a] * Peter B. O'Connor [a] *

Supporting Information



SI Figure SF1: Structures of the metal anticancer complexes used to test the SNAP-LC algorithm. Left to right, Os1 an organo-osmium(II) arene ethylenediamine piano-stool complex, Ir2 an aldehyde-functionalised organo-iridium(III) Cp* phenylpyridine complex, and : Pt3, a photoactivatable platinum(IV) compound. The Ir and Os complexes can readily bind to targets by displacement of the chlorido ligand. The Pt(IV) complex is reduced to square-planar Pt(II) on irradiation with light and usually retains the two pyridine ligands.

In this work the major metallodrug fragments found on the peptide targets are $\{(biph)Os(en)\}^{2+}$; $\{Cp^*Ir(Phepy)\}^{+}$; $\{Pt(py)_2\}^{2+}$; $\{Pt(py)_2(N_3)\}^{+}$; $\{Pt(N_3)\}^{+}$ It should be noted that understanding of metallodrug-target interactions requires knowledge of the metal ion itself (and oxidation state) as well as the nature of the bound ligands. Both can play important roles in downstream biological processes.

SI Figure SF2: The SNAP-LC code used for analysis of metal-containing LCMS data:

```
'****************************************
****************************************
Option Explicit
SetLocale(1033) ' Localise for US (. instead , accepted as decimal separator for calculations)
Dim startSpectrum, NumSpectra, ProcessFullChrom, spectraToProcess, firstMass, lastMass
'****************************************
****************************************
ProcessFullChrom = TRUE 'Set to TRUE to process whole chromatogram, FALSE will use parameters below
startSpectrum =1   ' specify spectrum number for method to start from
NumSpectra = 2000          'in spectra
firstMass = 300 'lowest mass for SNAP
lastMass = 5000   'highest mass for SNAP
'****************************************
****************************************
Const cDebug = False
Const delim = ","
Const sHeader = "Rt,m/z,Charge,Intensity"
spectraToProcess = 0
Dim objMasses, objMassList
Const ForAppending = 8
Call Main()
objMassList.Close
Call Analysis.Save()
Call Form.Close()
 Sub Main
        Call ClearPreviousData
        Set objMasses = CreateObject("Scripting.FileSystemObject")
        Set objMassList = objMasses.OpenTextFile(Analysis.Path & "\" & "MassList.csv", ForAppending, True)
        objMassList.WriteLine(sHeader)
        spectraToProcess = NumSpectra+startSpectrum
        If ProcessFullChrom then
                spectraToProcess = Analysis.chromatograms(1).size
                startSpectrum = 1
        End If
        Call CreateSpectra(startSpectrum, spectraToProcess)
        Call AddChrom
        Analysis.Compounds.Clear
End Sub
Function CreateSpectra(startSpectrum, NumSpectra)
        Dim spec, x, intensity, rt, rtinsec, count, spectrum, peak
        Dim results()
        ReDim results(NumSpectra,1)
        count = 0
        Set spec = analysis.spectra
        For x = startSpectrum To (NumSpectra+startSpectrum)
                spec.add x, daProfileOnly
                spec.masslistfind firstMass,lastMass
                If spec.count >0 Then
                        Dim name, test
                        Set spectrum = spec(1)
                        name = spectrum.name
                        rt = CDbl(ExtractDecimal(spectrum.name))
                        rtinsec = rt*60
        If InStr(name, "MS2") Then 'If this is MSMS spectrum them write 0 and skip to next
                results(count,0) = rtinsec
                        results(count,1) = 0
                        Set spectrum = Nothing
                        name = ""
                Analysis.compounds.clear
                Else
                If cDebug then
        WScript.Echo(spectrum.name)
        WScript.Echo(AddIntensities(spectrum))
                End If
        results(count,0) = rtinsec
        results(count,1) = AddIntensities(spectrum)
        For Each peak In spectrum.msPeakList
        objMassList.WriteLine(rtinsec & delim & peak.m_over_z & delim & peak.ChargeState & delim & peak.Intensity)
        Next
        Analysis.compounds.clear
                End if
                End If
                If cDebug Then
                        WScript.Echo(x)
                End If
                count = count +1
        Next
        Call WriteIntensities(results)
End Function
Function ExtractDecimal(str)
        Dim re, newstr, result
        Set re = New RegExp
        With re
                .Pattern = "(\d+\.\d+)"
                Set    newstr = .Execute(str)
        End With
        If newstr.count <> 0 Then
```

```
                result =
newstr.Item(0).submatches.Item(0)
        End If
        ExtractDecimal = result
End Function
Function WriteIntensities(results)
        Dim objFSO, objTextFile, spec, y, sum, i,
rt, rtinsec, count
                Const ForAppending = 8
        Set objFSO =
CreateObject("Scripting.FileSystemObject")
        Set objTextFile =
objFSO.OpenTextFile(Analysis.Path & "\" &
"intensities.csv", ForAppending, True)
        For count = 0 To UBound(results)
                objTextFile.WriteLine(count & ","
& results(count,0) & "," & results(count,0) & "," &
results(count,1))
        Next
        objTextFile.Close
End Function
Sub ClearPreviousData
        Analysis.Compounds.Clear
        Call Delete_File(Analysis.Path &
"\Intensities.csv")
        Call Delete_File(Analysis.Path &
"\MassList.csv")
End Sub
Sub Delete_File (FileName)
        Dim fso
        Set fso =
CreateObject("Scripting.FileSystemObject")
        If fso.FileExists(FileName) Then
                fso.DeleteFile(FileName)
        End If
End Sub
Sub AddChrom
        Dim Chrom
  Set Chrom =
CreateObject("DataAnalysis.ImportedDataChromat
ogramDefinition")
  Chrom.FileName = Analysis.Path &
"\Intensities.csv"
  Analysis.Chromatograms.AddChromatogram
Chrom
End Sub
Function AddIntensities(spec)
        Dim intensities, peak
        intensities = 0
        For Each peak In spec.msPeakList
                intensities = intensities +
peak.Intensity
        Next
        AddIntensities = intensities
End Function
**********
```

SI Figure SF3: The iPython code used to search for unique species from the individual mass lists generated by SNAP-LC and generate the condensed list of modified species

****
****

In [1]:
```
import numpy as np
import math
import copy
import matplotlib
import matplotlib.pyplot as plt
import csv
from scipy import interpolate
```

In [2]:
```
def is_number(s):
    try:
        float(s)
        return True
    except ValueError:
        return False
```

In [3]:
```
def rounding_up_ppm(m, n):
    # n is the number of ppms and
m is the m/z ratio.
    p = -
int(math.log(m*n/1000000.0, 10))
    return round(m, p)
```

In [4]:
```
def
smooth(x,window_len=11,window='han
ning'):
    """smooth the data using a
window with requested size.
    This method is based on the
convolution of a scaled window
with the signal.
    The signal is prepared by
introducing reflected copies of
the signal
    (with the window size) in both
ends so that transient parts are
minimized
    in the begining and end part
of the output signal.
    input:
        x: the input signal
        window_len: the dimension
of the smoothing window; should be
an odd integer
        window: the type of window
from 'flat', 'hanning', 'hamming',
'bartlett', 'blackman'
            flat window will
produce a moving average
smoothing.
    output:
        the smoothed signal
    example:
    t=linspace(-2,2,0.1)
    x=sin(t)+randn(len(t))*0.1
    y=smooth(x)
    see also:
    numpy.hanning, numpy.hamming,
numpy.bartlett, numpy.blackman,
numpy.convolve
    scipy.signal.lfilter

    TODO: the window parameter
could be the window itself if an
array instead of a string
    NOTE: length(output) !=
length(input), to correct this:
return y[(window_len/2-1):-
(window_len/2)] instead of just y.
    """

    if x.ndim != 1:
        raise ValueError, "smooth
only accepts 1 dimension arrays."

    if x.size < window_len:
        raise ValueError, "Input
vector needs to be bigger than
window size."
    if window_len<3:
        return x
    if not window in ['flat',
'hanning', 'hamming', 'bartlett',
'blackman']:
        raise ValueError, "Window
is on of 'flat', 'hanning',
'hamming', 'bartlett', 'blackman'"
    s=np.r_[x[window_len-1:0:-
1],x,x[-1:-window_len:-1]]
    #print(len(s))
    if window == 'flat': #moving
average
        w=np.ones(window_len,'d')
    else:
w=eval('np.'+window+'(window_len)'
)
y=np.convolve(w/w.sum(),s,mode='va
lid')
    return y
```

In [5]:
```
"""Detect peaks in data based on
their amplitude and other
features."""
from __future__ import division,
print_function
import numpy as np
__author__ = "Marcos Duarte,
https://github.com/demotu/BMC"
__version__ = "1.0.4"
__license__ = "MIT"
def detect_peaks(x, mph=None,
mpd=1, threshold=0, edge='rising',
                 kpsh=False,
valley=False, show=False,
ax=None):
```

```
    """Detect peaks in data based
on their amplitude and other
features.
    Parameters
    ----------
    x : 1D array_like
        data.
    mph : {None, number}, optional
(default = None)
        detect peaks that are
greater than minimum peak height.
    mpd : positive integer,
optional (default = 1)
        detect peaks that are at
least separated by minimum peak
distance (in number of data).
    threshold : positive number,
optional (default = 0)
detect peaks (valleys) that are
greater (smaller) than `threshold`
in relation to their immediate
neighbors.
    edge : {None, 'rising',
'falling', 'both'}, optional
(default = 'rising')
        for a flat peak, keep only
the rising edge ('rising'), only
the falling edge ('falling'), both
edges ('both'), or don't detect a
        flat peak (None).
    kpsh : bool, optional (default
= False)
        keep peaks with same
height even if they are closer
than `mpd`.
    valley : bool, optional
(default = False)
        if True (1), detect
valleys (local minima) instead of
peaks.
    show : bool, optional (default
= False)
        if True (1), plot data in
matplotlib figure.
    ax : a matplotlib.axes.Axes
instance, optional (default =
None).
    Returns
    -------
    ind : 1D array_like
        indeces of the peaks in
`x`.
    Notes
    -----
    The detection of valleys
instead of peaks is performed
internally by simply
    negating the data:
`ind_valleys = detect_peaks(-x)`
    The function can handle NaN's
    See this IPython Notebook
[1]_.
    References
    ----------
    .. [1]
http://nbviewer.ipython.org/github
/demotu/BMC/blob/master/notebooks/
DetectPeaks.ipynb
    Examples
    --------
    >>> from detect_peaks import
detect_peaks
    >>> x = np.random.randn(100)
    >>> x[60:81] = np.nan
    >>> # detect all peaks and
plot data
    >>> ind = detect_peaks(x,
show=True)
    >>> print(ind)
    >>> x =
np.sin(2*np.pi*5*np.linspace(0, 1,
200)) + np.random.randn(200)/5
    >>> # set minimum peak height
= 0 and minimum peak distance = 20
    >>> detect_peaks(x, mph=0,
mpd=20, show=True)
    >>> x = [0, 1, 0, 2, 0, 3, 0,
2, 0, 1, 0]
    >>> # set minimum peak
distance = 2
    >>> detect_peaks(x, mpd=2,
show=True)
    >>> x =
np.sin(2*np.pi*5*np.linspace(0, 1,
200)) + np.random.randn(200)/5
    >>> # detection of valleys
instead of peaks
    >>> detect_peaks(x, mph=0,
mpd=20, valley=True, show=True)
    >>> x = [0, 1, 1, 0, 1, 1, 0]
    >>> # detect both edges
    >>> detect_peaks(x,
edge='both', show=True)
    >>> x = [-2, 1, -2, 2, 1, 1,
3, 0]
    >>> # set threshold = 2
    >>> detect_peaks(x, threshold
= 2, show=True)
    """
    x =
np.atleast_1d(x).astype('float64')
    if x.size < 3:
        return np.array([],
dtype=int)
    if valley:
        x = -x
    # find indices of all peaks
    dx = x[1:] - x[:-1]
    # handle NaN's
    indnan =
np.where(np.isnan(x))[0]
```

```
    if indnan.size:
        x[indnan] = np.inf
  dx[np.where(np.isnan(dx))[0]] =
np.inf
    ine, ire, ife = np.array([[],
[], []], dtype=int)
    if not edge:
        ine =
np.where((np.hstack((dx, 0)) < 0)
& (np.hstack((0, dx)) > 0))[0]
    else:
        if edge.lower() in
['rising', 'both']:
            ire =
np.where((np.hstack((dx, 0)) <= 0)
& (np.hstack((0, dx)) > 0))[0]
        if edge.lower() in
['falling', 'both']:
            ife =
np.where((np.hstack((dx, 0)) < 0)
& (np.hstack((0, dx)) >= 0))[0]
    ind =
np.unique(np.hstack((ine, ire,
ife)))
    # handle NaN's
    if ind.size and indnan.size:
        # NaN's and values close
to NaN's cannot be peaks
        ind = ind[np.in1d(ind,
np.unique(np.hstack((indnan,
indnan-1, indnan+1))),
invert=True)]
    # first and last values of x
cannot be peaks
    if ind.size and ind[0] == 0:
        ind = ind[1:]
    if ind.size and ind[-1] ==
x.size-1:
        ind = ind[:-1]
    # remove peaks < minimum peak
height
    if ind.size and mph is not
None:
        ind = ind[x[ind] >= mph]
    # remove peaks - neighbors <
threshold
    if ind.size and threshold > 0:
        dx =
np.min(np.vstack([x[ind]-x[ind-1],
x[ind]-x[ind+1]]), axis=0)
        ind = np.delete(ind,
np.where(dx < threshold)[0])
    # detect small peaks closer
than minimum peak distance
    if ind.size and mpd > 1:
        ind =
ind[np.argsort(x[ind])][::-1]  #
sort ind by peak height
        idel = np.zeros(ind.size,
dtype=bool)
        for i in range(ind.size):
            if not idel[i]:
                # keep peaks with
the same height if kpsh is True
                idel = idel | (ind
>= ind[i] - mpd) & (ind <= ind[i]
+ mpd) \
                    & (x[ind[i]] >
x[ind] if kpsh else True)
                idel[i] = 0  #
Keep current peak
        # remove the small peaks
and sort back the indices by their
occurrence
        ind = np.sort(ind[~idel])
    if show:
        if indnan.size:
            x[indnan] = np.nan
        if valley:
            x = -x
        _plot(x, mph, mpd,
threshold, edge, valley, ax, ind)
    return ind
def _plot(x, mph, mpd, threshold,
edge, valley, ax, ind):
    """Plot results of the
detect_peaks function, see its
help."""
    try:
        import matplotlib.pyplot
as plt
    except ImportError:
        print('matplotlib is not
available.')
    else:
        if ax is None:
            _, ax =
plt.subplots(1, 1, figsize=(8, 4))

        ax.plot(x, 'b', lw=1)
        if ind.size:
            label = 'valley' if
valley else 'peak'
            label = label + 's' if
ind.size > 1 else label
            ax.plot(ind, x[ind],
'+', mfc=None, mec='r', mew=2,
ms=8,
                    label='%d %s'
% (ind.size, label))
            ax.legend(loc='best',
framealpha=.5, numpoints=1)
        ax.set_xlim(-.02*x.size,
x.size*1.02-1)
        ymin, ymax =
x[np.isfinite(x)].min(),
x[np.isfinite(x)].max()
        yrange = ymax - ymin if
ymax > ymin else 1
        ax.set_ylim(ymin -
0.1*yrange, ymax + 0.1*yrange)
```

```
        ax.set_xlabel('Data #',
fontsize=14)
        ax.set_ylabel('Amplitude',
fontsize=14)
        mode = 'Valley detection'
if valley else 'Peak detection'
        ax.set_title("%s (mph=%s,
mpd=%d, threshold=%s, edge='%s')"
                     % (mode,
str(mph), mpd, str(threshold),
edge))
        # plt.grid()
        plt.show()
```

In [6]:

```
rt = []
mz6 = []
charge = []
I = []
with open('D:/Chris/SNAP1.csv',
'rb') as csvfile:
    reader = csv.reader(csvfile,
delimiter=',', quotechar='|')
    for row in reader:
        if is_number(row[0]):
rt.append(float(row[0]))
mz6.append(float(row[1]))
charge.append(int(row[2]))
I.append(float(row[3]))
```

In [8]:

```
mz3 = [rounding_up_ppm(mz6[i], 5)
for i in range(len(mz6))]
list_of_unique_mz = set(mz3)
print(list_of_unique_mz,
len(list_of_unique_mz))
set([583.25, 621.25, 924.75,
332.88, 643.25, 620.24, 1032.58,
686.27, 648.18, 1032.57, 666.76,
628.73, 1301.41, 1042.49, 337.39,
637.25, 725.81, 1330.49, 1207.93,
1049.57, 704.78, 1314.5, 1040.41,
361.15, 553.18, 665.76, 569.17,
906.2, 1301.4, 825.35, 910.46,
1038.56, 638.25, 1044.23, 778.19,
1038.25, 751.8, 1061.53, 1049.56,
793.43, 1130.42, 1022.4, 722.3,
1057.57, 314.63, 603.98, 920.75,
813.82, 930.42, 811.72, 919.68,
1153.25, 806.91, 904.86, 603.97,
920.41, 1239.46, 567.16, 551.16,
1050.55, 772.22, 1006.4, 796.65,
906.45, 362.91, 1170.88, 1084.19,
970.39, 1066.19, 904.98, 526.13,
696.25, 852.38, 1089.87, 674.76,
806.66, 1317.99, 766.77, 1044.26,
925.09, 510.14, 629.21, 648.75,
588.18, 1171.22, 717.81, 1202.6,
788.17, 887.34, 1318.0, 1213.93,
919.43, 512.72, 1075.2, 937.95,
805.38, 817.86, 1170.89, 537.15,
928.09, 825.86, 1046.26, 503.71,
930.09, 673.75, 1239.47, 1061.86,
930.08, 788.16, 629.25, 657.76,
912.86, 714.3, 692.72, 1043.89,
694.27, 842.38, 970.38, 887.0,
1348.5, 306.13, 512.15, 847.06,
1314.49, 1208.26, 920.42, 1548.35,
725.8, 528.15, 520.71, 1084.54,
902.2, 1166.93, 511.71, 995.79,
618.29, 592.19, 611.25, 1050.21,
936.88, 1057.56, 910.71, 878.42,
904.97, 878.67, 1075.54, 734.81,
1158.92, 630.26, 1548.34, 902.45,
787.91, 512.71, 612.25, 1139.5,
1084.53, 337.38]) 157
```

In [13]:

```
S = sorted(list_of_unique_mz,
key=float)
np.savetxt('D:/Chris/SNAP1_list_of
_unique_mz.csv', S, delimiter =
',')
```

In [10]:

```
B=[]
for j in range(len(S)):
    A=[]
    for i in range(len(mz3)):
        if mz3[i] ==S[j]:
            A.append(i)
    B.append(A)
```

In [12]:

```
for j in range(len(B)):
    C = []
    for i in B[j]:
        C.append((rt[i], mz6[i],
charge[i], I[i]))
        filename =
'D:/Chris/SNAP1_sorted/%.2f.csv'%S
[j]
        np.savetxt(filename, C,
delimiter = ',', header = 'm/z
%.2f'%S[j])
```

In [21]:

```
Chris = []
for j in range(len(S)):
    if len(B[j]) > 10:
        retention_time = [rt[i]
for i in B[j]]
        Intensity = [I[i] for i in
B[j]]
        m_over_z = [mz6[i] for i
in B[j]]
        Z = [charge[i] for i in
B[j]]
        f=
interpolate.interp1d(retention_tim
e, Intensity)
        RTnew =
np.arange(min(retention_time),
max(retention_time), 0.1)
        Inew = f(RTnew)
        if len(Inew)>121:
            Ismooth = smooth(Inew,
121, 'flat')
```

```
            T =
detect_peaks(Ismooth, 1000000,
1000, 170, 'rising', False, False,
False, None)
            for k in T:
                if k <=
len(RTnew):
                    for l in
range(len(B[j])):
                        if
RTnew[k]-0.5<
retention_time[l]<RTnew[k]+0.5:
Chris.append((S[j],
retention_time[l], m_over_z[l],
Z[l], Intensity[l]))
np.savetxt('D:/Chris/SNAP1_sorted.
csv', Chris, delimiter = ',',
header = 'unique m/z, retention
time (s), exact m/z, charge,
Intensity')
```

In [ ]:

```
**********************************
*******************************
```