

Supplementary Information

Bayesian optimization with known experimental and design constraints for chemistry applications

Riley J. Hickman,^{1,2,*} Matteo Aldeghi,^{1,2,3,4,*} Florian Häse,^{1,2,3,5} Alán Aspuru-Guzik^{1,2,3,6,7,8}

¹Chemical Physics Theory Group, Department of Chemistry, University of Toronto, Toronto, ON M5S 3H6, Canada

²Department of Computer Science, University of Toronto, Toronto, ON M5S 3G4, Canada

³Vector Institute for Artificial Intelligence, Toronto, ON M5S 1M1, Canada

⁴Department of Chemical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, United States

⁵Department of Chemistry and Chemical Biology, Harvard University, Cambridge, MA 02138, United States

⁶Department of Chemical Engineering & Applied Chemistry, University of Toronto, Toronto, ON M5S 3E5, Canada

⁷Department of Materials Science & Engineering, University of Toronto, Toronto, ON M5S 3E4, Canada

⁸Lebovic Fellow, Canadian Institute for Advanced Research, Toronto, ON M5G 1Z8, Canada

*These authors contributed equally

S.1. CONSTRAINED OPTIMIZATION OF THE ACQUISITION FUNCTION

Minima of the acquisition function, $\alpha(\mathbf{x})$ (main text Eq. 1), determine the parameter space point \mathbf{x} to be proposed for objective function measurement, thus, its optimization is an important subroutine in Bayesian optimization. Acquisition function optimization in GRYFFIN consists of two main steps. First, a set of N initial parameter space points are drawn from the domain \mathcal{X} using rejection sampling according to constraint function $c(\mathbf{x})$ producing the set of initial samples $\mathcal{P}_{\text{init}} = \{\mathbf{x}_i\}_{i=1}^N$, where $\mathbf{x}_i \sim \mathcal{X}$; s.t. $c(\mathbf{x}_i) \mapsto$ feasible. These samples are then refined by one of several optimization strategies, returning a set of refined proposals $\mathcal{P}_{\text{ref}} = \{\mathbf{x}_i\}_{i=1}^N$. In the following subsections, we detail the each acquisition function optimization strategy and compare their computational scaling.

A. Adam/Hill optimizer

For continuous parameters, the Adam optimizer¹ is used to optimize the acquisition function, with built in checks for whether the optimized samples obey the defined known constraints. For discrete and categorical parameter types, we use a “hill-climbing” strategy, in which each initial sample is randomly updated for a predefined number of iterations, with the candidate that has the best merit maintained and eventually returned. Algorithm 1 shows this strategy’s basic pseudocode.

B. Genetic optimizer

Our genetic algorithm implementation is based on the DEAP library^{2,3} and consists of crossover (\mathcal{C}), mutation (\mathcal{M}) and tournament selection (\mathcal{S}) operations. The size of the initial population is determined by the number of initial random samples, N . At each generation, offspring are chosen using tournament-style selection (tournament size of 3); elitism is applied to 5% of the population. Each parent in the resulting population is given a chance to mate (using either uniform or two-point crossover, depending on the dimensionality of the parameter space, with a crossover probability of 0.5), as well as a chance to mutate. We employ a custom mutation operator, \mathcal{M} which can handle continuous, discrete and categorical parameter types. The mutation probability is set to 0.4, with the probability of mutating each individual parameter set to 0.2. The maximum number of generations is set to 10, but the search is terminated early if the diversity of the population reaches a specified threshold. Specifically, if the population is concentrated in a small subvolume of parameter space, where each attribute does not span more than 10% of the allowed range, then the search is terminated. \mathcal{M} consists of different mutations for each parameter type. For continuous parameters, a perturbation x' is sampled from a Gaussian distribution with scale 0.1, i.e. $\mathcal{M}(x) = x + x'$, $x' \sim \mathcal{N}(0, 0.1)$. The same is true for discrete parameter types, except x' is first rounded to the nearest integer value. For categorical parameters, the mutated offspring are sampled randomly from the set of options for that parameter.

In order to constrain the genetic optimization procedure according to the known constraint function, a subroutine is used to project infeasible population offspring onto the feasibility boundary using a binary search procedure. After each application of \mathcal{C} or \mathcal{M} to parent \mathbf{x}_p , the feasibility of each resulting offspring \mathbf{x}_o is tested with $c(\mathbf{x}_o)$, where c is the user-defined constraints function. If $c(\mathbf{x}_o)$ returns **False**, i.e. the constraint is not satisfied and \mathbf{x}_o is in the infeasible region, the following procedure is employed project \mathbf{x}_o onto the boundary of the feasible region. For

Algorithm 1: Constrained acquisition function optimization with Adam/Hill climbing

Data: initial samples $\mathcal{P}_{\text{init}}$, acquisition function $\alpha(\cdot)$, constraint function $c(\cdot)$, max iterations i_{max}
Result: refined samples \mathcal{P}_{ref}

```

 $\mathcal{P}_{\text{ref}} \leftarrow \emptyset$ ;
for  $\mathbf{x}_n$  in  $\mathcal{P}_{\text{init}}$  do
  for  $i$  in  $i_{\text{max}}$  do
     $\mathbf{x}_n \leftarrow \text{AdamStep}(\mathbf{x}_n^{\text{cont}})$ ;           /* continuous optimization step */
     $\mathbf{x}_n \leftarrow \text{HillStep}(\mathbf{x}_n^{\text{cat}})$ ;       /* categorical optimization step */
     $\mathbf{x}_n \leftarrow \text{HillStep}(\mathbf{x}_n^{\text{disc}})$ ;     /* discrete optimization step */
    if  $c(\mathbf{x}_n) \mapsto \text{infeasible}$  then
       $\mathcal{P}_{\text{ref}} \stackrel{+}{\leftarrow} \mathbf{x}_n^{i-1}$ ;           /* add feasible sample from previous iteration to refined samples */
    else
       $\mathcal{P}_{\text{ref}} \stackrel{+}{\leftarrow} \mathbf{x}_n$ ;                 /* add current sample to refined samples */
    end
  end
end
Function HillStep( $\mathbf{x}_n$ ):
   $y_{\text{best}} \leftarrow \alpha(\mathbf{x}_n)$ ;
  for  $\mathbf{z}_d$  in  $\mathbf{x}_n^{\text{cat}}$  do
     $\mathbf{z}_d^{\text{new}} \leftarrow \mathbf{z}_d^{\text{new}} \sim \mathbb{S}_d$ ;           /* sample from set of categorical options  $\mathbb{S}_d$  */
     $\mathbf{x}_n^{\text{new}}$  updated with  $\mathbf{z}_d^{\text{new}}$ ;
     $y_{\text{new}} \leftarrow \alpha(\mathbf{x}_n^{\text{new}})$ ;
    if  $y_{\text{new}} < y_{\text{best}}$  then
       $y_{\text{best}} \leftarrow y_{\text{new}}$ ;
       $\mathbf{x}_n \leftarrow \mathbf{x}_n^{\text{new}}$ ;
    end
  end
return  $\mathbf{x}_n$ 

```

continuous parameters, we consider the midpoint \mathbf{x}_m of the line segment bounded by \mathbf{x}_p and \mathbf{x}_o . If \mathbf{x}_m is feasible, then the parent is set to the midpoint; if it is infeasible, the offspring is set to the midpoint. This process is repeated until the distance between \mathbf{x}_o and \mathbf{x}_p is below a tolerance threshold. As a default, we use the criterion $\|\mathbf{x}_p - \mathbf{x}_o\|_\infty < 0.01$ to terminate the search, i.e., when we are guaranteed to be within a 1% of relative distance from the feasibility boundary in all parameter dimensions. Throughout this procedure, \mathbf{x}_p is guaranteed to always be feasible while \mathbf{x}_o is always infeasible. Hence, once the search is terminated, the \mathbf{x}_o is set equal to \mathbf{x}_p and is returned. For discrete parameters, the same process is performed but the search is terminated when the closest point to the feasibility boundary is identified. Categorical parameters of infeasible offspring are instead simply reset to those of the feasible parent. When mixed continuous, discrete, and categorical parameters are present, we (i) set the categorical parameters of \mathbf{x}_o to those of its parent, to obtain \mathbf{x}'_o . If \mathbf{x}'_o is feasible, we return it, otherwise we (ii) perform the binary search procedure described above for the continuous and/or discrete parameters and obtain \mathbf{x}''_o . Then, we (iii) reset the categorical parameters of \mathbf{x}''_o to their original values in \mathbf{x}_o , obtaining \mathbf{x}'''_o . If \mathbf{x}'''_o is feasible, we return it, otherwise we return \mathbf{x}''_o . Given this approach relies on binary searches, it has a favorable logarithmic scaling and adds little overhead to the optimization of the acquisition function.

Algorithm 2 shows the basic pseudocode for our implementation. We show pseudocode for our custom mutation function \mathcal{M} (referred to as `Mutation`), but we omit definition of our subroutine which projects infeasible points to the feasible boundary for brevity. The function is referred to in Algorithm 2 by `ProjectToFeasible`. We direct the interested reader to the source code of GRYFFIN for more details (<https://github.com/aspuru-guzik-group/gryffin>).

C. Empirical time complexity of the Adam and genetic acquisition optimizers

Computational scaling experiments were carried out to compare the relative cost of the Adam and Genetic acquisition optimization strategies. The time taken to optimize the acquisition function was measured with increasing number of past observations while keeping the problem dimensionality constant, and with increasing number of parameter dimensions while keeping the number of observations constant. All parameters were continuous and in $[0, 1]$. Tests were performed with and without optimization constraints. When relevant, the constraint used was $\sum_{i=1}^d x_i \leq 0.5d$, where d is dimension of the parameter space and x_i are the individual elements of the d -dimensional parameter vector. That is, we assumed half of the optimization domain to be infeasible. Results of these tests are shown in Fig. S2. Each datapoint was obtained as the average elapsed time for 60 repeated acquisition function optimizations

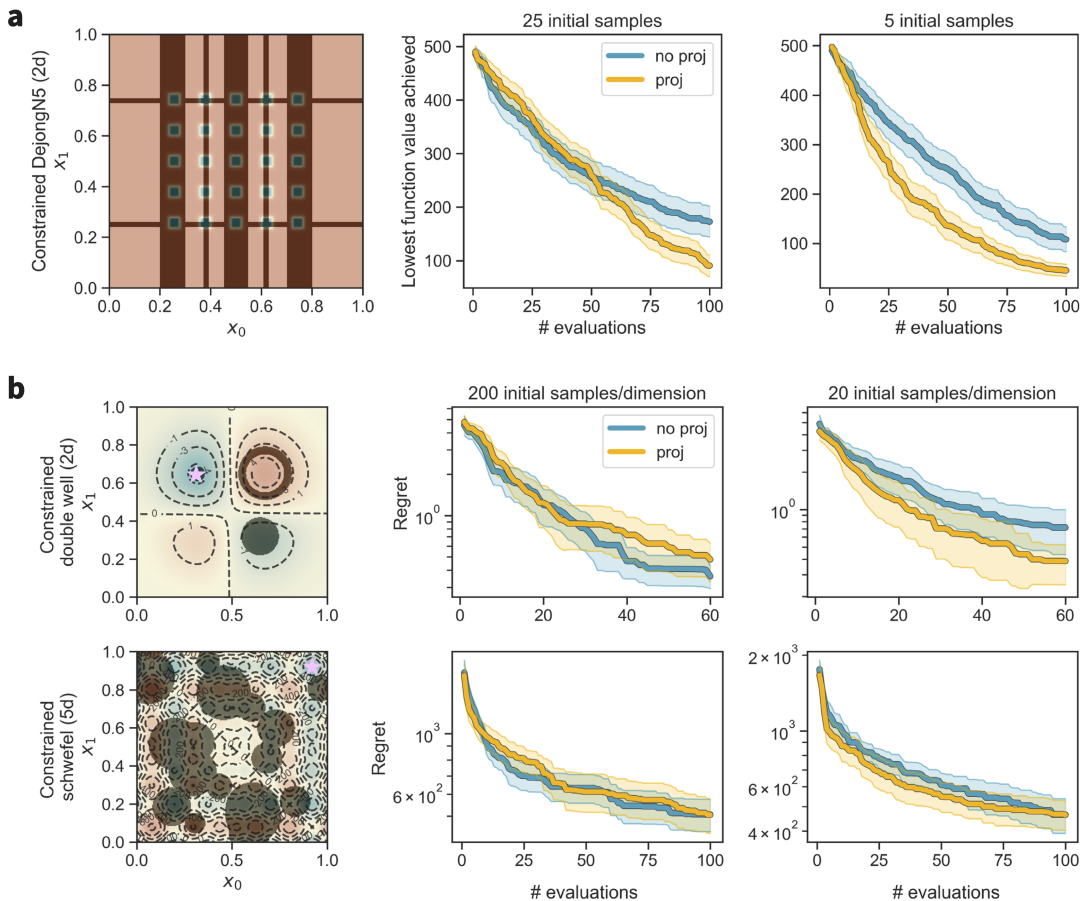


Figure S1. Performance of the genetic optimizer implemented in GRYFFIN, with (**proj**) and without (**no_proj**) the use of the `ProjectToFeasible` subroutine. (a) Effect of `ProjectToFeasible` on the optimization performance of the genetic optimizer, applied to the optimization of a two-dimensional surface. This additional feature of the optimizer improved optimization performance. As one may expect, the improvement becomes more evident as the size of the population (“initial samples”) is decreased. (b) Effect of `ProjectToFeasible` on the optimization performance of *Gryffin (Genetic)*. While less evident, here too the added feature can improve performance when the global optimum is located on the feasibility boundary and with low population sizes. Shaded regions indicate 95% confidence intervals from 100 independent executions.

(20 for each $\lambda = \{-1, 0, 1\}$). No appreciable overhead was observed when constraints were present. When keeping the dimensionality constant, the Genetic strategy showed favourable scaling compared to Adam, being roughly 20% as expensive as Adam after 100 observations. Similar results were observed in the experiments with a constant number of observations, where the optimization cost with the Genetic strategy took, on average, $\sim 60\%$ less time than Adam.

S.2. CONSTRAINED OPTIMIZATION OF ANALYTICAL FUNCTIONS

In this section, we provide more details about the constrained analytical functions used for testing GRYFFIN’s implementation and performance.

A. Benchmark functions and constraints used

Our synthetic benchmark experiments consisted of four continuous and four discrete surfaces in two dimensions. The original implementations of the surfaces can be accessed via the OLYMPUS package.⁴ We used Python wrappers for each of the surfaces to implement constraints on the parameter space. While the full implementation is available on GitHub, code snippets are provided here as well to show a user may implement different constraint functions to be used by GRYFFIN. These constraint functions, called `is_feasible()`, expect a dictionary, `params`, containing the

Algorithm 2: Constrained acquisition function optimization with genetic algorithm

Data: $\mathcal{P}_{\text{init}}$, $\alpha(\cdot)$, $c(\cdot)$, i_{max} , crossover operator \mathcal{C} with prob p_c , custom mutation operator \mathcal{M} with prob $p_{\mathcal{M}}$ and independent prob $p_{\mathcal{M}}^{\text{indep}}$, tournament selection operator \mathcal{S}

Result: refined population \mathcal{P}_{ref}

```

 $\mathcal{P} \leftarrow \mathcal{P}_{\text{init}}; f \leftarrow \alpha(\mathcal{P});$ 
for  $i$  in  $i_{\text{max}}$  do
   $\mathcal{O} \leftarrow \mathcal{S}(\mathcal{P});$  /* tournament selection of offspring  $\mathcal{O}$  from population  $\mathcal{P}$  */
  for  $\mathbf{x}_{\text{parent},1}^i, \mathbf{x}_{\text{parent},2}^i$  in mating pairs do
    if crossover sample  $\sim \mathcal{U}(0,1) < p_c$  then
       $\mathbf{x}_{\text{child},1}^i, \mathbf{x}_{\text{child},2}^i \leftarrow \mathcal{C}(\mathbf{x}_{\text{parent},1}^i, \mathbf{x}_{\text{parent},2}^i);$ 
       $\mathbf{x}_{\text{child},1}^i \leftarrow \text{ProjectToFeasible}(\mathbf{x}_{\text{child},1}^i, \mathbf{x}_{\text{parent},1}^i);$ 
       $\mathbf{x}_{\text{child},2}^i \leftarrow \text{ProjectToFeasible}(\mathbf{x}_{\text{child},2}^i, \mathbf{x}_{\text{parent},2}^i);$ 
    end
  end
  for  $\mathbf{x}_{\text{parent}}^i$  in  $\mathcal{O}$  do
    if mutant sample  $\sim \mathcal{U}(0,1) < p_{\mathcal{M}}$  then
       $\mathbf{x}_{\text{mutant}}^i \leftarrow \mathcal{M}(\mathbf{x}_{\text{parent}}^i);$ 
       $\mathbf{x}_{\text{mutant}}^i \leftarrow \text{ProjectToFeasible}(\mathbf{x}_{\text{mutant}}^i, \mathbf{x}_{\text{parent}}^i);$ 
    end
  end
   $f \leftarrow \alpha(\mathcal{O});$  /* evaluate the fitness  $f$  of offspring  $\mathcal{O}$  */
   $\mathcal{O} \stackrel{+}{\leftarrow} \mathcal{E};$  /* add elites  $\mathcal{E}$  to the offspring  $\mathcal{O}$  */
   $\mathcal{P} \leftarrow \mathcal{O};$  /* set population  $\mathcal{P}$  as the offspring  $\mathcal{O}$  for next generation */
end
 $\mathcal{P}_{\text{ref}} \leftarrow \mathcal{P};$ 
Function Mutation( $\mathbf{x}, p_{\mathcal{M}}^{\text{indep}}$ ):
  for  $x_d$  in  $\mathbf{x}$  do
    if independent mutation sample  $\sim \mathcal{U}(0,1) < p_{\mathcal{M}}^{\text{indep}}$  then
      if  $x_d$  is continuous then
         $x_d \leftarrow x_d + x';$  /* sample perturbation from unit Gaussian, i.e.  $x' \sim \mathcal{N}(0,1)$  */
      else if  $x_d$  is discrete then
         $x_d \leftarrow x_d + \text{round}(x', \text{integer});$  /* sample perturbation from unit Gaussian, i.e.  $x' \sim \mathcal{N}(0,1)$  */
      else if  $x_d$  is categorical then
         $x_d \leftarrow x';$  /* sample  $x'$  from set of categorical options, i.e.  $x' \sim \mathbb{S}_d$  */
      end
    end
  end
return  $\mathbf{x}$ 

```

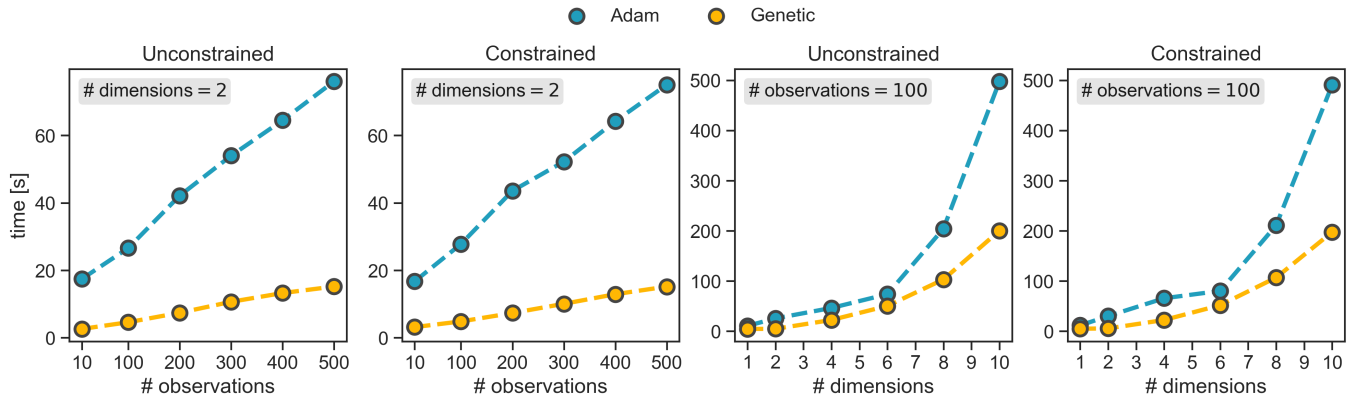


Figure S2. Empirical measurements of the time required by GRYFFIN to optimize its acquisition function. The Adam and Genetic optimization strategies were compared at varying number of past observations and optimization domain dimensions, with and without the presence of constraints.

parameter values and evaluate their feasibility; **True** is returned for feasible, **False** for infeasible. Note that, while here we report the definition of the analytical functions and the input domain typically used, OLYMPUS normalizes the input domain to the unit hypercube for ease of use (i.e. all analytical function can be expected to be supported in $[0, 1]^d$). As such, the constraint functions below assume each parameter to be normalized between zero and one.

- *Branin*: This surface is evaluated on the domain $x_1 \in [-5, 10]$, $x_2 \in [0, 15]$, and has the form $f(\mathbf{x}) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1 - t)\cos(x_1) + s$, with $a = 1$, $b = 5.1/4\pi^2$, $c = 5/\pi$, $t = 1/8\pi$. There are three degenerate global minima at $(x_1, x_2) = (-\pi, 12.275)$, $(\pi, 2.275)$ and $(9.42478, 2.475)$. Two of these minima were removed by the constraints defined below.

```
def is_feasible(params):
    x0 = params['x0']
    x1 = params['x1']
    y0 = (x0-0.12389382)**2 + (x1-0.81833333)**2
    y1 = (x0-0.961652)**2 + (x1-0.165)**2
    if y0 < 0.2**2 or y1 < 0.35**2:
        return False
    else:
        return True
```

- *Schwefel*: This surface is a complex optimization problem with many local minima. In d dimensions, it is evaluated on the hypercube $x_i \in [-500, 500] \forall i = 1, \dots, d$ and is described by the expression $f(\mathbf{x}) = 418.9829d - \sum_{i=1}^d x_i \sin(\sqrt{|x_i|})$. The surface has a global optima at $\mathbf{x} = (420.9687, \dots, 420.9687)$.

```
def is_feasible(params):
    np.random.seed(42)
    N = 20
    centers = [np.random.uniform(low=0.0, high=1.0, size=2) for i in range(N)]
    radii = [np.random.uniform(low=0.05, high=0.15, size=1) for i in range(N)]
    x0 = params['x0']
    x1 = params['x1']
    Xi = np.array([x0, x1])
    for c, r in zip(centers, radii):
        if np.linalg.norm(c - Xi) < r:
            return False
    return True
```

- *Dejong*: This surface generalizes a parabola to higher dimensions. It is convex and unimodal an evaluated on the d -dimensional hypercube $x_i \in [-5, 5]$, $\forall i = 1, \dots, d$. In two dimensions, this surface has a global minimum at $(x_0, x_1) = (0, 0)$ with $y = 0$.

```
def is_feasible(params):
    x0 = params['x0']
    x1 = params['x1']
    y = (x0-0.5)**2 + (x1-0.5)**2
    if np.abs(x0-x1) < 0.1:
        return False
    if 0.05 < y < 0.15:
        return False
    else:
        return True
```

- *DiscreteAckley*: This surface is the discrete analogue to the *Ackley* function.

```
def is_feasible(self, params):
    x0 = params['x0']
    x1 = params['x1']

    if np.logical_or(0.41 < x0 < 0.46, 0.54 < x0 < 0.59):
        return False
    if np.logical_or(0.34 < x1 < 0.41, 0.59 < x1 < 0.66):
```

```

    return False
return True

```

- *Slope* This surface generalizes a plane to discrete domains. The surface's values linearly increase along each dimension. Constraints form three area elements defined by circles with increasing radii.

```

def is_feasible(params):
    x0 = params['x0']
    x1 = params['x1']
    y = x0**2 + x1**2
    if 5 < y < 25:
        return False
    if 70 < y < 110:
        return False
    if 200 < y < 300:
        return False
    return True

```

- *Sphere*: This surfaces generalizes a parabola to discrete spaces. It features a degenerate global minimum if the number of options along at least one dimension is even, and a well-defined minimum if the number of options for all dimensions is odd. Constraints remove the same two integer inputs, 9 and 11, from consideration in both dimensions.

```

def is_feasible(params):
    x0 = params['x0']
    x1 = params['x1']
    if x0 in [9, 11]:
        return False
    if x1 in [9, 11]:
        return False
    return True

```

- *Michalewicz*: This surface features a sharper well where the global optimum is located. The number of psuedo-local minima scales factorially with the number of dimensions. Constraints consist of the area element between a circle centred around $(x_0, x_1) = (14, 10)$ with radii $\sqrt{5}$ and $\sqrt{30}$, as well as two rectangular areas.

```

def is_feasible(params):
    x0 = params['x0']
    x1 = params['x1']
    y = ((x0-14)**2 + (x1-10)**2)
    if 5 < y < 30:
        return False
    if 12.5 < x0 < 15.5:
        if x1 < 5.5:
            return False
    if 8.5 < x1 < 11.5:
        if x0 < 9.5:
            return False
    return True

```

- *Camel*: This surface features a degenerate and pseudo-disconnected global minimum. In 2-dimensions, it has global minima at $(x_0, x_1) = (7, 11)$ and $(x_0, x_1) = (14, 10)$. Constraints are generated by randomly sampling 100 infeasible locations and excluding the $(x_0, x_1) = (7, 11)$ optima.

```

def is_feasible(params):
    # choose infeasible points at random

```

```

num_opts = 21
options = [i for i in range(0,num_opts,1)]
num_infeas = 100
np.random.seed(42)
infeas_arrays = np.array([np.random.choice(options, size=num_infeas,replace=True),
                          np.random.choice(options, size=num_infeas, replace=True)])
infeas_tuples = [tuple(x) for x in infeas_arrays]

# always exclude the other minima
infeas_tuples.append((7, 11))
infeas_tuples.append((7, 15))
infeas_tuples.append((13, 5))

x0 = params['x0']
x1 = params['x1']
sample_tuple = (x0, x1)
if sample_tuple in infeas_tuples:
    return False
return True

```

In addition to the above functions, which were tested in two dimensions, we used higher-dimensional versions of the *AckleyPath* and *Schwefel* functions in Fig. S4. *AckleyPath* is the continuous analogue of the *DiscreteAckley* function and may be generalized to d dimensions. For *AckleyPath*, the constraints were randomly-generated hyper-rectangles in the parameter domain. For a d -dimensional function defined on the unit hypercube $[0,1]^d$, each rectangular constraint has $d - 1$ width parameters $\mathbf{w} = (w_1, w_2, \dots, w_{d-1})$ along with a start and end coordinate, e.g., $(x, 0, z)$ and $(x, 1, z)$. We generate $10d$ hyper-rectangles with parameters \mathbf{w} and coordinates sampled according to the following class definition.

```

class AckleyPathConstr(AckleyPath):
    def __init__(self, param_dim=2):
        AckleyPath.__init__(self, param_dim=param_dim)
        self.param_dim = param_dim
        np.random.seed(43)
        num_rect = int(param_dim*10)
        max_width = 0.025*(param_dim**1.8)
        self.ws, self.coords = [], []
        for _ in range(num_rect):
            w = np.random.uniform(0.05, max_width, size=(param_dim,))
            coord_1 = np.random.uniform(size=(param_dim,))
            coord_2 = coord_1.copy()
            ix = np.random.randint(param_dim)
            w[ix] = 0.
            coord_1[ix] = 0.
            coord_2[ix] = 1.
            self.ws.append(w)
            self.coords.append([coord_1, coord_2])

    def is_feasible(self, Xi):
        for w, coord in zip(self.ws, self.coords):
            bools = []
            for param_ix in range(self.param_dim):
                bool_ = np.logical_and(
                    Xi[param_ix] > coord[0][param_ix] - (w[param_ix]/2.),
                    Xi[param_ix] < coord[1][param_ix] + (w[param_ix]/2.)
                )
                bools.append(bool_)
            if all(bools):
                return False
        return True

```

For the *Schwefel* function generalized to d parameter dimensions, the constraint function is generalized from the random circles from our 2-dimensional example to randomly generated d -spheres. The equation of a d -sphere with

radius r centered at coordinate $\mathbf{c} = (c_1, c_2, \dots, c_d)$ is $r^2 = \sum_{i=1}^d (x_i - c_i)^2$. We sampled $10d$ spheres with randomly generated parameters r and \mathbf{c} according to the following class definition.

```

class SchwefelConstr(Schwefel):
    def __init__(self, param_dim=5):
        Schwefel.__init__(self, param_dim=param_dim)
        np.random.seed(42)
        num_spheres = int(param_dim*10)
        max_radius = np.sqrt(param_dim*1.**2) / 4.6
        # generate d-sphere centres and radii
        self.centers = [np.random.uniform(size=(param_dim,)) for _ in range(num_spheres)]
        self.radii = [np.random.uniform(0.1, max_radius) for _ in range(num_spheres)]

    def is_feasible(self, Xi):
        for c, r in zip(self.centers, self.radii):
            if np.linalg.norm(c-Xi) < r:
                return False
        return True

```

B. Results of the constrained optimization benchmarks

Fig. S3 shows the results of the continuous optimization benchmarks where regret is displayed on a linear scale, which highlights how performance differences between GRYFFIN and DRAGONFLY on *Branin* and *Dejong* are marginal. Table S1 reports the optimization performance achieved by the strategies tested on the discrete surfaces.

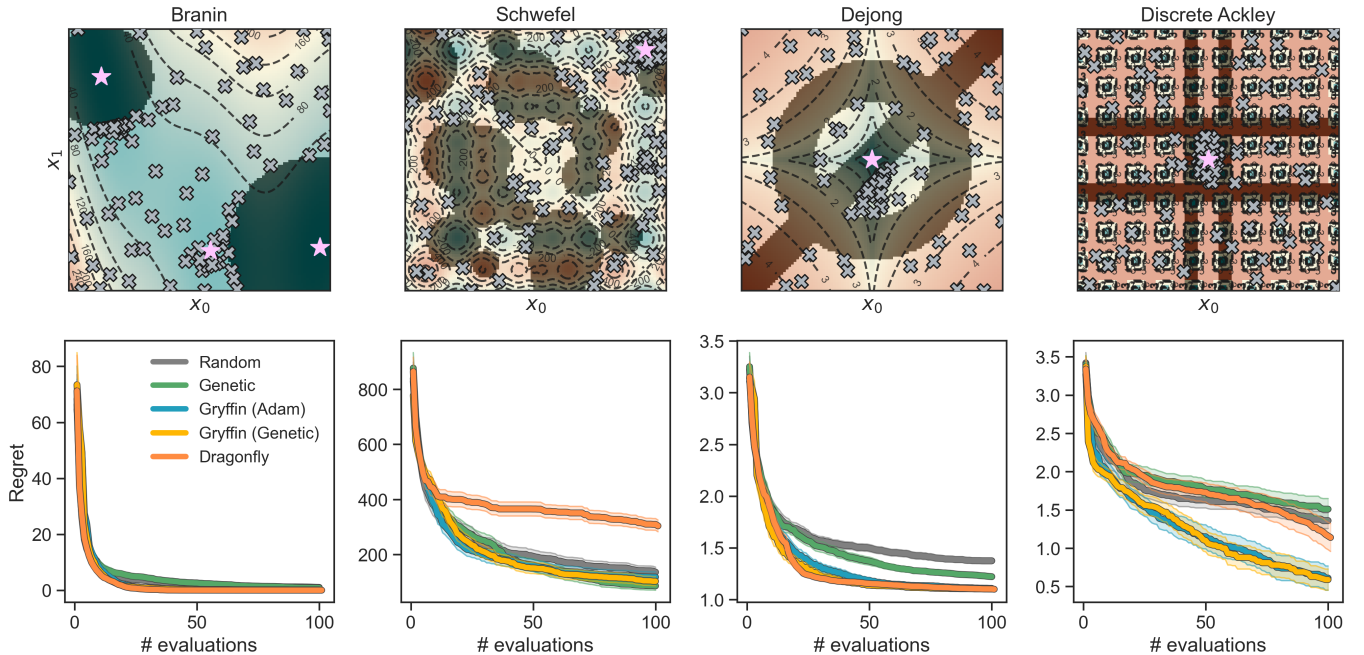


Figure S3. Constrained optimization benchmarks on analytical functions with continuous parameters. The upper row shows contour plots of the surfaces with constrained regions darkly shaded. Gray crosses show sample observation locations and purple stars denote the location(s) of unconstrained global optima. The bottom row show optimization traces for each strategy. Shaded regions around the solid trace represent 95% confidence intervals.

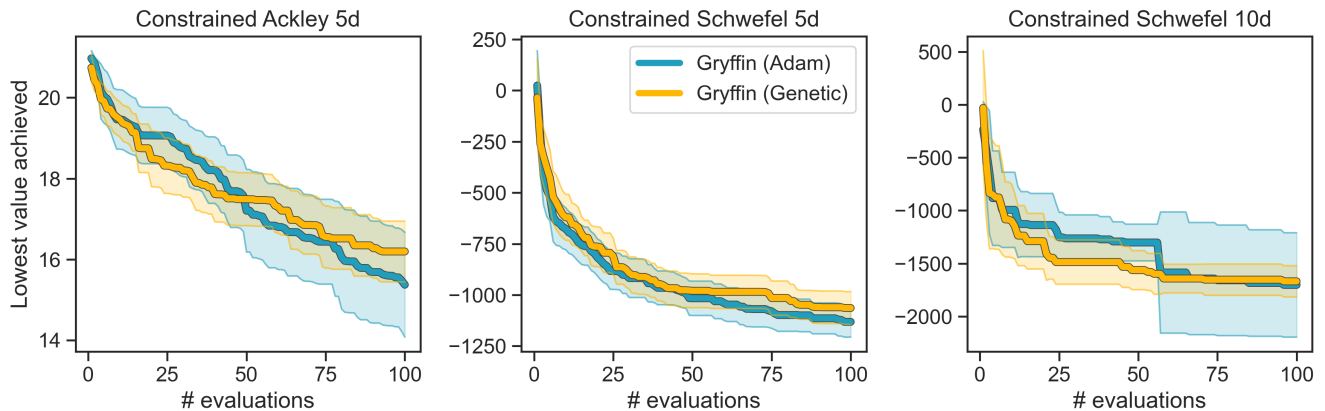


Figure S4. Performance comparison of *Gryffin (Adam)* and *Gryffin (Genetic)* on high-dimensional analytical functions. These results show that, even for higher dimensional surfaces, there is no significant difference in performance between *Gryffin (Adam)* and *Gryffin (Genetic)*. While one may expect the genetic approach to be more suited to more tortuous search landscapes, GRYFFIN’s acquisition function is generally smooth, such that it is unlikely to pose a particularly challenging optimization problem for either approach. Shaded regions indicate 95% confidence intervals from 25 independent executions of each strategy.

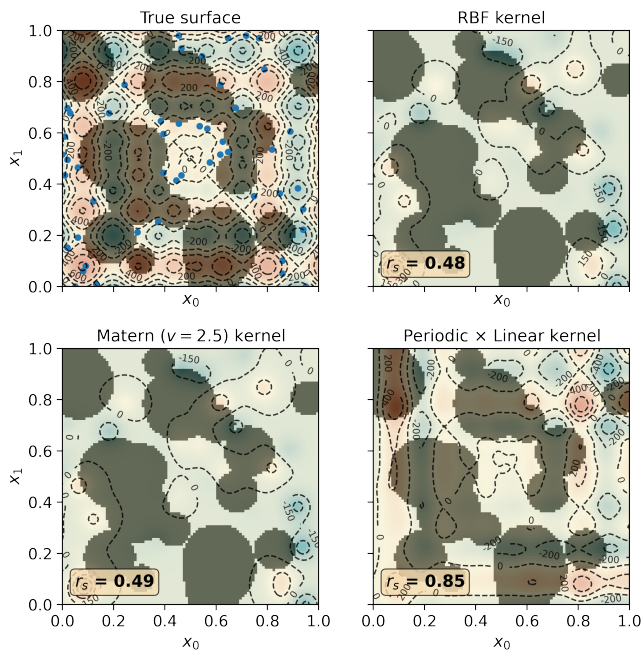


Figure S5. Gaussian process regression on the *Schwefel* surface using different kernels. The training data (50 samples) is shown as blue, circular markers on the top-left plot. The Spearman correlation coefficients (r_s) between the predictions of and the true Schwefel function evaluated in the feasible region are reported. While RBF and Matern are the most common kernels used for Bayesian optimization, a periodic kernel provides a better inductive bias for this surface. However, this is not known at the beginning of a black-box optimization.

C. Empirical comparison of sampling in Gryffin and Dragonfly

In this section, we examine the sampling tendencies of GRYFFIN and DRAGONFLY on the constrained, continuous analytical benchmark functions. Specifically, we compare the tendency of each algorithm to suggest parameter point which are in close proximity to past observations. The first row of Fig. S6 shows the minimum Euclidean distance between any two parameter points selected during an optimization campaign by each planner (boxplots show this metric over the 100 independently seeded runs). *Dragonfly* is able to recommend parameter points which are significantly closer to past observations than *Gryffin (Adam)* or *Gryffin (Genetic)*. The greater exploitative tendency of

–	Slope (311)	Sphere (362)	Michalewicz (323)	Camel (347)
Random	157.3 ± 9.4	162.3 ± 9.7	167.8 ± 9.2	171.0 ± 10.8
Genetic	55.2 ± 2.7	61.5 ± 2.7	47.7 ± 2.4	92.9 ± 6.2
GRYFFIN (Hill)	12.7 ± 1.0	19.0 ± 0.8	18.4 ± 0.9	33.8 ± 1.6
GRYFFIN (Genetic)	12.4 ± 1.0	20.2 ± 0.8	18.7 ± 0.8	36.0 ± 2.6
DRAGONFLY	11.0 ± 0.1	13.6 ± 0.3	29.8 ± 1.2	39.0 ± 2.3

Table S1. Mean and standard error of the number of evaluations needed by each strategy to identify the global optimum of each constrained discrete surface tested. The integer in parentheses in the header is the number of feasible tiles for the surface after the constraint is applied, out of a total of $21 \times 21 = 442$ input combinations.

Dragonfly is beneficial on smooth continuous surfaces as it allows for marginal improvement on regret values (main text Fig. 2). *Gryffin* strategies, on the other hand, contain a self-avoidance routine which biases the search away from past observations in an attempt to avoid redundant measurements. For practical experimental applications in chemistry, the resolution on input parameters is determined by precision of laboratory equipment and/or human error, and should be considered before commencing the experiment. The bottom two rows show the location of observations for *Gryffin (Adam)* and *Dragonfly* strategies around the minima of each surface. Visually, it is apparent that *Dragonfly* has a greater tendency to recommend parameter points which are considerably closer to past observations than does *Gryffin*.

S.3. PROCESS-CONSTRAINED OPTIMIZATION OF *O*-XYLENYL C₆₀ ADDUCTS SYNTHESIS

A. Details of the Bayesian neural network experiment emulator

To emulate the process-constrained synthesis of C₆₀ adducts, we trained a Bayesian neural network (BNN) to return stochastic outcomes based on a set of controllable parameters. The trained emulator takes a vector containing the experimental conditions (T , $F_{C_{60}}$, and F_S) and predicts the mole fractions of the products, the un- ($[X_0]$), singly- ($[X_1]$), doubly- ($[X_2]$), and triply-functionalized ($[X_3]$) C₆₀. The BNN consisted of 3 densely-connected variational layers with reparameterized Monte Carlo estimators⁵ and was implemented in PyTorch.⁶ Each hidden layer had 64 nodes and featured a ReLU non-linearity, while the output layer had 4 nodes, one for each of the aforementioned C₆₀ adducts. The output layer used the softmax activation function, which normalizes the outputs to a probability distribution where $\sum_{i=0}^3 [X_i] = 1$. Network weights w_i and biases b_i followed Gaussian distributions whose priors were set to have zero mean and unit standard deviation, i.e. $w_i \sim \mathcal{N}(0, 1)$, $b_i \sim \mathcal{N}(0, 1)$. The network was trained using variational Bayesian inference. The ELBO loss was minimized using the Adam optimizer¹, and resulting gradients were used to adjust the weights and biases of the network’s parameter distributions during training. Fig. S7 shows parity plots of our model’s predictions against the true C₆₀ adduct mole fractions, where 500 experimental measurements were used for training and 100 for testing. The BNN displayed excellent interpolation performance across the parameter space for each adduct type, with Pearson correlation coefficients on the test sets between 0.93 and 0.96.

B. Estimating the experimental cost

The overall goal of the process-constrained optimization of *o*-xylenyl C₆₀ adducts synthesis is to adjust reaction conditions such that the combined yield of first- and second-order adducts is maximized and reaches at least 90%, while the cost of reagents is minimized. In order to estimate the cost of the experiments, we considered the listed price of sultine and C₆₀ by the chemical supplier Sigma-Aldrich. The cost of dibromo-*o*-xylene cost on Sigma Aldrich was \$191 for 100 g. The cost of C₆₀ was \$422 for 5g. In the experiments by Walker *et al.*⁷, the concentration of sultine was 1.4 mg/mL, while the concentration of C₆₀ was 2.0 mg/mL. The amount of C₆₀ used in the experiments will therefore have much greater influence on overall experiment cost than sultine. Our optimization experiments target the adjustment of volume flow rates of each of these chemicals. Thus, we seek a measure of per-unit-time operation cost to be minimized. Converting to per-litre costs, we have 2.674 \$/L for sultine, and 168.8 \$/L for C₆₀. Finally, from the flow rates used in the experiment, F_C and F_S (with units of $\mu\text{L}/\text{min}$), we obtain an estimate of per-minute operation cost of the flow-reactor from Walker *et al.*⁷ with units of \$/min as

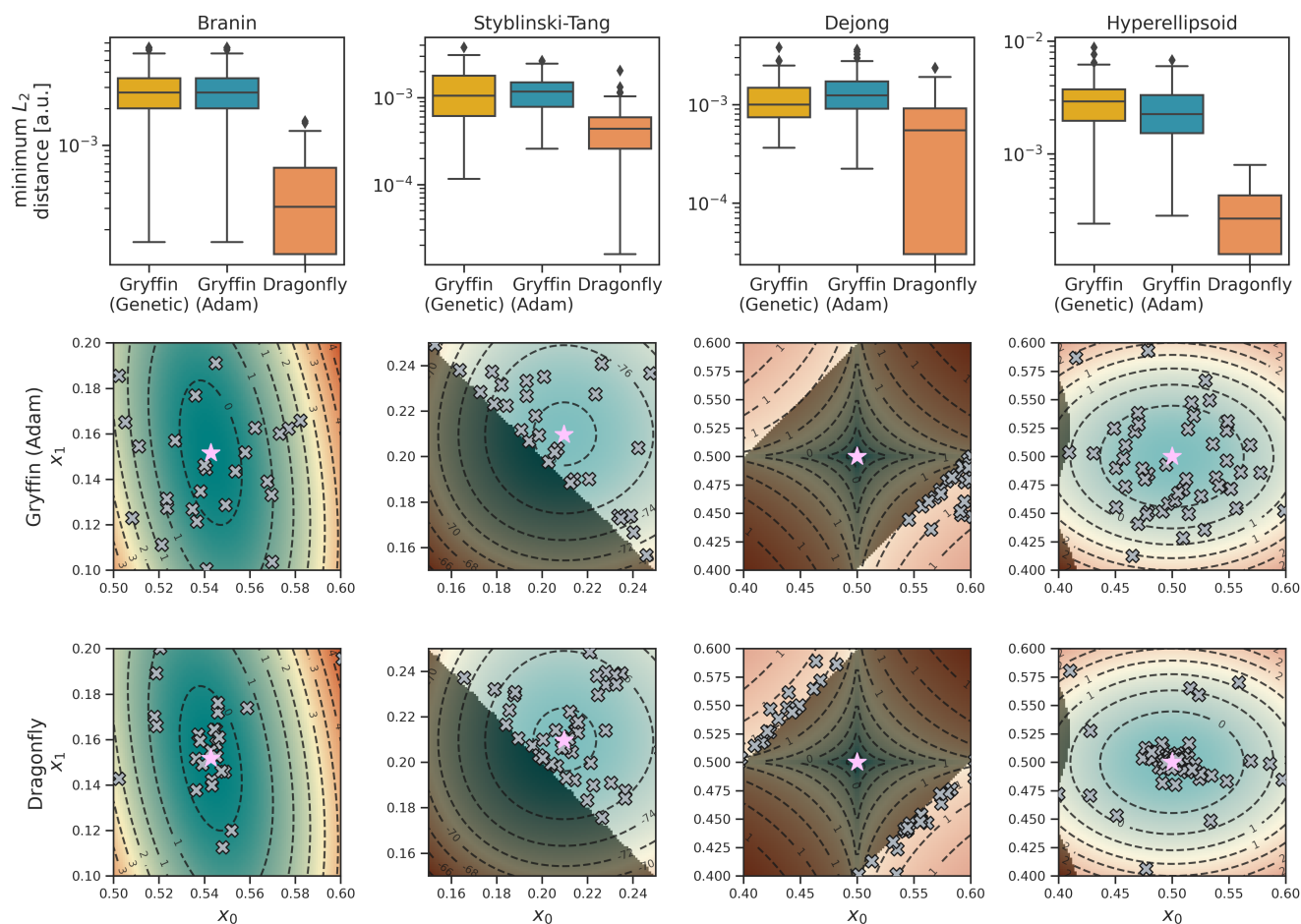


Figure S6. Empirical evaluation of the sampling behaviour of *Gryffin* and *Dragonfly* on constrained continuous surfaces. The first row shows the minimum Euclidean distance between any two parameter points selected by the each optimization strategy. For each continuous constrained surface, *Dragonfly* allows for recommendation of parameter points which are significantly closer to past observations than does *Gryffin (Adam)* or *Gryffin (Genetic)*. The second and third rows shows the location of *Gryffin (Adam)* and *Dragonfly* samples (grey crosses) in the vicinity of the surface minima (pink star).

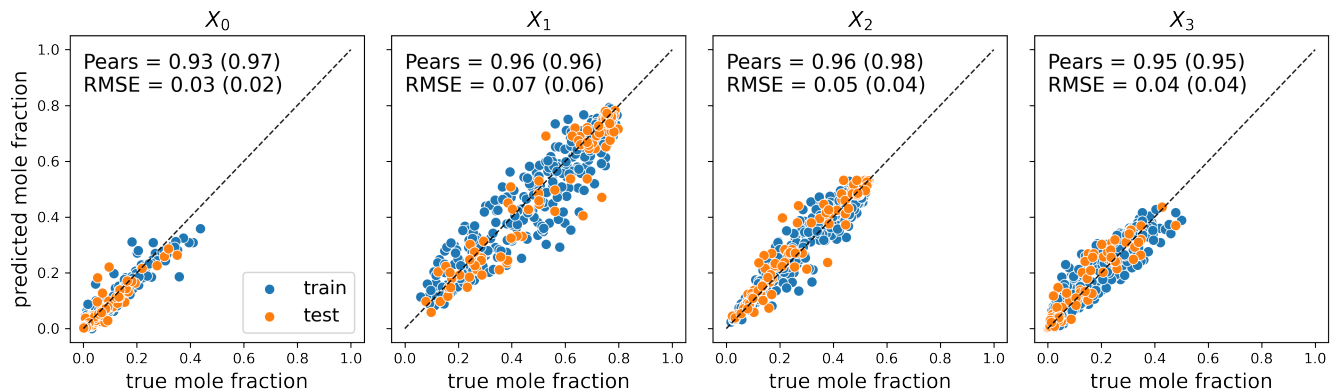


Figure S7. Parity plots for each mole fraction predicted by our Bayesian neural network emulator, averaged over 50 network parameter samples. Horizontal axes plot the true mole fraction, and vertical axes plot the predicted mole fraction. The dashed diagonal line indicates perfect agreement. The Pearson correlation coefficient and root-mean-square error is given for the training set (in parentheses) and test set for each mole fraction target. Train (test) set points are shown in blue (orange).

$$\text{cost} = \frac{1\text{L}}{10^6 \mu\text{L}} F_C \times \frac{\$168.8}{\text{L}} + \frac{1\text{L}}{10^6 \mu\text{L}} F_S \times \frac{\$2.674}{\text{L}}. \quad (3)$$

Fig. S8 shows the mean and 95% confidence interval for parameter values corresponding to the best observed objective values achieved by each optimization strategy. We report the flow rate in terms of *mass* per unit time (mass flow rate) to account for the difference in concentrations of each reagent and compare the rates on an equal footing. To improve upon our secondary cost objective, each strategy decreases the F_C parameter, as it's value dominates the cost in Eq. 3. Decrease in F_C is however accompanied by a decrease in F_S to preserve the high (≥ 0.9) mol fractions of the X_1 and X_2 adducts. For most optimization runs, the temperature of the best performing reactions varies between 116 and 130 °C.

Fig. S9 shows distributions of $F_C - F_S$ values for the best reaction conditions achieved by each optimization strategy in units of $\mu\text{g}/\text{min}$. For each strategy, we note that this distribution favours positive values, meaning that, in the majority of the best achieved reaction conditions, the mass flow rate of C_{60} was greater than that of sultine. Crucially, the *Gryffin* strategies, which exhibited the best optimization performance on this application, achieved narrower distributions around $F_C - F_S = 0$ than other strategies.

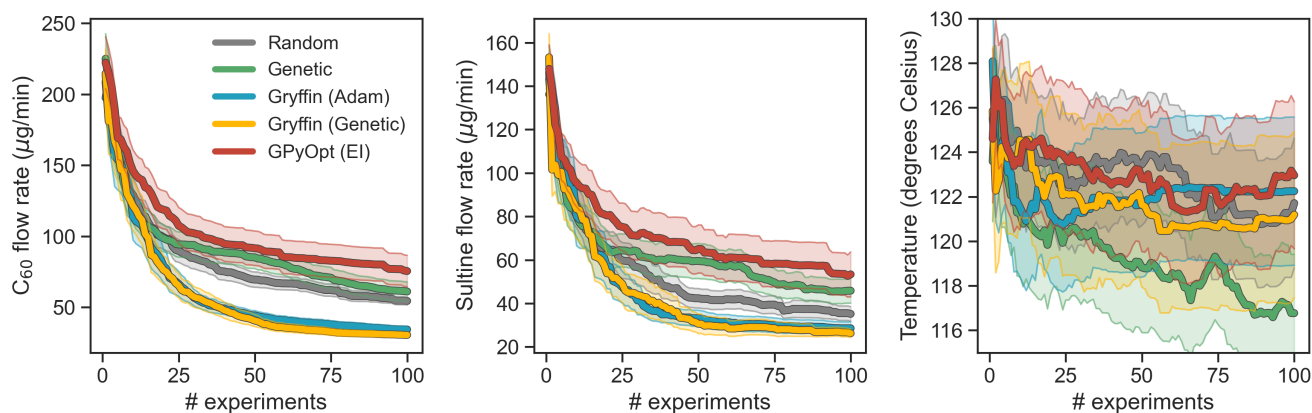


Figure S8. Mean and 95% confidence interval for parameter values corresponding to the best objective values found by each optimization strategy at each iteration of the optimization campaign. C_{60} and sultine flow rates are both shown with units of $\mu\text{g}/\text{min}$.

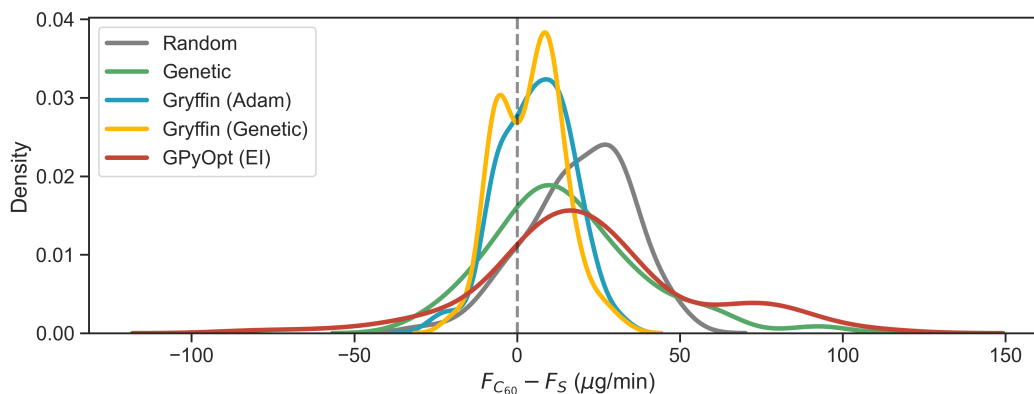


Figure S9. Kernel density estimates show the distribution of $F_C - F_S$ for the best reactions conditions achieved by each optimization strategy in units of $\mu\text{g}/\text{min}$. Each distribution is comprised of 100 such values, one for each independently seeded optimization. Positive values indicate that the best achieved reaction conditions had $F_C > F_S$.

S.4. DESIGN OF REDOX-ACTIVE MATERIALS FOR FLOW BATTERIES WITH SYNTHETIC ACCESSIBILITY CONSTRAINTS

A. Computation of reduction potential tolerance

To set the reduction potential (E^{red}) upon which we would like to improve, and which is used as an absolute tolerance in CHIMERA⁸, we computed E^{red} for the base scaffold molecule **H-AcBzC₆**.⁹ We computed E^{red} with the same computational protocol used by Agarwal *et al.*⁹. The DFT calculation was performed using Gaussian 16¹⁰ at the wb97xd/6-31+G-(d,p)^{11,12} level of theory. Optimized neutral and anionic geometries were subject to frequency calculations to compute the free energies. The SMD continuum model¹³ was used with acetonitrile as the solvent. The reduction potential was calculated using Eq. 1 in Agarwal *et al.*⁹,

$$E^{\text{red}} = \frac{-\Delta G^{\text{red}}}{nF} - 1.24 \text{ V}, \quad (4)$$

where $\Delta G^{\text{red}} = G^{\text{reduced}} - G^{\text{neutral}}$, n is the number of electrons added to the neutral molecules ($n = 1$), F is Faraday’s constant in eV, and 1.24 is a constant subtracted to convert the Gibbs free energy change to reduction potential (with Li/Li⁺ reference electrode). The E^{red} for **H-AcBzC₆** was computed to be 2.038372 V. Of the 1408 functional derivatives subject to computation by Agarwal *et al.*⁹, only 243 had better (lower) E^{red} .

B. Prediction of the synthetic accessibility of redoxmer candidate molecules

As a constraint on the redoxmer candidates space, we enforce a retrosynthetic accessibility threshold below which the candidate is considered infeasible. The goal was to have an indication of synthetic accessibility that could be used to constrain the search space to candidates that likely to be synthesizable in practice.

Fig. S10 shows the distributions of different synthetic accessibility scores for the set of 1408 redoxmer candidates considered in this application. Specifically, it includes the *RAScore*¹⁴ predicted by an XGBoost classifier (XGB) and a neural network (NN), the fragment-based synthetic accessibility score *SAScore*¹⁵, and the synthetic Bayesian classifier (*SYBA*)¹⁶. The *RAScore* is a recently reported synthetic accessibility score that tries to capture the probability of *AiZynthFinder* being able to identify a synthetic route for the molecule being evaluated.¹⁴ *AiZynthFinder* is a retrosynthetic planning tool that can generate synthetic routes for organic molecules.¹⁷ Hence, an *RAScore* of 1 indicates a synthetic path to the desired molecule is likely to exist, while a score of 0 indicates that finding a synthetic path is likely to be challenging and potentially impossible. For the purpose of our constrained optimization experiments, we decided to use the *RAScore* based on a NN model given its reported performance¹⁴ and intuitive interpretation.

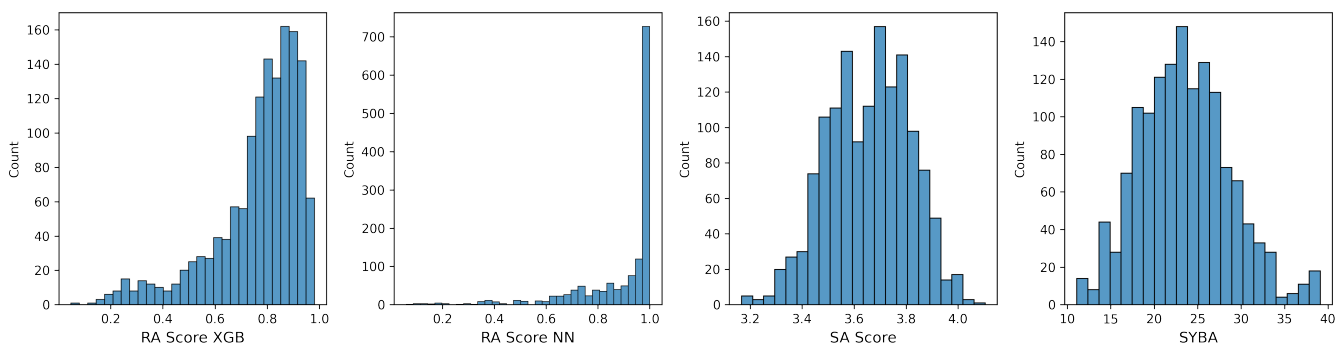


Figure S10. Histograms showing the distributions of four synthetic accessibility scores computed for the 1408 redoxmer candidates.

C. Generation of descriptors for benzothiadiazole scaffold substituents

In this example application of constrained Bayesian optimization, we employed the *Dynamic* version of GRYFFIN for combinatorial optimization,¹⁸ which can take advantage of physicochemical descriptors in the search for optimal

molecules. Specifically, we provided GRYFFIN with a total of seven simple descriptors associated with each of the four substituent groups considered (R_{1-4} in Fig. 5a). The physicochemical descriptors were computed with the *Mordred* Python package.¹⁹ As summarized in Table S2, the following descriptors were considered: the number of hetero atoms (**nHetero**), molecular weight (**MW**), topological polar surface area (**TopoPSA**), number of heavy atoms (**nHeavyAtom**), atomic polarizability (**apol**), fraction of sp^3 hybridized carbons (**FCSP3**), and geometric diameter (**Diameter**). All seven descriptors were used for substituent groups R_{2-4} , but only four of them are used for R_1 . We eliminate **apol**, **FCSP3** and **Diameter** from consideration because they each have equal value for both R_1 substituent options, and therefore are not informative. Table S2 summarizes the Pearson correlation of each descriptor with each objective value over the entire set of 1408 molecules. N/A entries show the cases where the descriptor is omitted for the R_1 substituent. In addition, Table S2 reports the Pearson correlations between the descriptors for all four R-groups (ρ_1 , ρ_2 , ρ_3 , and ρ_4) and each optimization objective ($\Delta\lambda^{\text{abs}}$, E^{red} , and G^{solv}). Table S3 reports instead the pairwise correlation between each descriptor, averaged over all R-groups.

Mordred name	$\Delta\lambda^{\text{abs}}$				E^{red}				G^{solv}			
	ρ_1	ρ_2	ρ_3	ρ_4	ρ_1	ρ_2	ρ_3	ρ_4	ρ_1	ρ_2	ρ_3	ρ_4
nHetero	0.17	0.13	0.12	0.19	0.22	0.27	0.26	0.35	0.62	0.24	0.22	-0.12
MW	0.17	0.06	0.05	0.15	0.22	0.21	0.20	0.19	0.62	0.23	0.21	-0.13
TopoPSA	-0.17	-0.12	-0.11	-0.14	-0.22	0.13	0.11	0.30	-0.62	0.00	0.01	-0.22
nHeavyAtom	0.17	0.09	0.07	0.20	0.22	0.22	0.21	0.22	0.62	0.22	0.20	-0.21
apol	-0.17	-0.25	-0.31	-0.14	-0.22	-0.26	-0.27	-0.16	-0.62	-0.06	-0.04	-0.41
FCSP3	N/A	-0.11	-0.06	-0.04	N/A	-0.31	-0.32	-0.36	N/A	0.13	0.14	0.02
Diameter	N/A	0.02	-0.04	0.17	N/A	0.16	0.16	0.23	N/A	0.14	0.12	-0.29

Table S2. Mordred descriptors used to describe R-groups for the battery application optimization. The right most three columns show the Pearson correlation between each descriptor and each optimization objective for the 1408 redoxmer candidates considered. The correlations for each of the four R groups are comma separated, i.e. $\rho_{R1}, \rho_{R3}, \rho_{R4}, \rho_{R5}$. The largest correlation for each objective and R-group is bolded. N/A entries indicate that this descriptor was not considered for this particular R group. For the R_1 group, we do not consider **apol**, **FCSP3** and **Diameter** since their values are the same for both R_1 options and therefore provide no additional information.

—	nHetero	MW	TopoPSA	nHeavyAtom	apol	FCSP3	Diameter
nHetero	1.00	0.92	0.01	0.91	-0.04	0.2	0.61
MW	0.92	1.00	-0.06	0.93	0.24	0.29	0.71
TopoPSA	0.01	-0.06	1.00	-0.14	0.03	-0.20	-0.13
nHeavyAtom	0.91	0.93	-0.14	1.00	0.32	0.35	0.86
apol	-0.04	0.24	0.03	0.32	1.00	0.47	0.62
FCSP3	0.2	0.29	-0.2	0.35	0.47	1.00	0.32
Diameter	0.61	0.71	-0.13	0.86	0.62	0.32	1.00

Table S3. Pairwise Pearson correlations between Mordred descriptors used to describe the R-groups of the redoxmer candidates.

D. Additional optimization experiments

In addition to GRYFFIN optimizations taking advantage of physicochemical descriptors (*Dynamic Gryffin*), we also carried out optimizations without this additional information using *Naive Gryffin*. Fig. S11 shows the optimization performance of all strategies tested, including the latter. The results show how the use of descriptors provide an edge to GRYFFIN to achieve superior performance to all other strategies. Regardless, *Naive Gryffin* still outperforms model-free optimization strategies *Random* and *Genetic*. All these optimizations were constrained to molecules with high synthetic accessibility scores, as described above.

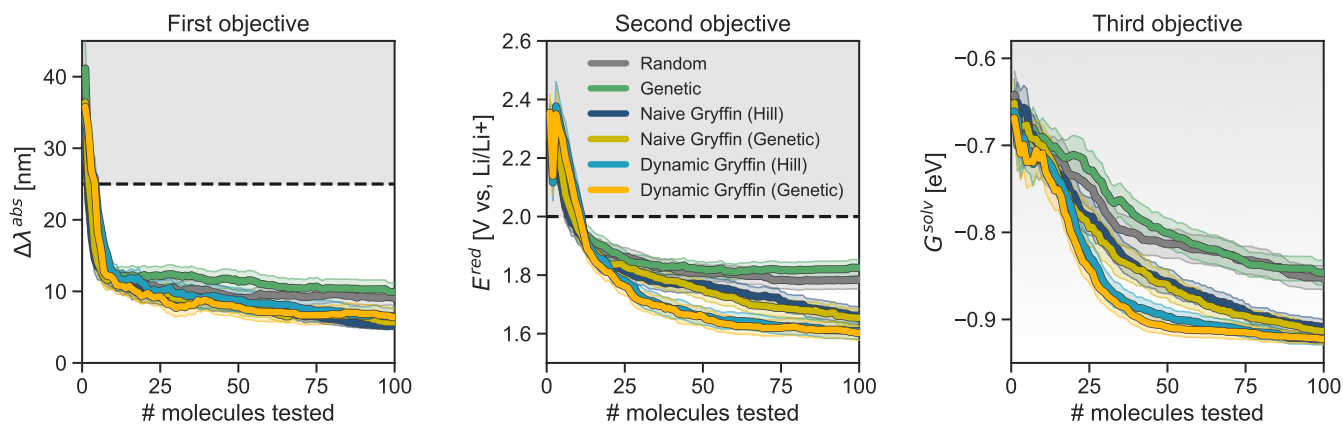


Figure S11. Results of the constrained optimization experiments for the design of redox-active flow battery materials. Grey shaded regions indicate objective values failing to achieve the desired objectives. Traces depict the objective values corresponding to the best achieved merit at each iteration, where error bars represent 95% confidence intervals.

- [1] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Jan. 2017.
- [2] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, 2012.
- [3] F.-M. De Rainville, F.-A. Fortin, M.-A. Gardner, M. Parizeau, and C. Gagné, "Deap: A python framework for evolutionary algorithms," in *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, pp. 85–92, 2012.
- [4] F. Häse, M. Aldeghi, R. J. Hickman, L. M. Roch, M. Christensen, E. Liles, J. E. Hein, and A. Aspuru-Guzik, "Olympus: a benchmarking framework for noisy optimization and experiment planning," *Machine Learning: Science and Technology*, vol. 2, p. 035021, July 2021.
- [5] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," 2015.
- [6] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.
- [7] B. E. Walker, J. H. Bannock, A. M. Nightingale, and J. C. deMello, "Tuning reaction products by constrained optimisation," *Reaction Chemistry & Engineering*, vol. 2, no. 5, pp. 785–798, 2017.
- [8] F. Häse, L. M. Roch, and A. Aspuru-Guzik, "Chimera: enabling hierarchy based multi-objective optimization for self-driving laboratories," *Chemical Science*, vol. 9, no. 39, pp. 7642–7655, 2018.
- [9] G. Agarwal, H. A. Doan, L. A. Robertson, L. Zhang, and R. S. Assary, "Discovery of Energy Storage Molecular Materials Using Quantum Chemistry-Guided Multiobjective Bayesian Optimization," *Chemistry of Materials*, vol. 33, pp. 8133–8144, Oct. 2021.
- [10] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, G. A. Petersson, H. Nakatsuji, X. Li, M. Caricato, A. V. Marenich, J. Bloino, B. G. Janesko, R. Gomperts, B. Mennucci, H. P. Hratchian, J. V. Ortiz, A. F. Izmaylov, J. L. Sonnenberg, D. Williams-Young, F. Ding, F. Lipparini, F. Egidi, J. Goings, B. Peng, A. Petrone, T. Henderson, D. Ranasinghe, V. G. Zakrzewski, J. Gao, N. Rega, G. Zheng, W. Liang, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, K. Throssell, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. J. Bearpark, J. J. Heyd, E. N. Brothers, K. N. Kudin, V. N. Staroverov, T. A. Keith, R. Kobayashi, J. Normand, K. Raghavachari, A. P. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, J. M. Millam, M. Klene, C. Adamo, R. Cammi, J. W. Ochterski, R. L. Martin, K. Morokuma, O. Farkas, J. B. Foresman, and D. J. Fox, "Gaussian 16 Revision C.01," 2016.
- [11] J.-D. Chai and M. Head-Gordon, "Long-range corrected hybrid density functionals with damped atom–atom dispersion corrections," *Physical Chemistry Chemical Physics*, vol. 10, pp. 6615–6620, Nov. 2008.
- [12] V. A. Rassolov, M. A. Ratner, J. A. Pople, P. C. Redfern, and L. A. Curtiss, "6-31G* basis set for third-row atoms," *Journal of Computational Chemistry*, vol. 22, no. 9, pp. 976–984, 2001.
- [13] A. V. Marenich, C. J. Cramer, and D. G. Truhlar, "Universal Solvation Model Based on Solute Electron Density and on a Continuum Model of the Solvent Defined by the Bulk Dielectric Constant and Atomic Surface Tensions," *The Journal of Physical Chemistry B*, vol. 113, no. 18, pp. 6378–6396, 2009.
- [14] A. Thakkar, V. Chadimová, E. J. Bjerrum, O. Engkvist, and J.-L. Reymond, "Retrosynthetic accessibility score (RAScore) – rapid machine learned synthesizability classification from AI driven retrosynthetic planning," *Chemical Science*, vol. 12,

- pp. 3339–3349, Mar. 2021.
- [15] P. Ertl and A. Schuffenhauer, “Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions,” *Journal of Cheminformatics*, vol. 1, p. 8, June 2009.
 - [16] M. Voršilák, M. Kolář, I. Čmelo, and D. Svozil, “Syba: Bayesian estimation of synthetic accessibility of organic compounds,” *Journal of Cheminformatics*, vol. 12, no. 1, p. 35, 2021.
 - [17] S. Genheden, A. Thakkar, V. Chadimová, J.-L. Reymond, O. Engkvist, and E. Bjerrum, “AiZynthFinder: a fast, robust and flexible open-source software for retrosynthetic planning,” *Journal of Cheminformatics*, vol. 12, p. 70, Nov. 2020.
 - [18] F. Häse, M. Aldeghi, R. J. Hickman, L. M. Roch, and A. Aspuru-Guzik, “Gryffin: An algorithm for Bayesian optimization of categorical variables informed by expert knowledge,” *Applied Physics Reviews*, vol. 8, p. 031406, Sept. 2021.
 - [19] H. Moriwaki, Y.-S. Tian, N. Kawashita, and T. Takagi, “Mordred: a molecular descriptor calculator,” *Journal of Cheminformatics*, vol. 10, p. 4, Feb. 2018.