JCIM JOURNAL OF CHEMICAL INFORMATION AND MODELING

pubs.acs.org/jcim Article

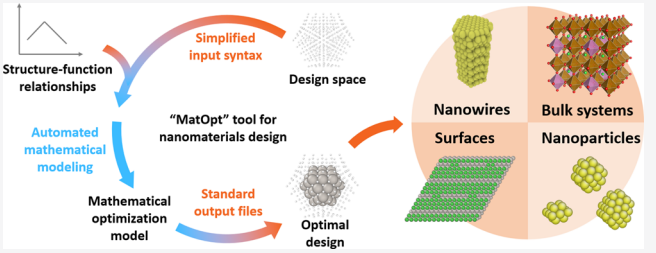
MatOpt: A Python Package for Nanomaterials Design Using Discrete Optimization

Christopher L. Hanselman, Xiangyu Yin, David C. Miller, and Chrysanthos E. Gounaris*

Cite This: J. Chem. Inf. Model. 2022, 62, 295−308 Read Online

ACCESS | Metrics & More | Article Recommendations | Supporting Information

ABSTRACT: Novel materials are being enabled by advances in synthesis techniques that achieve ever better control over the atomic-scale structure of materials. The pace of materials development has been further increased by high-throughput computational experiments guided by informatics and machine learning. We have previously demonstrated complementary approaches using mathematical optimization models to search through highly combinatorial design spaces of atomic arrangements, guiding the design of nanostructured materials. In this paper, we highlight the common features of materials optimization problems that can be efficiently modeled via mixed-integer linear optimization models. To take advantage of these commonalities, we have created *MatOpt*, a Python package that formalizes the process of representing the design space and formulating optimization models for the on-demand design of nanostructured materials. This tool serves to bridge the gap between practitioners with expertise in materials science and those with expertise in formulating and solving mathematical optimization models, effectively lowering the barriers for applying rigorous numerical optimization capabilities during nanostructured materials development.



INTRODUCTION

Advances in nanoscale material synthesis are resulting in better control over the atomic-scale structure of materials.[1−4] Important techniques include the generation of twinned defects,[5] synthesis of hollow nanostructures,[6] and tuning of strain,[6] to name but a few. In parallel to synthesis efforts, computational advances in the area of density functional theory (DFT) are enabling the computational design of materials from first-principles.[7−9] While there are many approaches for controlling structure and predicting properties at the nanoscale, there are relatively few systematic methods for designing nanomaterials algorithmically.

Several research efforts take a screening approach to identify highly functional materials by comparing and interpolating in shared databases of experimental and computational results.[10−12] Recently, high-throughput computational approaches and machine learning have been used to massively increase the pace of calculations and to smartly sample the combinatorial materials design space.[13−18] Another approach has been to develop simplified structure−function relationships for the purpose of predicting material properties as a function of simple geometric descriptors.[19,20] A notable result can be found in the work of Calle-Vallejo et al.,[21] where the authors used a model for activity as a function of generalized coordination number to design defects on nanostructured surfaces. However, it can be argued that approaches relying on chemist intuition to identify reactive sites might not always provide optimal surfaces, as they lack the ability to extensively search the design space. To improve upon the quality of the resulting structures, Rück et al.[22] used a similar relationship in conjunction with particle swarm optimization to design symmetric nanoparticles and rods. In the work of Núñez et al.,[23] the authors used another coordination based structure−function model in conjunction with simulated annealing to design defects on nanostructured surfaces.

Approaches that leverage metaheuristic optimization methods can find good solutions, but they lack rigorous guarantees on the optimality of the identified designs. In this paper, we seek to identify provably optimal designs by applying exact *mathematical optimization* algorithms to arrange nanostructured materials from their fundamental building blocks. Previously, we have developed mathematical optimization models for the design of nanostructured surfaces,[24,25] doped perovskites,[26,27] metallic nanoclusters,[28,29] and nanowires.[30] In each case, we showed how mathematical optimization was well-suited to search the combinatorial design space of nanostructures by formulating appropriate models.

Mathematical optimization represents decision problems via variables, algebraic constraints, and an objective function. In eqs 1−4, we express the search for novel materials as a

Received: August 12, 2021
Published: January 13, 2022

https://doi.org/10.1021/acs.jcim.1c00984
J. Chem. Inf. Model. 2022, 62, 295−308

mathematical optimization model with generic notation. We denote a vector of variables representing the design as d, which must attain values consistent with a design space D, and we seek to maximize a desired functionality against several constraints on the stability and fabricability of the design. In this abstract notation, there are no restrictions on the type or scale of materials that can be addressed; however, it is practically important to narrow the scope of problems to focus on cases that are both chemically relevant and that can be tractably modeled. In the next section, we shall discuss some common properties of nanostructured materials and chemistries, providing guidance on the subset of systems that can be designed via the below optimization model.

$$\max_{d} \quad \text{Functionality}(d) \tag{1}$$

$$\text{s.t.} \quad \text{Stability}(d) \geq \varepsilon \tag{2}$$

$$\text{Fabricability}(d) \geq 0 \tag{3}$$

$$d \in \mathcal{D} \tag{4}$$

The abstract functions referenced in the above materials optimization model can be used to represent a wide variety of functionalities of interest and constraints on the design space, including crystallinity, composition bounds, geometric restrictions, thermodynamic relationships, reaction rates, and mass transport, among others. In this context, we should highlight that the ability to solve the resulting optimization models is closely tied to the number and nature of variables and constraints it features, and hence, there is a clear need to make simplifying approximations and carefully match chemical systems to suitable optimization formulations. In general, a modeler will have to properly limit its design efforts to sizes (e.g., unit cells) that strike the right balance between quality of material representation and resulting model tractability. Having said that, it can be argued that solver technology for mixed integer linear programming (MILP) models is more mature compared to that for other model classes, all the while being actively improved at a fast pace. To that end, our proposed design paradigm is largely committed to the casting of linear optimization models, in order to capitalize on the current state-of-the-art and anticipated future improvements of solvers that will effectively enable its application in ever more complex and larger systems.

In the remainder of this paper, we shall first describe the common features of material design domains of interest and show how these commonalities result in simple patterns for modeling materials via mathematical optimization. Then, we will describe the implementation of *MatOpt*, a Python based toolkit for specifying and solving these types of material optimization problems. Finally, we will provide three case studies to illustrate possible use cases of *MatOpt* for nanoclusters, periodic bulk materials, and catalytic surfaces, identifying the most functional design in each case via the use of well-established numerical optimization software.

COMMON CHARACTERISTICS OF NANOSTRUCTURED MATERIALS

Nanostructured materials generally share a few characteristics that lead to the complexity of the design space and, thus, the combinatorial difficulty of finding optimal structures. These common material characteristics provide a basis for defining the scope for our optimization framework and result in simplifying assumptions for our framework to take advantage of. Specifically, we benefit from the discrete nature of lattices, local descriptors of functionality, and periodicity of designs.

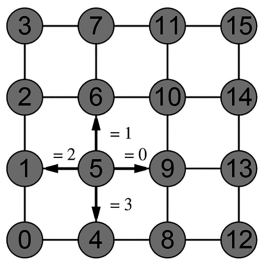
Discrete Lattices. We generically refer to the individual components of the design as "building blocks." In the example case of nanostructured transition metal catalyst surfaces, the building block can be thought of as an atom, while in larger systems such as metal organic frameworks or supramolecular assemblies,[31] the building block can be a molecule.

Conceptually, one could optimize the design of materials by choosing the Cartesian coordinates for all of the building blocks of interest. However, at the nanoscale, many solid materials tend to form architectures with atoms placed on lattices with specific, discrete locations. While exceptions to this observation can be found (i.e., lattice relaxations around defects, atomic restructuring on nanoclusters), we note that this observation can serve as a good first approximation in many systems of interest. We note that, while we typically work with regular crystalline lattices, our approaches can be applied to other design spaces where the possible placement of matter is ordered but does not satisfy strict definitions of a lattice (i.e., 5-fold symmetry around icosahedral nanoclusters). The discrete nature of nanomaterials can let us simplify the search space to only require "yes" or "no" decisions on the choices of building blocks to place in the design. These binary decisions also allow us to preprocess geometric information and avoid encoding nonlinear constraints in our optimization models. For example, as we shall discuss later, we can use piecewise-linear formulations or conformation (pattern) based modeling to encode nonlinear relationships. Reformulated MILP models with binary decision variables are in general expected to be significantly more tractable than their nonlinear counterparts, and thus more amenable to tackle nanomaterials design at a larger scale.

Local Functionality Descriptors. Given a set of discrete locations on which building blocks can be placed, there are also a discrete number of interactions that can contribute to the functionality of a design. In many cases, the desired functionality of the material can be broken down into contributions from sites in the structure. Furthermore, it is also typically the case that the functionality of a site is dependent on the presence of building blocks in a small subset of sites that can be thought of as "neighbors." Strictly speaking, the definition of neighbors does not need to be based on any kind of chemical information (e.g., chemical bonds), nor even on the physical distance between sites, though this is typically a good first approximation. The combination of discrete sites and their neighborhoods (i.e., ordered lists of neighboring sites) leads to the basic data structure that we denote as a "canvas."

The canvas data structure is generically composed of a list of site coordinates in conjunction with a graph, where the latter contains nodes for each site and directed arcs for the neighbor connections. The connections are considered directed because, in general, it is possible for site functionality to depend on each other site asymmetrically. In addition to the standard components of the graph data structure, the canvas specifies an ordering of neighbors to represent specific types of connections. This is useful for the ability to represent the specific alignment of neighbors in the lattice, which is necessary when trying to indicate a particular configuration of atoms in a design. In Figure 1, we present an example canvas with the corresponding data structure for the site neighbor-

https://doi.org/10.1021/acs.jcim.1c00984
J. Chem. Inf. Model. 2022, 62, 295−308

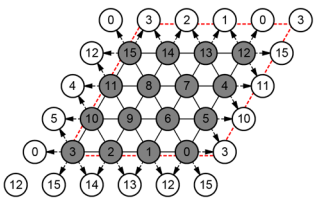


Site i	Neighbor ℓ			
	0	1	2	3
0	4	1	None	None
1	5	2	None	0
2	6	3	None	1
3	7	None	None	2
4	8	5	0	None
5	9	6	1	4
6	10	7	2	5
7	11	None	3	6
8	12	9	4	None
9	13	10	5	8
10	14	11	6	9
11	15	None	7	10
12	None	13	8	None
13	None	14	9	12
14	None	15	10	13
15	None	None	11	14

Figure 1. Example ordered neighbor connections for a simple canvas. (left) Graphical representation depicting connections between the 16 sites indexed from 0 to 15; the concept of neighbors is illustrated with site 5, where the arrows point to its four neighbors in four directions indexed from 0 to 3. (right) Tabular representation for use in code.

hoods. In this example, the ordering of neighbors consistently corresponds to sites along the four cardinal directions.

Material Periodicity. Nanostructured materials are modeled via cells containing periodicity in 0, 1, 2, or 3 dimensions, corresponding to clusters, wires, surfaces, and bulk materials, respectively. From an implementation standpoint, the presence of periodicity can lead to nonintuitive neighborhoods with connections that cross periodic boundaries. These connections can be identified from the combination of the canvas sites with a set of rules for transforming points that cross the tiling boundaries. Given a fixed canvas, these connections can be tabulated prior to formulating a mathematical optimization model, while care should be taken to ensure that the canvas neighborhoods are geometrically consistent. Figure 2 illustrates a simple two-dimensional lattice with connections defined by the periodicity of the canvas. In this example, the shape of the periodic tile is denoted with the dashed red parallelogram and neighbors crossing the tile edge are shown with directed arcs.



Site i	Neighbor ℓ					
	E	NE	NW	W	SW	SE
0	3	5	6	1	12	15
1	0	6	9	2	13	12
2	1	9	10	3	14	13
3	2	20	5	0	15	14
4	11	12	13	7	5	10
5	10	4	7	6	0	3
6	5	7	8	9	1	0
7	4	13	14	8	6	5
8	7	14	15	11	9	6
9	6	8	11	10	2	1
10	9	11	4	5	3	2
11	8	15	12	4	10	9
12	15	0	1	13	4	11
13	12	1	2	14	7	4
14	13	2	3	15	8	7
15	14	3	0	12	11	8

Figure 2. Example neighbor connections across the boundary of a periodically tiled canvas. (left) Graphical representation depicting connections between the 16 sites indexed from 0 to 15 and their periodic counterparts; connections across the periodic boundary (red dashed line) are shown with arrows. (right) Tabular representation for use in code.

It is important to highlight that the size of a periodic canvas constrains the design space; hence, it might lead the resulting optimization model to find an "optimal" solution but miss better designs that are only possible with a larger repeating pattern. To that end, in the case of periodic designs, a natural algorithm for quickly identifying good solutions is to iteratively expand the size of the periodic canvas and solve again until no further improvements are found with increasing size, assuming the available computational resources are not exceeded in the process. While we may not know the truly optimal solution *a priori*, a typical observation would be to see the same design repeated, as optimal solutions are obtained under consecutively increasing canvas sizes that are consistent with the periodicity of the design.

MODELING ELEMENTS

Given the commonalities in the materials of interest, there are several optimization modeling patterns that we can use to modularly formulate models to be used for rigorous design. In particular, the discrete nature of the design space leads to formulations that naturally employ binary variables. Furthermore, as we will demonstrate in our case studies, there are several strategies to exactly encode information via linear constraints and implication logic, even in cases where complex or nonlinear structure−function relationships are of interest.

Model Variables. The common features of nanostructured materials leads to a natural set of variables from which mathematical optimization models can be formulated. Below, we present four types of basic variables in addition to several types of auxiliary variables representing their aggregation.

Presence of Building Blocks. The presence of a building block of a given type at a particular site is the most fundamental variable in the optimization model. In eq 5, we denote as $\tilde{Y}_{ik}$ the presence of a building block of type k in site i. We denote the set of sites in the canvas as I and the set of building blocks as K.

$$\tilde{Y}_{ik} \in \{0, 1\} \quad \forall\, k \in K, \forall\, i \in I \tag{5}$$

Presence of Bonds. Given the variables for the presence of specific building block types, we can encode information about the connections between building blocks. For convenience, we refer to these variables as "bond" type variables, but in general, they do not have to correspond to bonds in the chemical sense. In eqs 6−9, we denote as $\tilde{X}_{ijkl}$ the presence of a bond between a building block of type k at site i and a building block of type l at site j. In the below, N_i denotes the set of neighboring sites to location i.

$$\tilde{X}_{ijkl} \le \tilde{Y}_{ik} \qquad \forall\, l \in K, \forall\, k \in K, \forall\, j \in N_i, \forall\, i \in I \tag{6}$$

$$\tilde{X}_{ijkl} \le \tilde{Y}_{jl} \qquad \forall\, l \in K, \forall\, k \in K, \forall\, j \in N_i, \forall\, i \in I \tag{7}$$

$$\tilde{X}_{ijkl} \ge \tilde{Y}_{ik} + \tilde{Y}_{jl} - 1 \qquad \forall\, l \in K, \forall\, k \in K, \forall\, j \in N_i, \forall\, i \in I \tag{8}$$

$$\tilde{X}_{ijkl} \in \{0, 1\} \qquad \forall\, l \in K, \forall\, k \in K, \forall\, j \in N_i, \forall\, i \in I \tag{9}$$

https://doi.org/10.1021/acs.jcim.1c00984
J. Chem. Inf. Model. 2022, 62, 295−308

Neighbor Counts. Applications at the nanoscale often correlate functionality to counts of neighbors around a particular type of site. In general, this functionality can also be broken down into contributions from sites and neighbors of particular types. In eqs 10−11, we denote as $\tilde{C}_{ikl}$ the count of bonds between a building block of type k at site i and neighboring building blocks of type l. In this definition, if there is not a building block of type k at site i, then all $\tilde{C}_{ikl}$ counts are set to zero.

$$\tilde{C}_{ikl} = \sum_{j \in N_i} \tilde{X}_{ijkl} \qquad \forall\, l \in K, \forall\, k \in K, \forall\, i \in I \tag{10}$$

$$\tilde{C}_{ikl} \in \{0, 1, ..., |N_i|\} \qquad \forall\, l \in K, \forall\, k \in K, \forall\, i \in I \tag{11}$$

The domain of variables $\tilde{C}_{ikl}$ could in principle be equivalently relaxed to the continuous interval $[0, |N_i|]$, since the integrality of the variables is enforced by eqs 10. For example, this may be advantageous when one wants to limit the number of integer variables (as opposed to continuous ones) in the overall formulation. However, our computational investigations suggest that most often the integer variable definition results in a more tractable optimization model, something that could be attributed to the fact that the branch-and-bound type algorithm employed in the solver can in this case exploit those variables during branching. To that end, the "default" *MatOpt* implementation of $\tilde{C}_{ikl}$ variables is to define them internally as integer variables.

Aggregate Variables. Variables with type-dependent information may not always be necessary or useful for modeling a given system. For example, many descriptors are expressed with conditionals such as the presence of "any atom" or "any bond" being present in the material. For each of the type-dependent variables previously discussed, we also present aggregated versions that indicate the presence of any building block or bond, or the count of neighbors. In eqs 12−21 we denote as Y_i the presence of any building block at site i, we denote as X_{ij} the presence of any type of bond between sites i and j, and we denote as C_i the count of any type of neighbor next to any type of building block present at site i. Similarly to variables $\tilde{C}_{ikl}$, variables C_i can be relaxed to continuous variables attaining values in their corresponding continuous interval.

$$Y_i \in \{0, 1\} \qquad \forall\, i \in I \tag{12}$$

$$X_{ij} \le Y_i \qquad \forall\, j \in N_i, \forall\, i \in I \tag{13}$$

$$X_{ij} \le Y_j \qquad \forall\, j \in N_i\ \forall\, i \in I \tag{14}$$

$$X_{ij} \ge Y_i + Y_j - 1 \qquad \forall\, j \in N_i, \forall\, i \in I \tag{15}$$

$$X_{ij} \in \{0, 1\} \qquad \forall\, j \in N_i, \forall\, i \in I \tag{16}$$

$$C_i = \sum_{j \in N_i} X_{ij} \qquad \forall\, i \in I \tag{17}$$

$$C_i \in \{0, 1, ..., |N_i|\} \qquad \forall\, i \in I \tag{18}$$

$$Y_i = \sum_{k \in K} \tilde{Y}_{ik} \qquad \forall\, i \in I \tag{19}$$

$$X_{ij} = \sum_{k \in K} \sum_{l \in K} \tilde{X}_{ijkl} \qquad \forall\, j \in N_i, \forall\, i \in I \tag{20}$$

$$C_i = \sum_{k \in K} \sum_{l \in K} \tilde{C}_{ikl} \qquad \forall\, i \in I \tag{21}$$

Presence of Conformations. In addition to variables for indicating bonds and neighbor counts, we also encode variables for indicating specific combinations of neighbors, a.k.a. "conformations." In eqs 22−25, we denote with $\tilde{Z}_{ic}$ the presence of a conformation of type c at site i. We use the parameter ξ_{icjl} to indicate if a conformation of type c that is located at site i should feature a building block of type l in neighboring location j.

$$\tilde{Z}_{ic} \le \tilde{Y}_{jl} \qquad \forall\, l \in K\colon \{\xi_{icjl} = 1\}, \forall\, j \in N_i, \ \forall\, c \in C, \forall\, i \in I \tag{22}$$

$$\tilde{Z}_{ic} \le 1 - \tilde{Y}_{jl} \qquad \forall\, l \in K\colon \{\xi_{icjl} = 0\}, \forall\, j \in N_i, \ \forall\, c \in C, \forall\, i \in I \tag{23}$$

$$\tilde{Z}_{ic} \ge 1 - \sum_{j \in N_i} \sum_{\substack{l \in K\colon \\ \{\xi_{icjl}=1\}}} (1 - \tilde{Y}_{jl}) - \sum_{j \in N_i} \sum_{\substack{l \in K\colon \\ \{\xi_{icjl}=0\}}} \tilde{Y}_{jl} \qquad \forall\, c \in C, \forall\, i \in I \tag{24}$$

$$\tilde{Z}_{ic} \in \{0, 1\} \qquad \forall\, c \in C, \forall\, i \in I \tag{25}$$

Common Constraint Patterns. Given the various sets of model variables introduced above, one may define a wide variety of additional, application-specific material descriptors, which can later be utilized to impose restrictions on the design space. In this section, we provide several basic patterns of constraints for defining new descriptors. In each case, we present a few basic patterns, noting that there are many ways to apply these patterns for combinations of site and bond types.

Linear and Piecewise-Linear Constraints. The most straightforward constraints to incorporate in MILP models are simple linear equalities or inequalities. In our models, these often take the form of budget constraints or summations. Similarly, we can incorporate piecewise-linear expressions into the model by introducing additional binary variables and implication logic.

Because the basic decision variables are discrete, all subsequent variables also attain discrete value. Therefore, we can practically consider piecewise-linear constraints to incorporate any nonlinear function into material design models without introducing approximation error. We achieve this by encoding breakpoints to coincide with the discrete feasible points, as illustrated in Figure 3. Assuming the number of required breakpoints is not too large, this technique can be used to exactly encode nonlinear functions without sacrificing accuracy. In other cases, it may still be necessary to estimate the original function to maintain computational tractability. In eqs 26−27, we present the general notation for representing simple constraints, where $g(x)$ and $P(x)$ denote linear and piecewise-linear expressions, respectively. Here, the vector x is used to generically represent the variables and parameters used in the expression. Also note that, although eqs 26−27 are cast here as ≤ 0 constraints, one may readily model any combination of $=$, $\ge$, and nonzero right-hand sides.

$$g(x) \le 0 \tag{26}$$

$$P(x) \le 0 \tag{27}$$

https://doi.org/10.1021/acs.jcim.1c00984
J. Chem. Inf. Model. 2022, 62, 295−308

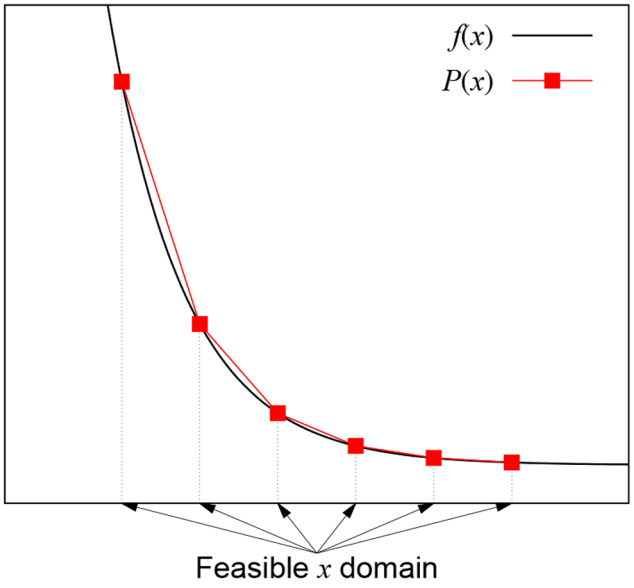


Figure 3. Example piecewise-linear function to exactly encode a nonlinear function over a discrete domain.

Implication Constraints. Frequently, material descriptors are defined in terms of logical predicates or conclusions. In such cases, we can use implication logic to conditionally apply constraints on the design space. We identify three general patterns of logical implications that are encountered in material design optimization models. In each case, a linear expression is conditionally set to zero if a binary indicator variable is active (i.e., equal to 1). Note that some predefined model variable (e.g., $\tilde{Y}_{ik}$, $\tilde{X}_{ijkl}$, $\tilde{Z}_{ic}$), or some user-defined descriptor, will serve as the indicator variable (generically denoted below as $D_{(\cdot)}$) in each case.

Site and Bond Descriptor Implications. The first general type of logical constraint forces an expression to be zero if a given condition is true for the site. In eqs 28−29, $g_i(x)$ corresponds to a general linear expression that is required to be equal to 0 if binary indicator D_i is equal to 1 at site i. The parameters M^{LB} and M^{UB} are so-called "big-M" parameters that can be automatically calculated to correctly encode the design space in the case that the binary indicator is inactive. The conditional expression can also be negated so that an expression is forced to be zero if the condition is not met. An example of invoking this pattern is the case when one requires a site's generalized coordination number to be equal to a target value if the site is considered a reactive site, and equal to zero otherwise.

$$M^{LB}(1 - D_i) \le g_i(x) \le M^{UB}(1 - D_i) \qquad \forall\, i \in I \tag{28}$$

$$D_i \in \{0, 1\} \qquad \forall\, i \in I \tag{29}$$

Similarly, logical implication constraints can be written across the bonds in the canvas, where the linear expression and associated indicator variables are suitably indexed over neighbor location pairs, as eqs 30−31 show. An example of this pattern is requiring the binding energy to be equal to an expression only if a bond is actually present between two sites.

$$M^{LB}(1 - D_{ij}) \le g_{ij}(x) \le M^{UB}(1 - D_{ij}) \qquad \forall\, j \in N_i, \forall\, i \in I \tag{30}$$

$$D_{ij} \in \{0, 1\} \qquad \forall\, j \in N_i, \forall\, i \in I \tag{31}$$

Site Combination Implications. The second pattern of logical implications involves combinations of neighboring sites. In this case, while the linear expressions of interest are location-specific, the indicator variables are indexed over the set of neighbor pairs in a canvas to create individual constraints on each of the sites. This pattern is used to encode the basic variables for the presence of bonds and can also be used in user-defined descriptors for encoding neighboring pairs of conformations.

$$M^{LB}(1 - D_{ij}) \le g_i(x) \le M^{UB}(1 - D_{ij}) \qquad \forall\, j \in N_i, \forall\, i \in I \tag{32}$$

$$M^{LB}(1 - D_{ij}) \le g_j(x) \le M^{UB}(1 - D_{ij}) \qquad \forall\, j \in N_i, \forall\, i \in I \tag{33}$$

$$D_{ij} \in \{0, 1\} \qquad \forall\, j \in N_i, \forall\, i \in I \tag{34}$$

Neighborhood Implications. The third pattern of logical implications introduces constraints on all neighboring sites to a location in the canvas. This pattern is used to encode the basic variables for the presence of conformations and is also generally useful for introducing more complicated constraints on combinations of sites in the canvas.

$$M^{LB}(1 - D_i) \le g_j(x) \le M^{UB}(1 - D_i) \qquad \forall\, j \in N_i, \forall\, i \in I \tag{35}$$

$$D_i \in \{0, 1\} \qquad \forall\, i \in I \tag{36}$$

THE *MATOPT* TOOLKIT

To assist researchers in casting and solving rigorous mathematical optimization models for nanostructured materials design, we have developed *MatOpt*, an object-oriented Python based library that implements the modeling framework discussed in the previous section. This toolkit formalizes and simplifies the process for carrying out nanostructured materials optimization via two major contributions. First, we provide several modeling objects for specifying the materials design space from simple input. Second, we provide a framework for specifying structure−function relationships without needing a detailed understanding of how that specification can be converted into a suitable mathematical optimization model. While the explanation given here is kept at a high level, we encourage the interested reader to refer to the detailed documentation and open-source code available online as part of the Institute for the Design of Advanced Energy Systems (IDAES) code distribution.[32]

Software Availability and Solver Dependencies. *MatOpt* is bundled with the open-source IDAES PSE framework[33] and can be downloaded freely at https://github.com/IDAES/idaes-pse/tree/main/idaes/apps/matopt. Users should follow the instructions described in the online documentation[34] to install and access the toolkit. Along with the source code, we also provide several use cases in the form of Jupyter notebooks,[35] which demonstrate the features and

https://doi.org/10.1021/acs.jcim.1c00984
J. Chem. Inf. Model. 2022, 62, 295−308

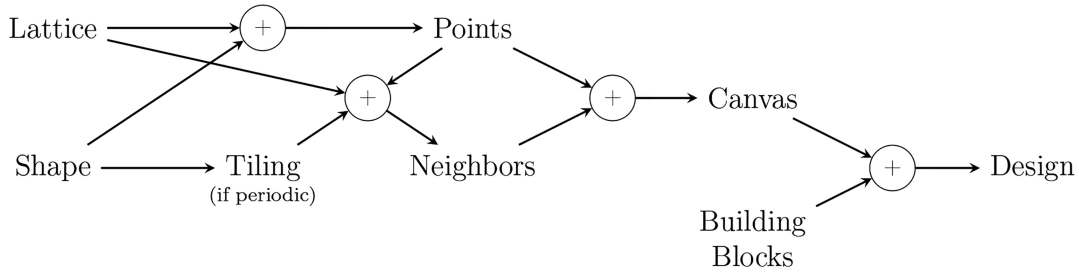


Figure 4. Flow of information for creating objects to represent materials.

functionalities of the toolkit as well as offer useful Python scripts from which users can start to develop new codes for their own materials design applications. The scripts and source code can be downloaded from the IDAES example repository at https://github.com/IDAES/examples-pse/tree/main/src/Examples/MatOpt.

At its core, *MatOpt* translates a materials optimization problem into a Python Optimization Modeling Objects (Pyomo) model object that can then be solved by an available MILP solver, with *MatOpt* providing appropriate shortcut functions to facilitate this step. Experienced Pyomo users with access to a licensed MILP solver can easily select their solver of choice during this process. For users who either do not have such access or are not familiar with how to set up and use optimization solvers in Pyomo, *MatOpt* further automates the invocation of the web based free solver NEOS-CPLEX that can be utilized for this purpose.[36−38]

Materials Representation Objects. The focus of this section is to describe the ability of the toolkit to easily instantiate data structures necessary to represent nanoscale materials from simple input. Figure 4 illustrates the interactions between several objects and data structures to represent nanostructured material designs in *MatOpt*. The basic data structure required to cast an optimization model is a `Canvas`, which is essentially a list of Cartesian points coupled with a graph, whose nodes correspond to sites and whose arcs correspond to bonds. This object establishes a mapping from the abstract mathematical modeling of materials as graphs to the geometry of the material's lattice.

The list of points necessary to create a `Canvas` can be obtained from the combination of a `Lattice` and a `Shape` object by scanning over the set of lattice sites that fall inside the shape. The set of neighbor connections to be encoded in the `Canvas` can be identified from rules that are attached to the `Lattice` object. In the simplest case, these rules for indicating neighbors can be simple cutoff distances, but we also note cases where asymmetric definitions of neighbors can be useful for representing materials. For example, in Hanselman et al.,[27] the authors utilized an asymmetric definition of neighborhoods to establish different definitions of neighbors around oxygen atoms compared to metallic B-site atoms. Complementary to the rules defined by the lattice, a `Tiling` object can be used to make the neighbor connections consistent across the boundary of a periodic tile.

While a `Canvas` object holds topological and geometric information, a `Design` object is composed of a `Canvas` in conjunction with a list of building blocks placed in the sites of the material. The necessary attributes of a building block are encoded in the `BuildingBlock` class and can in principle be extended to represent many materials. As a starting point, we have implemented the `Atom` class that is of interest in our work.

It is worthy to note that several existing tools provide interfaces for setting up and analyzing materials at the atomic scale.[39−41] Their emphasis is on setting up computational experiments that can efficiently calculate material properties during atomic-scale simulations. Although these tools typically include optimization routines, the latter serve to minimize the energy of atomic configurations by relaxing the atomic coordinates in the vicinity of lattice sites; in this sense, their use of *optimization* is conceptually different from the design paradigm discussed in this work. Furthermore, these codes are tailored to work with more traditional workflows in computational materials screening and, therefore, do not immediately adapt to the same level of generality that the mathematical optimization approach can exhibit. Whereas *MatOpt*'s basic data structures were defined to support the representation of the materials design space at the level of detail required by the various transition metallic systems we wanted to contemplate, there exist opportunities for extending these language features. Future versions could extend *MatOpt* by incorporating connections between other open-source packages and our optimization modules.

Algebraic Modeling Language Features. While the goal of *MatOpt*'s material modeling modules is to help organize and create prerequisite data structures, the goal of the optimization modules is to enable users to cast optimization models from minimal input. One of the key features of our modeling approach is the automatic encoding of logical expressions into MILP models. Additionally, we automatically generate indexed expressions for a wide variety of patterns encountered in material optimization models. In this way, our tool can be used by practitioners without a background in mathematical optimization and from simplified user input. Conversely, a user who is already familiar with Pyomo can use *MatOpt* to generate a Pyomo model with a partial formulation defined and then modify it as necessary (e.g., by adding customized constraints).

Given a defined `Canvas` and a list of `BuildingBlock` objects to place in the design, we start by automatically generating the set of basic descriptors for the presence of building blocks, bonds, neighbor counts, and conformations. We note that *MatOpt* will infer from user inputs what basic descriptors are needed, generating and indexing constraints accordingly, but there is no guarantee that *MatOpt* will identify all possible simplifications and return the optimization model that is most simplified in this regard. In principle, however, power users can implement further simplifications by modifying the resulting Pyomo model directly. Beyond these basic descriptors, *MatOpt* enables users to specify additional descriptors using a combination of `Expression` and `Rule`

https://doi.org/10.1021/acs.jcim.1c00984
J. Chem. Inf. Model. 2022, 62, 295−308

objects. The general flow of information for specifying a model is presented in Figure 5. The `Expression` objects are

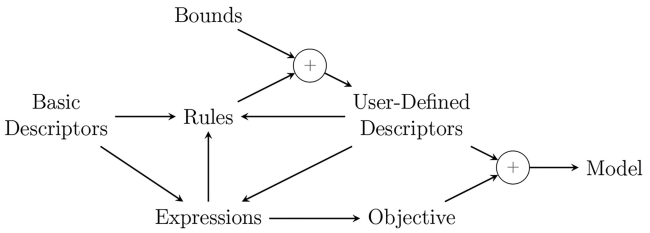


Figure 5. Flow of information for instantiating *MatOpt* optimization models.

generated from basic or user-defined descriptors via a set of predefined patterns. Two examples include `SumSites` and `SumBonds` to allow simple summations. A more complicated example is `SiteCombination` which creates expressions indexed over bonds by adding contributions from descriptors at two neighboring sites. The `Expression` objects are used in conjunction with `Rule` objects to define descriptors and to generate constraints on the design space without requiring the user to specify the details of the underlying constraints or necessary reformulations. A full list of available expressions and descriptor rules, along with some example use cases, can be found in Section A of the Supporting Information, with the expectation that additional patterns can be introduced as part of ongoing open-source development.

When the user requests to optimize the model, the code will automatically interpret `Rule` objects to generate a Pyomo optimization model.[42,43] Several rules, such as `LessThan`, `EqualTo`, and `GreaterThan`, can be immediately interpreted as linear constraints in the model. Other rules might require some more specialized treatment. For example, the `Implies` rule creates implication constraints that are enforced only if the site descriptor is true (e.g., if a site is present, if a bond is present, etc.). The `FixedTo` rule is included to allow descriptors to be explicitly fixed to a value. While, in principle, the same effect could be achieved by introducing equality constraints into the model, the `FixedTo` routines have the additional efficiency of fixing logically implied basic variables when possible. The `PiecewiseLinear` rule allows the user to specify the variable domain, values, and breakpoints to equate a descriptor to a piecewise linear function. When generating constraints, *MatOpt* converts `PiecewiseLinear` rules to specialized objects in Pyomo. In general, this leverages advances in Pyomo and its supported optimization solvers and enables interested users to quickly try several alternate mathematical reformulations of piecewise functions provided by Pyomo.

One of the useful features of our modeling framework is the automatic handling of expression and rule indexing. For example, if a user-defined variable is indexed over a subset of sites in the canvas (e.g., over only the oxygen sites in a lattice), then the derived expressions and constraints are likewise indexed over that subset. Alternatively, if a variable indexed over canvas sites is multiplied by a parameter that is indexed over site types, then the resulting expression is automatically indexed over the set product of sites with site types. Figure 6 presents an example descriptor definition and highlights the data structures for indexing that are maintained for each descriptor, expression, and rule in the model. As an example descriptor, we present an abstract structure−function relationship that assigns the pairwise binding energy (BE_{ij}, programmatically `m.BEij`) to an expression formed from a weighted combination of the coordination number (C_i, programmatically `m.Ci`) conditionally on the type of binding that is present ($\tilde{X}_{ijkl}$, programmatically `m.Xijkl`). The constraint includes bond type-indexed parameters (A_{kl}, programmatically `m.Akl`) that abstractly represent the weighting contributions.

The last component necessary to specify an optimization model is a single expression to minimize or maximize as an objective function. While optimization solvers typically only accept a single objective, we note that it is possible to combine multiple objectives via a weighted sum implemented as a `LinearExpr` object. One of our case studies in the next section exemplifies this and illustrates how optimization models can be used to generate Pareto-optimal frontiers.

CASE STUDIES

As a demonstration of the *MatOpt* framework, we present three example case studies. In the first example, we build a simple nanocluster stability optimization model using minimal programming effort. This example illustrates the basic workflow and simplified input syntax, exemplifying how *MatOpt* can lower the barriers to apply rigorous mathematical optimization in the area of nanostructrured materials design. In

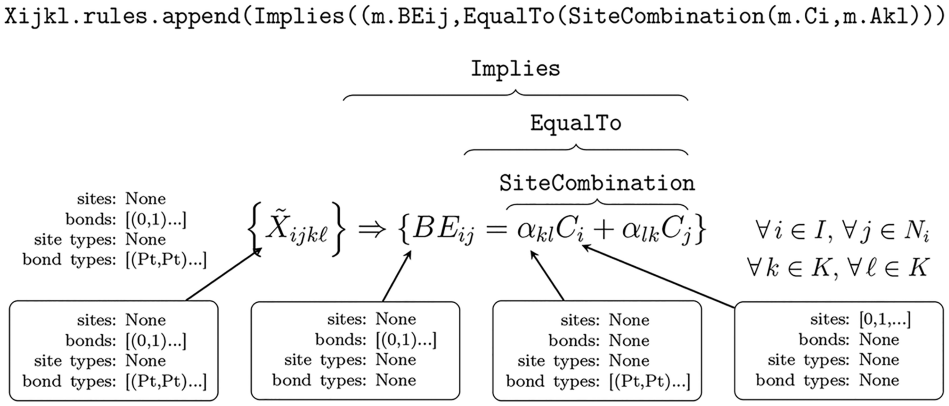


Figure 6. Example creation of site descriptor rules. Example code is given in the top row, and the equivalent mathematical notation is given in the middle. Attributes implied by automatic indexing are presented in the bottom row.

https://doi.org/10.1021/acs.jcim.1c00984
J. Chem. Inf. Model. 2022, 62, 295−308

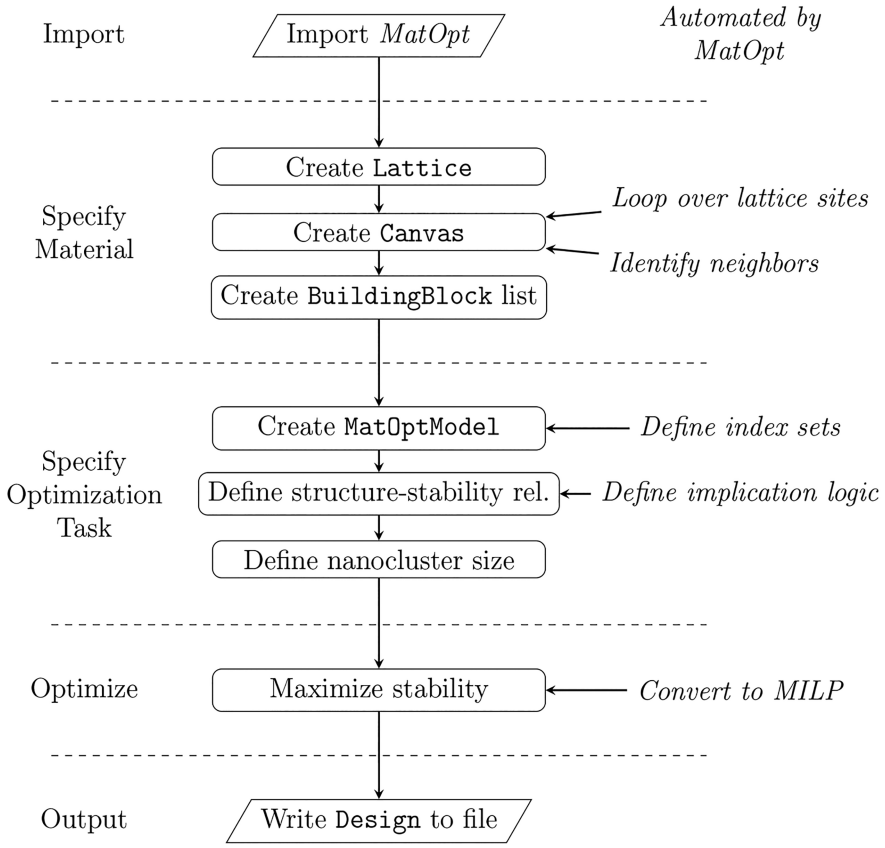


Figure 7. Conceptual flowchart for the nanocluster cohesive energy example.

the second example, we identify the best placement of dopants in bulk perovskite materials for optimal oxygen diffusivity. We introduce syntax to deal with unit cell periodicity and conformation based design. This example also showcases how DFT calculations can inform a mathematical optimization model in the context of a materials design problem. In the third example, we show how *MatOpt* can be employed to design a bifunctional catalyst surface, targeting both reactivity and stability, in the context of a system studied recently in the literature.[44] This example further demonstrates two-dimensional periodicity syntax and the flexibility of modeling a more complex multiobjective problem with only a modest increase in programming effort.

Cohesive Nanocluster Design. The recent work of Isenberg et al.[28] demonstrated the use of mathematical optimization models to identify cohesive nanoclusters via a set of tailored algorithms. Here, we present a simplified version of their models as an example system on which *MatOpt* can be employed. As the structure−stability relationship, we consider a normalized version of the Tománek model for cohesive energy[45] shown in eq 37. Here, $\hat{E}^{\text{coh}}$ corresponds to the normalized cohesive energy of a nanocluster, CN_{bulk} is the material's bulk coordination number, N is the size of the nanocluster, and C_i corresponds to the coordination number of atom i.

$$\hat{E}^{\text{coh}} = \frac{1}{N\sqrt{CN_{\text{bulk}}}} \sum_{i \in I} \sqrt{C_i} \tag{37}$$

Figure 7 presents the conceptual steps that we will be following to specify this nanocluster energy minimization problem; that is, importing the *MatOpt* module, providing input about the materials system of interest, specifying the relevant structure−function and constraint information, invoking the optimizer, and finally, outputting the identified designs.

In the following section, we present the minimal code necessary to set up and solve the nanocluster cohesive energy optimization problem. First, we import the *MatOpt* package and standard Python modules.

```
1 from matopt import *
2 import numpy as np
```

Next, we define the objects necessary to represent the material information in the problem. For this, we create a `Lattice` object that specifies the sites to consider in the design space along with their neighbor connections. In this case, the object will codify information pertaining to a face-centered cubic (FCC) lattice of a specific interatomic distance.

```
3 Lat = FCCLattice(IAD=2.77)
```

In the next few lines of code, we build a `Canvas` object composed of three shells of FCC lattice locations around the origin. The particular choice of three shells was based on the original work of Isenberg et al.,[28] where an adaptive canvas resizing technique was used to identify the best canvas size. As discussed, it is important for users to carefully choose the size of the canvas so as to properly navigate the trade-off between global optimality and model tractability. Note that, since we are using a lattice defined in three-dimensional space, the

https://doi.org/10.1021/acs.jcim.1c00984
J. Chem. Inf. Model. 2022, 62, 295−308

expected data type of a point is a NumPy[46] array of length three with floating point precision.

```
4  Canv = Canvas()
5  Canv.addLocation(np.array([0,0,0],dtype=float))
6  Canv.addShells(3,Lat.getNeighbors)
```

In the following couple of lines, we define a parameter, `N`, to be the number of building blocks to consider in the nanocluster; in this case, 17 atoms. Additionally, we define the set of `Atom` objects to place in the design, specifying their elemental identify as gold atoms, for convenience.

```
7  N = 17
8  Atoms = [Atom('Au')]
```

Once the material information is specified, we can begin to generate a mathematical optimization model via an object of class `MatOptModel`. The model object is initialized from a `Canvas` and a list of `Atom` objects from which we can identify the applicable index sets for sites, bonds, site types, and bond types. These index sets automatically used as new descriptors are defined.

```
9  m = MatOptModel(Canv,Atoms)
```

In the following steps, we introduce material descriptors to encode the structure−stability relationship as well as to constrain the design space of interest. First, we introduce a descriptor for the square root of the coordination number. Because the coordination number is itself discrete, we can encode its square root without approximation error via a piecewise linear formulation, thereby preserving the linearity of the model. Note that, since the input descriptor (i.e., the coordination number `m.Ci`) is indexed over all sites, the resulting rule and descriptor are also indexed over all sites. Therefore, we utilize the method `addSitesDescriptor`, which registers the new user-defined descriptor with the model.

```
10 m.addSitesDescriptor('CNRi',bounds=(0,sqrt(12)),integer=False,
11           rules=PiecewiseLinear(values=[np.sqrt(CN) for CN in range
                 (0,13)],
12                                 breakpoints=[CN for CN in range
                                       (0,13)],
13                                 input_desc=m.Ci))
```

Next, we introduce the descriptor for the normalized cohesive energy. Here, we utilize the number of atoms in the cluster, `N`, and the square root of the bulk coordination number (i.e., $\sqrt{12}$) as a normalizing coefficient. Since this descriptor is a single scalar quantity, we invoke the `addGlobalDescriptor` method.

```
14 m.addGlobalDescriptor('Ecoh',
15            rules=EqualTo(SumSites(desc=m.CNRi,
16                                   coefs=(1/(N*sqrt(12))))))
```

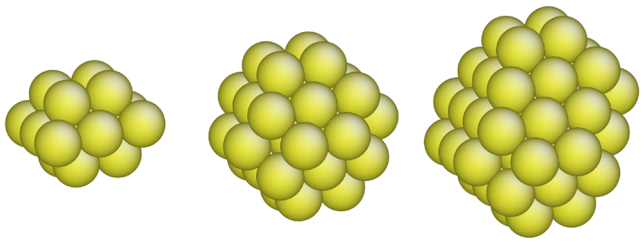
Finally, we introduce a descriptor for the size of the nanocluster. Note how, by setting the lower and upper bounds on the descriptor to be equal, we essentially constrain the nanocluster to be of a specific size.

```
17 m.addGlobalDescriptor('Size',bounds=(N,N),
18                rules=EqualTo(SumSites(desc=m.Yi)))
```

At this point, the feasible space for the optimization model is fully specified. We finalize the model and instruct *MatOpt* to solve it by specifying an objective. In this example, the objective to be maximized will be the previously defined (code line 14) descriptor `m.Ecoh`, noting that any `Expression` object can be used to that purpose, in general. The `maximize` function will formulate an undelying Pyomo model object and call the appropriate optimization solver routine.

```
19 D = m.maximize(m.Ecoh)
```

In this specific example, *MatOpt* will generate a model with 5292 variables (1914 continuous, 3231 binary, 147 integer) and 8079 constraints (7488 inequalities, 591 equalities). Upon testing with our default optimization solver, CPLEX 12.10, on an 8-core Intel i7 laptop, the model could be solved to global optimality (i.e., relative and absolute gap tolerances set to zero) in approximately 33 s. The solution attains a globally optimal normalized cohesive energy of 0.707 and is depicted in Figure 8a.



(a) $N = 17$ (b) $N = 34$ (c) $N = 55$

Figure 8. Example optimally cohesive monometallic nanoclusters.

We note that the size of the model depends on the size of the canvas (code line 6). More specifically, if we use four shells of FCC lattice locations around the origin (309 sites) instead of three shells (147 sites), we will obtain a model with 11 322 variables and 17 583 constraints. The model will grow to 20 790 variables and 32 631 constraints when using 5 shells (561 sites) and to 34 476 variables and 54 503 constraints, when using 6 shells (923 sites). Whereas a larger canvas increases our confidence that the optimality of the solution has not been affected by boundary effects, it also makes it more difficult to solve the optimization model. In particular, our testing laptop device could not solve the model to global optimality when using five- or six-shell canvases, yielding relative optimality gaps of 18.18% and 26.90%, respectively, at the 1 h time limit. We should highlight, however, that increasing the canvas size in this case would not be justified, as it would have no impact on the ability of the tool to identify the optimal solution.

If the optimization solver is successful, a `Design` object is returned that can be written to standard material file formats. Here, we create a Protein Data Bank (PDB) file,[47] noting that additional parsers are also implemented and can be used as necessary.

While the sample code presented above is intentionally terse, there are many ways to modify the script to collect and analyze

https://doi.org/10.1021/acs.jcim.1c00984
J. Chem. Inf. Model. 2022, 62, 295−308

```
20 if(D is not None):
21     D.toPDB('result_{}.pdb'.format(N))
```

a variety of results. The most obvious extension is to place the model generation and solution in a Python loop to generate designs across a range of sizes N. In Figure 8, we present a few example results from solving the above simple model within such a loop. Here, we should remark that, although the parameter N may impact the solution times from imposing various inherent combinatorial complexities, it does not directly impact the size of the model, and since we used same size canvases in all those examples, the computational times required to solve them were all in the same order of magnitude (around 100 s).

The code can be further tailored by changing the contributions to cohesive energy with specially regressed coefficients to better represent a metal of interest, as presented in the work of Isenberg et al.[28]

Before we conclude this case study, we note that, in addition to the basic mathematical optimization model presented above, the original work of Isenberg et al.[28] made several customized problem-specific modifications to improve the optimization model's tractability and the interpretability of its solution. One major such modification was the introduction of a set of symmetry-breaking constraints to eliminate certain equivalent feasible solutions (i.e., nanostructures symmetric to others via rotation and reflection) while making sure that at least one representative nanostructure remained feasible. We note that symmetry-breaking constraints are typically problem-specific, depending strongly on the applicable lattice topology. As a general-purpose set of symmetry breaking constraints is hard to devise, *MatOpt* does not automatically apply such constraints. Here, it should be emphasized that the lack of symmetry-breaking constraints will not affect the optimality of the obtained solutions, rather the solutions obtained by *MatOpt* will indeed be optimal albeit in an arbitrary orientation. In any case, a user could in principle define additional descriptors and rules to formulate symmetry-breaking constraints on an ad-hoc basis. However, it is always advisable that such action be empirically investigated on a case-by-case basis, as declaring symmetry-breaking constraints may not necessarily result in the most efficient MILP model formulation and thus may not help with overall tractability.

Perovskite Dopant Design. Hanselman et al.[27] proposed a mathematical optimization based framework for designing doped perovskites. Their study identified the optimal atomic-scale patterns of In dopants at the B-sites in $BaFe_{1-x}In_xO_{3-\delta}$ perovskites that achieve the highest oxygen diffusivity, an important property for such perovskites to be used as oxygen carriers in a chemical looping system. The optimization model proposed in the aforementioned study identified oxygen excess energy as the target functionality of interest, decomposing it into contributions from conformations associated with each oxygen atom in the lattice. More specifically, the paper investigated oxygen atom conformations that consisted of 10 nearest B-sites and represented all possible ways of placing between zero to five dopants around a particular oxygen atom. The set of 74 rotationally unique conformations, so-called "motifs," was then identified and their oxygen excess energies were evaluated via DFT calculations. The 74 evaluations formed a structure−function relationship that was then encoded into an MILP model that attempted to maximize the number of oxygen sites exhibiting lowest excess energies. We highlight how this conformation based encoding is another way of expressing nonlinear structure−function relationships via linear constraints that utilize binary variables.

The above approach and model can be readily reproduced using *MatOpt*. The full implementation is provided in Section B of the Supporting Information, while below we highlight only some key aspects. To implement this optimization model using *MatOpt*, we first construct a `PerovskiteLattice` object to hold perovskite-specific lattice site information. We confine our optimization to a periodic supercell using a `Shape` object, which defines the geometry and size of the supercell, in conjunction with a `Tiling` object to handle boundary neighbor relationships. In particular, we create a rectangular prism-shaped supercell `Canvas` with periodicity in all three dimensions using `RectPrism` and `CubicTiling` objects, which constitute available specializations of the `Shape` and `Tiling` classes, respectively. Here, we note a small trick to shift the shape slightly to avoid duplication of sites on the border of the periodic cell, which is necessary to do for proper visualization of the canvas. We highlight that such shift has no impact on solution optimality and is in fact not necessary for the correctness of the canvas itself. The visualization of this perovskite canvas can be found in Figure 9a. Next, we load motifs from standard material file formats,

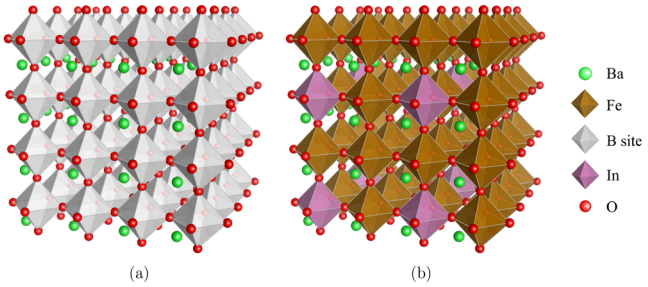


Figure 9. (a) Doped perovskite design canvas and (b) optimal design that maximizes the number of sites displaying one of the three motifs with lowest oxygen excess energy.

noting that parsers for several common formats such as PDB, POSCAR, and XYZ are available. We then initialize a `MatOptModel` object from the defined conformations and sites. Note how *MatOpt* will automatically encode basic variables and constraints to indicate those conformations (eqs 22−25). To specify additional constraints in the model, we create several descriptors for the activity, local dopant concentration, and the global dopant concentration. We specify these descriptors over different subsets of sites and atom types. For example, the activity descriptor is defined over the oxygen sites only, while the dopant budgets are expressed only over B-sites. Note the use of the `addSitesTypesDescriptor` and `addGlobalTypesDescriptor` methods, as appropriate.

In this case study, we shall specify our objective as the maximization of the number of sites displaying conformations with the lowest three oxygen excess energies among all possible conformations. We note, however, that additional objectives stated in the original work can also be modeled and solved via *MatOpt* by simply changing parameters of the activity descriptor. We specifically consider a unit cell with four atoms on the edge, which corresponds to the size chosen by

https://doi.org/10.1021/acs.jcim.1c00984
J. Chem. Inf. Model. 2022, 62, 295−308

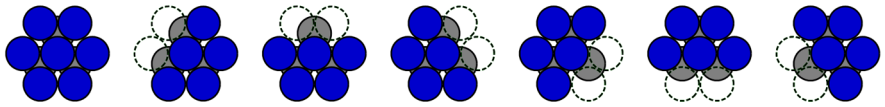


Figure 10. Material conformations relevant for the bifunctional catalyst example. The gray circles represent an extended {111} layer of Pt atoms while the blue circles represent the (patchy) overlayer of Ni atoms. Dashed circles represent missing Ni atoms, resulting in reactive step sites next to the central Ni atom.

Hanselman et al.[27] who carefully investigated the trade-off between solution optimality and model tractability in this setting. The *MatOpt*-generated model consists of 6723 variables (195 continuous, 6528 binary) and 16 259 constraints (16 000 inequalities, 259 equalities) and can be solved to global optimality (using similar zero gap tolerances and machine as before) in approximately 33 min. The global optimal solution is shown in Figure 9b. It features an objective value of 0.5 and perfectly reproduces the original result from Hanselman et al.[27]

Before we conclude, we highlight that the code provided in the Supporting Information constitutes an example *MatOpt* implementation for conformation based optimization models. Starting from this code as the basis, a user may adapt the definition of motifs and the way in which properties of interest are calculated from first-principles, and then rely on *MatOpt* to automatically instantiate and solve the rigorous mathematical optimization model that searches for the best combinations of motifs in the design space.

Bifunctional Catalyst Design. To further demonstrate the capabilities of the *MatOpt* toolkit, we present here a surface design problem inspired by the recent work of Núñez and Vlachos.[44] In that work, the authors presented a machine-learned model for predicting surface reactivity of ammonia decomposition on a patchy Ni−Pt bifunctional catalyst and then designed a surface by applying a simulated annealing optimization algorithm. While their approach is tailored to make accurate predictions of reactivity, the use of simulated annealing lacks guarantees of producing the globally optimal design. In this example, we present a complementary MILP based approach that can identify provably optimal structures for a simplified version of the relevant chemistry.

This design problem calls for identifying the optimal patterning of a single layer of Ni atoms on top of a Pt{111} surface. The placement of Ni atoms creates facet and edge sites that contribute to the turnover of the ammonia decomposition reaction. As a first order approximation of the microkinetic model presented in the work of Núñez and Vlachos,[44] we shall simply assume that the slab reactivity is proportional to the sum of pairs of edge sites immediately next to a facet site. This simplified viewpoint neglects contributions to reactivity from adsorbates that take a longer path diffusing across the catalyst surface between sites further away than nearest neighbors. However, we remark that more complicated contributions could be conceptually represented (and encoded in *MatOpt*) via additional variables, constraints, and parameters for the turnover on the catalyst surface.

The full *MatOpt* implementation of this design problem can be found in section C of the Supporting Information. To design a periodic surface, we create a `Parallelepiped` shaped canvas with periodicity defined by `PlanarTiling`. We again follow a conformation based modeling strategy to encode nonlinear structure−function relationships and utilize *MatOpt*'s capabilities to indicate identified conformations automatically. For reference, we plot in Figure 10 the seven relevant conformations, including an FCC{111} facet site and six orientations of edge sites.

As a simplified model of catalyst turnover, we consider here that the combination of a facet site next to an edge site gives rise to an "ideal" reactive step site. To represent this in our optimization model, we define additional material descriptors to indicate the conjunction of a conformation of type "A" (i.e., a facet site) next to conformation of type "B" (i.e., an edge site). We then fix the possible chemical identity of the various layers of the design. Additionally, we can improve the tractability of the model by arbitrarily placing one conformation on the surface, breaking symmetry and simplifying the design space.

Furthermore, we model the surface energy of the resulting patterns with the goal of mapping the Pareto-optimal frontier of activity against stability in this design space. To obtain the surface energy, we first employ the formation energy model proposed for this system in Núñez et al.[23] In this model, each atom contributes to lowering the formation energy in a manner that is proportional to the square root of its coordination number; that is, lower coordinated atoms contribute to a higher formation energy. More specifically, eq 38 presents the descriptor for the formation energy of the slab (normalized by the cohesive energy of the metal), $\hat{E}^{\text{form}}$, as a function of coordination numbers C_i at each site i. The surface energy is then approximated in eq 39 by dividing the formation energy with the surface area, where as a proxy for the latter we use the square of the number of atoms on the edge of the canvas, N_{edge}. Note here that, since we are using a fixed size periodic canvas during each run, we will identify the same optimal results when using surface energy or formation energy as the stability indicator. After some experimentation using differently sized periodic tiles, in order to explore the impact of periodicity limitations on the obtained designs as well as the tractability of the resulting models, we chose to work with tiles of size 8 × 8 atoms, which were found to be a good compromise inasmuch as they were generally tractable and did not show solution differences compared to using 9 × 9 atom tiles.

$$\hat{E}^{\text{form}} = \sum_{i \in I} \left(1 - \sqrt{\frac{C_i}{CN_{\text{bulk}}}}\right) \tag{38}$$

$$\hat{E}^{\text{surf}} = \hat{E}^{\text{form}} / N_{\text{edge}}^2 \tag{39}$$

Note how the values of the per-atom contributions to the surface energy have been normalized so that they all lie between 0 and 1, with a value of 0 corresponding to bulk atoms that do not contribute to the surface energy and a value of 1 corresponding at the limit to the hypothetical contribution of an isolated atom that would have increased the surface energy the most. In calculating surface energy via the above method, we exclude sites in the bottom two layers of the canvas, which are fixed to be fully occupied with Pt atoms.

https://doi.org/10.1021/acs.jcim.1c00984
J. Chem. Inf. Model. 2022, 62, 295−308

To build a Pareto frontier of activity against stability of such nanostructured surfaces, we perform a loop to solve the model for a range of thresholds on the surface activity (i.e., lower bound on the number of active sites). The resulting frontier is presented in Figure 11. We note that each frontier point

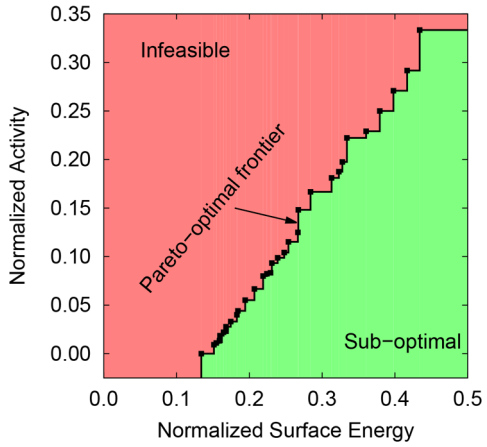


Figure 11. Pareto-optimal frontier for the bifunctional catalyst design example.

represents the optimal solution of a separate *MatOpt*-generated optimization model, each featuring 10 240 variables (1667 continuous, 8189 binary, 384 integer) and 38 659 constraints (37 376 inequalities and 1283 equalities). On average, these models took approximately 410 s to solve to global optimality (zero gap tolerance) using the same computational setup as the previous case studies.

A few representative Pareto-optimal solutions identified are plotted in Figure 12. We observe that nanostructuring frequent grooves on the Ni layer yields a larger density of edge sites and, hence, increases activity. However, it has a negative effect on overall stability, which is promoted when there are fewer grooves. A middle ground can be found when atoms are placed intermittently as bridges that disrupt the grooves, creating a stabilizing effect without significant reduction in activity. Finally, it is worth noting that the maximally active design was equivalent to the design proposed by Núñez and Vlachos,[44] suggesting that, at least to a first approximation, their identified design was in fact optimally active.

CONCLUSIONS

In this paper, we presented a general-purpose approach for designing nanostructured materials. For this, we compiled a set of shared features of nanomaterial design problems and then developed modular variables and constraints to represent the basic features of their respective optimization models. To enable wider adoption of this materials design paradigm, we have created *MatOpt*, a Python module that offers object-oriented classes for specifying materials information and for casting mathematical optimization models for their nanostructured design. This paper introduced this software and demonstrated its usage via three examples. A detailed example of the module applied to a nanocluster cohesive energy minimization problem illustrated the basic syntax and logic for creating models. The perovskite dopant optimization model illustrated how to handle periodicity and utilize data from DFT calculations in the context of this optimization task. Finally, the implementation of a bifunctional catalyst surface design model illustrated more complex features and demonstrated the facile generation of Pareto-optimal results. *MatOpt* is distributed as part of the IDAES software distribution, and we hope that it shall enable a rigorous mathematical optimization paradigm in the domain of nanostructured materials design.

DATA AND SOFTWARE AVAILABILITY

The software *MatOpt* is freely available at https://github.com/IDAES/idaes-pse/tree/main/idaes/apps/matopt. All data required to run the case studies presented in this paper is included within the code provided in sections B and C of the Supporting Information.

ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available free of charge at https://pubs.acs.org/doi/10.1021/acs.jcim.1c00984.

Additional information about some of *MatOpt*'s key modeling objects as well as code listings for the perovskite design problem and the bifunctional catalyst design problem (PDF)

AUTHOR INFORMATION

Corresponding Author

Chrysanthos E. Gounaris − *Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, United States;* orcid.org/0000-0001-5779-2510; Email: gounaris@cmu.edu

Authors

Christopher L. Hanselman − *Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, United States*

Xiangyu Yin − *Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, United States;* orcid.org/0000-0003-2868-1728

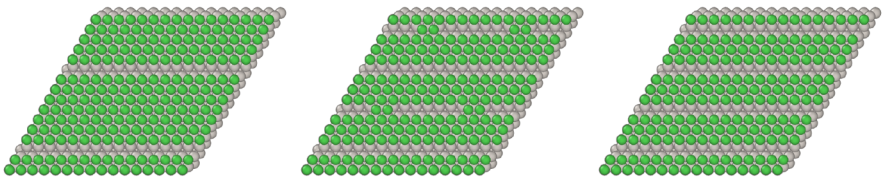


Figure 12. Example Pareto-optimal results for the Ni/Pt {111} bifunctional catalyst example. Normalized activity (A) and surface energy (E) values are provided below each tile.

https://doi.org/10.1021/acs.jcim.1c00984
J. Chem. Inf. Model. 2022, 62, 295−308

David C. Miller − *National Energy Technology Laboratory, Pittsburgh, Pennsylvania 15236, United States;* orcid.org/0000-0002-7378-5625

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.jcim.1c00984

Notes

This paper was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

This work was conducted as part of the Institute for the Design of Advanced Energy Systems (IDAES) with support through the Simulation-Based Engineering, Crosscutting Research Program within the U.S. Department of Energy's Office of Fossil Energy and Carbon Management.

■ REFERENCES

(1) Barth, J. V.; Costantini, G.; Kern, K. Engineering atomic and molecular nanostructures at surfaces. *Nature* **2005**, *437*, 671.

(2) Xia, Y.; Xiong, Y.; Lim, B.; Skrabalak, S. E. Shape-controlled synthesis of metal nanocrystals: simple chemistry meets complex physics? *Angew. Chem.* **2009**, *48*, 60−103.

(3) Xia, Y.; Xia, X.; Peng, H.-C. Shape-controlled synthesis of colloidal metal nanocrystals: thermodynamic versus kinetic products. *J. Am. Chem. Soc.* **2015**, *137*, 7947−7966.

(4) Satyavolu, N. S. R.; Loh, K. Y.; Tan, L. H.; Lu, Y. Discovery of and insights into DNA "codes" for tunable morphologies of metal nanoparticles. *Small* **2019**, *15*, 1900975.

(5) King, M. E.; Personick, M. L. Defects by design: synthesis of palladium nanoparticles with extended twin defects and corrugated surfaces. *Nanoscale* **2017**, *9*, 17914−17921.

(6) Park, J.; Kwon, T.; Kim, J.; Jin, H.; Kim, H. Y.; Kim, B.; Joo, S. H.; Lee, K. Hollow nanoparticles as emerging electrocatalysts for renewable energy conversion reactions. *Chem. Soc. Rev.* **2018**, *47*, 8173−8202.

(7) Nørskov, J. K.; Bligaard, T.; Rossmeisl, J.; Christensen, C. H. Towards the computational design of solid catalysts. *Nat. Chem.* **2009**, *1*, 37−46.

(8) Maurer, R. J.; Freysoldt, C.; Reilly, A. M.; Brandenburg, J. G.; Hofmann, O. T.; Björkman, T.; Lebègue, S.; Tkatchenko, A. Advances in density-functional calculations for materials modeling. *Annu. Rev. Mater. Res.* **2019**, *49*, 1−30.

(9) Garlyyev, B.; Liang, Y.; Xue, S.; Watzele, S.; Fichtner, J.; Li, W.-J.; Ding, X.; Bandarenka, A. S. Theoretical and experimental identification of active electrocatalytic surface sites. *Curr. Opin. Electrochem.* **2019**, *14*, 206−213.

(10) Jain, A.; Ong, S. P.; Hautier, G.; Chen, W.; Richards, W. D.; Dacek, S.; Cholia, S.; Gunter, D.; Skinner, D.; Ceder, G.; Persson, K. A. Commentary: the Materials Project: a materials genome approach to accelerating materials innovation. *APL Mater.* **2013**, *1*, 011002.

(11) Saal, J. E.; Kirklin, S.; Aykol, M.; Meredig, B.; Wolverton, C. Materials design and discovery with high-throughput density functional theory: the open quantum materials database (OQMD). *JOM* **2013**, *65*, 1501−1509.

(12) Curtarolo, S.; Setyawan, W.; Hart, G. L.; Jahnatek, M.; Chepulskii, R. V.; Taylor, R. H.; Wang, S.; Xue, J.; Yang, K.; Levy, O.; Mehl, M. J.; Stokes, H. T.; Demchenko, D. O.; Morgan, D. AFLOW: An automatic framework for high-throughput materials discovery. *Comput. Mater. Sci.* **2012**, *58*, 218−226.

(13) Liu, Y.; Zhao, T.; Ju, W.; Shi, S. Materials discovery and design using machine learning. *J. Materiomics* **2017**, *3*, 159−177.

(14) Ulissi, Z. W.; Singh, A. R.; Tsai, C.; Nørskov, J. K. Automated discovery and construction of surface phase diagrams using machine learning. *J. Phys. Chem. Lett.* **2016**, *7*, 3931−3935.

(15) Ulissi, Z. W.; Medford, A. J.; Bligaard, T.; Nørskov, J. K. To address surface reaction network complexity using scaling relations machine learning and DFT calculations. *Nat. Commun.* **2017**, *8*, 1−7.

(16) Ulissi, Z. W.; Tang, M. T.; Xiao, J.; Liu, X.; Torelli, D. A.; Karamad, M.; Cummins, K.; Hahn, C.; Lewis, N. S.; Jaramillo, T. F.; Chan, K.; Nørskov, J. K. Machine-learning methods enable exhaustive searches for active bimetallic facets and reveal active site motifs for CO2 reduction. *ACS Catal.* **2017**, *7*, 6600−6608.

(17) Goldsmith, B. R.; Esterhuizen, J.; Liu, J.-X.; Bartel, C. J.; Sutton, C. A. Machine learning for heterogeneous catalyst design and discovery. *AIChE J.* **2018**, *64*, 2311−2323.

(18) Tran, K.; Palizhati, A.; Back, S.; Ulissi, Z. W. Dynamic workflows for routine materials discovery in surface science. *J. Chem. Inf. Model.* **2018**, *58*, 2392−2400.

(19) Calle-Vallejo, F.; Martínez, J. I.; García-Lastra, J. M.; Sautet, P.; Loffreda, D. Fast prediction of adsorption properties for platinum nanocatalysts with generalized coordination numbers. *Angew. Chem.* **2014**, *53*, 8316−8319.

(20) Calle-Vallejo, F.; Loffreda, D.; Koper, M. T.; Sautet, P. Introducing structural sensitivity into adsorption−energy scaling relations by means of coordination numbers. *Nat. Chem.* **2015**, *7*, 403−410.

(21) Calle-Vallejo, F.; Tymoczko, J.; Colic, V.; Vu, Q. H.; Pohl, M. D.; Morgenstern, K.; Loffreda, D.; Sautet, P.; Schuhmann, W.; Bandarenka, A. S. Finding optimal surface sites on heterogeneous catalysts by counting nearest neighbors. *Science* **2015**, *350*, 185−189.

(22) Rück, M.; Bandarenka, A.; Calle-Vallejo, F.; Gagliardi, A. Fast identification of optimal pure platinum nanoparticle shapes and sizes for efficient oxygen electroreduction. *Nanoscale Adv.* **2019**, *1*, 2901−2909.

(23) Núñez, M.; Lansford, J.; Vlachos, D. Optimization of the facet structure of transition-metal catalysts applied to the oxygen reduction reaction. *Nat. Chem.* **2019**, *11*, 449−456.

(24) Hanselman, C. L.; Gounaris, C. E. A mathematical optimization framework for the design of nanopatterned surfaces. *AIChE J.* **2016**, *62*, 3250−3263.

(25) Hanselman, C. L.; Zhong, W.; Tran, K.; Ulissi, Z. W.; Gounaris, C. E. Optimization-based design of active and stable nanostructured surfaces. *J. Phys. Chem. C* **2019**, *123*, 29209−29218.

(26) Hanselman, C. L.; Tafen, D. N.; Alfonso, D. R.; Lekse, J. W.; Matranga, C.; Miller, D. C.; Gounaris, C. E. Design of Doped Perovskite Oxygen Carriers Using Mathematical Optimization. *Comput.-Aided Chem. Eng.* **2018**, *44*, 2461−2466.

(27) Hanselman, C. L.; Tafen, D. N.; Alfonso, D. R.; Lekse, J. W.; Matranga, C.; Miller, D. C.; Gounaris, C. E. A framework for optimizing oxygen vacancy formation in doped perovskites. *Comput. Chem. Eng.* **2019**, *126*, 168−177.

(28) Isenberg, N. M.; Taylor, M. G.; Yan, Z.; Hanselman, C. L.; Mpourmpakis, G.; Gounaris, C. E. Identification of optimally stable nanocluster geometries via mathematical optimization and density-functional theory. *Mol. Sys. Des. Eng.* **2020**, *5*, 232−244.

(29) Yin, X.; Isenberg, N. M.; Hanselman, C. L.; Dean, J. R.; Mpourmpakis, G.; Gounaris, C. E. Designing stable bimetallic nanoclusters via an iterative two-step optimization approach. *Mol. Sys. Des. Eng.* **2021**, *6*, 545−557.

(30) Yin, X.; Gounaris, C. E. Search methods for inorganic materials crystal structure prediction. *Curr. Opin. Chem. Eng.* **2022**, *35*, 100726.
(31) Goronzy, D. P.; Ebrahimi, M.; Rosei, F.; Arramel; Fang, Y.; De Feyter, S.; Tait, S. L.; Wang, C.; Beton, P. H.; Wee, A. T.; Weiss, P. S.; Perepichka, D. F. Supramolecular assemblies on surfaces: Nanopatterning, functionality, and reactivity. *ACS Nano* **2018**, *12*, 7445−7481.
(32) National Energy Technology Laboratory, Institute for the Design of Advanced Energy Systems (IDAES). www.idaes.org/download/.
(33) IDAES PSE framework documentation. https://idaes-pse.readthedocs.io/en/stable/getting_started/index.html.
(34) MatOpt documentation. https://idaes-pse.readthedocs.io/en/stable/user_guide/modeling_extensions/matopt/index.html.
(35) MatOpt use cases. https://idaes.github.io/examples-pse/latest/Examples/MatOpt/index.html.
(36) Gropp, W.; Moré, J. J. Optimization Environments and the NEOS Server. *Approximation Theory and Optimization*, Buhman, M. D., Iserles, A., Eds.; Cambridge University Press, 1997; pp 167−182.
(37) Czyzyk, J.; Mesnier, M. P.; Moré, J. J. The NEOS server. *IEEE Comput. Sci. Eng.* **1998**, *5*, 68−75.
(38) Dolan, E. D. NEOS Server 4.0 Administrative Guide. *arXiv.org*, 2001, cs/0107034. https://arxiv.org/abs/cs/0107034.
(39) Larsen, A. H.; Mortensen, J. J.; Blomqvist, J.; Castelli, I. E.; Christensen, R.; Dułak, M.; Friis, J.; Groves, M. N.; Hammer, B.; Hargus, C.; Hermes, E. D.; Jennings, P. C.; Jensen, P. B.; Kermode, J.; Kitchin, J. R.; Kolsbjerg, E. L.; Kubal, J.; Kaasbjerg, K.; Lysgaard, S.; Maronsson, J. B.; Maxson, T.; Olsen, T.; Pastewka, L.; Peterson, A.; Rostgaard, C.; Schiotz, J.; Schutt, O.; Strange, M.; Thygesen, K. S.; Vegge, T.; Vilhelmsen, L.; Walter, M.; Zeng, Z.; Jacobsen, K. W. The atomic simulation environment—a Python library for working with atoms. *J. Phys.: Condens. Matter* **2017**, *29*, 273002.
(40) Ong, S. P.; Richards, W. D.; Jain, A.; Hautier, G.; Kocher, M.; Cholia, S.; Gunter, D.; Chevrier, V. L.; Persson, K. A.; Ceder, G. Python Materials Genomics (PyMatGen): a robust, open-source python library for materials analysis. *Comput. Mater. Sci.* **2013**, *68*, 314−319.
(41) Plimpton, S. Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.* **1995**, *117*, 1−19.
(42) Hart, W. E.; Watson, J.-P.; Woodruff, D. L. Pyomo: modeling and solving mathematical programs in Python. *Math. Program.* **2011**, *3*, 219−260.
(43) Hart, W. E.; Laird, C. D.; Watson, J.-P.; Woodruff, D. L.; Hackebeil, G. A.; Nicholson, B. L.; Siirola, J. D. *Pyomo-optimization modeling in python*; Springer, 2017; Vol. *67*.
(44) Núñez, M.; Vlachos, D. G. Multiscale modeling combined with active learning for microstructure optimization of bifunctional catalysts. *Ind. Eng. Chem. Res.* **2019**, *58*, 6146−6154.
(45) Tomanek, D.; Mukherjee, S.; Bennemann, K. Simple theory for the electronic and atomic structure of small clusters. *Phys. Rev. B* **1983**, *28*, 665.
(46) Oliphant, T. E. *A guide to NumPy*; Trelgol Publishing USA, 2006; Vol. *1*.
(47) Berman, H. M.; Westbrook, J.; Feng, Z.; Gilliland, G.; Bhat, T. N.; Weissig, H.; Shindyalov, I. N.; Bourne, P. E. The protein data bank. *Nucleic Acids Res.* **2000**, *28*, 235−242.

https://doi.org/10.1021/acs.jcim.1c00984
J. Chem. Inf. Model. 2022, 62, 295−308