University of Antwerp

Faculty of Pharmaceutical, Biomedical and Veterinary Sciences
Department of Pharmaceutical Sciences

Computational design of synthesizable molecules by imitating reference chemistry

PhD thesis submitted for the degree of Doctor of Pharmaceutical Sciences at the University of Antwerp to be defended by Alan Kerstjens

Supervisor(s):
Hans De Winter

Antwerp, 2024

Table of contents

Summary

De novo molecular design is the practice of generating molecules with desirable properties from scratch. When done computationally the proposed molecules tend to be difficult to synthesize and overall chemically unappealing. In this work we present methods to extract patterns from available data and bias molecular design towards synthetically accessible chemistry. Given a list of known synthesizable compounds, we design molecules with the same chemical features, under the assumption that this resemblance increases the likelihood of them being synthesizable as well.

Molecules were designed using evolutionary algorithms that breed populations of molecules by modifying their molecular graphs. The designed molecules were constrained to be composed of chemical features that are prevalent in reference chemistry. Different ways of defining chemical features were explored, and we determined that mimicking small circular atomic environments allowed us to design reasonably fit and easy to synthesize molecules.

We developed an evolutionary algorithm that constructs molecules with desirable chemical features by assembling molecular fragments in a computationally efficient way, and showed how it outperformed competing algorithms in both the quality of the generated molecules and its ability to navigate chemical space effectively.

We also developed a molecule correction algorithm that can identify flaws in molecules and sanitize them to make the molecules more desirable. Said tool can be used to post-process molecules generated elsehow, or integrated into molecule generators to enforce chemical constraints in a hands-off fashion.

Dutch Summary

De novo moleculair ontwerp betreft het genereren van moleculen met wenselijke eigenschappen vanuit het niets. Wanneer dit computationeel wordt gedaan, zijn de voorgestelde moleculen vaak moeilijk te synthetiseren en chemisch onaantrekkelijk. In dit werk presenteren we methoden om patronen te extraheren uit beschikbare data en moleculair ontwerp te sturen in de richting van synthetisch toegankelijke chemie. Gegeven een lijst van bekende synthetiseerbare moleculen, ontwerpen we moleculen met dezelfde chemische eigenschappen, in de veronderstelling dat deze gelijkenis de waarschijnlijkheid vergroot dat ze ook synthetiseerbaar zijn.

Moleculen werden ontworpen met evolutionaire algoritmen die populaties van moleculen fokken door hun moleculaire grafieken te wijzigen. We dwongen de moleculen om samengesteld te zijn uit chemische eigenschappen die veel voorkomen in de referentiechemie. Verschillende manieren om chemische eigenschappen te definiëren werden onderzocht en we stelden vast dat het nabootsen van kleine cirkelvormige atomaire omgevingen ons in staat stelde om redelijk geschikte moleculen te ontwerpen.

We ontwikkelden een evolutionair algoritme dat moleculen construeert met gewenste chemische eigenschappen door moleculaire fragmenten op een computationeel efficiënte manier samen te voegen, en toonden hoe het beter presteerde dan concurrerende algoritmen in zowel de kwaliteit van de gegenereerde moleculen als het vermogen om effectief door de chemische ruimte te navigeren.

We ontwikkelden ook een algoritme voor molecuulcorrectie dat gebreken in moleculen kan identificeren en ze kan zuiveren om de moleculen wenselijker te maken. Dit gereedschap kan worden gebruikt om moleculen die op een andere manier zijn gegenereerd achteraf te bewerken of kan worden geïntegreerd in moleculengeneratoren om chemische beperkingen op een hands-off manier af te dwingen.

Chapter 1 Introduction

1.1 Molecular design

Molecular design is the art of crafting molecules with specific properties and/or functions. It lays at the core of the pharmaceutical industry.

When designing a molecule one formulates a hypothesis about its properties, acquires said molecule, presumably by synthesizing it, and assays the molecule for its properties to test the hypothesis. Unfortunately, our limited understanding of the universe makes formulating strong hypotheses challenging. Consequently, molecular design is an iterative process with many “design-make-test” cycles. As of today the “make” and “test” parts of this cycle have to be performed by humans in a laboratory, making molecular design a resource intensive process. Intelligent prioritization of molecules during the “design” stage could dramatically reduce the cost of developing novel chemical entities.

Computers can aid in the molecular design process. A task of special interest is the virtual prediction of molecular properties. *In silico* assays, generically referred to as *scoring functions* or *objective functions*, tend to be more resource efficient than their *in vitro*, and especially *in vivo*, counterparts. These assays can be used on a large scale to screen virtual libraries of molecules and identify promising compounds for further testing in the lab. This process, known as *virtual screening*, has proven its worth as a useful tool in molecular discovery [1]. Commonly the molecules being screened are either commercially available [2] or predicted to be easy to synthesize [3–5], enabling a fast transition from *in silico* to *in vitro* studies. However, given that even the largest virtual libraries [6] dwarf in size compared to chemical space, which is commonly cited to contain somewhere between 10^{23} and 10^{60} [7–9] drug-like molecules, it’s unlikely that the library will contain the most potent and attractive molecules. Preferences for certain chemotypes and synthetic reactions [10, 11] often make their way to virtual libraries, leading to a small and non-uniform coverage of chemical space [12, 13]. This, coupled to the fact that publicly available libraries may have been screened previously or even contain patent-protected molecules, raises concerns about a lack of chemical novelty. Virtual screening can be thought of as a blind search through chemical space, with molecules being tested randomly. This constitutes a rather inefficient use of computational resources.

In the early 1990s it was postulated that given a scoring function that predicts a molecule’s properties one could directly construct molecules with desirable properties as opposed to searching for them in pre-enumerated libraries. This process was described as “inverse QSAR” [14, 15]. Early on the focus was on optimizing already validated hits with the assistance of a scoring function. It did not take long for some to become more ambitious and aim to design molecules from scratch using solely the feedback from the scoring function [16–21]. This gave rise to the field of computational *de novo molecular design*. The field boomed up until the late 2000s, when enthusiasm started to die down. Two reasons are commonly cited for this decay in popularity [22]. Firstly, the scoring functions available at the time were lackluster. Most methods employed crude structure-based scoring functions, and ligand-based scoring functions were still in their infancy. Secondly, and perhaps most importantly, the focus was almost entirely on designing molecules with high predicted scores, with other properties being neglected in the process. Many of the designed molecules were difficult to synthesize and, given the inaccuracies of the scoring functions, one can imagine many chemists undertook challenging syntheses only to be disappointed by false positives.

Recently advances in computational power, scoring function accuracy and molecule manipulation techniques, as well as greater access to molecular data, have reinvigorated the field. The present work is part of this rebirth and revolves around the development of software to construct and optimize molecules based on the feedback of some problem-specific scoring function.

1.2 Molecular representation

Computational chemists describe molecules and their behavior mathematically. These descriptions may be two- or three-dimensional, and may obey different levels of chemical theory. The most pervasive and basic molecular description is the *molecular graph*. While the name may be foreign to some, virtually everyone is familiar with the concept of molecular graphs, as they are the *de facto* standard for molecule depiction (Figure 1.1). Graphs are data structures containing objects or *vertices*, where relationships between said objects are expressed as *edges*. In a molecular graph vertices represent atoms, and edges represent bonds. Molecular graphs represent the molecule's *topology* or connectivity. Molecular graphs are usually:

- *Undirected*. Edges have no directionality and can be traversed in both directions.
- *Unweighted*. Edges have no associated weights, or alternatively, unit weights.
- *Simple*. Parallel edges and self-edges or loops are disallowed. In other words, two vertices may be connected by at most one edge, and connected vertices must be distinct.
- *Connected*. A path exists between any two vertices of the graph. Occasionally disconnected graphs are used to represent distinct molecules functioning jointly or stabilizing each other, such as in the case of salts.

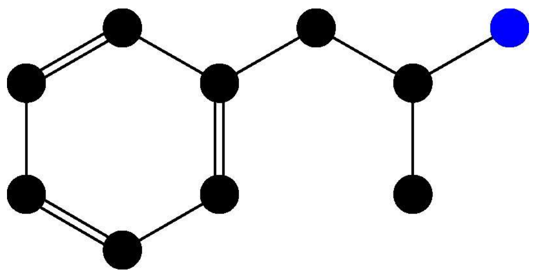


Figure 1.1. An example molecular graph. Balls (vertices) represent atoms, while sticks (edges) represent bonds.

Two things set molecular graphs apart from other undirected, unweighted and simple graphs. Firstly, the vertices and edges store (numeric) information about the atoms and bonds they represent. This may include the atomic number, mass number, formal charge, bond order etc. Hydrogens are usually not included explicitly in the topology but rather treated implicitly as a property of the non-hydrogen atoms instead. Vertices may also convey some information about the molecule’s 3D structure in the form of stereochemistry annotations or plain 3D coordinates.

Secondly, molecular graphs are expected to follow a set of rules founded in chemical theory to represent reasonable molecules. Yet it is important to recognize that these are merely expectations, and that mathematically there are no limits to a graph’s topology and annotations. Theoretically a molecular graph can represent chemically unstable, unreasonable or impossible entities.

Some graph concepts of importance later in this work are defined as follows:

- The *degree* of a vertex is equal to the number of edges associated with it.
- A *path* is a sequence of connected vertices in a graph. It is a way to traverse from one vertex to another following a sequence of adjacent edges. More than one path may exist between two vertices, with the shortest one known as the *shortest path*.
- The *topological distance* between two vertices is equal to the number of edges in the shortest path between both vertices. For example, two adjacent atoms are at a topological distance of 1.
- A *cycle* is a closed path where the starting and ending vertex are the same. In cheminformatics cycles are often called *rings*.
- The *minimum cycle basis* of a graph is a set of cycles that contains the fewest possible number of cycles while still representing all the cycles in the graph. Each cycle in the basis is unique and cannot be formed by combining other cycles in the set. In cheminformatics the minimum cycle basis is often referred to as the *Smallest Set of Smallest Rings* (SSSR) [23].

1.3 Molecular characterization

Given a molecular representation such as a molecular graph one can calculate certain properties of the represented molecule. These calculated properties are called *molecular descriptors*. Examples of simple descriptors include the molecular weight and octanol-water partition coefficient (logP). Descriptor values can be predictive of higher order molecular properties. For example, the renowned Lipinski's rule of five predicts bioavailability based on simple physicochemical descriptors [24].

However, not all properties can be satisfactorily predicted as a function of simple physicochemical descriptors. For instance, binding affinity is a highly complex trait that depends on the molecule's topology, as specific functional groups must be in specific positions to be able to interact with a biological target. This triggered the development of molecular descriptors that characterize a molecule's topology. Early efforts revolved around the development of topological indices, which are single numbers characterizing some aspect of a molecule's connectivity [25–28]. While topological indices capture some information about a molecule's topology, they do not provide explicit details about functional groups, substructures, or atom arrangements. Structural keys were developed in response [29, 30]. A structural key is a boolean array where each boolean or bit encodes the absence or presence of a chemical substructure, for example a functional group (Figure 1.2). Structural keys have two major drawbacks. Firstly, searching for substructures in a molecule is a variant of the subgraph isomorphism problem, which is known to be computationally expensive to solve [31]. Secondly, and perhaps most importantly, structural keys suffer from a lack of generality, as the substructures encoded in the key may not be relevant in every problem domain.

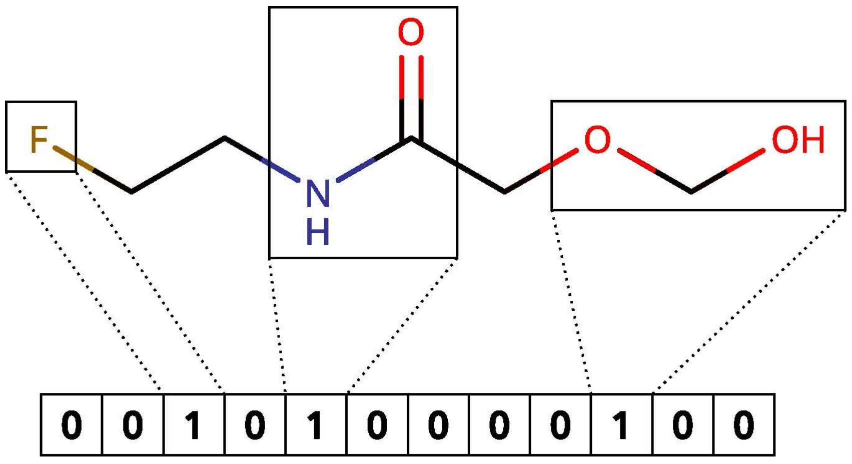


Figure 1.2. Example structural key. Certain functional groups map to certain bits in a bitstring. The bit is set if the functional group is present, and unset otherwise.

Molecular fingerprints are a generalization of structural keys that bypasses the need for defining substructures of interest. Much like structural keys they are usually represented as bit arrays, with the difference being that each bit maps to a large number of undefined patterns instead of mapping to a single predefined pattern. Chemical features are generated algorithmically. These features may include atom pairs [32], topological paths [33] and/or circular environments [34]. Thereafter they are *hashed* to an integer, which acts as the bit index, and can be understood as the feature's identifier (Figure 1.3). *Hashing* is the process of deterministically converting an input of arbitrary size to an output of fixed size. In our case given an input chemical feature we generate a seemingly random integer between 0 and some maximum value, typically a power of two such as 2^{32}. Two similar yet distinct chemical features are hashed to entirely unrelated integers. Since the output space is smaller than the input space there is also a probability of two distinct features hashing to the same integer by chance. This is known as a hash *collision*. The probability of a collision is larger the smaller the output space. A good hashing function distributes hash values uniformly over the output space to minimize the probability of a hash collision.

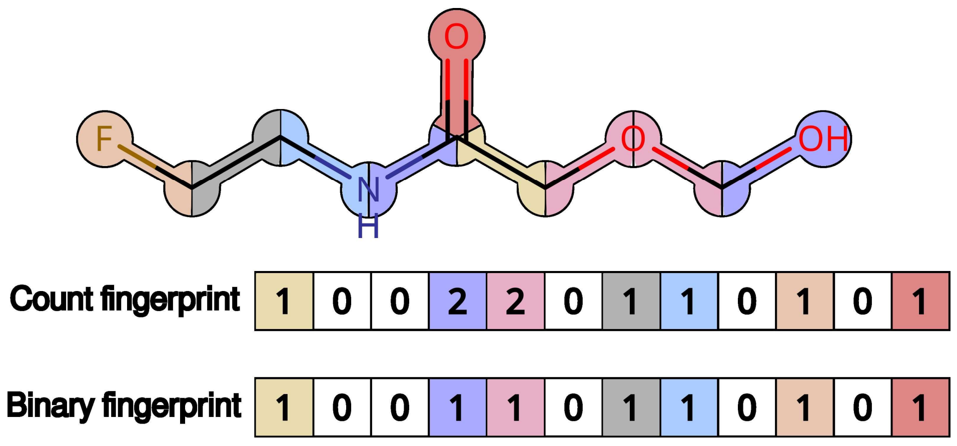


Figure 1.3. Example path-based molecular fingerprint. Each bond is mapped to a bit, typically through hashing. Two different bonds can map to the same bit by chance (hash collision). One can either count how many times each feature occurs, or simply denote its binary presence/absence.

Molecular fingerprints are a convenient and efficient way of characterizing the topology of a molecule as a set, that is, a collection of ordered integers. One can efficiently calculate the similarity between two sets, and therefore molecular fingerprints, using a *similarity index* such as the Tanimoto, Dice or Tversky index [35]. Since molecular similarity is one of the cornerstones of computational chemistry [36, 37] it is no surprise that molecular fingerprints are applied to a wide variety of tasks ranging from database searches [38, 39] to molecular clustering [39, 40], featurization of machine learning training data [41] and ligand-based virtual screening [42–44].

Arguably the most popular type of molecular fingerprint is the *Extended Connectivity Fingerprint* (ECFP) [34]. ECFP falls under the category of circular fingerprints. In a circular fingerprint a bit signals the absence or presence of a *circular atomic environment*. A circular atomic environment is defined as a central atom and all atoms and bonds within a given topological distance of it, called the environment's radius *r*. The ECFP algorithm starts by assigning initial atom identifiers to a molecule's atoms. Commonly these identifiers are the hash of some atomic properties. These atomic properties or atomic invariants typically include the atom's degree, valence, atomic number, mass number, formal charge, number of hydrogens and sometimes ring membership or stereochemistry annotations [34, 45]. Atomic identifiers are iteratively updated following the Morgan algorithm [46], which combines the identifiers of a central atom and its adjacent atoms (Figure 1.4). In modern implementations this is achieved with a hashing function [34]. Since the ECFP algorithm is based on the Morgan algorithm some re-implementations of the original ECFP have been called Morgan fingerprints [47].

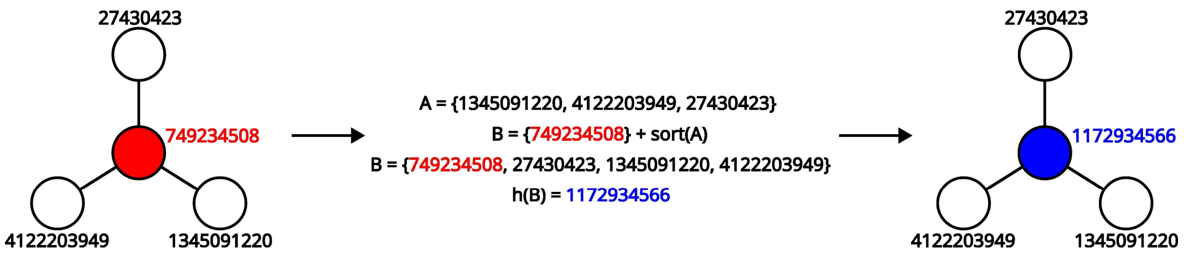

27430423
749234508
4122203949
1345091220
A = {1345091220, 4122203949, 27430423}
B = {749234508} + sort(A)
B = {749234508, 27430423, 1345091220, 4122203949}
h(B) = 1172934566
27430423
1172934566
4122203949
1345091220

Figure 1.4. Illustration of a single iteration of the Morgan algorithm. Each atom is characterized with an integer identifier. In the first iteration this is commonly the hash of some of the atom's properties. For each atom the identifiers of itself and its neighbors are collected and aggregated with a hashing function, resulting in a seemingly random number. Said random number becomes the new atom's identifier for the next iteration, as well as serving as a bit index into a molecular fingerprint.

Different ECFP variants, denoted as ECFP{*2r*}, can be distinguished depending on the number of iterations *r* of the algorithm. For example, *r = 2* iterations of the algorithm would yield ECFP*4*. There is a direct correlation between the number of algorithm iterations and the radius of the resulting atomic environments (Figure 1.5).

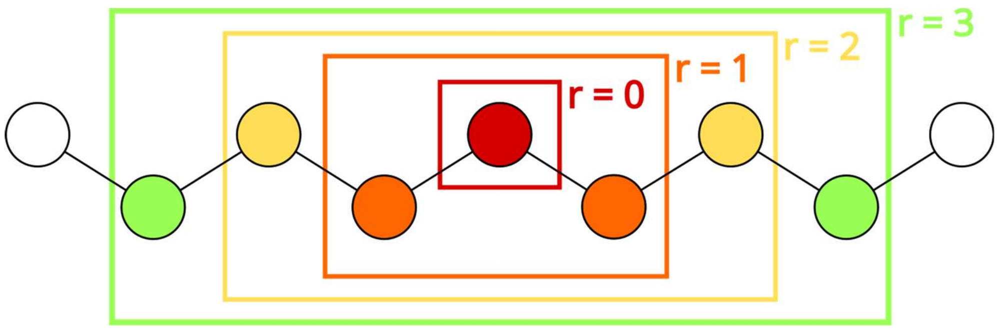


Figure 1.5. Circular atomic environments (centered on the red atom) defined by r iterations of the Morgan algorithm. Note that bigger environments encompass smaller environments.

1.4 Chemical space

Chemical space is an abstract concept referring to the theoretical collection of all possible chemical entities. It is common to narrow down general chemical space to subspaces containing specific types of molecules. For example, one could define a drug-like chemical space containing exclusively drug-like molecules. The size of drug-like chemical space has been a topic of much debate. Frequently quoted figures range between 10^{23} and 10^{60} drug-like molecules [7–9, 48]. In any case, it is clear that its magnitude is astronomical.

Chemical space can be thought of as a multidimensional similarity-based arrangement of molecules. A molecule corresponds to a point in chemical space, and similar molecules, according to some criterion, are close to each other in chemical space. Depending on the application different similarity criteria may be used to define chemical space [49–51].

Of interest to us is a chemical space defined based on molecular graph similarity, where molecules with similar topologies are proximal. Since molecular properties are generally assumed to be linked to the structure of the molecule, it stands to reason that molecules with similar structures will also have similar properties [36, 37]. Indeed, this has become one of the cornerstone theorems of molecular design. The go-to molecular design strategy is to explore regions of chemical space surrounding reference molecules with promising properties, with the hopes of finding even more appealing molecules in their neighborhood.

For visualization purposes it is common to represent chemical space as a two-dimensional coordinate system using physicochemical descriptors or principal components thereof as axes (Figure 1.6A) [49]. Alternatively one could use chemical features like the ones in molecular fingerprints as dimensions instead [13, 51]. The aforementioned techniques embed molecules in a continuous space by representing them as descriptor vectors. Since molecules are discrete states, and chemical space is a discrete space, continuous representations can be a bit misleading. It may be preferable to represent chemical space as a graph, where each vertex is a molecule and edges represent relationships between molecules (Figure 1.6B). When considering molecules as static entities these edges could represent topological similarities. In this work molecules will be treated as dynamic entities instead, in the sense that their molecular graph can be altered. In this case given a graph-like representation of chemical space edges would represent transitions between related molecules [52–54].

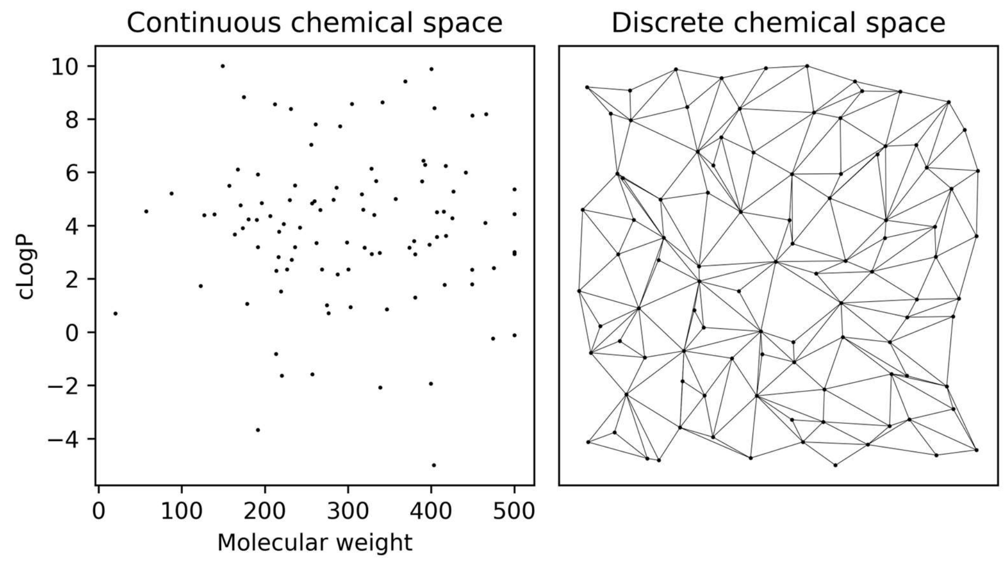


Figure 1.6. Examples of chemical space representations. Chemical space is commonly thought of as a continuum, with the dimensions given by molecular descriptors (left). Alternatively, one can view chemical space as a dimensionless transition graph, where edges represent transformations between related molecules (right).

1.5 Objective functions

Many scientific problems can be formalized as optimization problems. Optimization involves finding the best or sufficiently good solutions among a set of possible *solutions* or *states*. The quality of a solution is quantified with one or more *objective functions*. The value of the objective function for a specific solution is termed *objective value*, *fitness* or *score*, and the process of measuring it is called *evaluating* the objective function.

Objective function design and selection is entirely dependent on the problem to be tackled. Depending on whether the objective function measures desirability or undesirability the function ought to be maximized or minimized, respectively. In drug discovery and development one might be interested in maximizing the binding potency or biological effect of a molecule with respect to its biological target. These properties can be measured with constants of dissociation (K_d or K_i) and effective concentrations (EC_{50} or IC_{50}), respectively. As these metrics relate to the concentration required to achieve a certain degree of binding or activity, the goal is to minimize them. Alternatively, one might be interested in maximizing the viability or survival rate of some organism when exposed to some substance.

A peculiarity of chemical and biological research is that the preferred objective functions are commonly wet laboratory experiments. The results of these experiments are widely trusted and often taken as ground truth. Unfortunately, their cost and throughput may limit the number of times the objective functions can be evaluated. *Ex silico* optimization strategies minimize the number of molecules to synthesize and evaluate through human expertise. In absence of said expertise, *in silico* optimization strategies rely on evaluating objective functions orders of magnitude more times than what wet lab experiment can ordinarily support. Hence, in computational molecular optimization the objective functions are usually surrogate objectives that mathematically and rapidly *predict* the underlying ground truth measured in the lab (Figure 1.7). For example, instead of measuring a K_i one could measure an interaction energy according to molecular docking, or instead of measuring an IC_{50} one could predict inhibitory effect using a machine learning model. While the accuracy of any one individual prediction is a topic of hot debate [55–58], the hope is that at a larger scale the objective function captures the statistical distribution of the ground truth. Additionally, the higher throughput enables a broader exploration of chemical space than what would have been possible *in vitro* [2, 59].

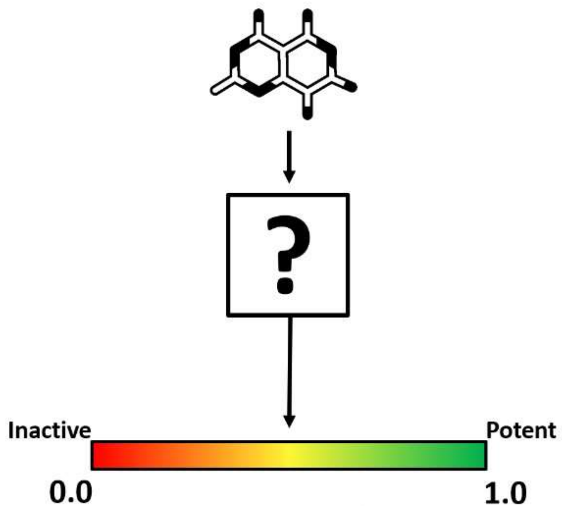


Figure 1.7. A black box objective function predicting the biological activity of a molecule.

1.6 Multi-objective optimization

While we tend to focus on biological activity, it is important to note that drug discovery is inherently a *multi-objective* problem. Other objectives include synthetic accessibility, good Absorption, Distribution, Metabolism and Excretion (ADME) properties, limited toxicity and off-target effects and chemical novelty.

Optimizing multiple objectives simultaneously can be challenging. In fact, if two objectives are conflicting it may be impossible to satisfy both simultaneously, forcing us to settle on a trade-off. By far the most popular approach to finding molecules satisfying multiple criteria is combining multiple objectives into a single composite objective, usually through linear combination, with each objective being assigned *a priori* some importance or weight. Some of these objectives may be implemented as filters, with the solution being unacceptable if a certain objective is not met [60–62]. When using composite objectives the optimizer is free to sacrifice some of them in favor of others, or return average molecules fulfilling all objectives to some extent but at an overall weak level [63].

Some optimization strategies do not require the user to define an objective trade-off [60, 64–66, 66–71]. These techniques generate Pareto optimal solutions, that is, solutions for which no objective can be improved without hurting another. Pareto optimal solutions define a so called Pareto front representing different objective trade-offs from which the user selects solutions *a posteriori*. Pareto optimization is not perfect either, for it is known to scale poorly with the number of objectives [72]. As the number of dimensions increases so does the number of Pareto optimal solutions. This, coupled to challenges in visualizing and interpreting high-dimensional data, complicates solution selection. Moreover the cost of evaluating Pareto optimality and the objectives themselves increases rapidly.

Given the aforementioned challenges, it is perhaps unsurprising that many researchers try to sidestep formal multi-objective optimization. Instead of considering every objective explicitly some objectives can be enforced implicitly by constraining the way in which solutions are generated. By reducing the number of objectives to optimize one has access to simpler optimization algorithms and analysis tools. Furthermore the cost of explicitly evaluating objectives is negated.

When optimizing molecules, objectives such as drug-likeness and synthesizability can be captured implicitly by the way in which molecules are modified or constructed. A historically popular approach has been to construct chemicals as combinations of smaller molecular fragments. Fragments may be systematically extracted from reference molecules [61, 73–79] or sourced from commercial reagent libraries [80, 81]. The use of fragments is not sufficient to guarantee that the assembled molecules will be synthesizable, as one must ensure that the bonds formed between fragments are also reasonable. Fragment combination can be governed by rules that range in chemical sophistication from knowledge-based bonding [73–75] to simulated chemical reactions [76, 77, 80, 80–85].

More recent research efforts have focused on generative models, that is, machine learning models trained on sets of (synthesizable) molecules to learn chemical distributions, and capable of sampling molecular representations from them [86–88]. Variational Auto-Encoders (VAE) can translate back and forth between discrete and abstract continuous numerical molecular representations, with molecule optimization taking place in the latter [67, 89, 90]. Recurrent Neural Networks (RNN) can sample molecules from chemical space by iteratively growing a molecule, conditioning the next action on the existing molecular context [91–93].

1.7 Optimization algorithms

In the context of optimization, *solution space* refers to the set of potential solutions for an optimization problem. An *optimization algorithm* - sometimes called a *search algorithm* - is a procedure or method to find solutions within the solution space that minimize or maximize the objective function. Due to optimization constraints, the optimization algorithm may be able to explore only a subset of the solution space. This subset is called the *search space*.

The objective values for all states in the solution space define a surface called *objective landscape* or *fitness landscape*. In the case of multiple objective functions, each objective function has an associated fitness landscape. The low and high points on these landscapes are called *minima* and *maxima*, respectively. The term *extrema* can be used to generically refer to both. Extrema can be local or global. A *local* extremum is a solution for which the objective value is either lower or higher than the objective values of its neighbors. The *global* extrema are those for which the objective values are at their absolute lowest or highest. Complete knowledge of the fitness landscapes would enable perfect retrieval of solutions residing in fitness extrema. Unfortunately, the sort of problems that are usually tackled with optimization algorithms tend to have extremely large solution spaces. Hence, in practice only small sections of the fitness landscape are characterized and assumptions are made about the landscape that lay beyond.

The choice of optimization algorithm for a specific problem depends on the nature of the solution space and fitness landscape. In molecular design the solution space is chemical space. The sheer size of chemical space forbids substantial systematic exploration. Chemical space is discrete. Without embedding [89, 90], this rules out gradient-based optimization algorithms that require a continuous solution space, the likes of which are common in other fields. Objective functions used in molecular design attempt to model chemical and biological reality. Chemistry and biology are complex fields rife with exceptions and partially understood phenomena. When coupled with inevitable inaccuracies of the objective function, one should expect the occasional abrupt fitness change or *activity cliff* between neighboring molecules [94, 95]. Indeed, fitness landscapes relevant to molecular design are rugged and bountiful in peaks and troughs (Figure 1.8). This can pose major challenges to an optimization algorithm. Firstly, assumptions made by an algorithm about the topography of the fitness landscape may be incorrect. Secondly, the abundance of local extrema increases the likelihood of an algorithm getting “stuck” in one of these extrema as no better solution may be found in their immediate proximity.

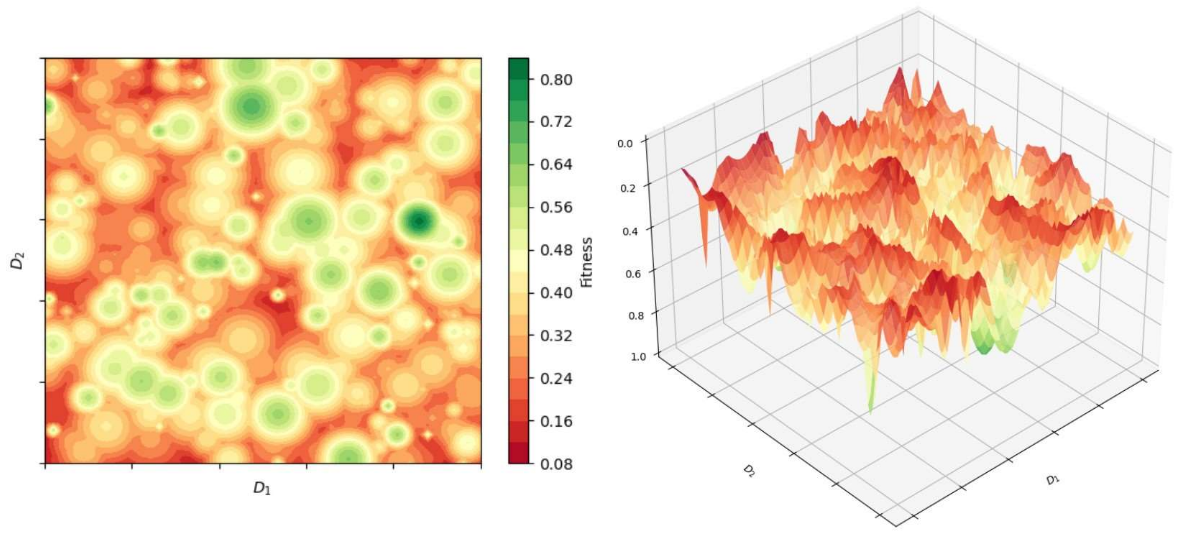


Figure 1.8. A hypothetical drug-discovery fitness landscape. In this example chemical space is a function of two continuous dimensions (D_1 and D_2). The Z-axis or height represents the fitness values of molecules. The left pane is a contour plot of the fitness landscape while the right pane is a 3D view. In this example the fittest molecules are found in the deepest valleys, and therefore our task is to minimize the fitness function.

Summarizing, a chemical space exploration algorithm must cope with a large and discrete solution space and rugged fitness landscapes. Algorithms meeting these requirements are usually *heuristic optimization algorithms*. Heuristic algorithms are approximate problem-solving methods. They often involve iterative steps of evaluating and improving candidate solutions, and frequently incorporate a stochastic component. While they cannot guarantee the discovery of global extrema, they can find reasonably good solutions in a timely manner. A wide variety of heuristic optimization algorithms have been employed in molecular design including simulated annealing [96, 97], Markov chain models [73], particle swarm optimization [98, 99], Monte Carlo tree search [100] and reinforcement learning [86, 92, 101], just to name a few.

Two search algorithms particularly relevant to this work are tree searches and evolutionary algorithms, which will be described in more detail in the following sections.

1.7.1 Tree searches

Trees are acyclic, connected and directed graphs expressing hierarchical relationships between vertices through directed edges. Vertices directly connected to a given vertex are called *adjacent* or *neighboring* vertices. When the tree is directed a vertex may have *incoming* and *outgoing* edges, and the adjacent vertices can be classified into *predecessors* or *parents* and *successors* or *children*, respectively. Vertices without successors are called *leaf* vertices. A tree has a single vertex without predecessors called the *root* vertex.

A tree search is a graph traversal technique that starting from the root vertex progressively visits connected vertices until all vertices have been visited once or an alternative termination criterion is met. It is this traversal that defines the tree. A vertex is chosen and one of its non-traversed edges is traversed leading to another vertex which, if it has not been visited yet, is added to the tree. This process is referred to as *expanding* the vertex. In an unweighted tree the distance between a vertex and the root vertex is the vertex's *depth*.

Tree searches are immensely flexible in that one can devise infinitely many strategies to select which vertex to expand. These strategies are sometimes called *policies*. The simplest search strategies are greedy search and Breadth-First Search (BFS), which are purely exploitative or purely explorative, respectively. In a greedy search the fittest vertex is always chosen for expansion. In BFS the expandable vertex with lowest depth is chosen for expansion, causing the tree to be expanded in a breadthward motion, "level by level". Greedy search finds solutions rapidly, but it favors deep searches and is likely to miss the optimal solution. BFS favors shallow searches and always find the optimal solution, but if the size of the tree is large it becomes computationally intractable. Practical policies are usually a hybrid of both, offering a balance between exploitation and exploration (Figure 1.9).

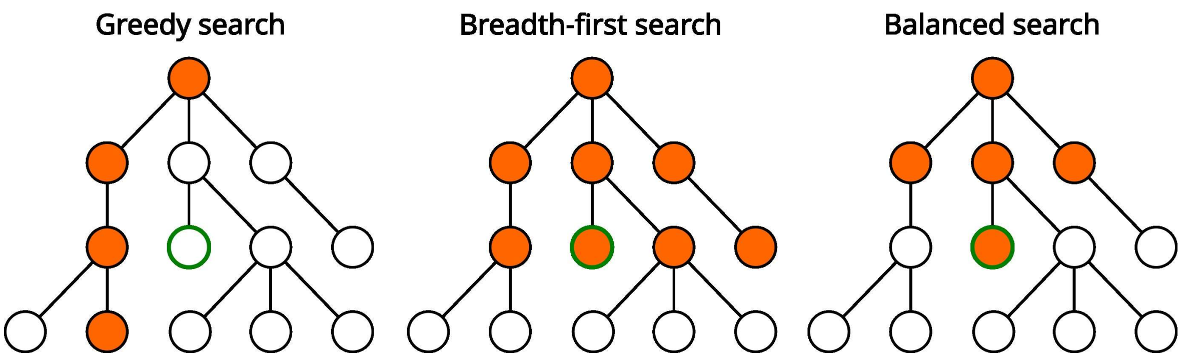


Figure 1.9. Different types of tree search policies applied to the same tree. The goal is to find the optimal vertex (green) starting from the root vertex. The vertices visited by the search are highlighted in orange. The more vertices are visited, the larger the cost of the search. Greedy search is the cheapest among the three, but missed the goal vertex. BFS finds the goal vertex but spends the largest amount of resources doing so. An ideal policy would find the goal vertex without exploring the whole tree.

It is common to apply tree searches to finite and fixed graphs that are fully defined and enumerated. As mentioned previously, chemical space can also be contextualized as a graph. However, due to its size it cannot be fully enumerated. In fact, if molecular size is not a consideration, chemical space is an infinite graph. Search algorithms such tree searches are still applicable, provided that vertex expansions procedurally generate the graph on the go [52–54].

1.7.2 Evolutionary algorithms

Evolutionary algorithms are population-based heuristic optimization algorithms inspired by genetics and biological evolution. A population of candidate solutions or states is considered. Each state or *individual* is expressed as some form of data structure, sometimes called the *chromosome*. Each generation, some of the individuals reproduce to generate offspring. The children are genetically distinct from their parents due to stochastic *mutations* and *recombination*, typically some form of crossover. The fitness of individuals is evaluated by a fitness function, and the fittest individuals are most likely to reproduce and survive. The fitness function exerts selective pressure on the population, driving it towards optimality in a process analogous to Darwinian natural selection (Figure 1.10).

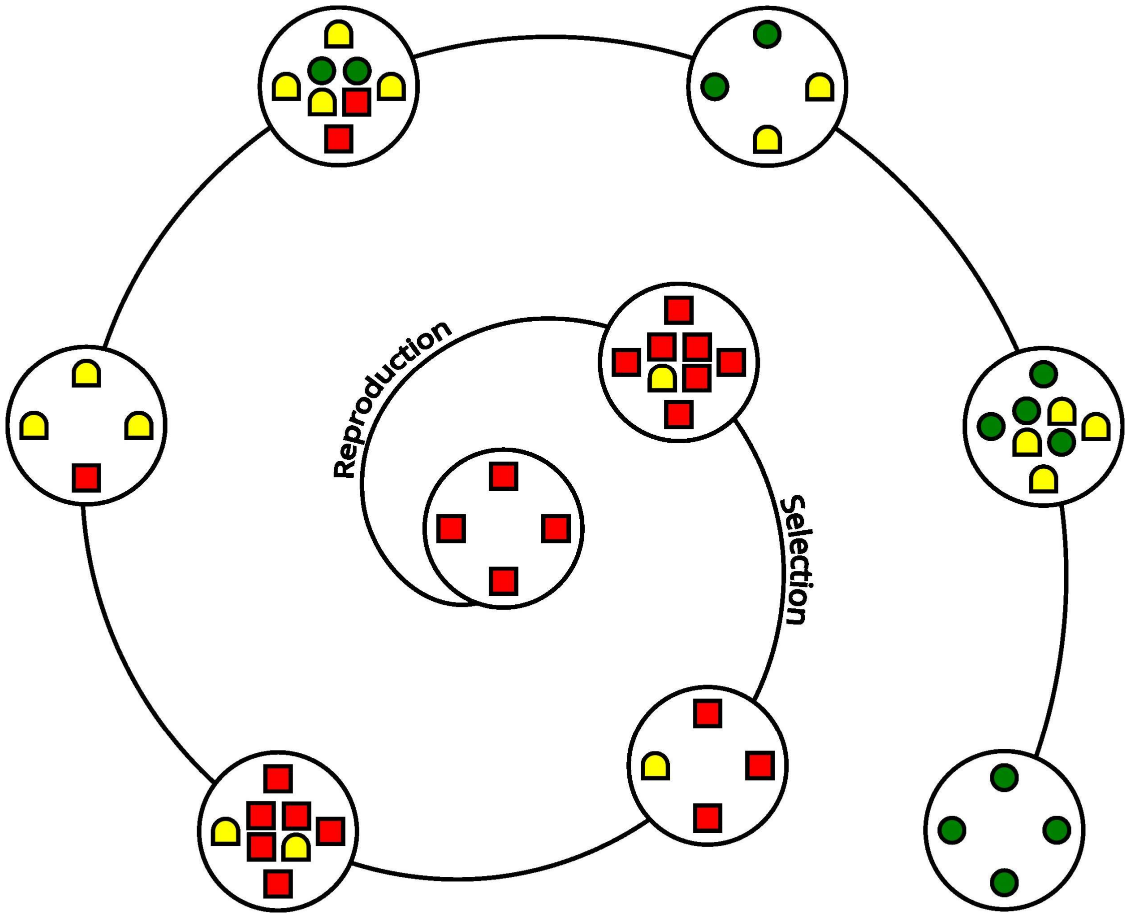


Figure 1.10. Evolutionary algorithm example. A population of states, in this example shapes, is iteratively evolved through alternating reproduction and selection events. The fitness function favors round shapes, and shapes are color coded according to their fitness. In the first generation a mutation gives rise to a yellow Norman window (i.e. "pac-man ghost"). Since its fitness is superior to the red square it survives into the next generation. Selective pressure amplifies the roundness shape until the population is made up of only green circles.

There are two key steps to an evolutionary algorithm: reproduction, which encompasses mutation and recombination, and selection. How individuals are reproduced is highly dependent on their chromosomal representation and sometimes the optimization domain. Some states can be encoded as linear data structures. Within the context of molecular design, nucleotides [102], proteins [14, 103, 104], synthetic polymers [105, 106] and even simple Markush structures [78] have all been encoded as arrays. The main appeals of using arrays as chromosomes are the ease of manipulation and the direct correlation with nucleotide sequences from which they draw inspiration, making the implementation of mutation and recombination straightforward. However, when it comes to diverse and arbitrary molecules like drugs this representation can be very restrictive. Arguably the most natural representation for molecules is their molecular graph. Accordingly, efforts have been undertaken to mutate and recombine molecular graphs.

Mutations stochastically modify molecules, adding external variability to the population. Molecular mutation techniques can be classified into atom- and fragment-based approaches based on the granularity of the molecular representation. It should be noted that this classification is purely didactical in nature, and in practice many methods blur the lines between atoms and fragments. Atom-based approaches modify the molecular graph one atom or one bond at a time, whereas fragment-based approaches construct molecules as a combination of multi-atomic fragments [63, 107]. Both have inherent advantages and disadvantages. Atom-based molecule construction tends to be simpler and enables the exploration of virtually the entirety of chemical space. Their biggest drawback is that when applied naively the resulting molecules may be difficult to synthesize [108, 109]. Nonetheless many applications of this methodology have been reported [12, 54, 60, 64, 79, 100, 110–112]. Fragment-based approaches were pitched as a solution to the poor synthesizability problem [60, 64, 73, 74, 78, 79, 98, 113, 114]. When fragments are manually curated or algorithmically extracted from desirable molecules they can capture, and therefore reproduce, recurring chemical features. Besides potentially improving the synthesizability of designed molecules it can also bias the search towards specific areas of chemical space of interest. Ultimately, as one biases molecular design towards known chemistry the designed molecules become more pleasing to the eye, but at the expense of ignoring large sections of chemical space, and therefore reduced chemical novelty.

Molecular recombination consists in exchanging chemical features between molecules, exploiting the internal variability of the population. For simplicity's sake this exchange usually takes place between pairs of molecules. Historically, recombination has been executed in two ways. The first and most popular procedure is the "digestion" approach [12, 64, 100, 115, 116], where certain bonds of each molecule are broken yielding fragments. Fragments are exchanged and reconnected to complete the recombination. The second procedure could be described as the "match and swap" approach [81, 117, 118]. One starts off by finding the maximum common substructure shared by the molecules. This common substructure acts as scaffold, and the remainder of the molecule is taken as substituents, which can then be exchanged between molecules and reconnected in the same positions. It can be argued that this form of recombination is more natural. However, it only works as intended when there is a sizable common substructure between molecules. Much like substructure matching it is also a computationally expensive approach, as finding the maximum common substructure is a hard problem.

Selection of which individuals to reproduce and which individuals to carry over into the next generation is more standardized than reproduction. A very simple approach is *truncation selection*, where the *N* fittest individuals are selected. This leads to a phenomenon known as *elitism*, where the best individuals are always preserved and reproduced. This favors exploitation and skews the population heavily towards a specific solution. In the process the genetic diversity of the population may decline, sometimes degenerating to the point where all individuals are identical. This can seriously hamper exploration. As an alternative one can opt for selection schemes such as *fitness-proportionate selection*, sometimes also called roulette wheel selection. As the name suggests, following this strategy the probability of selecting an individual is proportional to its fitness. This reduces the probability of premature convergence. Other selection schemes such as steady state selection and tournament selection exist, but won't be discussed here as they were not applied within this work.

It should be remarked that it is common to find related terms such as evolutionary strategy, genetic algorithm and evolutionary algorithm being used as synonyms. This would be considered a misnomer by some. The consensus seems to be that evolutionary algorithm is a broad term encompassing more specific variants such as genetic algorithms. However, in the literature there are some disagreements about the distinctive features of the different variants. In practice the terms are often used interchangeably, especially outside the world of optimization theory. Nonetheless, to err on the side of caution I will refer to them broadly as evolutionary algorithms.

1.8 Benchmarking molecular design algorithms

1.8.1 Objective values

Despite many molecule generators having been reported in the literature, it is not always clear how they compare to each other and where their strengths and weaknesses lie. For a long time it was common to test molecular design algorithms on arbitrary in-house problems. Skeptics might argue that the test cases had been contrived to showcase methodologies in a positive light. Recently efforts have been undertaken to standardize the test suits for molecule generators [108, 119, 120]. In principle this enables straightforward comparisons between algorithms without the need of re-running any benchmarks, although this may be partly wishful thinking.

One such test suite we use throughout this work is the goal-directed GuacaMol benchmark suite [108]. This benchmark suite measures how good an optimization algorithm is at designing molecules that maximize some pre-defined objectives. The GuacaMol benchmark suite is a collection of individual benchmarks. The most important component of each benchmark is an objective function that scores molecules in the [0, 1] range, with higher values being better. Molecule generator are tasked with designing a population of molecules maximizing the specified objective. This population is then evaluated to yield a final benchmark score, also in the [0, 1] range, typically as a weighted average of molecule scores. Optionally some benchmarks may also provide a starting population of molecules.

Different versions of the GuacaMol benchmark suite have been developed. Two are of interest to us. The "trivial" version includes 7 benchmarks and is comprised mostly of tasks to design molecules with specific physicochemical properties (e.g. logP). The "V2" version is more diverse. It consists of 20 benchmarks including tasks to find molecules that (1) are identical to a reference molecule, (2) are similar to reference molecule(s), (3) have specific chemical formulas, (4) have/have not certain substructures, and (5) combinations of the aforementioned tasks, the so called Multi-Parameter Optimization (MPO) tasks.

The GuacaMol benchmark scores are suitable to assess the "optimization power" (OP) of an algorithm, but they do not provide any information about the chemical quality of the designed molecules. To assess the latter we have used synthesizability metrics. Note however that a common element to many of the GuacaMol benchmarks is that they score molecules based on topological similarity to drugs. In other words, the perfect solution is oftentimes related or identical to a known drug. Since drugs are obviously synthesizable, the GuacaMol scoring functions unintentionally provide some implicit guidance to design synthesizable molecules. Throughout this work we will occasionally resort to custom benchmarks to better elucidate the effects attributable to the scoring function.

1.8.2 Synthetic accessibility

As has already been touched upon, molecular design is a multi-objective problem. One of the objectives that the designed molecules must fulfill is synthetic accessibility. *Synthetic accessibility*, *synthetic feasibility* or *synthesizability* refers to a molecule's ease of synthesis, and is a rather vague and subjective concept. It could be interpreted as a binary property, with a molecule either being synthesizable or not. However, not all synthesizable molecules are equally easy to synthesize. Some molecules may be theoretically synthesizable, but the effort required to synthesize them in large enough quantities may be beyond the willingness of the medicinal chemist. We will consider synthesizability as a continuous property, and we aim to design molecules that a chemist might agree to synthesize. This typically demands a short synthetic route (preferably less than 5 steps), starting from commercially available building blocks and employing well established reactions with high yields and simple reaction conditions [10], for example amide coupling. Easy to synthesize molecules are typically small, have little to no stereochemistry, and should be comprised of small and conventional ring systems the chemist is comfortable with [121].

Chemists rely on their intuition and expertise when assessing synthesizability. While the algorithm underlying a chemist's decision making is poorly understood and highly subjective [121–123], generally speaking chemists prefer familiar chemistry. Computers can exploit this preference by finding patterns in known examples of familiar chemistry and comparing them to the patterns of subject molecules whose synthesizability ought to be assessed.

An early attempt to do so was the SAScore [124]. The SAScore is a heuristic that estimates the synthesizability of a molecule on a scale from 1 to 10, with lower values suggesting an easier synthesis. The SAScore consists of two components that are summed together. The first component is the *fragment score*. It measures the topological similarity of a query molecule to reference synthesizable molecules through means of ECFP features [34]. If a molecule contains features that are prevalent among known synthesizable molecules, it is presumed to be syntesizable as well. The reverse is true for features that are rare or even absent in the reference library. The fragment score implicitly captures and expresses chemical preferences as well as reactant availability. The second component is a *complexity penalty*. As the name implies, molecules exhibiting properties hindering synthesis receive a penalty. These properties include size, the number of chiral centers and ring complexity descriptors such as the number of macrocycles, spirocycles or bridged ring systems. The appeal of the SAScore is that it is simple and fast to calculate. Despite its simplicity it correlates surprisingly well with a chemist's understanding of synthesizability [124, 125], and is predictive of more complex synthesizability assessment techniques [109, 126, 127].

The golden standard for synthesizability assessments is retrosynthesis. If a molecule is retrosynthesizable it is presumably also synthesizable. Retrosynthesis not only evaluates if a molecule is synthesizable; it also proposes a synthetic route for it. In retrosynthesis an input molecule is successively broken down by the inverse of synthetic reactions until sufficiently simple or commercially available building blocks are generated [128, 129]. Unfortunately retrosynthesis is a computationally expensive problem. *In silico* reactions must be preceded by the detection of functional groups involved in the reaction, which is an expensive procedure [31]. Making matters worse, retrosynthesis is typically implemented as a tree search. In combinatorial spaces like chemical space deep tree searches are only tractable when paired with intelligent policies [130–133]. Even then retrosynthesizing a single molecule may take several minutes (compared to the fractions of a second it takes to compute heuristics like the SAScore), forbidding its iterative application. Whether a molecule is retrosynthesizable or not can be rapidly predicted with classifiers [127, 134], but in doing so one sacrifices the proposed synthetic route and with it any validation of the prediction.

Chapter 2 Research aims

The purpose of this work is two-fold. From a practical perspective we aim to develop algorithms and software for molecular design, with a special focus on designing small organic molecules with potential therapeutic uses. During this process we hope to gain theoretical insights into the nature of chemical space and how to best explore it.

As a starting point for our method development we are equipped with the following:

1. The “similar structure, similar property” principle. We make the assumption that the likelihood of two molecules having similar properties is proportional to how structurally similar they are. Within this work molecule similarity will be expressed as a function of shared chemical features, and measured using topological fingerprint similarity coefficients.
2. A black-box objective function that explicitly and numerically expresses how well a molecule fulfills some arbitrary objective. What the objective function measures, and whether objective values ought to be minimized or maximized is task dependent. For simplicity the reader may assume that the objective function predicts the magnitude of a biological response under exposure to the molecule in question.
3. A virtual library of chemically desirable molecules. Molecules comprising the library ought to (1) be easy to synthesize and (2) have favorable ADMET properties (presumably for oral bioavailability). Ideally the library should also be large and diverse enough to be representative of the chemical state of the art. Throughout this work we will use ChEMBL as reference library.

Designed molecules ought to:

- Have extreme objective values, as measured by the aforementioned objective function.
- Be likely to be "chemically desirable". Leveraging the "similar structure, similar property" principle we can quantify chemical beauty by measuring the molecule's similarity to the aforementioned reference chemistry library.

The developed methods must be capable of designing molecules that meet the above criteria. Beyond this evident requirement we strive to develop methods that:

- Design molecules *de novo*, that is, the molecular starting point shouldn't be an existing chemical entity but rather vacuum. Accordingly, we expect the methods to yield novel molecules that haven't been described or enumerated previously.
- Competently explore complex fitness landscapes without requiring assumptions about the nature of the fitness function.
- Explore said landscape in a computationally efficient manner, finding good solutions expending as few computational resources as possible. Opportunities for computational optimization include (1) the routines to construct / modify molecules with desirable properties, and (2) minimizing the number of molecules to be evaluated by the objective function. The latter will push us to explore heuristic optimization algorithms.
- Are open-source and modular, so the scientific community can easily integrate them in their workflows.

Chapter 3 Molecular constraints as a means to improve molecule quality

3.1 Source

This chapter is based on the publication:

Kerstjens, A., De Winter, H. **A molecule perturbation software library and its application to study the effects of molecular design constraints**. *J Cheminform* 15, 89 (2023). https://doi.org/10.1186/s13321-023-00761-5

3.2 Problem statement

Designing molecules that optimize many objectives simultaneously can be challenging. Some authors try to evade the challenges of multi-objective optimization by considering explicitly only the primary objective and capturing the secondary objectives implicitly by constraining the molecular generation process to imitate known and desirable chemistry [73–76, 80, 82–84, 91, 92, 135]. This effectively blocks access to certain areas of chemical space (Figure 3.1). A large corpus of enumerated molecules with desirable secondary objectives exists [2, 136, 137], and it's reasoned that constraining the molecular design process to only generate compounds similar to those in the corpus will yield molecules with desirable properties.

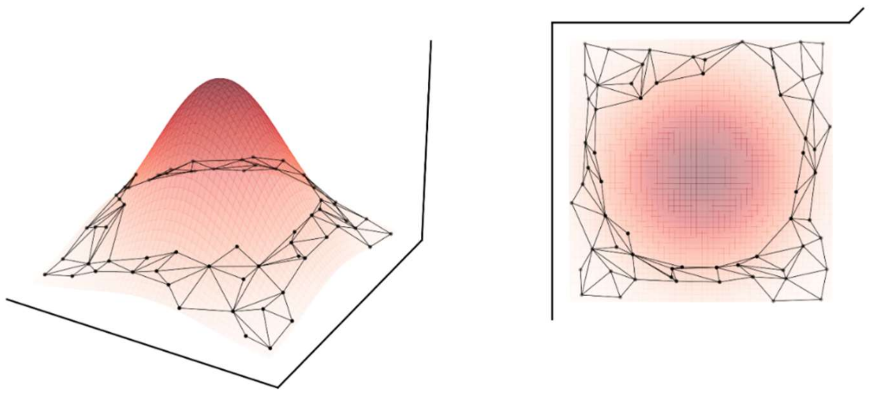


Figure 3.1. A section of graph-like chemical space with an excluded area (center). The exclusion stems from molecular construction constraints and corresponds to a maximum on an undesirability objective landscape (red).

Many accounts describe the effectiveness of this approach to improve the drug-likeness and synthetic accessibility of generated molecules [73–76, 80, 82–84, 91, 92, 135], but it is not without drawbacks. The constraints imposed on molecule construction manifest themselves as barriers in search space, restricting the optimization algorithm’s freedom [75, 138]. These barriers may prevent accessing undesirable molecules, but inadvertently they may also hinder or impede discovering potentially appealing molecules, especially those that are most novel and resemble known chemistry the least.

Consider some molecular generation scheme that can modify a reference molecule to yield related molecular entities. In this case chemical space can be visualized as a transition graph (previously termed a “morph graph” [54]), where vertices symbolize accessible molecules, and edges symbolize transitions between them (Figure 3.2). The topology of this graph is dependent on the constraints of the chosen molecule generator. Generally speaking, atom-based approaches define a more populous graph than fragment-based approaches since a larger number of chemical states is accessible. The density of the graph (that is, the ratio between the existing number of edges and the theoretical maximum number of edges) depends on the strictness of the perturbation rules. Approaches with strict rules will define a sparse graph, while approaches with lax rules will define a dense graph.

Suppose that a chemical space search starts at a known molecule A. The goal is to find some unknown molecule B that exhibits good objective values. The more populous the transition graph, the more probable it will be that desirable molecules are part of it and therefore discoverable. The perfect optimization algorithm would define the shortest path between A and B. Such an ideal algorithm would benefit from a very populous and dense transition graph, as in these graphs paths between pairs of vertices tend to be shorter (Figure 3.2).

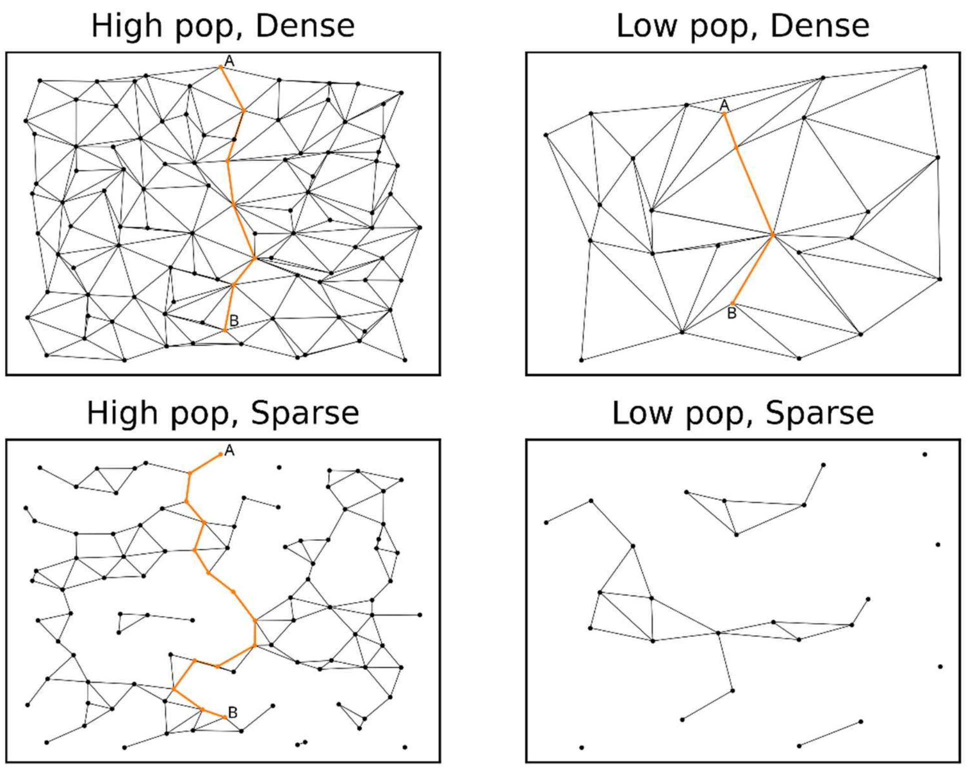


Figure 3.2. Examples of transition graphs of different population and density. The shortest path between two vertices A and B is highlighted in orange. Note that the path is shorter if the graph's population is lower or the density is higher. As the population and density decrease the probability of two vertices being connected decreases.

Sadly, we do not have access to these utopian search algorithms. In absence of omniscient oracles that reveal B and the path towards it, our algorithms must err on the side of exploration. Thorough exploration of very populous and dense graphs is computationally intractable. Trimming the size and density of the search graph in a chemically meaningful way could provide guidance to algorithms that otherwise would wander around unpromising regions of chemical space without a clear sense of direction.

In summary, when it comes to predicting the effect of molecular construction constraints on the fitness of the designed molecules, we are faced with two opposing hypotheses. The constraints may either hinder or facilitate chemical space exploration, and what the outcome will actually be is poorly understood.

Pieces of the answer lay scattered throughout the literature. Unfortunately, every study performs different experiments using different software, making it impossible to isolate the effect of any one variable. Attempts have been made to standardize experiments with benchmark suites [108, 119, 120], yet software is rarely standardized. Fully standardizing software is an impossible and arguably undesirable task as scientific methodologies are ever evolving. However, when it comes to graph-based molecule edition many commonalities can be found between different implementations.

We set out to create a software library for graph-based molecular edition providing the common denominator of functionality of previous implementations [12, 54, 64, 112]. We have named this library **Molpert**. Key considerations during the design were flexibility, extendibility, interoperability and ease of use. Molecule perturbations are atom-based, as fragment-based edition can be described in function of the former but not vice versa. Molecules are treated as graphs and modified without any sort of chemical considerations. This is by design as we did not want to impose our own biases and ways on others. Instead, users can specify themselves the properties the designed molecules ought to fulfill through means of arbitrary constraints. Mechanisms are foreseen to extend the functionality of the library should the provided functionality not suffice. **Molpert** is built on top of the RDKit, a highly popular and open-source cheminformatics toolkit [47], and integrates well with RDKit molecules. It has no other dependencies. A C++ and Python Application Programming Interface (API) are both provided and made available on GitHub (https://github.com/AlanKerstjens/Molpert).

In this chapter we describe **Molpert** and showcase how it can be applied to cheminformatics research. Specifically, we use it to build an evolutionary algorithm for molecule design and try to answer the question: "What are the consequences of constraining atom-based molecular construction?".

3.3 Methodology

3.3.1 Atom/bond property perturbations

Perturbations included in the library can be broadly classified into those changing the molecular graph's annotations and those changing the graph's topology. The former are trivial to understand and implement: each vertex (i.e. atom) and edge (i.e. bond) have a set of mutable numeric properties that are independent from the rest of the graph and can be freely changed. For atoms these properties are (1) the atomic number, (2) formal charge and (3) number of explicit hydrogens. For bonds the only property of interest is the bond type, which in most instances is equivalent to the bond order. Each property has a list of allowed values and associated sampling weights, both being user specified. By default the sampling weights are proportional to the property values' frequencies in ChEMBL31 [136]. All properties have a corresponding perturbation to modify it.

Modifying the number of explicit hydrogens may seem superfluous as hydrogens are often treated implicitly. However, explicit hydrogens can be of importance to adjust the perception of implicit hydrogens. They are also one of the invariants used in topological fingerprint calculation [34, 45]. Hence, being able to modify the number of explicit hydrogens is essential for good interplay with fingerprint-based scoring functions.

3.3.2 Topological perturbations

Topological perturbations refer to insertions and deletions of atoms and bonds. These operations could be performed by simply creating or destroying a single atom or bond. However, the resulting transformations may not match a chemist's expectations about what these perturbations should entail.

Consider a molecular graph $\mathcal{G}(\mathcal{V},\mathcal{E})$ with vertices or atoms $\mathcal{V}$ and edges or bonds $\mathcal{E}$. Naive implementation of topological perturbations may result, among other things, in a disconnected graph, that is, a graph in which there is a pair of atoms *v* and *w* between which no path exists. This is commonly undesirable unless the disconnected fragments represent salts.

To ensure that the graph remains connected an atom insertion requires bond insertions as well. Hence, inserting a new atom *a* involves (1) selecting the atomic properties of *a*, (2) selecting a set of *k* existing atoms $\mathcal{N}$ to which *a* will bond with *k* new bonds $\mathcal{B}$ ($\mathcal{N} \subset \mathcal{V}$, $|\mathcal{N}| = k$, $|\mathcal{B}| = k$) and (3) selecting the bond types of $\mathcal{B}$. Possible values for *a* and $\mathcal{B}$'s properties are sampled from a list of allowed values. *k* ranges between 1 and a user specified parameter defaulting to 3 to avoid a combinatorial explosion in possible outcomes. Up to $k - 1$ cycles may be formed during this process. Cycle formation may be unwanted. For example, given an alkane one might want to extend the length of the chain without creating a cycle. In other words, one might want to insert an atom between other atoms. To do so we select as $\mathcal{N}$ a central atom *c* and some atoms $\mathcal{J}$ adjacent to *c* ($\mathcal{J} = \{j \mid c \sim j\}$, $\mathcal{N} = c \cup \mathcal{J}$), and define a "dropped" atom $p \in \mathcal{N}$. During insertion *a* and $\mathcal{B}$ are added and existing bonds between *p* and $\mathcal{N} - p$ are deleted. The destruction of some existing bonds allows the insertion of atoms in acyclic regions without the creation of cycles (Figure 3.3). This only holds true if $\mathcal{N}$ is selected as described above such that all members of $\mathcal{N}$ are adjacent to *p* ($\mathcal{N} = \{n \mid n \sim p\}$). If $\mathcal{N}$ comprises arbitrary atoms and two atoms $\{v,w\} \subset \mathcal{N}$ are separated by a topological distance $d(v,w) \geq 2$ a cycle necessarily forms. Nonetheless, specifying a dropped atom can help in the design of more relaxed topologies that are not as densely packed with cycles.

Bond insertion is simple, as it only involves the selection of two atoms *v* and *w* where the topological distance between them $d(v,w) > 1$, the selection of a bond type and the creation of the bond. Once again, this necessarily creates a cycle of $d(v,w) + 1$ atoms (Figure 3.3). A minimum and maximum $d(v,w)$ may be specified. This provides the user with some control over the size of the resulting cycles but more importantly limits the number of possible outcomes.

Bonds are defined by a pair of atoms. Consequently, deleting one such atom a destroys the bond. Consider a set of atoms $\mathcal{N}$ adjacent to a ($\mathcal{N} = \{n \mid a \sim n\}$). The degree g of a is defined as $g(a) = |\mathcal{N}|$. If $g(a) \leq 1$ it is peripheral, and if $g(a) > 1$ it is internal. Peripheral atoms and internal atoms that are members of a cycle can always be deleted without disconnecting the graph. Internal atoms that are not part of a cycle separate two parts of the graph through a unique path. Hence, their deletion would result in a graph disconnection. To prevent this the atom deletion may be followed by some bond formations. We define a "reconnection" atom $r \in \mathcal{V}$, and create new bonds between r and $\mathcal{N}$ - ($\mathcal{N} \cap r$). This ensures that a path passing through r exists between all pairs of atoms of $\mathcal{N}$ after the deletion of a and that the graph remains connected. Typically $r \in \mathcal{N}$. Intuitively, this corresponds to deleting a and one of its neighbors $n_i \in \mathcal{N}$ taking its place by bonding to the remainder of the neighbors $n_j \in \mathcal{N} - n_i$ (Figure 3.3). However, if the user allows it, one could also sample an arbitrary r within a given distance $d(a,r)$ of a. This will result in the formation of a cycle of size $d(a,r)$ when $d(a,r) \geq 3$.

Similarly to atom deletions, bond deletions result in graph disconnections if the bond is not a member of a cycle. To delete an acyclic bond without disconnecting the graph a new replacement bond vw must be formed. Similar operations have been previously described as "rerouting" the bond [54]. The newly bonded atoms v and w ought to be on opposite sides of the "chokepoint" defined by the deleted bond (Figure 3.3). They must also be separated by a distance $d(v,w) \geq 2$, as otherwise the same bond would be recreated. The user can specify a maximum distance $d(v,w)$ to alter the topology less drastically.

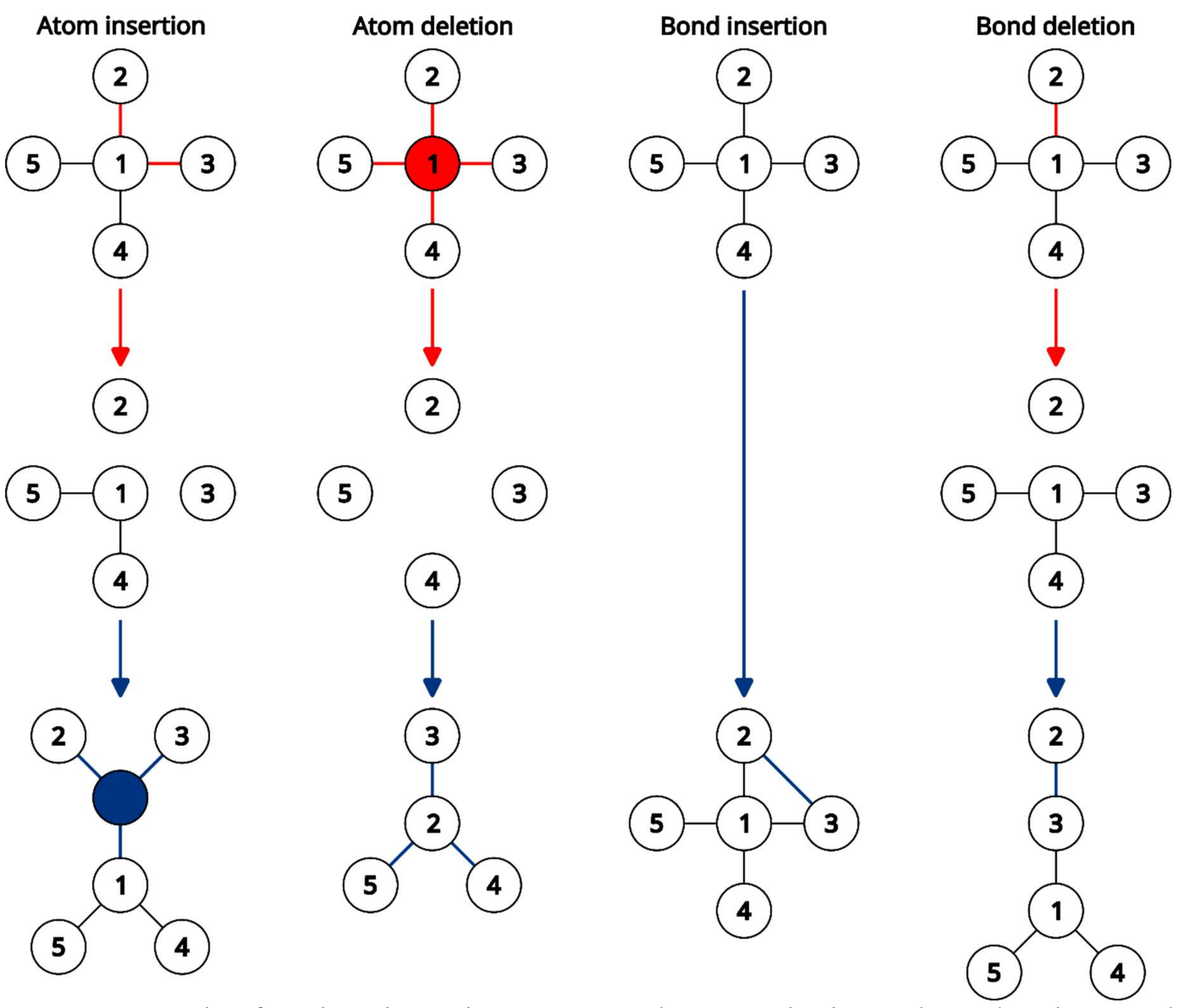


Figure 3.3. Examples of topological perturbations. Input and output molecules are depicted on the top and bottom, respectively. Deleted atoms and bonds are highlighted in red while inserted atoms and bonds are highlighted in blue. In the atom insertion example $\mathcal{N}$ = {1, 2, 3} and p = {1}. In the atom deletion example $\mathcal{N}$ = {2, 3, 4, 5} and r = {2}.

The described perturbations are sufficient to access the entirety of chemical space when executed in the right order. When sampled randomly specific long sequences of perturbations are statistically unlikely. It may be of interest to execute some of these sequences of perturbations as a unit. For example, one might want to insert a fragment corresponding to a specific functional group. It's possible to combine the above elemental perturbations to create more complex operations.

3.3.3 Molecule sanitization

Perturbations treat molecular graphs more like mathematical objects than chemical structures. Careless edition of the molecular graph is bound to result in chemically invalid structures. Notorious pain points include explicit hydrogen counts, stereochemistry and aromaticity. Cheminformatics toolkits like the RDKit [47] store atom and bond properties as integers within atoms and bonds themselves. These properties may be sensible when first calculated, but can lose their meaning after modifying the molecular graph. We employ a post-perturbation sanitization procedure that either alters these properties to sensible values or deletes them altogether.

A heavy atom's hydrogen count is modified to the value resulting in the lowest valid valence for said atom. The list of valid valences per element is provided by the RDKit. When no hydrogen count would result in a valid valence the count is set to zero. Chiral center stereochemistry labels are kept where possible. If a former chiral center is no longer chiral after a perturbation its stereochemistry label is erased. Newly formed chiral centers are not assigned any stereochemistry labels. Bond stereochemistry labels are always erased. Aromaticity presents the most egregious problem. Bonds may have been flagged as aromatic once upon a time, yet these flags are kept indefinitely even after modifying the molecule. The naive solution would be to convert aromatic bonds to single bonds once aromaticity has been broken. In the context of editing molecules, aromatic systems are fragile as most topological perturbations will cause aromaticity to be invalidated. On the other hand, creating an aromatic ring system is much more challenging, as it requires atoms and bonds of the right types to be placed in the right positions simultaneously. When modifying molecules stochastically the sequence of events leading to the creation of an aromatic ring system is highly unlikely. In practice this means that most designed ring systems won't be aromatic, which is uncharacteristic of organic molecules.

Molpert handles aromaticity in two different ways, depending on the user's preference. The simplest option is to work with kekulized molecules only, that is, molecules where aromatic systems are represented by alternating single and double bonds. Alternatively, one can work with "partially aromatic" molecules where the aromaticity flags are preserved, irrespective of whether they are valid at present time or not. For example, acyclic regions may be transiently labelled as aromatic. The former aromatic character of bonds is remembered and used to reestablish aromaticity in the future whenever conditions are right. When a molecule with valid aromaticity assignments is required, a sanitization procedure can be applied. Acyclic regions labelled as aromatic are kekulized. Rings are defined as components of the Smallest Set of Smallest Rings (SSSR) [23]. Rings that are correctly flagged as fully aromatic are left untouched. Kekulized rings fulfilling aromaticity criteria are aromatized. Partially aromatic rings are sorted in descending order according to their number of aromatic bonds and sanitized. If the number of aromatic bonds in the ring is greater than half and the ring otherwise meets the requirements to be aromatic it's aromatized. Otherwise it is kekulized. Starting the sanitization process with the most aromatic rings allows aromaticity to propagate throughout fused ring systems (Figure 3.4).

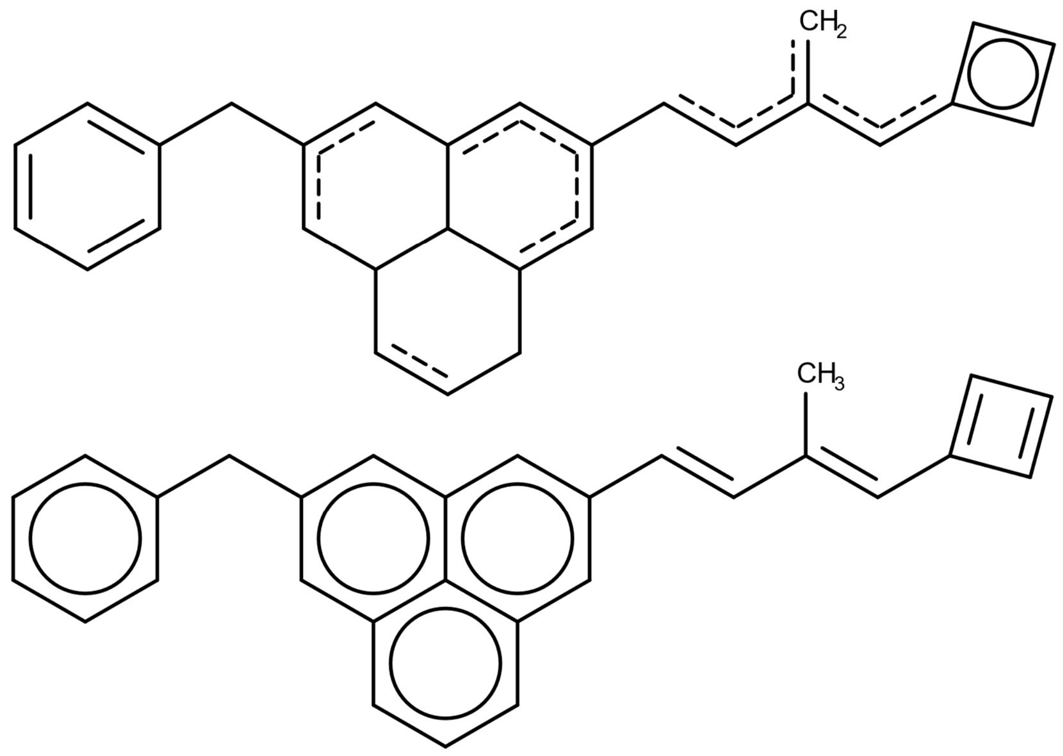


Figure 3.4. Aromaticity sanitization example. Aromatic bonds are depicted as dashed bonds. Aromatic ring systems where all bonds are aromatic are depicted with internal circles. Partially aromatic ring systems are either aromatized or kekulized depending on their "degree of aromaticity". Bonds incorrectly labelled as aromatic are kekulized.

3.3.4 Modes of operation

Perturbations are implemented as objects specifying how a molecule will be modified. These objects are callable and can be invoked when the perturbation ought to be executed. The user may construct these objects directly for fine-grained control over the outcome of a perturbation. For convenience we also provide factory functions that abstract away the details of constructing perturbations. Said factories can systematically enumerate all possible perturbations that could be applied to a molecule. Enumeration may be restricted to specific types of perturbations and/or atom/bond targets. When used deterministically all generated perturbations fulfilling the constraints are stored in a queue. When used stochastically the iteration order is randomized and the first generated perturbation fulfilling the constraints is returned. The randomization relies on a weighted shuffle in such a way that perturbations featuring common property values are most likely to be tried first [139]. This reduces the number of perturbations that need to be tried before one fulfilling the constraints is found.

3.3.5 Molecule constraints

While we designed the software to be able to generate any molecular graph, there may be instances where one wishes to use it to generate molecular graphs fulfilling specific criteria. This is enabled through constraints. In this context constraints are callback functions evaluating whether a molecule fulfills some arbitrary requirements. They take as input an atom, bond or molecule and return as output a boolean. A return value of “true” signals that the requirements are satisfied, whereas “false” signals the requirements are not met. Constraints may apply to one specific atom or bond. It’s therefore possible to constrain only certain parts of the molecule and to mix constraints as desired.

Constraints are enforced through trial and error. A queue of compatible perturbations is prepared. The perturbation at the front of the queue is applied to a copy of the molecule to simulate its outcome. The perturbed molecule is then forwarded to the constraints for evaluation. If any constraint evaluates to “false” the perturbed molecule is discarded and the next perturbation is simulated. This process repeats until a perturbation satisfying all constraints is found or the queue is empty. The stricter the constraints the higher the perturbation attrition rate and with it the computational performance degradation (Figure 3.5).

For our experiments we explored different variants of atom and bond constraints. The most basic constraints are **valence constraints**. Each element is assigned a maximum allowed valence, and any atoms of said element with a lower valence are accepted, under the assumption that hydrogens can be added to pad the valence up to the closest valid value. The remainder of the constraints are **key-based constraints**. Atoms and bonds are characterized with atom and bond keys, respectively (Table 3.1, Figure 3.6). An **atom key** is a tuple of integer properties characterizing the atom. Depending on the properties used to define the key we distinguish between **local atom keys** made up of common atomic invariants [23, 24] (degree *D*, valence *V*, atomic number *Z*, formal charge *Q* and number of explicit hydrogens *H*) and **ring-aware (RA) atom keys**, which on top of the aforementioned atomic invariants include the number of SSSR rings the atom is involved (*R*) and the sizes of the smallest and largest SSSR rings it is involved in (*NR* and *XR*, respectively). **Bond keys** are defined through combination of the bonded atoms’ keys and the bond’s type (*B*). Lastly, we also define **atomic environment keys** as the hashes of circular atomic environments, akin to the Extended Connectivity Fingerprint (ECFP) algorithm [34]. Environments of topological radii 1 (r = 1) and 2 (r = 2) were studied.

We determined the set of keys found in drug-like molecules, specifically ChEMBL31 [136], and recorded them in dictionaries, one for each key type. Given one such dictionary and a query molecule, a key based constraint calculates the same type of keys for the query molecule and compares said keys to the dictionary’s keys. If the query molecule exhibits keys that are not part of the dictionary the constraint evaluates to false and the query molecule is rejected.

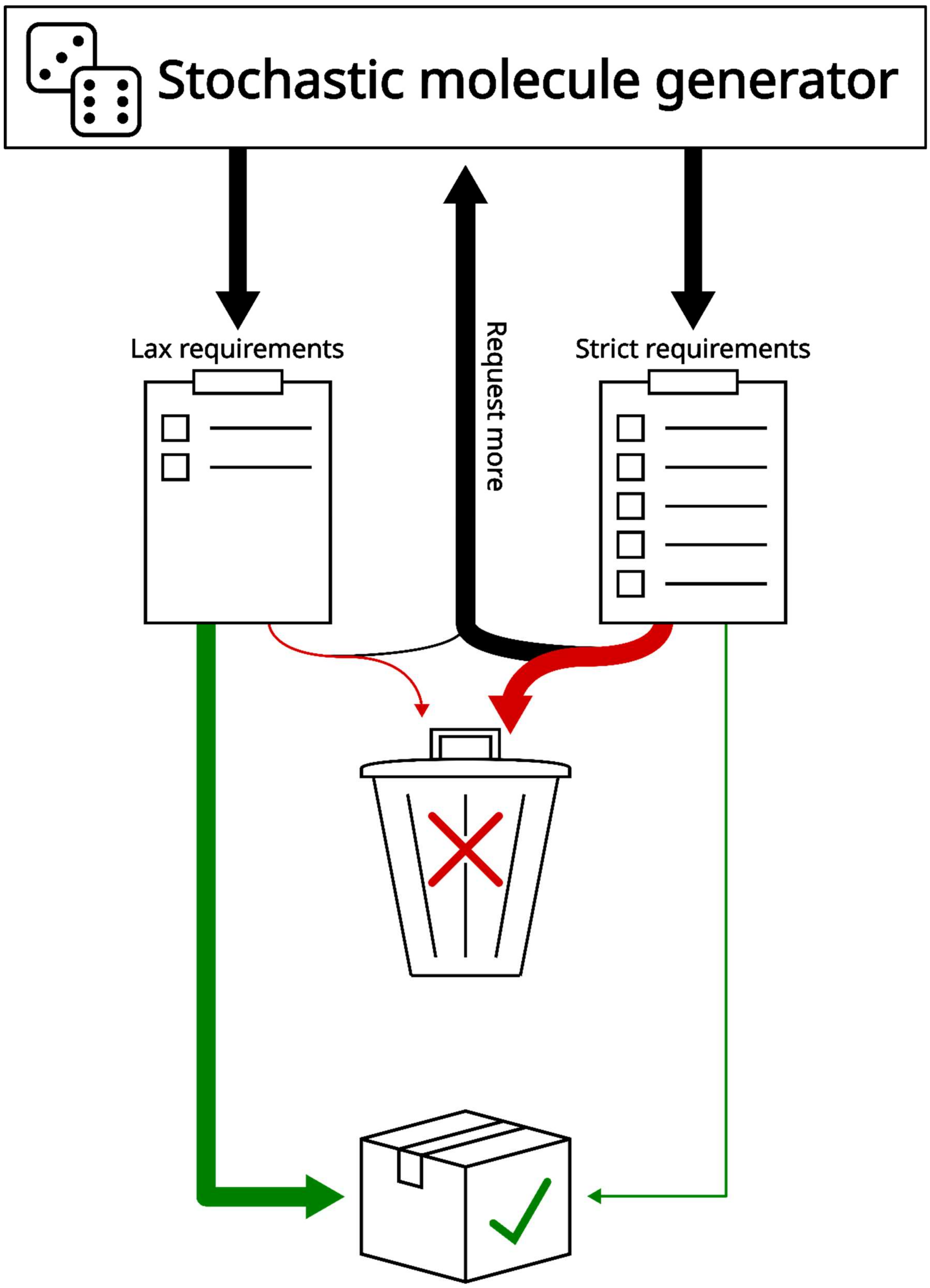
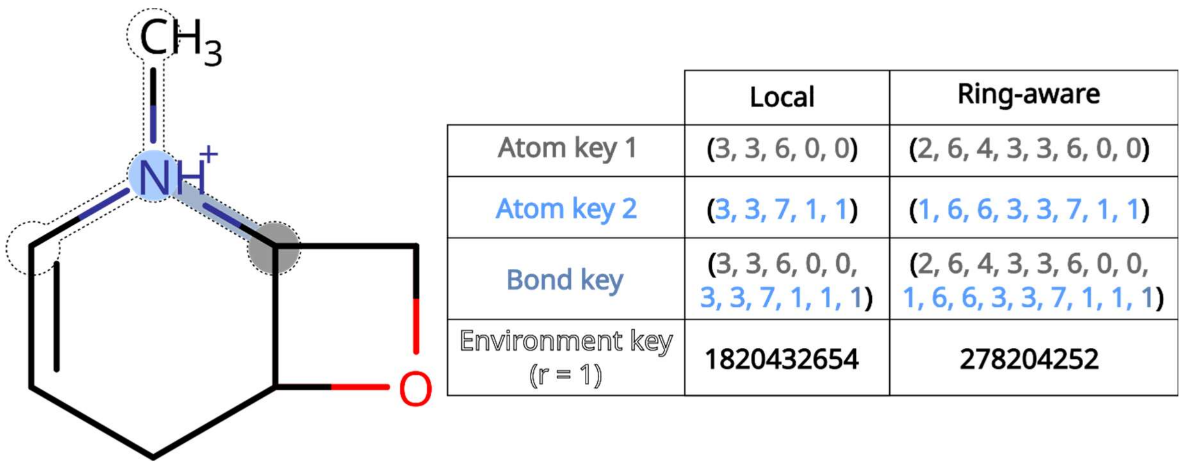


Figure 3.5. Illustration of different attrition rates for lax and strict constraints. When using strict constraints few molecules fulfill the desired criteria, resulting in the molecules being discarded and new molecules being generated.

Table 3.1. Overview of the molecular keys used to characterize molecules and constrain molecular perturbation.

Molecular key	**Key structure**
Local atom key	(D, V, Z, Q, H)
Ring-aware atom key	(R, XR, NR, D, V, Z, Q, H)
Local bond key (LB)	(D_1, V_1, Z_1, Q_1, H_1, D_2, V_2, Z_2, Q_2, H_2, B)
Ring-aware bond key (RAB)	(R_1, XR_1, NR_1, D_1, V_1, Z_1, Q_1, H_1, R_2, XR_2, NR_2, D_2, V_2, Z_2, Q_2, H_2, B)
Local environment key	hash({LB_1, LB_2 … LB_n})
Ring-aware environment key	hash({RAB_1, RAB_2 … RAB_n})



	Local	Ring-aware
Atom key 1	(3, 3, 6, 0, 0)	(2, 6, 4, 3, 3, 6, 0, 0)
Atom key 2	(3, 3, 7, 1, 1)	(1, 6, 6, 3, 3, 7, 1, 1)
Bond key	(3, 3, 6, 0, 0, 3, 3, 7, 1, 1, 1)	(2, 6, 4, 3, 3, 6, 0, 0, 1, 6, 6, 3, 3, 7, 1, 1, 1)
Environment key (r = 1)	1820432654	278204252

Figure 3.6. Example molecular keys. The color highlighted atoms are characterized with atom keys, and the color highlighted bond between them characterized with a bond key. The nitrogen's circular atomic environment of radius 1 is shown as a dotted outline and characterized with the hash of its bonds' keys, resulting in seemingly random numbers. For the meaning of each integer see Table 3.1.

3.3.6 Property and perturbation sampling

Some molecular perturbations, namely property perturbations, atom insertions and bond insertions must sample atom and/or bond properties. The properties being sampled are atomic numbers, formal charges, explicit hydrogen counts and bond types. Property values are sampled from pre-defined sets of allowed values. When perturbations are enumerated deterministically each allowed value is used to construct one perturbation. When perturbations are generated stochastically a single allowed value is randomly sampled to construct a single perturbation, with each value having an associated sampling probability.

While the user may provide their own sampling values and probabilities, we provide some reasonable defaults. For each property we recorded the frequency of occurring values in ChEMBL31 [136], as well as the mode (i.e. the most frequent value). Property values occurring with a frequency larger than 0.01% are considered allowed and may be sampled with probabilities proportional to the values' frequencies. The mode is taken as a default property value and, at the discretion of the user, may replace the list of allowed values to reduce the number of perturbations resulting from deterministic enumeration.

Property values for a specific atom or bond are sampled independently from the rest of the atom's or bond's properties and independently from their surrounding chemical environment. As an exception one may opt to sample atomic numbers and bond types with different probabilities depending on whether the atom/bond is part of a ring or not. The main motivation behind this exception is to preferentially place aromatic and double bonds in rings, and triple bonds in acyclic structures.

When generating perturbations stochastically the user may or may not specify the type of the perturbation. Should they choose to not do so the library will randomly sample a perturbation type for them. Property perturbations have sampling probabilities that are proportional to how often the property deviates from the mode. For example, since it is rare to encounter charged atoms the probability of sampling a "formal charge change" perturbation is low. Conversely, since it is relatively common to encounter non-carbon atoms the probability of sampling an "atomic number change" perturbation is comparatively high.

Weighted property sampling is supposed to reduce the probability of stochastically generating a constraint-infringing perturbation. To verify this assumption we took a subset of 10,000 ChEMBL molecules of varying sizes and generated 10 perturbations of each type for each molecule. We repeated the process twice sampling property values from either a uniform distribution or from the aforementioned ChEMBL-derived distribution. We then measured the perturbation rejection rate according to different molecular constraints.

3.3.7 Chemical space connectivity

Stricter constraints are associated with sparser chemical spaces (Figure 3.2). One can quantify the stringency of a set of constraints by calculating the average degree of the corresponding chemical transition graph. We stratified ChEMBL31 [136] according to the molecules' heavy atom counts (HAC), and sampled 1000 random molecules every 5 HAC between 10 and 50 HAC. Two molecules are considered to be neighbors in chemical space if they are separated by a single edge in the chemical transition graph, that is, a single perturbation. **Molpert** was used to deterministically enumerate all perturbations applicable to each molecule of the aforementioned ChEMBL subsets. Said perturbations were subsequently executed to enumerate the molecule's neighbors. Different perturbations may result in the same neighbor, but only unique neighbors were kept. The process was repeated using different sets of constraints, and the number of perturbations resulting in constraint-infringing neighbors was recorded. The average number of neighboring molecules, equal to the average degree of the transition graph, was taken as a measure of the constraints' stringency.

3.3.8 Benchmark

3.3.8.1 *Effect of constraints on optimization power*

To evaluate the effects constraints have on molecule fitness we developed an evolutionary algorithm using **Molpert** to mutate and recombine molecules. We initialized a population of 100 "empty" molecules with no atoms or bonds. Said population was evolved over the course of at most 10,000 generations. Evolution may be halted earlier if some termination criterion is met, such as no improvement being observed in the best individual's score for 1000 generations, or a molecule with a sufficiently good score being found. Every generation copies of the parent molecules are made and recombined or mutated to yield child molecules. The child molecules must (1) fulfill any constraints that may have been imposed by the user and (2) be topologically dissimilar from all other molecules in the population. The topological similarity between two molecules is calculated as the Tanimoto coefficient [35] between their ECFP4 fingerprints [34]. If the similarity of a child molecule to any of the current members of the population surpasses 0.9, the child is discarded. The internal similarity filter prevents the degeneration of the population to a single solution, and serves as the algorithm's main premature convergence guard. At the end of the generation the parent and child molecules are ranked according to their scores, as determined by some scoring function, and the best 100 are retained for the next generation. A flowchart of the algorithm can be seen in Figure 3.7.

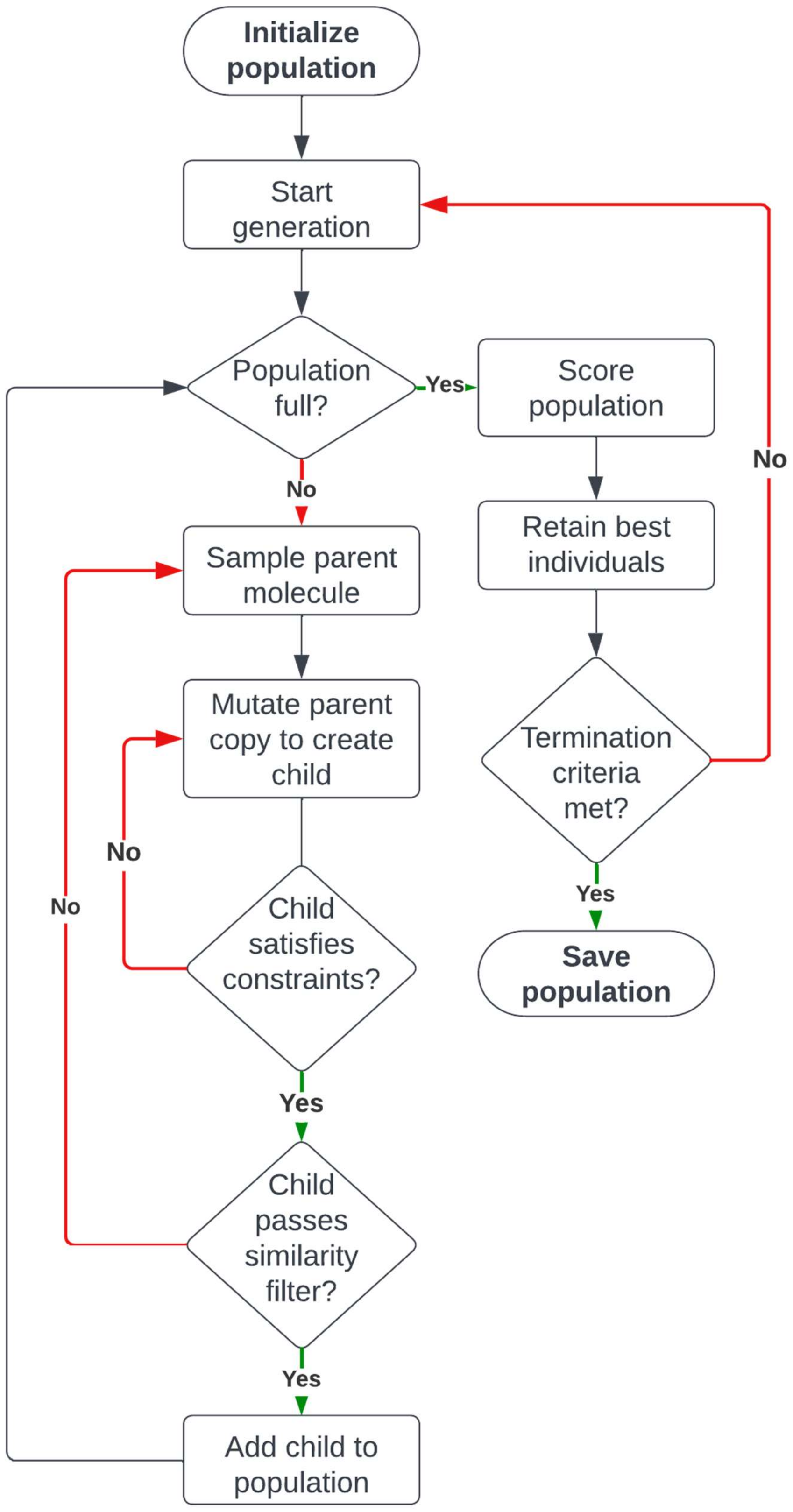


*Figure 3.7. Flowchart of the **Molpert**-based evolutionary algorithm used for benchmarking purposes.*

Recombination is not a core part of **Molpert**, but simple digestion-based recombination operators [12, 64, 100, 115, 116] are made available as addons. Subgraphs of approximately the same size are induced in two molecules, by randomly walking through their graphs. The size of the subgraph is chosen as a random integer in the range $[0.1 \cdot HAC, 0.5 \cdot HAC]$, where HAC is the molecule's Heavy Atom Count (HAC). Bonds flanking said subgraphs are broken and converted to attachment points resulting in fragments. If one fragment has less attachment points than the other, random hydrogens are replaced with attachment points within the former until the numbers are equalized. Fragments are exchanged between molecules, and attachment points are reconnected randomly. We forbid fragmentation to break ring systems, but this behaviour may be disabled upon request. An example of how subgraph exchange can be used to crossover molecular graphs is found in Figure 3.8.

The algorithm was tasked to design molecules maximizing the scores of GuacaMol goal-directed benchmark (v2) scoring functions [108]. For details on the GuacaMol benchmark suite please refer to section 1.8.1. Some benchmarks demand the generation of a population of molecules, in which case the total benchmark score is calculated as a population weighted average. We opted out of this last step and took as score the fitness of the top molecule only. In our algorithm population diversity is enforced through means of a topological similarity threshold. Due to this filter the remainder of the population is by design subpar and present solely to facilitate the evolution of the top molecule. Since evolutionary algorithms are stochastic one will presumably want to run multiple independent replicas anyways, sourcing the top molecule of each run. We ran the benchmark 50 times for each type of constraint recording the top molecule of each replica. Jobs were given a maximum of 72h core hours. Some jobs for strict constraints failed to complete within this time, reducing the sample size (Table 3.2). Differences in molecule fitness between the "no constraints" control group and constraints groups were analyzed using the non-parametric Kruskal–Wallis H-test [140] followed by pairwise Mann-Whitney U-tests [141] with Šidák correction [142]. $\alpha = 0.05$ was taken as significance level and family-wise error rate. Statistical tests and post hoc corrections were performed using the SciPy [143] and statsmodels [144] Python packages, respectively.

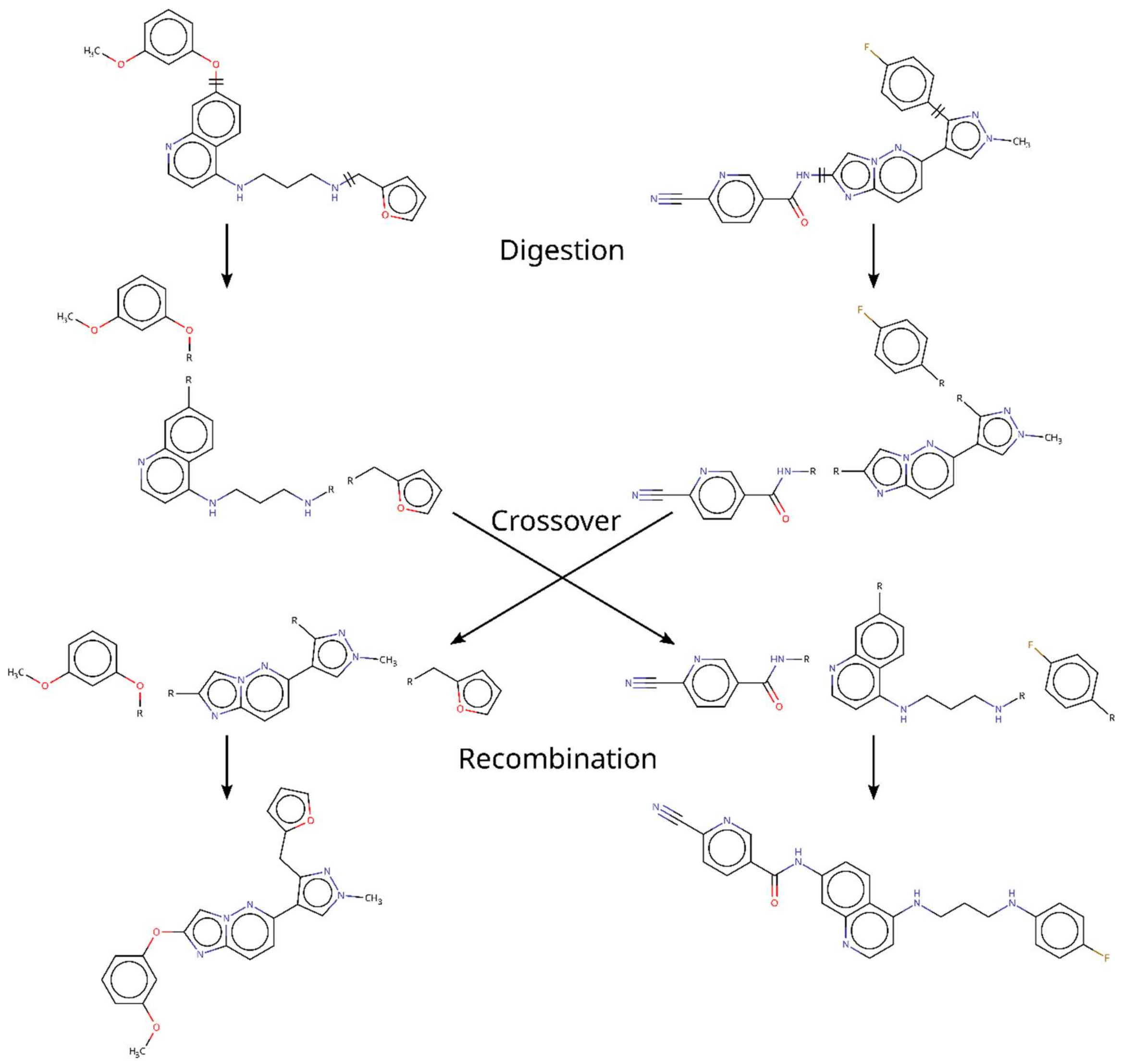


Figure 3.8. Example molecular graph crossover. Broken bonds are crossed. Attachment points resulting from digestion are shown as R-groups.

Table 3.2. Number of successfully completed GuacaMol benchmark replicas. We submitted 50 replicas for each of the 20 benchmarks within the benchmark suite. Missing replicas are due to them exceeding the allocated computational time limit.

Constraint type	Number of completed replicas
None	1000
Local atom	1000
Valence	1000
Local bond	1000
RA atom	1000
Local environment (r = 1)	1000
RA bond	1000
RA environment (r = 1)	999
Local environment (r = 2)	959
RA environment (r = 2)	942

3.3.8.2 *Effect of constraints on chemical appeal*

To evaluate the chemical appeal and novelty of molecules we designed 1000 random molecules using each set of constraints. Said molecules, hereon forward referred to as Randomly Designed Molecules (RDM), were constructed through successive atom and bond insertions, aiming to create a molecule of 29 heavy atoms and 32 bonds, which corresponds to the average number of heavy atoms and bonds of molecules in ChEMBL31 [136]. Synthesizability and drug-likeness were assessed through means of the SAScore [124] and Quantitative Estimation of Drug-likeness (QED) [145], respectively. ChEMBL31 was used as reference synthesizable chemistry for SAScore calculations. Differences between distributions were analyzed with one-way Analysis Of Variance (ANOVA) [146] followed by Dunnett’s test [147]. $\alpha = 0.05$ was taken as significance level and family-wise error rate. Chemical novelty was evaluated qualitatively by embedding the molecules into a 2D continuous chemical space and studying their location. Said chemical space was defined by characterizing molecules as binary 2048-bit ECFP4 fingerprints [34] and reducing their dimensionality with Principal Component Analysis (PCA) [148].

Optimizing molecules according to some objective function by design biases the regions of chemical space that are sampled. This is particularly true for the GuacaMol scoring functions, many of which incorporate topological similarity to some reference molecule as a component [75, 108]. We chose to study the chemical appeal and novelty of RDM as opposed to that of the optimized molecules resulting from the benchmark because we wanted to distinguish which effects are attributable to the constraints and which ones to the scoring function. Nonetheless, for the sake of completeness, all analyses on the optimized molecules of section 3.3.8.1 were repeated as well.

3.4 Results

3.4.1 Constraint stringency

To rationalize the effects constraints have on molecular design it is important to study the extent to which they trim the chemical space transition graph. Figure 3.9A shows the average degree of said graph for different types of constraints. A higher average degree is indicative of a denser graph and therefore laxer constraints, whereas a lower average degree is indicative of sparser graphs and stricter constraints. The differences in graph density can be explained by discrepancies in the number of perturbations or moves that are rejected by the constraints (Figure 3.9B).

Generally speaking, constraints are stricter the more variables define the corresponding key. Accordingly, ring-aware (RA) constraints are stricter than local constraints, environment constraints are stricter than bond constraints, and bond constraints are stricter than atom constraints. Counterintuively valence constraints appear to be stricter than certain key-based constraints that encompass valence. This stems from differences in the definitions of valid valences. For valence constraints a rather conservative list of allowed valences hard coded within the RDKit is used. For key-based constraints allowed values are extracted from a large library of reference molecules. Should one find within this library a few examples of atoms with unusual valences this would suffice for said valences to be considered valid. Moving forward, results will be color-coded according to the constraint stringency order described in Figure 3.9.

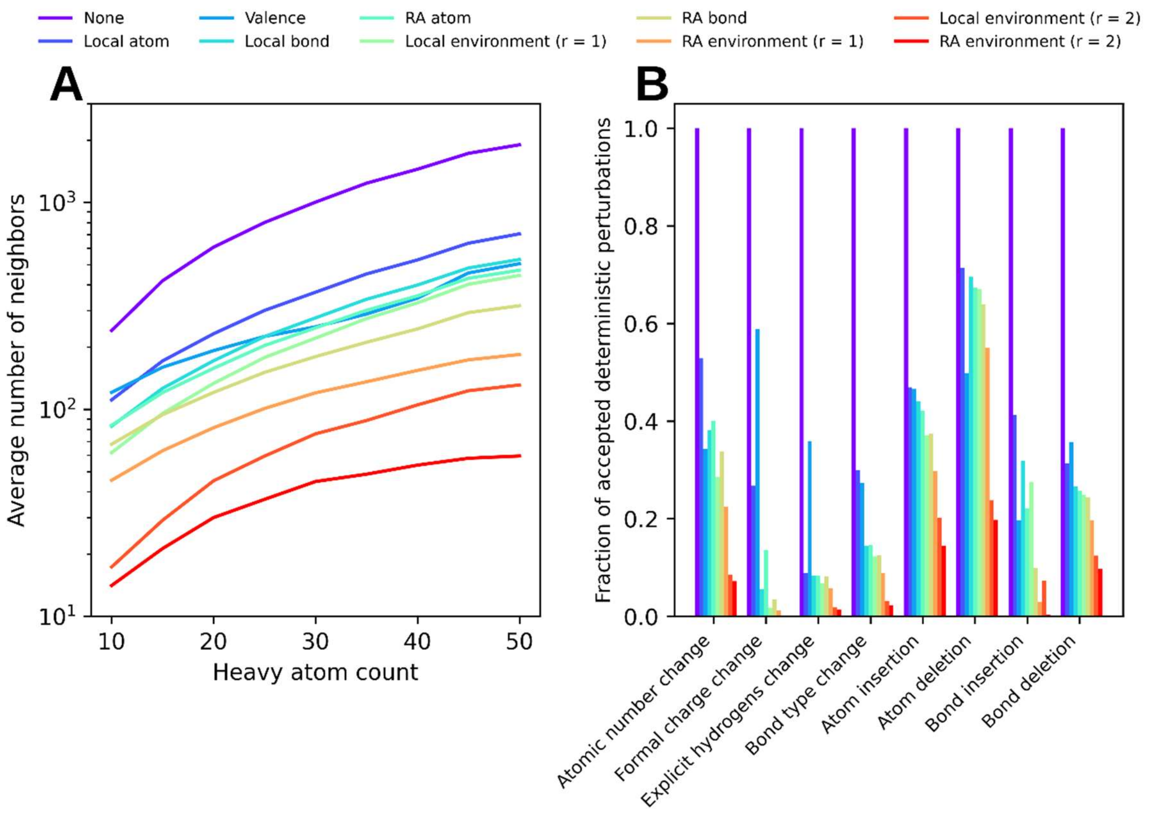


Figure 3.9. (A) Average number of neighboring molecules for molecules in ChEMBL based on their size and molecular constraints. The lower the number of neighbors the sparser the chemical transition graph. (B) Fraction of accepted perturbations broken down by perturbation type. The remainder of the perturbations were rejected by the molecular constraints.

3.4.2 Effect of constraints on synthesizability and drug-likeness

It is well established that constraining the way in which molecules are constructed increases the likelihood of designing chemically appealing molecules [73–76, 80, 82–84, 91, 92, 135]. Figure 3.10 shows how the synthetic accessibility of RDM, as measured with the SAScore [124], increases with design constraints becoming stricter. Our constraints restrict the designed molecules to topological features present in reference molecules, with the differences between them being in the granularity of these features. As the granularity increases so does the algorithm's ability to mimic the topology of reference molecules. Despite requesting RDM of a given size, when strict constraints are enforced, namely environment (r = 2) constraints, some sequences of random operations starting from vacuum can lead to "dead-end" molecules, that is, small molecules for which no other atom or bond can be added without infringing upon the constraints. Examples of such molecules can be found in Figure 3.11.

Similar, albeit tamer, results were observed for the molecules resulting from the GuacaMol benchmark optimization task (Figure 3.12, Figure 3.13). Note that the SAScores in Figure 3.12 are markedly better than in Figure 3.10 due to the benchmark's scoring functions pointing towards synthesizable molecules. This same effect also explains why the differences in synthesizability between constraint types are smaller for molecules resulting from the GuacaMol benchmark than for RDM.

The use of constraints also seems to improve the drug-likeness of the designed molecules as measured with the QED [145] (Figure 3.14). Unlike for the SAScores, this improvement was not observed for the optimized molecules as well (Figure 3.15, Figure 3.16). QED is calculated based mostly on physicochemical descriptors, yet our constraints do not consider physicochemical descriptors explicitly. Further analysis reveals that the main driver for QED improvements is a reduction in the number of undesirable substructures (i.e. structural alerts) (Table 3.7). A drop-off in QED is observed for RA environment (r = 2). This is explained by the designed molecules having over double the number of rotatable bonds one might expect to find in molecules designed with other constraints or drug-like molecules (Table 3.7). RA environment (r = 2) constraints are so strict that oftentimes the only allowed atom insertion is that of carbons in existing hydrocarbon features, resulting in long and flexible molecules (Figure 3.17).

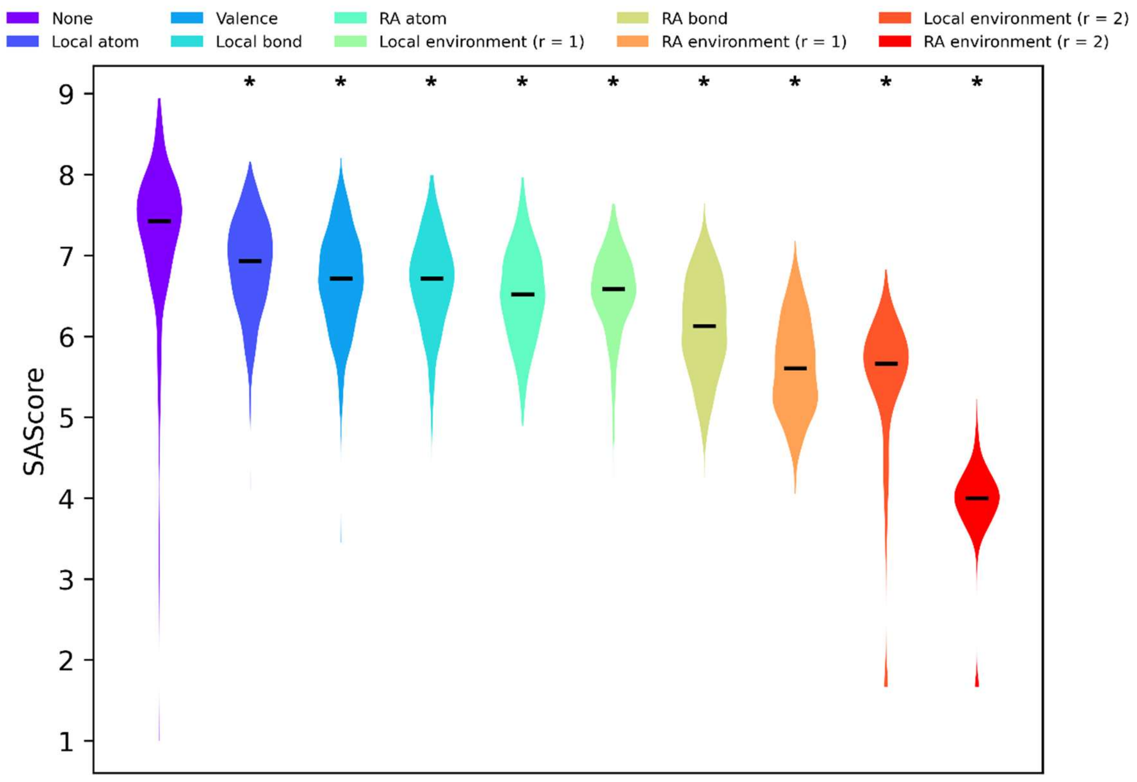


Figure 3.10. SAScore distributions of RDM using different types of constraints. Medians are shown as black lines. Lower SAScores are indicative of an easier synthesis. Stars on top of the distributions indicate statistically significant differences with the "no constraints" control group. A more detailed statistical analysis can be found in Table 3.3.

Table 3.3. Statistical analysis of RDM's SAScore differences between the no constraints control group and other groups. Pairwise comparisons were preceded by one-way ANOVA (statistic = 2161.557, p-value < 0.001).

Comparison	**Mean SAScore difference**	**Dunnett statistic**	**p-value**
None - Local atom	0.394	-13.472	< 0.001
None - Valence	0.586	-20.052	< 0.001
None - Local bond	0.604	-20.662	< 0.001
None - RA atom	0.761	-26.055	< 0.001
None - Local environment (r = 1)	0.740	-25.327	< 0.001
None - RA bond	1.148	-39.304	< 0.001
None - RA environment (r = 1)	1.645	-56.304	< 0.001
None - Local environment (r = 2)	1.845	-63.164	< 0.001
None - RA environment (r = 2)	3.332	-114.063	< 0.001

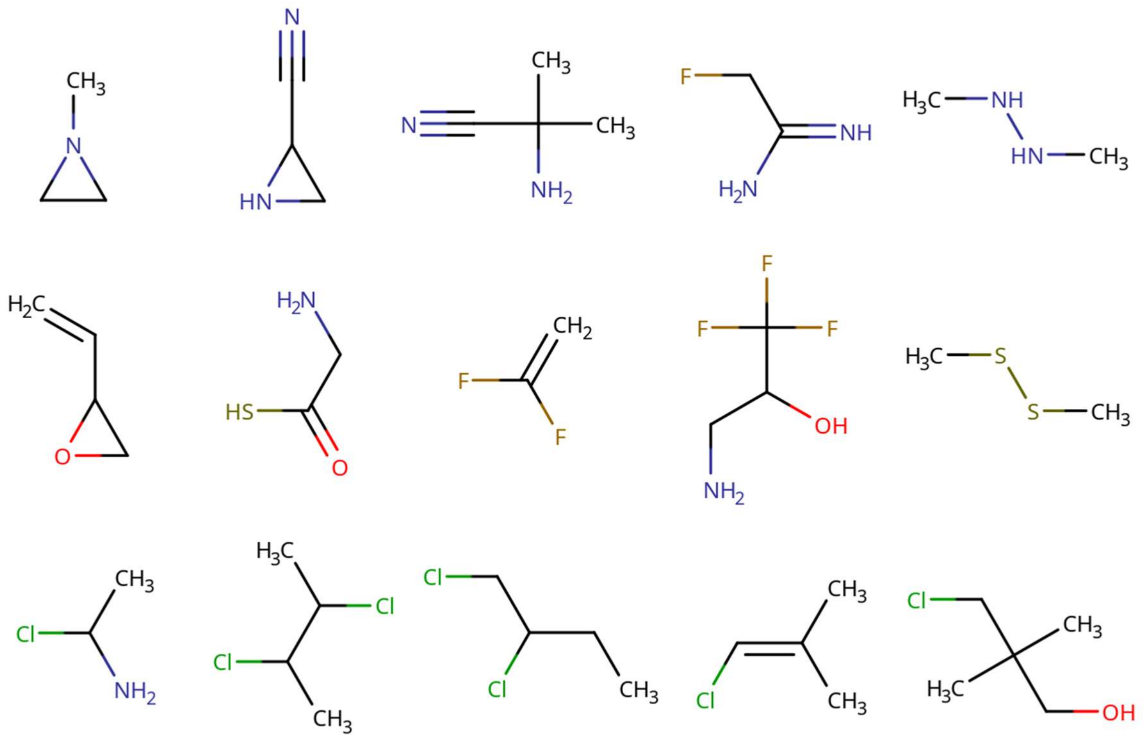


Figure 3.11. Examples of “dead-end” molecules, to which no other atom or bond can be added without infringing upon environment (r = 2) constraints.

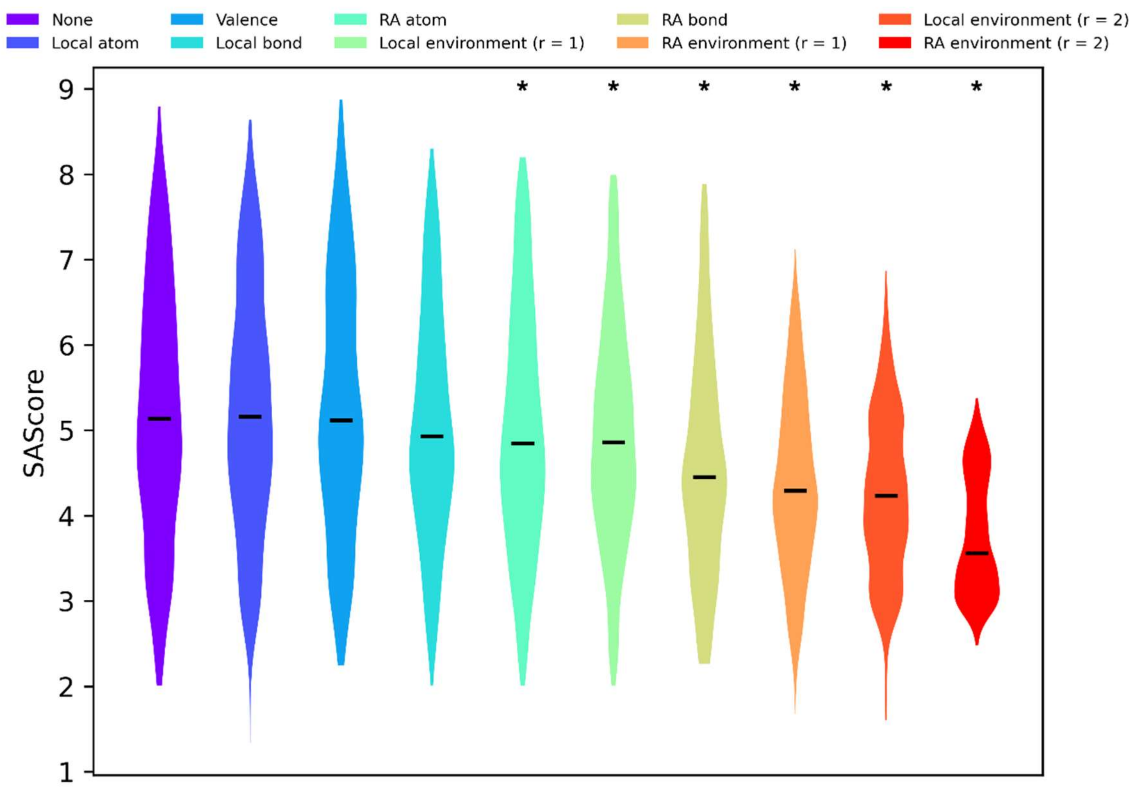


Figure 3.12. SAScore distributions of molecules designed during the GuacaMol benchmark using different types of constraints. Medians are shown as black lines. Lower SAScores are indicative of an easier synthesis. Stars on top of the distributions indicate statistically significant differences with the "no constraints" control group. A more detailed statistical analysis can be found in Table 3.4.

Table 3.4. Statistical analysis of optimized molecules' SAScore differences between the no constraints control group and other groups. Pairwise comparisons were preceded by one-way ANOVA (statistic = 153.643, p-value < 0.001).

Comparison	Mean SAScore difference	Dunnett statistic	p-value
None - Local atom	-0.035	0.618	0.995
None - Valence	-0.075	1.311	0.715
None - Local bond	0.145	-2.538	0.074
None - RA atom	0.203	-3.554	0.003
None - Local environment (r = 1)	0.271	-4.740	< 0.001
None - RA bond	0.612	-10.729	< 0.001
None - RA environment (r = 1)	0.839	-14.704	< 0.001
None - Local environment (r = 2)	0.970	-16.810	< 0.001
None - RA environment (r = 2)	1.438	-24.821	< 0.001

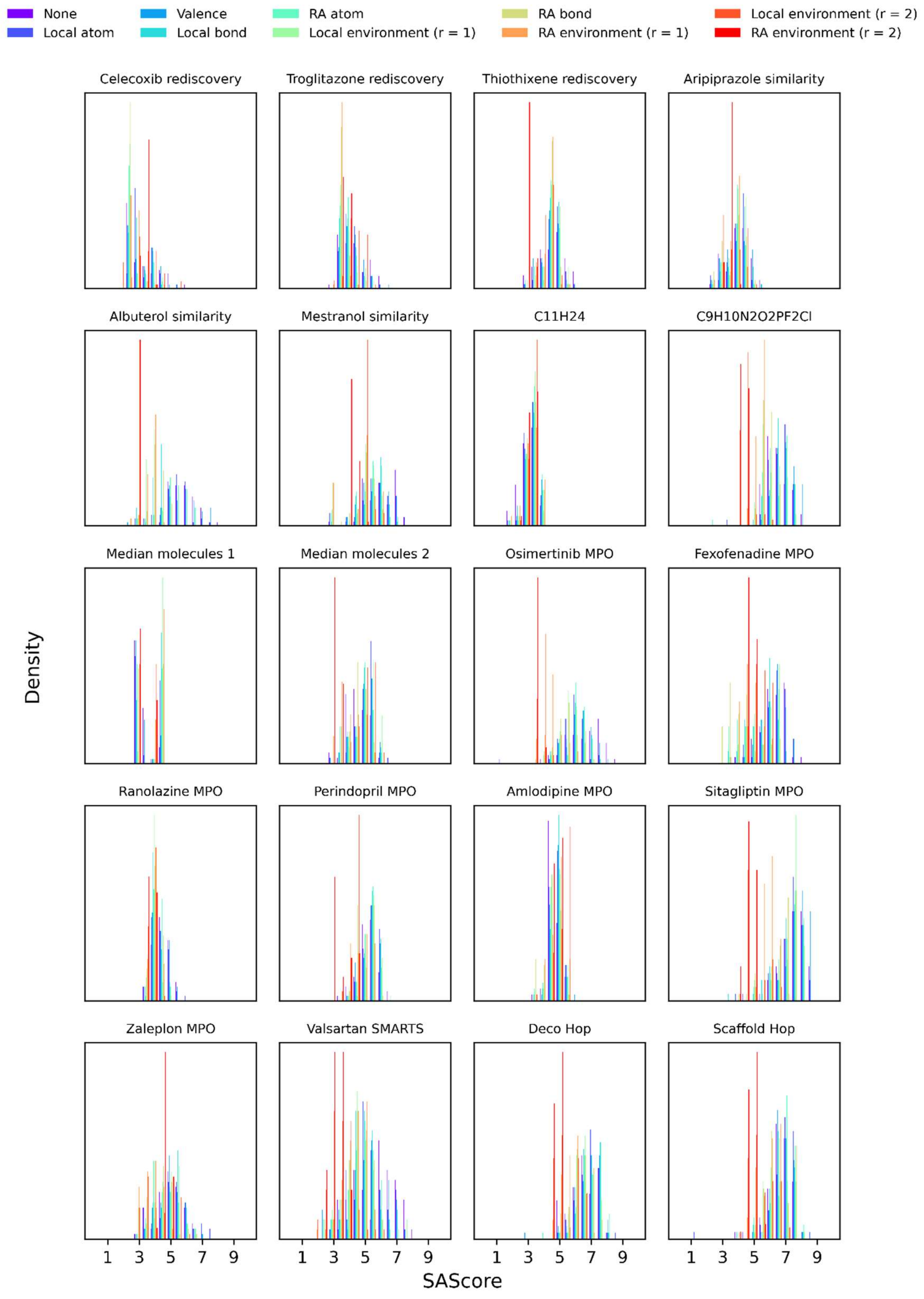


Figure 3.13. SAScore distributions of molecules designed during the GuacaMol benchmark using different types of constraints broken down by benchmark. Lower SAScores are indicative of an easier synthesis.

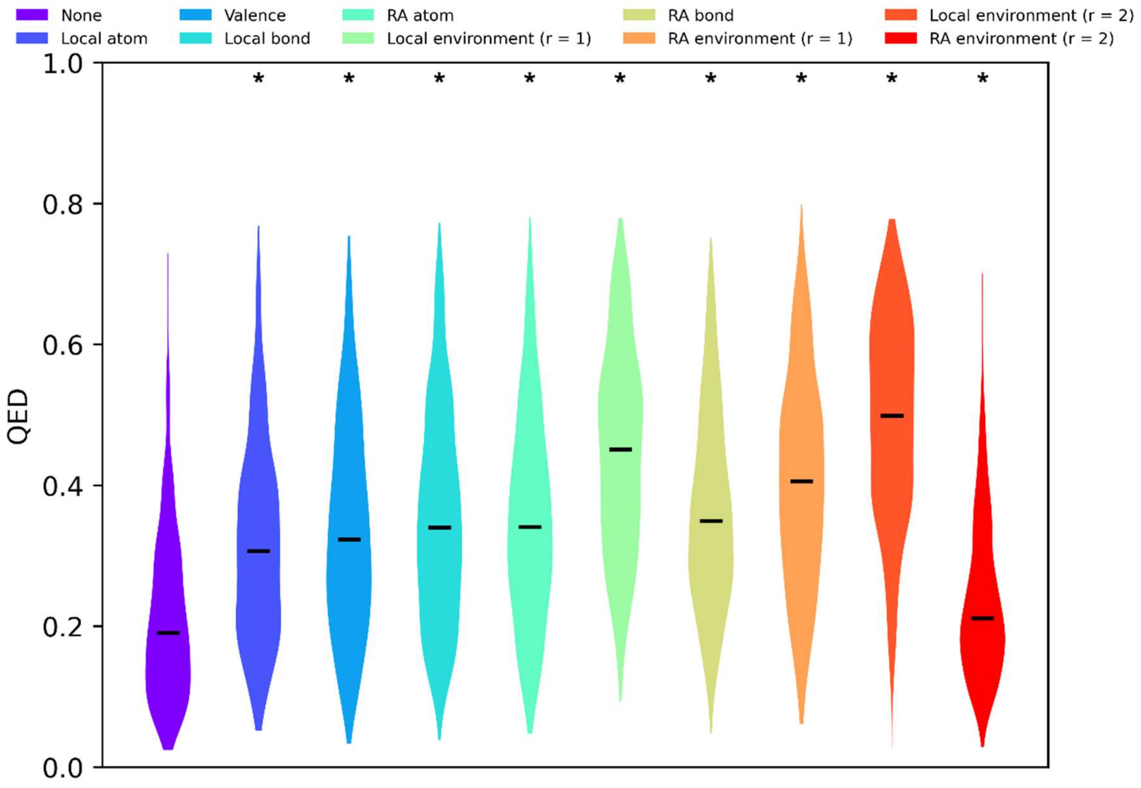


Figure 3.14. QED distributions of RDM using different types of constraints. Medians are shown as black lines. Higher values are indicative of more drug-like molecules. Stars on top of the distributions indicate statistically significant differences with the "no constraints" control group. A more detailed statistical analysis can be found in Table 3.5.

Table 3.5. Statistical analysis of RDM's QED differences between the no constraints control group and other groups. Pairwise comparisons were preceded by one-way ANOVA (statistic = 356.599, p-value < 0.001).

Comparison	**Mean QED difference**	**Dunnett statistic**	**p-value**
None - Local atom	-0.105	16.469	< 0.001
None - Valence	-0.122	19.148	< 0.001
None - Local bond	-0.141	22.118	< 0.001
None - RA atom	-0.139	21.890	< 0.001
None - Local environment (r = 1)	-0.233	36.710	< 0.001
None - RA bond	-0.150	23.603	< 0.001
None - RA environment (r = 1)	-0.191	30.011	< 0.001
None - Local environment (r = 2)	-0.277	43.660	< 0.001
None - RA environment (r = 2)	-0.024	3.825	0.001

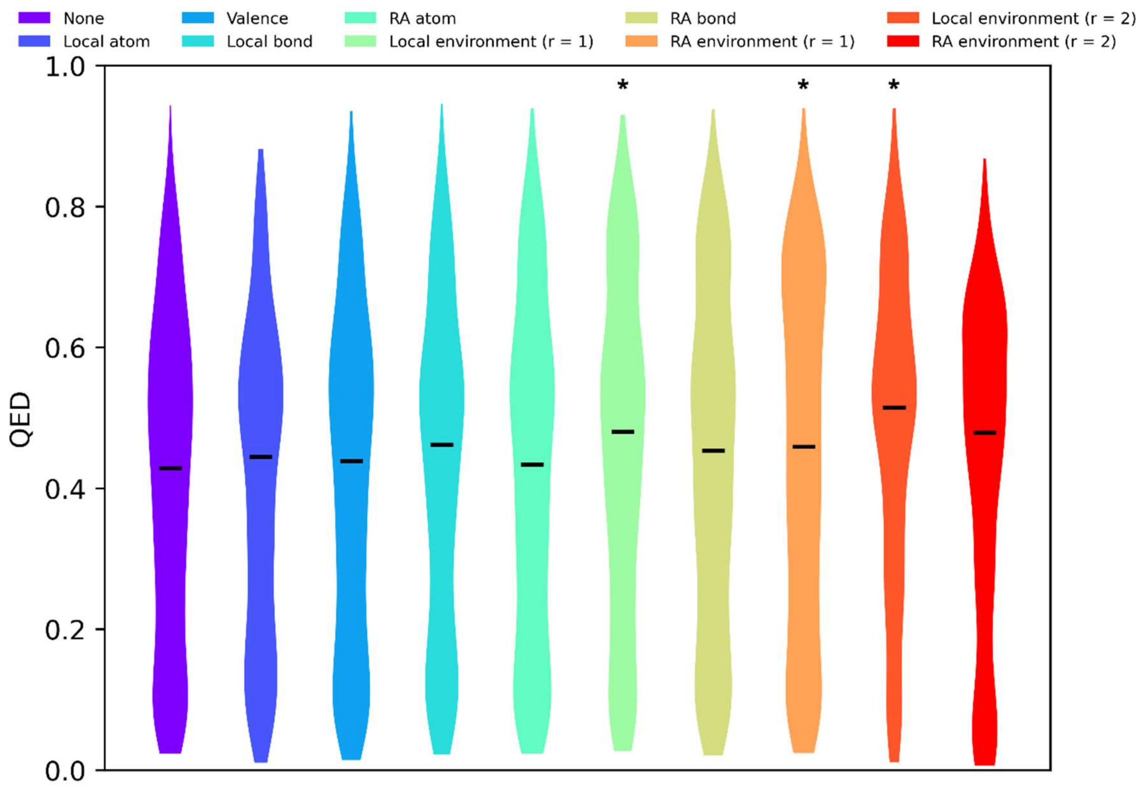


Figure 3.15. QED distributions of molecules designed during the GuacaMol benchmark using different types of constraints. Medians are shown as black lines. Higher values are indicative of more drug-like molecules. Stars on top of the distributions indicate statistically significant differences with the "no constraints" control group. A more detailed statistical analysis can be found in Table 3.6.

Table 3.6. Statistical analysis of optimized molecules' QED differences between the no constraints control group and other groups. Pairwise comparisons were preceded by one-way ANOVA (statistic = 10.134, p-value < 0.001).

Comparison	Mean QED difference	Dunnett statistic	p-value
None - Local atom	0.001	-0.120	> 0.999
None - Valence	-0.003	0.258	> 0.999
None - Local bond	-0.019	1.834	0.341
None - RA atom	-0.012	1.188	0.803
None - Local environment (r = 1)	-0.047	4.606	< 0.001
None - RA bond	-0.027	2.631	0.058
None - RA environment (r = 1)	-0.035	3.370	0.006
None - Local environment (r = 2)	-0.073	7.01	< 0.001
None - RA environment (r = 2)	-0.017	1.670	0.447

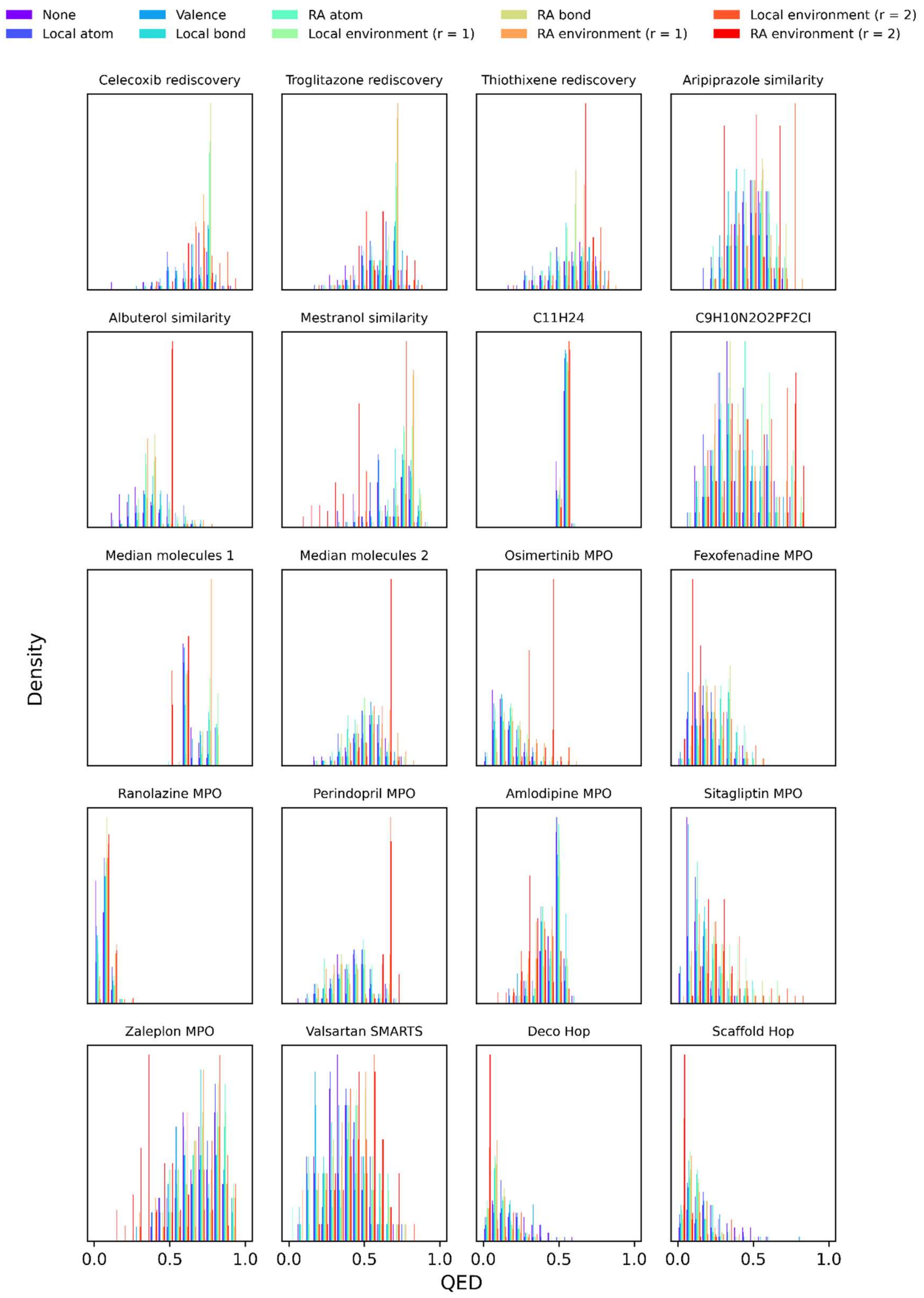


Figure 3.16. QED distributions of molecules designed during the GuacaMol benchmark using different types of constraints broken down by benchmark. Higher values are indicative of more drug-like molecules.

Table 3.7. Average values for QED components of RDM using different types of constraints. Note the decay in the number of structural alerts (ALERTS) as constraint stringency increases and the sudden spike in the number of rotatable bonds (ROTB) for RA environment (r = 2) constraints. MW = Molecular Weight, ALOGP = octanol-water partition coefficient, HBD = number of Hydrogen Bond Donors, HBA = number of Hydrogen Bond Acceptors, PSA = Polar Surface Area, AROM = number of aromatic rings.

Constraint	**MW**	**ALOGP**	**HBD**	**HBA**	**PSA**	**ROTB**	**AROM**	**ALERTS**
None	417.16	3.21	2.16	2.78	68.13	5.95	0.005	5.25
Local atom	408.68	3.22	2.04	3.62	70.10	6.45	0.008	3.61
Valence	409.18	3.30	2.27	4.85	68.04	7.26	0.016	3.18
Local bond	408.41	3.23	2.14	4.98	69.21	7.02	0.025	3.02
RA atom	410.06	3.35	2.26	4.95	70.84	7.24	0.018	2.97
Local environment (r = 1)	406.20	3.82	2.02	4.53	64.24	5.42	0.011	2.18
RA bond	408.34	3.57	2.20	4.84	69.00	7.35	0.024	2.80
RA environment (r = 1)	405.54	3.90	2.04	4.51	67.14	7.62	0.028	2.35
Local environment (r = 2)	378.15	3.67	1.50	4.03	58.51	2.91	0.001	1.45
RA environment (r = 2)	382.21	4.45	1.77	3.87	62.41	**14.31**	0.0	2.58

Figure 3.17 shows some examples of molecules designed using different types of constraints. While subjective, molecules designed using stricter constraints are chemically more appealing. It has been noted that one of the main factors explaining chemists' willingness to pursue synthesis or further development of a compound is the molecule's ring complexity [121]. The use of ring-aware constraints discourages the design of complex ring systems. One could argue that the use of strict constraints leads to the design of "plain" molecules, rich in carbons and single bonds yet poor in functional groups. This foreshadows that excessively restricting molecular construction may be undesirable. Regardless of the constraints used, RDM are unlikely to contain aromatic systems (Table 3.7, Figure 3.17). As discussed previously, the creation of aromatic rings requires very specific arrangements of single and double bonds that are unlikely to occur by chance. This illustrates the value of the proposed partial aromaticity treatment.

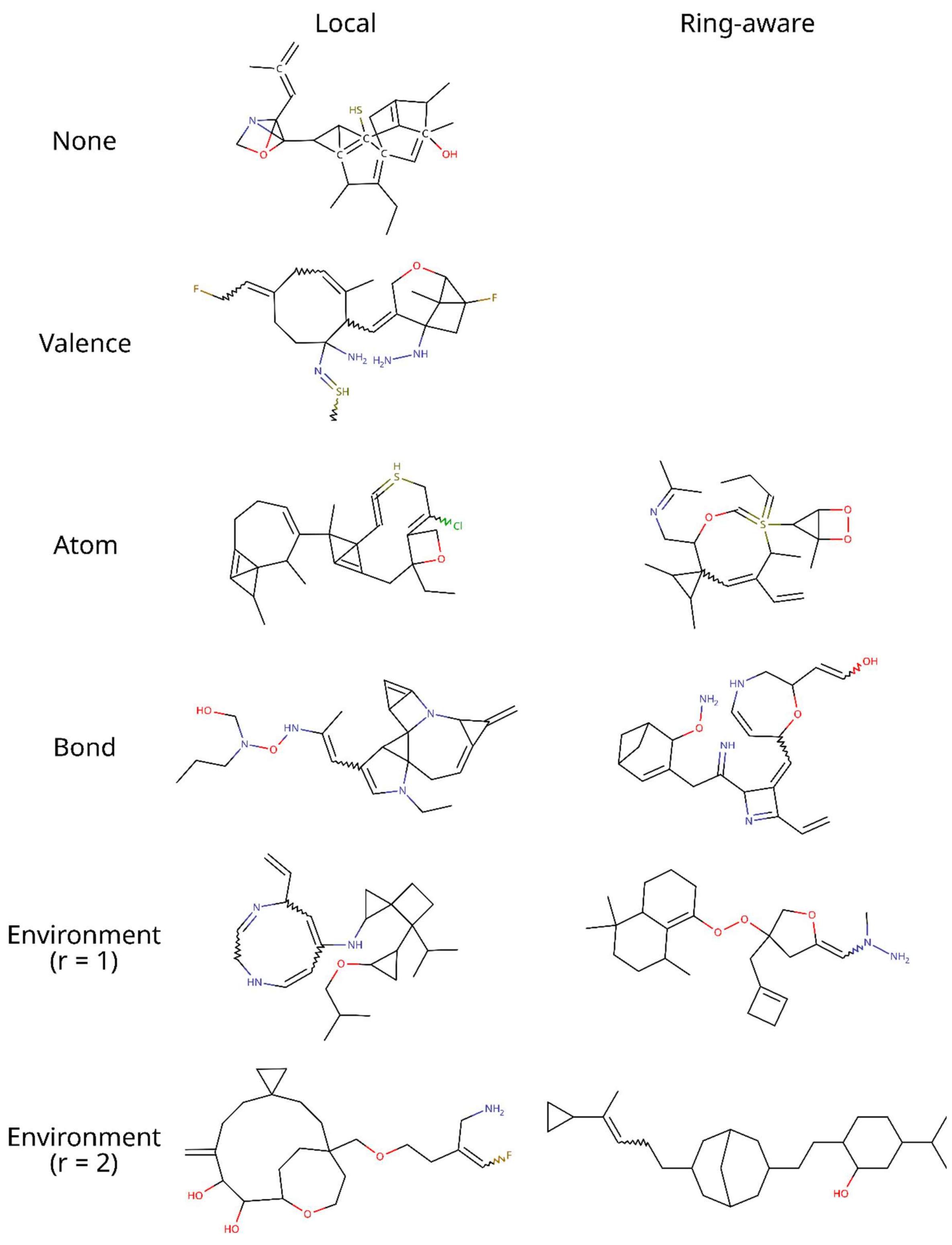


Figure 3.17. Examples of RDM molecules designed by successive random atom and bond insertions using different types of constraints.

We would like to clarify that within **Molpert** the only dependable source of molecule correctness are the molecular constraints. Weighted property sampling reduces the probability of stochastically generating perturbations that would infringe upon the constraints (Figure 3.18), but does not prevent it. Weighted sampling should thus be seen more as an algorithmic efficiency optimization than a strategy to design reasonable molecules.

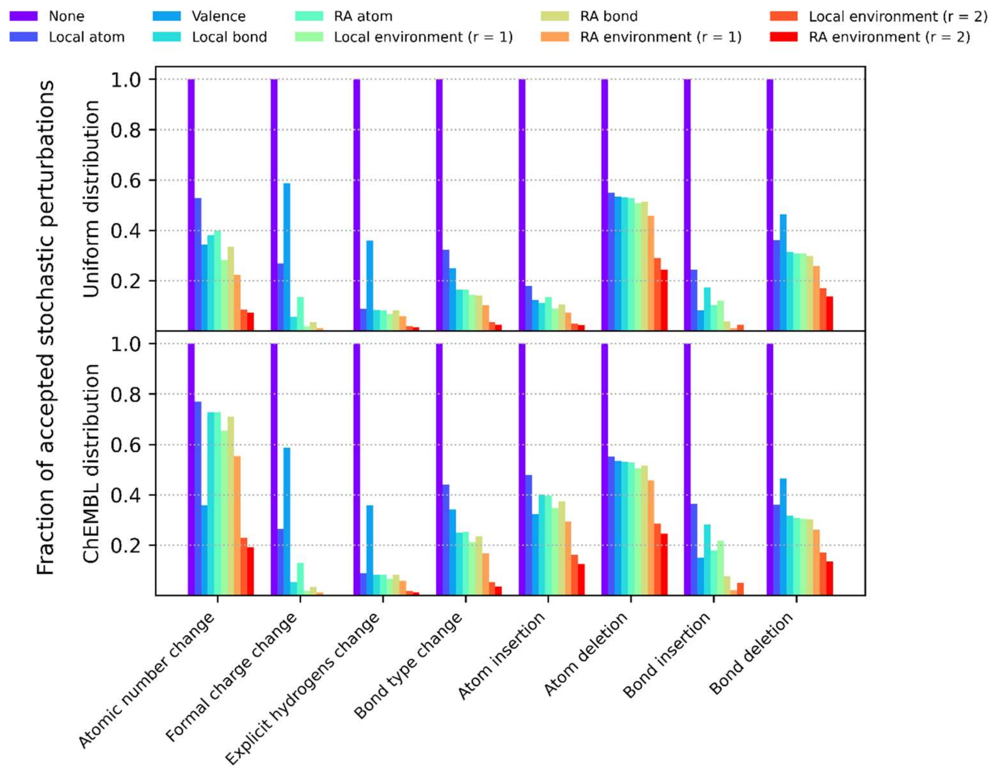


Figure 3.18. Fraction of stochastically generated perturbations accepted by the molecular constraints, broken down by perturbation type and property value sampling strategy. The upper panel corresponds to uniform random sampling of property values, whereas the lower panel corresponds to weighted random sampling of property values Note that the only perturbation types where property values are sampled are property perturbations (i.e. atomic number, formal charge, explicit hydrogens and bond type changes) and atom/bond insertions.

One could be concerned that imitating reference chemistry stifles chemical innovation. To investigate this concern, we visualized the positions of designed molecules in a 2D chemical space, using ChEMBL [136] as a reference space (Figure 3.19). There is some overlap between ChEMBL and RDM, but the latter are skewed towards the less densely populated areas of chemical space, regardless of the constraints used. Optimized molecules are more similar to known chemistry due to scoring function bias (Figure 3.20). It should be noted that a 2D projection of chemical space is overly simplistic, with distances between molecules appearing to be smaller than they truly are. Hence the designed molecules are more distinct from ChEMBL than what Figure 3.19 might indicate. We believe that the designed molecules are sufficiently novel.

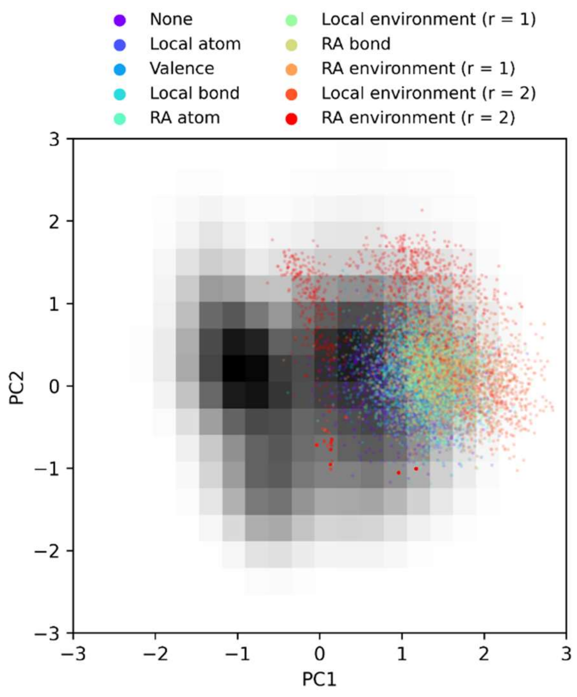


Figure 3.19. Positions of RDM in 2D PCA space. The grayscale grid represents the density of ChEMBL molecules in chemical space on a linear scale, with darker cells being more densely populated.

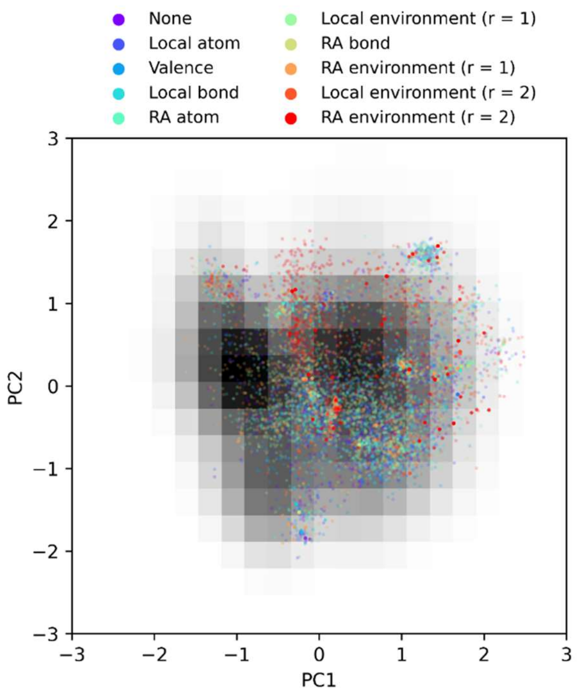


Figure 3.20. Positions of molecules designed during the GuacaMol benchmark in 2D PCA space. The grayscale grid represents the density of ChEMBL molecules in chemical space on a linear scale, with darker cells being more densely populated.

3.4.3 Effect of constraints on molecule fitness

The effect constraints have on compound fitness during molecular optimization is poorly understood. Figure 3.21 shows the optimization power of a **Molpert**-based evolutionary algorithm in the GuacaMol benchmark suite [108] using different types of constraints. As a reminder, constraints are enforced every time a molecule is mutated. Using mild constraints, that is, anything between local and RA bond constraints, leads to significantly improved molecule fitness over unconstrained molecular design. RA bond constraints performed best, followed closely by local environment (r = 1) and RA atom constraints. RA environment (r = 1) constraints are equivalent to unconstrained molecular design in terms of molecule fitness. Stricter constraints, namely environment (r = 2) constraints are markedly worse.

The results in Figure 3.21 suggest that there is a constraint stringency sweet spot that trims the search graph in just the right way to facilitate the optimization process. Upsettingly the exact location of this sweet spot depends on the individual benchmark (Figure 3.22). The most common pattern is a fitness maximum at some constraint stringency middle point such as RA bond constraints, with laxer and stricter constraints both performing worse. Even in the cases where fitness is unaffected by constraint choice most constraints seem to be tolerated. This is an encouraging result as the primary use of constraints in molecular design is to increase the likelihood of designing drug-like and synthesizable molecules.

Two peculiar cases are those of Celecoxib and Troglitazone rediscovery, where very pronounced fitness differences are observed between local and RA constraints (Figure 3.22). Visual inspection of the designed molecules reveals that when using local constraints the algorithm correctly rediscovers many of the reference molecule's features, but proposes alternative ring systems. In rediscovery benchmarks the goal is to re-design a reference molecule, with the score being given by the topological similarity to the reference molecule. Topological similarity is assessed through means of ECFP4 fingerprints similarity [34], with two molecules being similar if they share many chemical features. Crucially, it is not required for the features to be in the same position for two molecules to be deemed similar. Celecoxib and troglitazone possess multiple benzene rings, with paths of aromatic carbons as features. The algorithm is rewarded for designing molecules with aromatic carbons, but this reward is the same regardless of the topology and size of the ring systems. Limiting the sizes of designed rings with RA constraints can prevent the algorithm from being led astray and towards macrocycles by the scoring function (Figure 3.23).

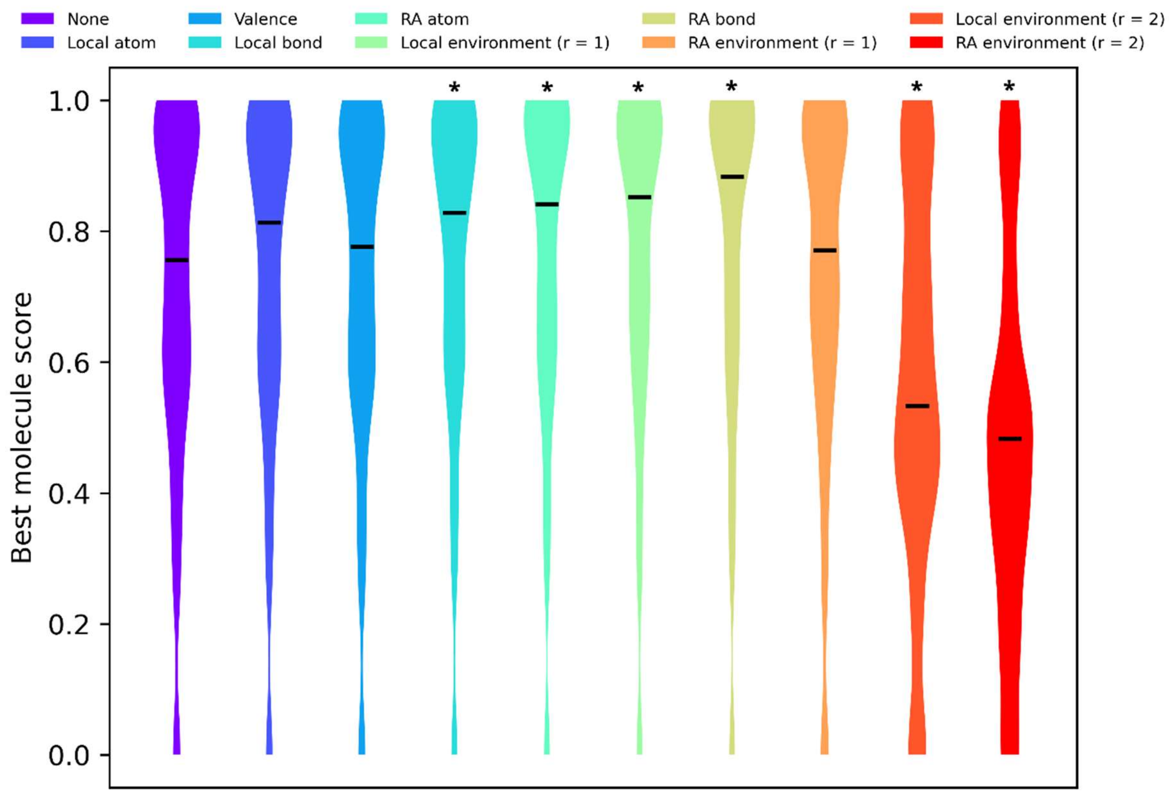


Figure 3.21. Distributions of top molecule scores, as assessed by the GuacaMol goal-directed scoring functions. Medians are shown as black lines. Only the best molecule of each population is included. The benchmark suite consists of 20 individual benchmarks, but for clarity's sake the results of all benchmarks were aggregated. A per-benchmark breakdown can be found in Figure 3.22. Stars on top of the distributions indicate statistically significant differences with the "no constraints" control group. A more detailed statistical analysis can be found in Table 3.8.

Table 3.8. Statistical analysis of molecule fitness differences between the no constraints control group and other groups. Pairwise comparisons were preceded by a Kruskal-Wallis test (statistic = 951.677, p-value <0.001).

Comparison	**Median molecule fitness difference**	**Mann-Whitney U-statistic**	**p-value**
None - Local atom	-0.058	472579.5	0.033
None - Valence	-0.020	489620.5	0.421
None - Local bond	-0.073	453589.5	< 0.001
None - RA atom	-0.085	434893.0	< 0.001
None - Local environment (r = 1)	-0.096	436572.0	< 0.001
None - RA bond	-0.128	411981.0	< 0.001
None - RA environment (r = 1)	-0.015	478862.5	0.109
None - Local environment (r = 2)	0.223	613857.0	< 0.001
None - RA environment (r = 2)	0.273	687198.0	< 0.001

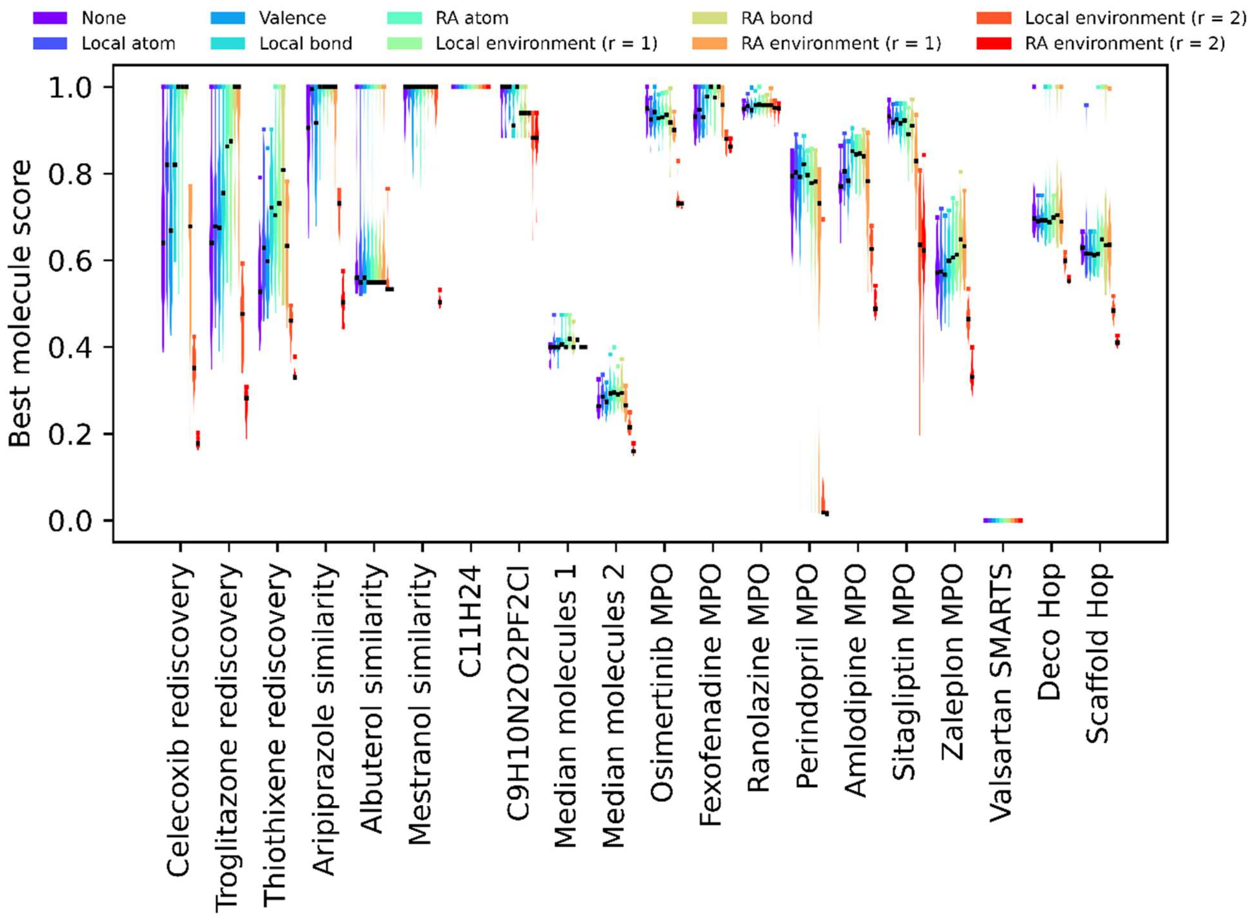


Figure 3.22. Distributions of top molecule scores, as assessed by the GuacaMol goal-directed scoring functions. Only the best molecule of each population is included. Black squares and colored squares represent median and maximum scores, respectively.

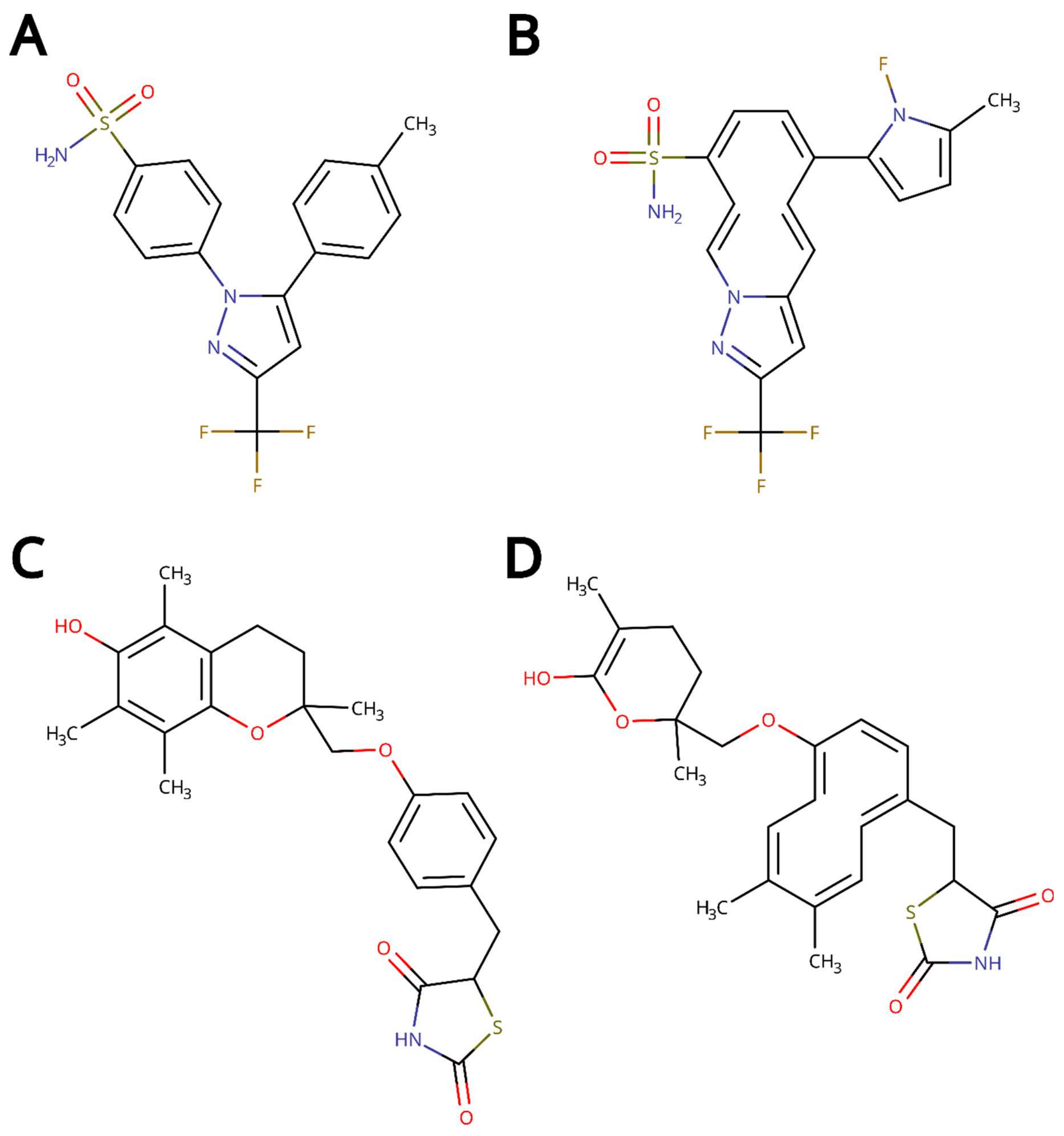


Figure 3.23. Celecoxib (A), troglitazone (C) and examples of molecules designed during their rediscovery benchmark using local bond constraints. The designed molecules B and D score relatively high (0.62 and 0.69, respectively) due to the presence of common chemical features albeit in different positions. Note that the 10-membered cycles in B and D are deemed aromatic by Hückel's rule [149] and the RDKit, despite not being aromatic due to ring strain [150].

3.4.4 Effect of constraints on computational performance

It’s worth noting that molecular design constraints can add considerable computational overhead (Figure 3.24). This is especially true for **Molpert** since constraints are enforced in a naive fashion. The slow down stems from a higher perturbation rejection rate for stricter constraints, prolonging the search for a suitable perturbation. Interestingly the number of molecules designed before the algorithm reaches convergence is moderately lower for stricter constraints. For the vast majority of objective functions this decrease is insufficient to offset the increased perturbation cost. Nonetheless, when working with very expensive objective functions the cost of perturbing molecules can be negligible compared to the cost of scoring them, making the use of constraints as convergence acceleration strategy an appealing proposition.

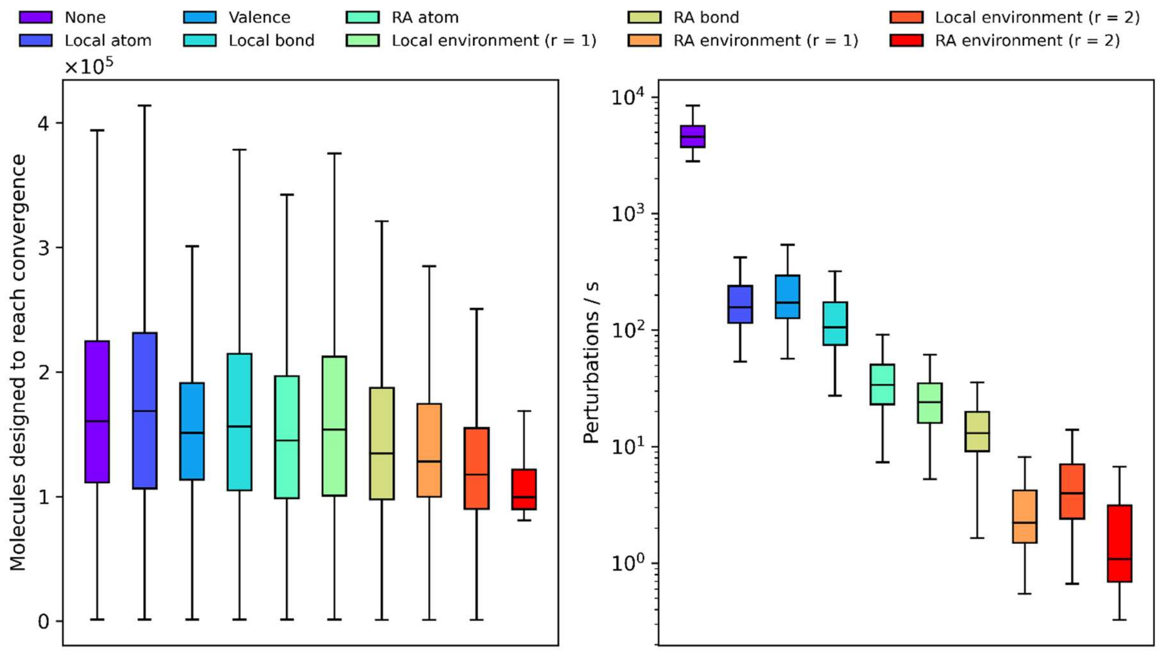


Figure 3.24. Number of molecules designed to reach convergence (left) and the number of perturbations executed per second (right) stratified per constraint type. Boxes represent interquartile ranges (IQR), the black line within them medians and the whiskers Q ± 1.5IQR. Performance numbers are for a single-threaded workload on an AMD Epyc 7452 CPU clocked at 2.35 GHz.

3.5 Discussion

Our results indicate that moderately constraining molecular construction has a net positive effect as it increases both the synthesizability and fitness of designed molecules. Nonetheless one must care to not choose excessively strict constraints as this can cause a sharp degradation of molecule fitness. As a guideline we recommend constraining bond or small environment properties and, if necessary, ring topologies. However, while the results presented herein apply to atom-based evolutionary algorithms, they may not be extrapolatable to alternative molecular optimization schemes. Evolutionary algorithms are powerful optimizers capable of navigating complex search spaces. Other algorithms such as tree searches may be less tolerant of barriers in search space and therefore construction constraints. We developed a simple tree search algorithm to test this hypothesis but found the fitness of the designed molecules too poor to extract any useful conclusions. Evolutionary algorithms are heuristic gradient-free optimization algorithms. They wander around chemical space until they stumble upon good solutions by chance. For an algorithm lacking a sense of direction the very dense chemical spaces characteristic of unconstrained molecular construction can seem like a maze with many “false paths”. Gradient-based optimization algorithms do have a sense of direction and may benefit from unconstrained molecular design.

Versatility was a major consideration when designing **Molpert**. Unfortunately, in software development versatility often comes at the cost of computational efficiency. **Molpert** is efficient at unconstrained molecular design, but this efficiency decreases with constraint stringency. Despite the decreased efficiency molecular design remained a tractable task. If one were to settle on an immutable set of constraints that molecules must fulfill it would indubitably be possible to write more specialized and performant algorithms. Should one wish to do so we would recommend using **Molpert** to build a prototype and confirm the effect of the envisioned algorithm and/or constraints before committing resources to developing a performant solution. We wanted to be able to re-use the code base in projects with differing requirements. Anecdotally during the development of the software we went through multiple iterations of more efficient yet less flexible constraint implementations, but kept encountering use cases that could not be covered by those alternative systems. This cemented our conviction to support truly arbitrary constraints.

In lieu of using constraints one could embrace unconstrained molecular design. When using stochastic molecule generators molecule fitness follows a distribution. While unconstrained molecular design may yield less fit molecules on average, it still may occasionally result in high scoring molecules (Figure 3.21). Sampling more times from a distribution with a lower median may be a superior strategy to sampling fewer times from a distribution with a higher median, provided that the variance is large enough (Figure 3.21). The faster unconstrained molecular generation allows one to roll the dice more often in the same amount of time. Biasing the design towards synthesizable molecules remains possible in absence of constraints by incorporating synthesizability into the objective function [62, 109]. Ideally the objective function should be able to evaluate the fitness of potentially invalid molecules resulting from unconstrained molecular design. Our benchmark shows that at least some scoring functions are able to do so, and we hypothesize that most ligand-based scoring functions will share this ability. Machine learning models may excel at this task given their interpolation capabilities. Structure-based scoring functions requiring conformation generation or relying on knowledge-based parameters, such as molecular mechanics, might be less suited for this purpose. Even then one could return null or negative fitness values when a molecule cannot be evaluated, in which case the objective function acts as a constraint itself. Lastly, while it is usually undesirable to design difficult to synthesize or even chemically invalid molecules some readers may find use in expressly generating these sorts of unreasonable molecules, for example as negative training data for machine learning models [151].

Chapter 4 Computationally efficient enforcement of molecular constraints

4.1 Source

This chapter is based on the publication:

Kerstjens, A., De Winter, H. **LEADD: Lamarckian evolutionary algorithm for de novo drug design**. *J Cheminform* 14, 3 (2022). https://doi.org/10.1186/s13321-022-00582-y

4.2 Problem statement

In the previous chapter we showcased the benefits of constrained molecular design. We determined that an evolutionary algorithm, when limited to constructing molecules composed of bonds or small circular atomic environments found in reference desirable chemistry, enables the design of fitter (i.e. higher objective values), easier to synthesize and more drug-like molecules.

Yet not all was sunshine and roses. Despite using constraints, some of the designed ring systems were unwieldly. Other molecules looked sane, but were rather boring hydrocarbon skeletons. Most concerningly, since the constraints were enforced in a computationally naive way the the throughput of the generator was poor, making it unsuitable for many use cases.

In this chapter we address these concerns by describing Lamarckian Evolutionary Algorithm for *de novo* Drug Design (**LEADD**). Much like the previously described algorithm, **LEADD** will mimic reference chemistry by ensuring that the bonds/environments in designed molecules have been observed in reference molecules. However it differs in the following:

- Molecules will be constructed as combinations of multi-atomic molecular fragments instead of single atoms. By capturing entire ring systems in fragments we rid ourselves of the challenge of designing reasonable ring systems atom-by-atom. Moreover, fragment mutations are associated with a bigger step size in chemical space, which may help the algorithm in escaping local fitness minima and discovering interesting chemical entities.

- A computationally efficient algorithm will be in charge of forming sane bonds, instead of cycling through all bonds that could be formed and determining which ones are acceptable according to some filters.

Additionally, we will explore a Lamarckian evolutionary mechanism that adjusts the future reproductive behavior of molecules based on the outcome of previous generations. **LEADD** attempts to strike a balance between optimization power (OP), synthetic accessibility (SA) of designed molecules and computational performance.

4.3 Methodology

4.3.1 Fragment library creation

A virtual library, assumed to be representative of drug-like chemical space, is fragmented to yield the fragments employed by **LEADD** during the design process.

Within this context, a fragment is a connectivity-encoding molecular subgraph of the source molecule from which it was extracted. A connection is an object describing the bond between two atoms and is directional by nature. It can be represented as a three-integer tuple, where the integers describe the starting atom type, ending atom type and bond type, respectively. Bonds are classified into either single, double or triple bond type (aromatic bond types do not occur since rings are not fragmented; see below). While any atom typing scheme may be used, we have implemented MMFF94 [152] and Morgan atom types in **LEADD**. Morgan atom types are integers describing an atom's circular chemical environment. They are the bit indices of a RDKit sparse Morgan fingerprint [34, 47] after *r* iterations of the Morgan algorithm [46] (see 1.3). For clarity, the examples and figures in this chapter use MMFF94 atom types.

We distinguish between connections, which are generic objects describing the type of an atom–atom bond, and connectors, which are specific instances of a connection centered on a fragment's atom. During molecule fragmentation, the bonds between the fragment's molecular subgraph and its extra-fragment adjacent atoms are recorded as connectors (Figure 4.1).

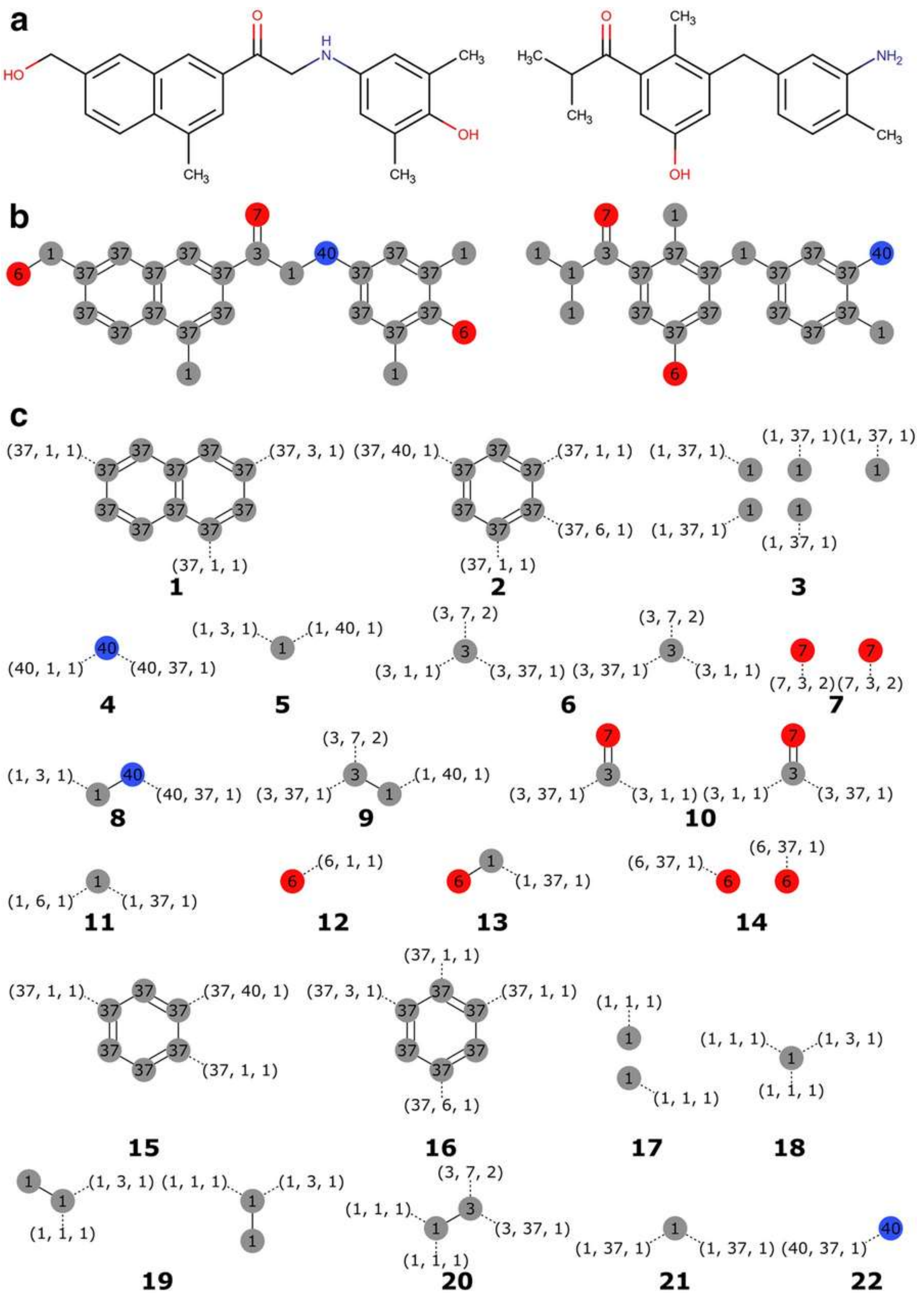


Figure 4.1. Fragmentation example of two molecules. The input molecules (A) are assigned MMFF94 atom types (B). Ring systems and all possible subgraphs from the remaining linkers and side chains of a given size (in this example s ϵ [0 .. 1]) are extracted as fragments (C). The bonds that were cut to extract fragments become connectors, and are represented as three-membered tuples in parenthesis. The number in bold below each fragment is its ID.

For each molecule, fragmentation starts by isolating ring systems from the acyclic regions. Rings pertaining to the Smallest Set of Smallest Rings (SSSR) [23] are considered to be part of the same ring system if they share at least one atom. Given the complexities of designing drug-like ring systems, we decided to consider whole ring systems as fragments. The remaining acyclic structures may either be taken as fragments as a whole or subjected to systematic fragmentation by extracting all possible molecular subgraphs of a given size from them, with each subgraph becoming a fragment (Figure 4.1). Hydrogens are treated implicitly. The size of the extracted subgraphs (*s*), given in number of bonds within the subgraph, is provided by the user. When $s = 0$, single atom fragments are generated. Fragments of different sizes can be combined by specifying a range of sizes.

Two fragments are considered equivalent only if both their molecular graph and connectors are the same. Both attributes are encoded as canonical ChemAxon extended SMILES (CXSMILES) [153] and molecular identity is assessed as canonical CXSMILES identity. The generated fragments, their connectors, frequencies, sizes and other convenience information are stored in a relational database. When a generated fragment is already present in the database its frequency is incremented by one.

4.3.2 Connection compatibility rules

Fragment compatibility is defined at the connection level. Two fragments can be bonded together if two of their free connectors are compatible. Whether two connections are compatible is determined by a set of pairwise and symmetric compatibility rules.

The compatibility rules are extracted from the connections table of the fragment database according to a user-specified compatibility definition. We employ two of those definitions, termed the “strict” and “lax” compatibility definitions. Both definitions are illustrated in Figure 4.2.

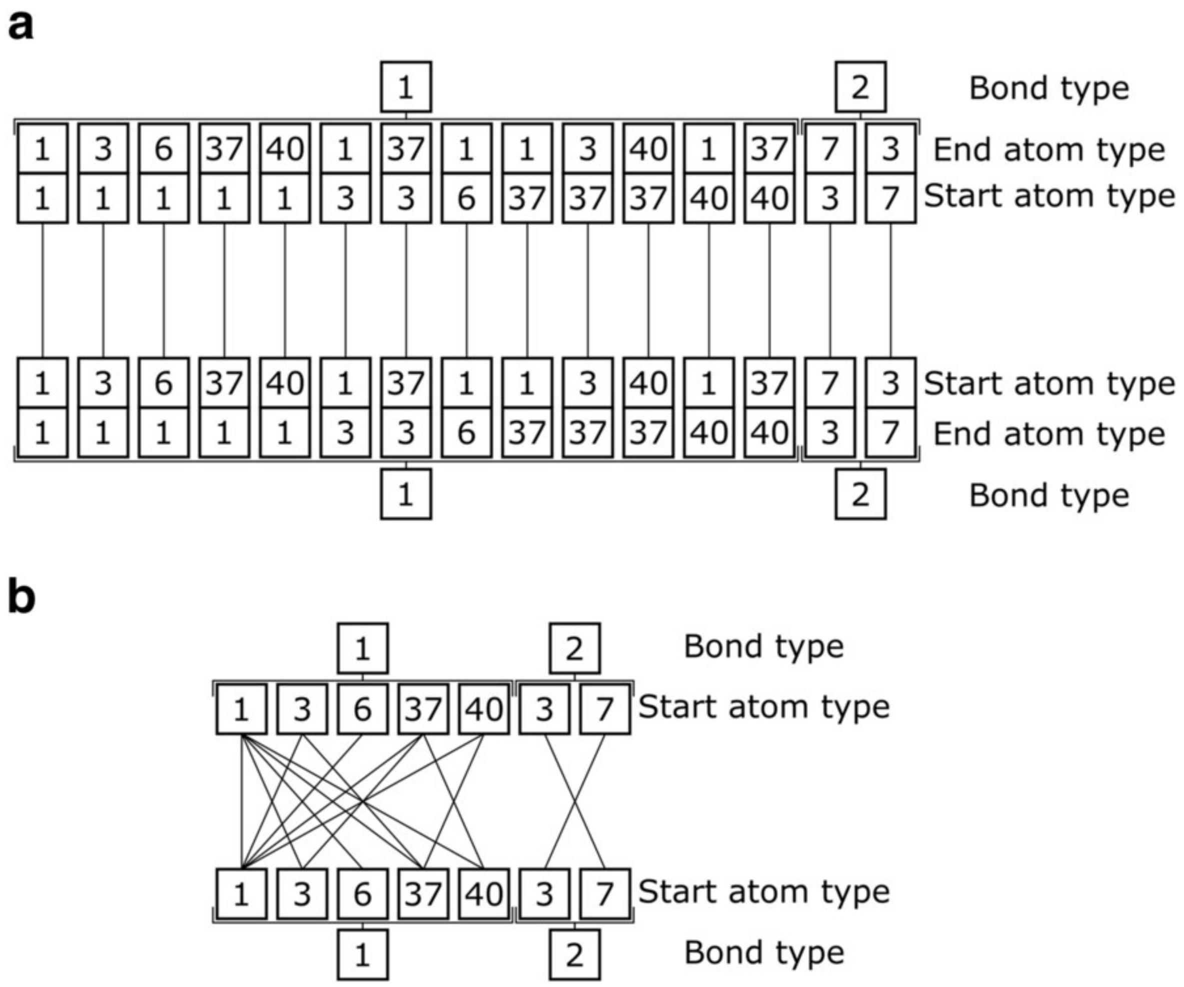


Figure 4.2. Connection compatibilities of the connections in Figure 4.1 according to the strict (A) and lax (B) compatibility definitions. Since in the lax definition the end atom type is irrelevant it is omitted.

According to the strict definition two connections are compatible only if (a) their bond types are the same, and (b) their atom types are mirrored (i.e. the start atom type of one is the end atom type of the other and vice versa). Consequently, only a single connection is compatible with each connection. During molecule design this entails that the connectivity of fragments to their flanking atoms in their source molecules is preserved. In other words, a fragment must be connected to atoms of the same atom type as those that flanked the fragment in the source molecule.

When following the lax compatibility definition two connections are compatible if (a) their bond types are the same, and (b) if the starting atom type of one has been previously observed paired with the starting atom type of the other in any connection. This definition expands the connectivity scope from the fragment's source molecule to the entire source molecules pool. In other words, two atom types can be connected if they have been observed paired together in any of the database's connections, which means they were bonded in at least one of the source molecules. As such, the strict compatibility definition is a subset of its lax counterpart.

4.3.3 Chromosomal representation and initialization

Molecules are represented internally as meta-graphs [64], where each vertex is a molecular graph corresponding to a fragment, and the edges describe which connectors bind the fragments (Figure 4.3). Due to the complexities of designing drug-like ring systems we treat ring systems as whole fragments, represented as a single vertex in the meta-graph. However, while the genetic operators do not create cycles in the meta-graph, they would work on existing cycles if one were to add a cyclization operator in the future.

The meta-graph chromosome can be translated into a single molecular graph by connecting the molecular graphs of all fragments (Figure 4.3). Thereafter, hydrogens are added to satisfy all incomplete valences. For elements with more than one valid valence like sulphur or phosphorus hydrogens are added up to the closest valid valence.

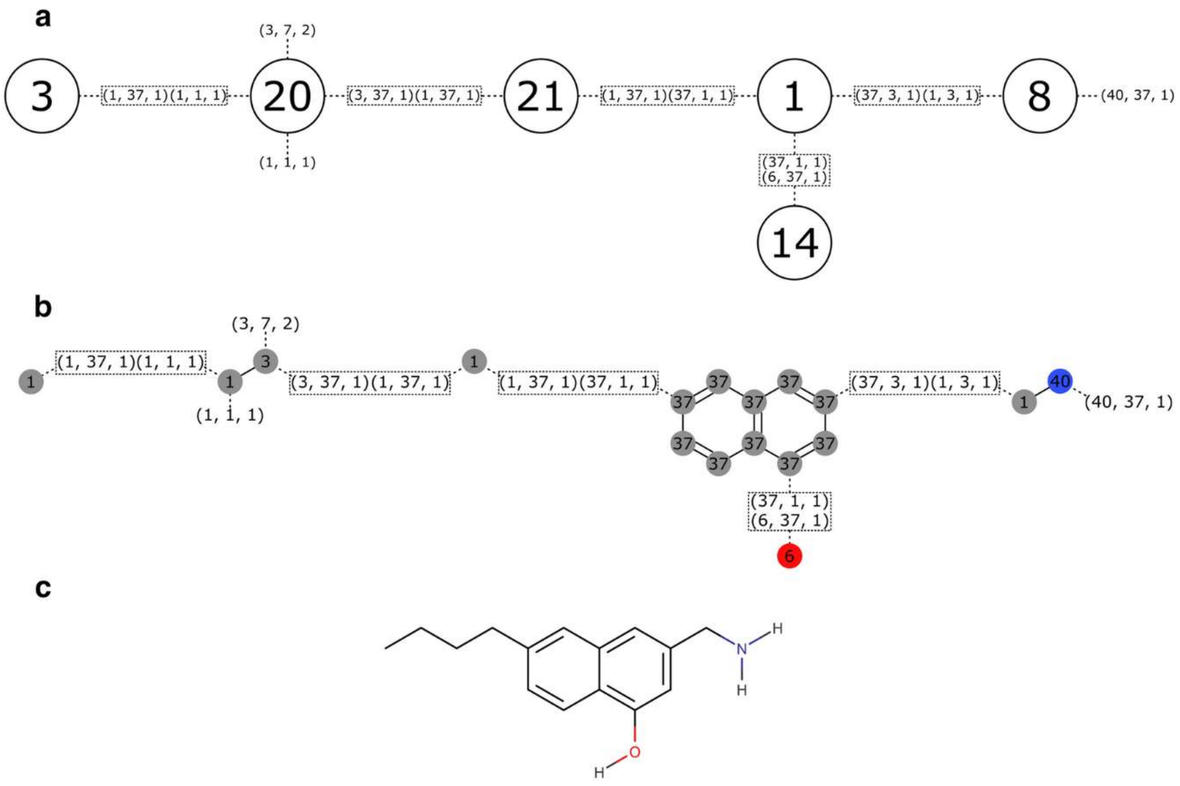


Figure 4.3. Chromosomal representation of a molecule created through combination of fragments in Figure 4.1 using the lax compatibility definition. (a) Chromosomal meta-graph. Numbered vertices correspond to fragment IDs. Numbers between parenthesis represent connector tuples. Bonds between connectors are represented as rectangles. (b) The chromosome with fragments shown as their molecular graphs. (c) Translation of the chromosome to the molecule seen by the user.

Upon initialization, for true *de novo* drug design random chromosomes are generated by successively combining random fragments. However, in some instances the user may want to perform molecule optimization instead, starting from a known population of molecules. In this case, it's possible to convert regular molecular graphs into meta-graphs by following the previously laid out fragmentation procedure using single atom acyclic fragments ($s = 0$). If any of the connections generated during the fragmentation of starting molecules do not appear in the database, connection compatibility information won't be available for them and the molecule will therefore be skipped.

4.3.4 Genetic operators

LEADD employs eight distinct genetic operators to modify the chromosome and generate offspring (Figure 4.4). Some of these operators have a peripheral and internal variant, referring to the location of fragments on which they operate. Peripheral fragments are those connected to one or less other fragments (vertex degree $d \leq 1$), while internal fragments are those connected to two or more fragments ($d \geq 2$). While peripheral operators are theoretically sufficient to access the entirety of the search space, in practice this relies on statistically unlikely sequences of operations, since to modify the core of the molecule one would have to "backtrack" and remove all peripheral fragments obstructing it. Hence, the algorithm would be very likely to get stuck in local minima on the fitness landscape.

The function of peripheral variants is mostly self-explanatory: peripheral expansions attach a fragment sampled from the database to a free connector, while peripheral deletions delete a peripheral fragment.

In internal expansions a fragment is inserted between a target fragment and one or more of its adjacent fragments. For this purpose, connectors involved in bonding the target fragment to the adjacent fragments are considered free.

In an internal deletion an internal target fragment is deleted. This is only possible if one of the fragments adjacent to the target fragment can "take its place" and bond to the remainder of the adjacent fragments.

In a substitution a target fragment is replaced by a fragment in the database. Connectors bonding the target fragment to its neighboring fragments are deemed free.

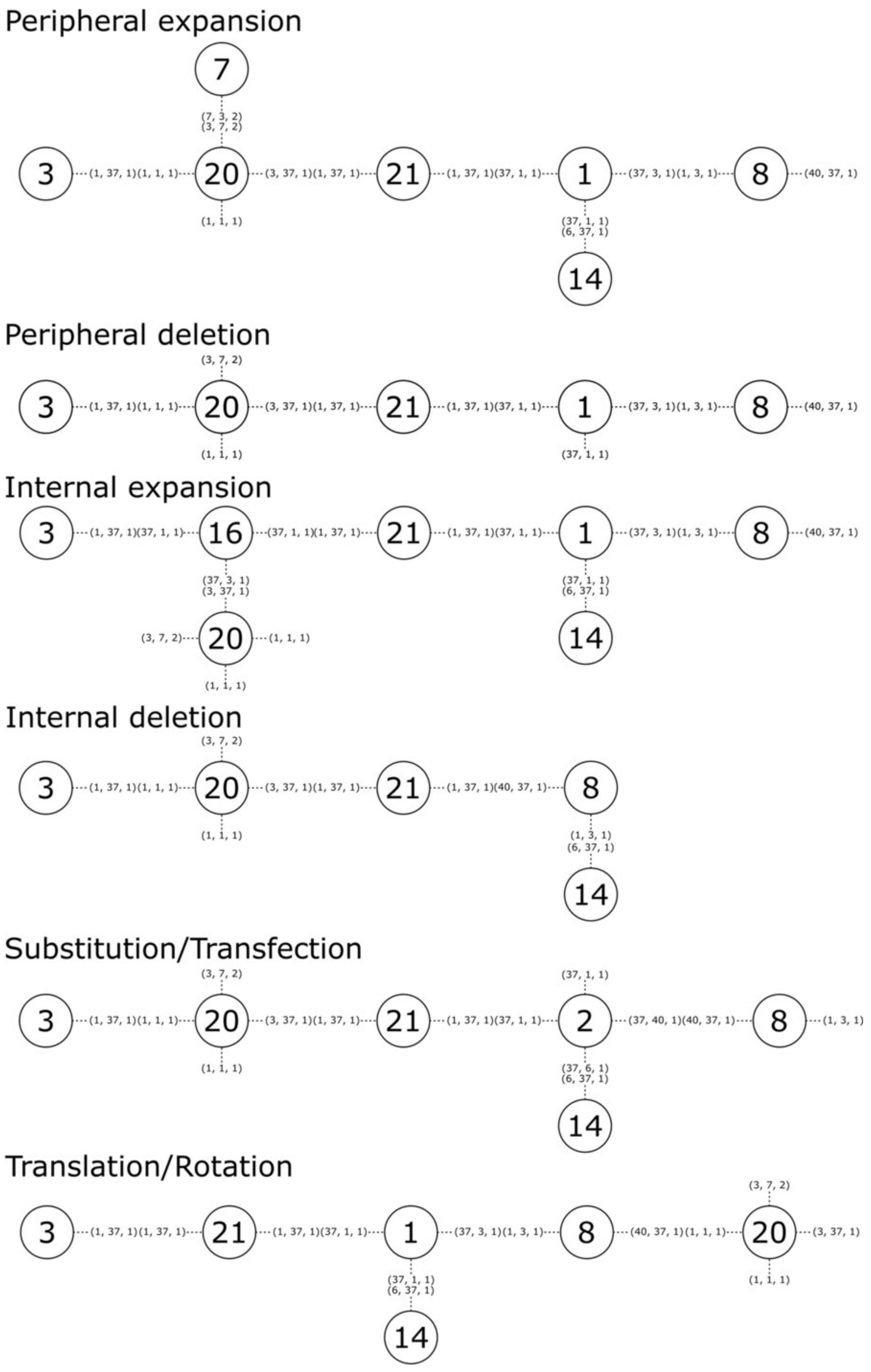


Figure 4.4. Illustration of the resulting chromosomes after applying each of the eight genetic operators to the chromosome given in Figure 4.3a.

Transfections derive their name from the corresponding biochemical technique of inserting genetic material into cells. Transfections are similar to substitutions in that they replace one fragment with another, with the difference being that the replacement fragments are sourced from the molecule population instead of the fragments database. Hence, they exploit the internal variability of the population, fulfilling a similar role to crossover operators in traditional genetic algorithms. We opted out of traditional crossover operators as none of the traditional approaches would have served us well. Subgraph exchange [12, 64, 100, 115, 116] (as described in section 3.3.8.1) without infringing upon the connection compatibility rules would have been challenging. Side chain exchange [81, 117, 118] requires the presence of a large common substructure, which is unlikely if the fragments are diverse and the number of fragments is large. While the transfection operator is less disruptive than a crossover operator, the unidirectional flow of genetic material in transfections is easier to implement, guarantees the success of the operation and reduces the time complexity from $O(n^2)$ to $O(n)$ compared to a bidirectional crossover.

Translations/rotations move a fragment from one position and orientation to another within the same molecule. They operate similar to a deletion and expansion in tandem. By inserting the fragment back in its starting position but with a different orientation it can effectively be rotated in place.

Lastly, for those scoring functions operating on 3D molecular structures, a stereochemistry flip operator is available. This operator chooses a random chiral atom or stereochemical double bond and inverts its stereochemistry.

4.3.4.1 *Connection rules enforcement*

LEADD’s genetic operators satisfy the connection compatibility rules by searching for fragments that can bond simultaneously to a given combination of neighbor fragments. Whether a specific query fragment fulfils the above condition can be expressed as a Maximum Bipartite Matching problem (MBPM). A bipartite graph is a graph with two separate vertex sets or “parts”, where edges involve one vertex of each set. Given a bipartite graph, a matching is a selection of edges such that no vertex is involved in more than one edge. In MBPM the goal is to find the matching of maximum cardinality, that is, the matching with the largest possible number of edges.

We construct the bipartite graph by placing the query fragment’s free connectors in one vertex set, and the fragments within the combination in the other vertex set (Figure 4.5). The edges between both vertex sets are drawn according to the lax connection compatibility rules (Figure 4.2B), with an edge representing that a connection is compatible with a fragment. This MBPM is then solved with a modified version of the Hopcroft-Karp algorithm [154]. The standard version of the algorithm is deterministic and always returns the same matching, even if multiple matchings with the same cardinality exist. By randomizing the order in which it iterates over vertices and edges it returns a random maximum cardinality matching instead. MBPM attempts to assign each of the neighboring fragments to one of the central fragment's connectors. If an arrangement is found where every neighbor fragment is bound to the central fragment without reusing any connectors, that is, the cardinality of the matching is equal to the number of neighbor fragments, the query fragment is compatible with said combination of fragments.

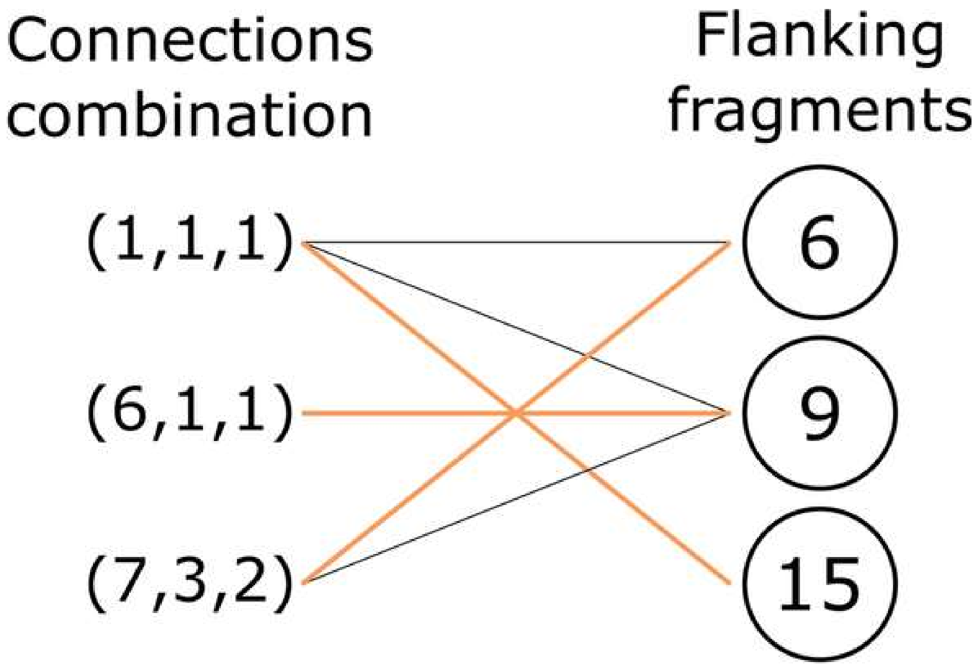


Figure 4.5. MBPM constructed to query whether a hypothetical fragment with a given set of connectors (left) is compatible with a combination of fragments (right). Black and orange edges represent compatibility relationships. The solution to the MBPM (i.e. the matching) is shown as the orange highlighted edges. Since the cardinality of the matching is equal to the number of flanking fragments our hypothetical fragment is compatible.

One can draw an analogy between finding suitable fragments and solving a jigsaw puzzle. If we equate fragments to be puzzle pieces, given a combination of flanking pieces the goal is to determine whether a query piece can connect to all of them simultaneously (Figure 4.6).

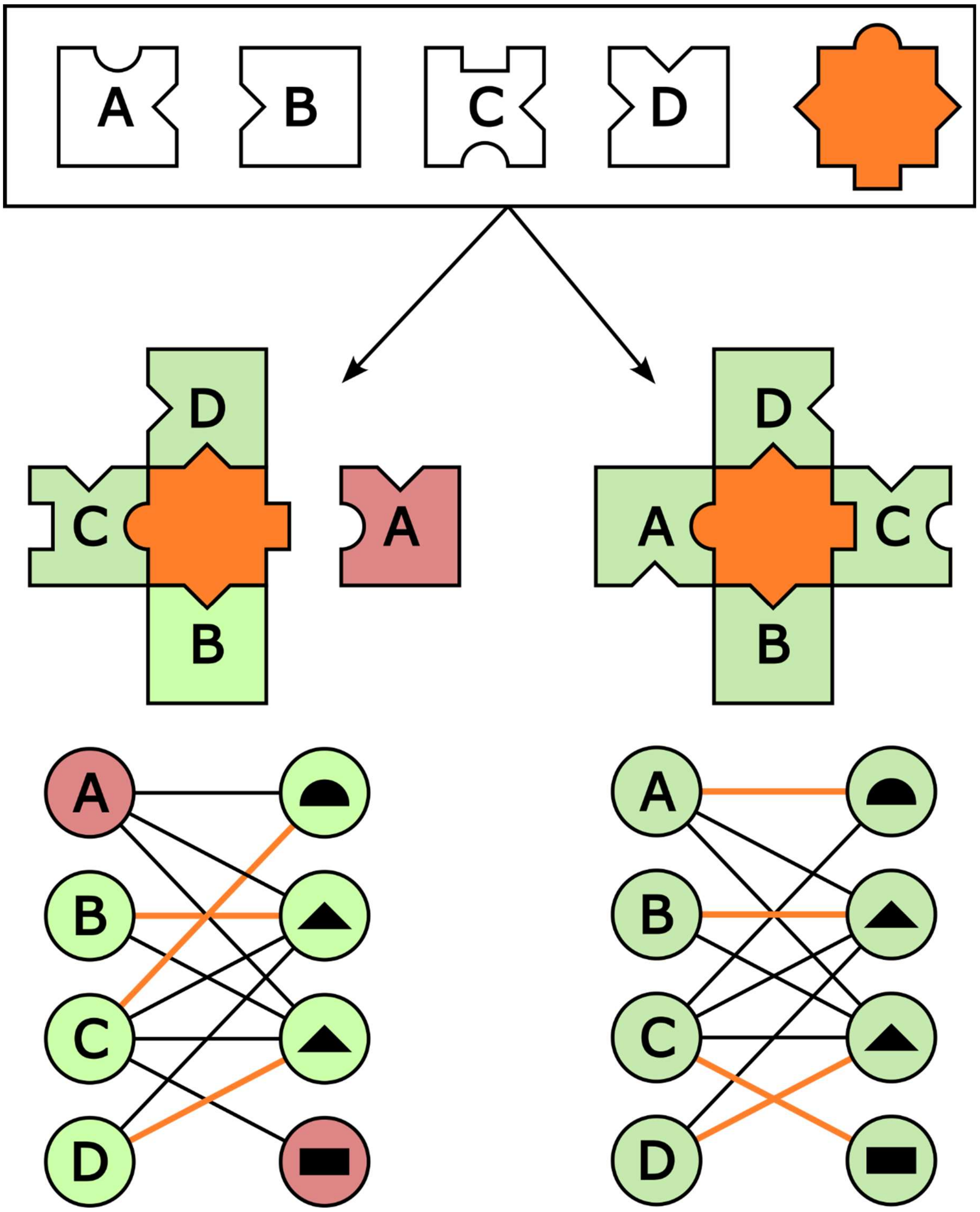
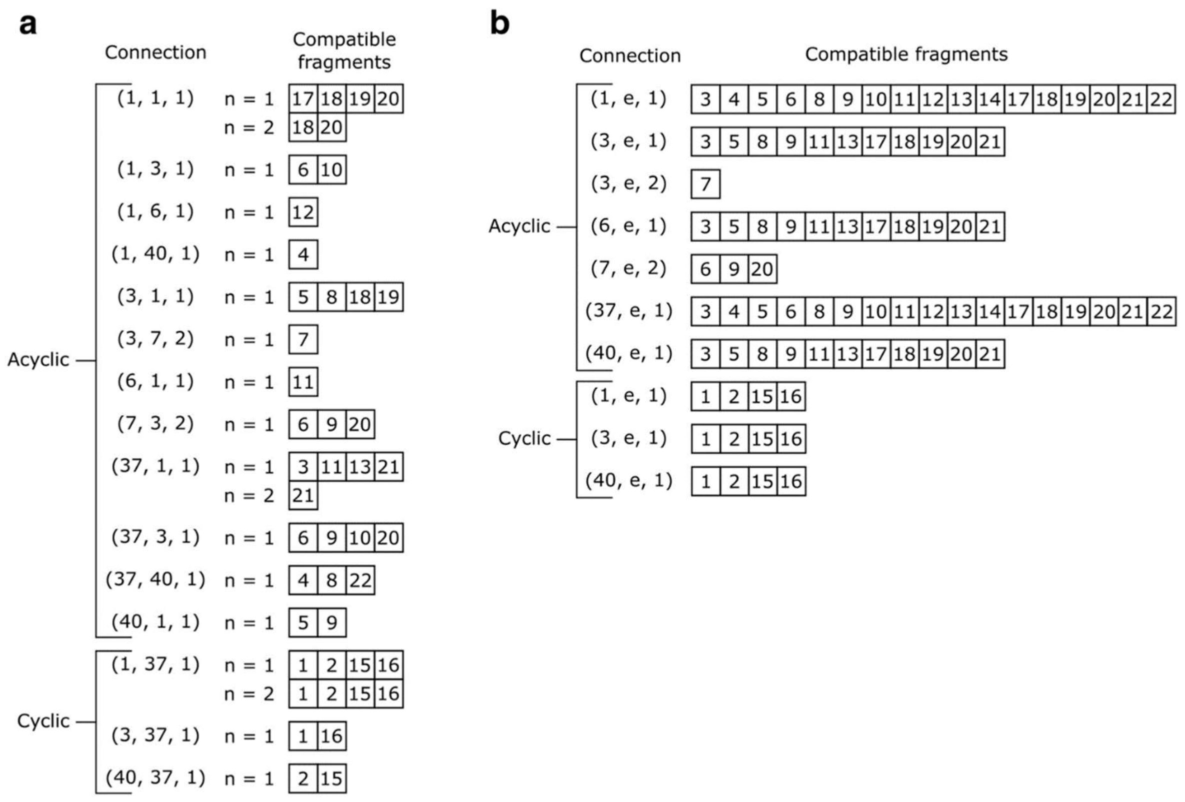


Figure 4.6. Illustration of how solving a jigsaw puzzle can be represented as a MBPM. The goal is to connect the orange central puzzle piece to all other flanking puzzle pieces (A-D). Each tab of the central piece must match a blank in one of the flanking pieces. Evidently not every piece arrangement allows for this (left), but some do (right). If an optimal arrangement exists we can find it through MBPM. We can verify its optimality by comparing the cardinality of the matching (number of highlighted edges) to the number of fragments.

To find all fragments that could bond to a combination of fragments one must interrogate all candidate fragments separately, which entails solving MBPM multiple times. This is computationally reasonable when the number of candidates is small, namely during internal deletions, transfections and translations/rotations. However, it becomes unreasonable for operations that sample fragments from the large fragments database, namely expansions and substitutions.

In those cases, we solve the problem through Multiple Set Intersection (MSI). Before **LEADD** is executed we precompute which fragments are compatible with each connection according to the strict connection compatibility rules and store their IDs in sets (Figure 4.7A). Since a connection combination may have repeats of the same connection, the compatible fragment IDs are stored stratified according to how many instances of compatible connections they have. If a fragment is compatible with *n* instances of a connection it is also compatible with *1* to *n−1* instances. To be able to control the number of ring fragments within the designed molecules, fragments are also stratified according to whether these are cyclic or acyclic.


a

	Connection		Compatible fragments
Acyclic	(1, 1, 1)	n = 1	17, 18, 19, 20
		n = 2	18, 20
	(1, 3, 1)	n = 1	6, 10
	(1, 6, 1)	n = 1	12
	(1, 40, 1)	n = 1	4
	(3, 1, 1)	n = 1	5, 8, 18, 19
	(3, 7, 2)	n = 1	7
	(6, 1, 1)	n = 1	11
	(7, 3, 2)	n = 1	6, 9, 20
	(37, 1, 1)	n = 1	3, 11, 13, 21
		n = 2	21
	(37, 3, 1)	n = 1	6, 9, 10, 20
	(37, 40, 1)	n = 1	4, 8, 22
	(40, 1, 1)	n = 1	5, 9
Cyclic	(1, 37, 1)	n = 1	1, 2, 15, 16
		n = 2	1, 2, 15, 16
	(3, 37, 1)	n = 1	1, 16
	(40, 37, 1)	n = 1	2, 15

b

	Connection	Compatible fragments
Acyclic	(1, e, 1)	3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 17, 18, 19, 20, 21, 22
	(3, e, 1)	3, 5, 8, 9, 11, 13, 17, 18, 19, 20, 21
	(3, e, 2)	7
	(6, e, 1)	3, 5, 8, 9, 11, 13, 17, 18, 19, 20, 21
	(7, e, 2)	6, 9, 20
	(37, e, 1)	3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 17, 18, 19, 20, 21, 22
	(40, e, 1)	3, 5, 8, 9, 11, 13, 17, 18, 19, 20, 21
Cyclic	(1, e, 1)	1, 2, 15, 16
	(3, e, 1)	1, 2, 15, 16
	(40, e, 1)	1, 2, 15, 16

Figure 4.7. Connection-fragment compatibilities of the fragments in Figure 4.1 according to (a) the strict compatibility rules and (b) lax compatibility rules, as described in Figure 4.2. Fragment weights are omitted for clarity purposes. Fragments are stratified according to their cyclicity, and in the case of the strict compatibility definition (a) also according to how many instances (n) of the connection the fragment has. In (b), "e" denotes any ending atom type. Note that in (a) higher strata are subsets of the lower strata, and that (a) is a subset of (b).

At runtime these arrays are loaded, and the list of fragments compatible with a combination of connections is calculated as the intersection of the fragment IDs compatible with each of its connections separately (Figure 4.8). Note that since fragments may have more than one free connector, if we wish to find fragments compatible with a combination of fragments, we must define all unique combinations of their free connectors and solve the MSI problem for each of them. The final result is the union of all resulting sets.

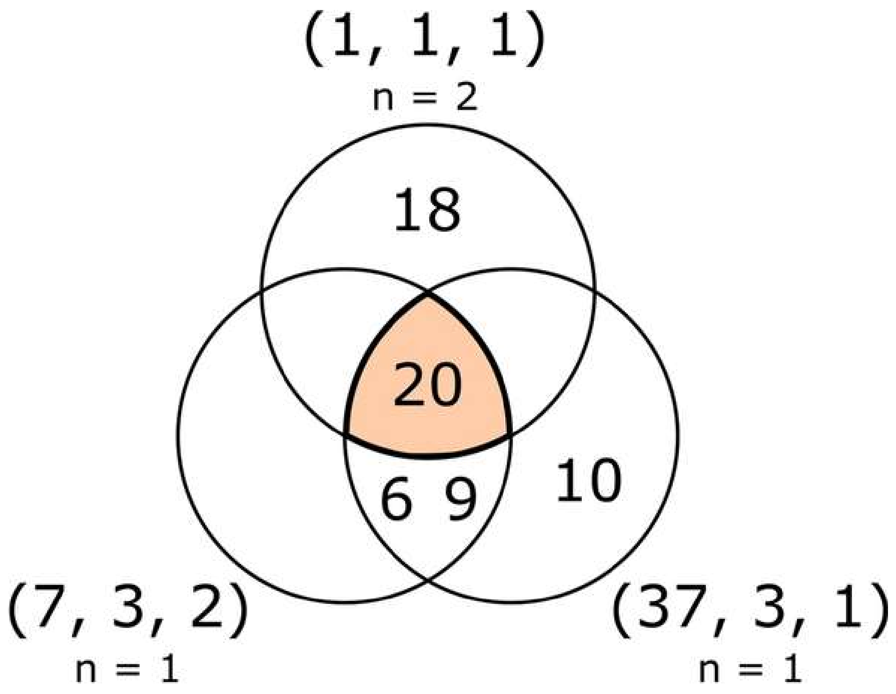


Figure 4.8. Venn diagram of the multiple intersection result for acyclic fragments compatible with the connections combination [(1,1,1), (1,1,1), (7,3,2), (37,3,1)], using the precalculated compatible fragments according to the strict compatibility definition (Figure 4.7A).

Pursuing the jigsaw puzzle analogy further, one could envision recording which puzzle pieces have certain connectors in a table. Given a combination of flanking pieces we can determine the connectors some central piece ought to have. This specification can then be used to search the table for suitable pieces (Figure 4.9).

The MSI connection-fragment compatibilities must be computed using the strict connection compatibility definition to ensure that the same connector does not contribute to a fragment showing up in more than one set of compatible fragments. Because of this, the MSI approach returns a subset of all fragments that would be deemed compatible according to the MBPM approach (Figure 4.7). Nonetheless, the final orientation of fragments retrieved with the MSI approach can still be determined through MBPM.

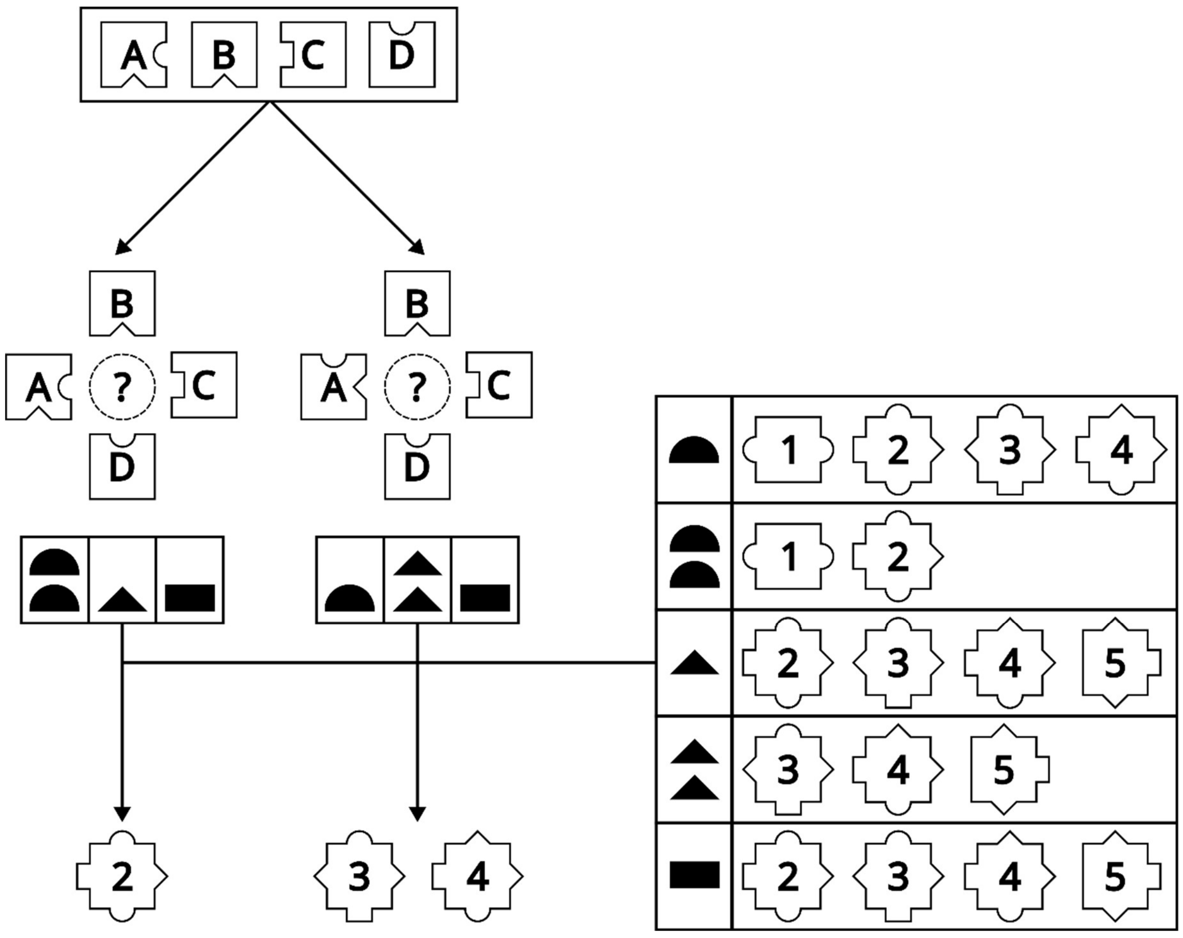


Figure 4.9. Illustration of how a jigsaw puzzle can be solved through MSI. Given a set of flanking puzzle pieces (A-D), the goal is to find a central piece that can connect to all of them simultaneously. Pieces B-D have a single connector, but A has two connectors. Accounting for the possibility that pieces may be rotated, we can define two combinations of connectors (left and right). Each of these combinations can be used as lookup key in a pre-computed table ordering candidate pieces according to their connectors to retrieve some compatible central pieces (left: 2, right: 3, 4). The final result is the union of pieces retrieved with each combination of connectors (2, 3, 4).

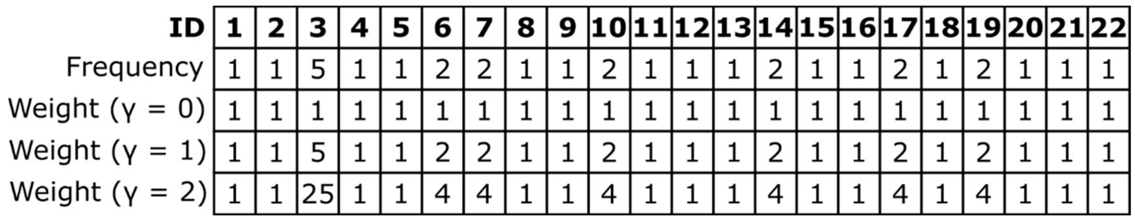
4.3.4.2 *Operation outcome sampling*

In the event that an operator finds multiple suitable operation outcomes a random one is chosen, typically through roulette wheel selection. For expansions, deletions and substitutions the weight W of a fragment F is calculated based on its frequency q in the database and its size *N*, in numbers of heavy atoms according to Equation 4.1.

Equation 4.1

$$W_F = q_F^{\gamma} \cdot N_F^{\lambda}$$

In Equation 4.1 exponents γ and λ are user parameters. γ determines how much the fragment selection should be guided by the fragment frequencies, with the default being $\gamma = 1$. If the user wishes true random fragment selection this can be done by setting $\gamma = 0$. λ is a size biasing term intended to be used when mixing fragments of different sizes. For efficiency reasons weights are precalculated and stored alongside the connection-fragment compatibilities (Figure 4.10).



ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Frequency	1	1	5	1	1	2	2	1	1	2	1	1	1	2	1	1	2	1	2	1	1	1
Weight (γ = 0)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Weight (γ = 1)	1	1	5	1	1	2	2	1	1	2	1	1	1	2	1	1	2	1	2	1	1	1
Weight (γ = 2)	1	1	25	1	1	4	4	1	1	4	1	1	1	4	1	1	4	1	4	1	1	1

Figure 4.10. Example of how the weights of the fragments in Figure 4.1 are calculated according to their database frequency and the γ exponent (λ = 0) (Equation 4.1). Note that in practice a single γ is chosen.

For transfections the weight is calculated following the same formula but with the score *S* of the fragment's owner molecule *R* as an additional variable term, with a corresponding user-specified exponent ζ signifying the transfection bias towards fragments contained in high scoring molecules (Equation 4.2).

Equation 4.2

$$W_F = q_F^{\gamma} \cdot N_F^{\lambda} \cdot S_F^{\zeta}$$

The translation/rotation and stereo flip operators select operation outcomes through uniform random sampling instead.

4.3.4.3 *Cyclicity control*

Fragment identity comprises both the molecular graph and connectors. Generally, the number of unique fragments increases with (1) the size of the fragments and (2) the atom type and connector diversity (Table 4.1). Differences in fragmentation procedure between acyclic and cyclic regions of source molecules can cause imbalances in the number of unique fragments, as well as their frequencies, which can lead to fragment sampling biases. Since cyclic fragments tend to outnumber their acyclic counterparts (Table 4.1), if fragments were sampled uniformly ($\gamma = 0$, Equation 4.1) it would be more likely to sample cyclic fragments. Conversely, under weighted sampling ($\gamma > 1$), and when defining acyclic fragments as subgraphs of $s > 0$, certain acyclic atoms are represented in more than one fragment. Since ring systems are not fragmented, this causes an overrepresentation of acyclic atoms in the fragment frequencies with respect to the cyclic ones. If these factors are not accounted for during fragment sampling, we risk designing either very rigid or very flexible and non-druglike molecules.

Table 4.1. Fragment database and connection compatibility statistics for the explored combinations of atom typing scheme, fragmentation scheme and MBPM compatibility stringency. [a] For dummy atom types the strict and lax compatibility definitions are equivalent since only one atom type exists. [b] According to the compatibility definition stringency used for MBPM (column 3).

Atom typing scheme	Acyclic region fragmentation scheme	MBPM compatibility stringency	Number of unique…				Average number of compatible…		
			atom types	connections	acyclic fragments	ring fragments	connections /connection (MBPM[b])	fragments /connection (strict)	fragments /connection (MBPM[b])
Dummy	Subgraph (s = 0)	Strict/Lax[a]	1	3	116	73287	1	37417	37417
Dummy	Subgraph (s ϵ [0 .. 2])	Strict/Lax[a]	1	3	3834	73287	1	39286.7	39286.7
Dummy	None	Strict/Lax[a]	1	2	95430	73287	1	104218	104218
MMFF	Subgraph (s = 0)	Strict	64	1316	4869	205637	1	436.8	436.8
MMFF	Subgraph (s = 0)	Lax	64	1316	4869	205637	708.2	436.8	180955
Morgan (r = 1)	Subgraph (s = 0)	Lax	14811	130494	139774	522517	1157.7	14.1	10127.3
Morgan (r = 1)	Subgraph (s = 0)	Strict	14811	130494	139774	522517	1	14.1	14.1
Morgan (r = 1)	Subgraph (s ϵ [0 .. 2])	Lax	14811	130494	937087	522517	1157.7	31.7	39008
Morgan (r = 1)	None	Lax	10472	62021	243964	522517	591.5	30.7	16367.3
Morgan (r = 2)	Subgraph (s = 0)	Lax	381252	1334292	799676	942568	223.9	3.2	845.2

To circumvent this issue the genetic operators with the capacity to modulate the number of ring atoms in a molecule (N_r), namely expansions, deletions, substitutions and transfections, decide whether and how N_r ought to be changed prior to selecting a suitable acyclic or cyclic fragment to do so, according to the current N_r.

How a genetic operator will modulate N_r is based on a pseudorandom number generator and the probabilities returned by up to two functions operating in tandem. The first function returns the probability of keeping the number of rings constant ($P^{=}$) based on the current N_r. It consists of a discrete function fit to the shape of a normal distribution's probability density function (*PDF*), and with its maximum scaled to an arbitrary user provided value (*M*). The equations of the normal distribution's PDF and discrete function are given in Equation 4.3 and Equation 4.4, respectively.

Equation 4.3

$$PDF(N_r) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}\left(\frac{N_r-\mu}{\sigma}\right)^2}$$

Equation 4.4

$$P^{=}(N_r) = \frac{PDF(N_r)}{\sum_{N_r=0}^{max(N_r)} PDF(N_r)} \cdot \frac{M}{PDF(\mu)}$$

Equation 4.5

$$P^{\neq}(N_r) = 1 - P^{=}(N_r)$$

The mean of the normal distribution (*μ*) describes the ideal N_r and its standard deviation (*σ*) the leniency in oscillating said number during evolution. Both parameters are user provided and ideally based on some notion of the desired N_r in a solution. Both *PDF(N_r)* and $P^{=}$ are maximum at *μ* and equal to the user's scaling target value *M*.

While for expansions and deletions this function suffices to decide how to modulate the number of rings, for substitutions and transfections, if in the preceding step it was decided to change N_r, a second function returns the probability of increasing it (P^{+}) (Equation 4.6). This function is a logistic function defined based on *μ* and *σ*.

Equation 4.6

$$P^{+}(N_r) = P^{\neq}(N_r) \cdot \frac{1}{1 + e^{0.341\sigma\cdot(N_r-\mu)}}$$

Equation 4.7

$$P^{-}(N_r) = P^{\neq}(N_r) - P^{+}(N_r)$$

The growth rate of the logistic function was empirically derived, and is set to be approximately the same as the normal distribution's "steepness", namely *0.682/2σ*, since in a normal distribution 68,2% of values are in *[μ – σ, μ + σ]*. Additionally, by setting the midpoint of the logistic function to *μ*, $P^{+}(\mu) = P^{-}(\mu)$. The edge case where $N_r = 0$ is treated by setting $P^{-} = 0$.

An example of the three aforementioned probability curves is shown in Figure 4.11.

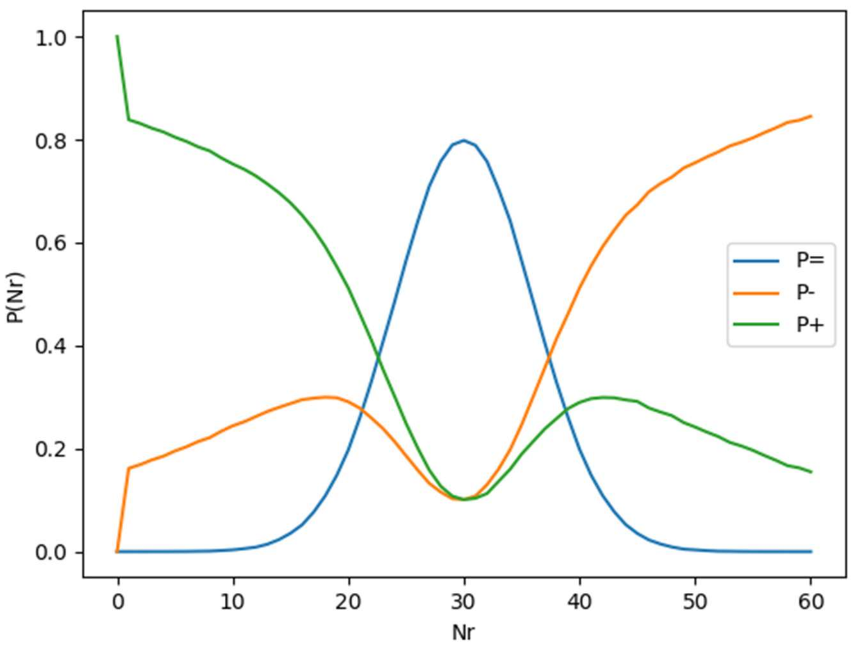


Figure 4.11. Probability distributions of keeping the number of ring atoms (N_r) constant ($P^=$), increasing it (P^+) or decreasing it (P^-) based on the current N_r, as described by Equation 4.4 - Equation 4.7. $\mu = 30$, $\sigma = 6$, $M = 0.8$.

4.3.5 Lamarckian evolution guidance

Given that the database fragment weights are static, so are the likelihoods of genetic operation outcomes, regardless of whether the same or similar operations proved beneficial or not in the past. In an attempt to improve the efficiency of the algorithm, as an extension, we conferred it with a certain ability to "learn" from the outcomes of previous genetic operations in hopes of increasing the likelihood of carrying out productive operations in the future. To this end, each connector within a molecule is endowed with a pair of arrays: one storing the IDs of compatible fragments F and one storing their corresponding weights W_F. The weights array is initialized to a copy of the database fragment weights (Figure 4.10), but it's free to change with each generation.

During evolution, a copy of a parent molecule P is subjected to a genetic operation, targeting some fragment V, to generate a child molecule C. The score S of C is compared to that of P (Equation 4.8). Scores are calculated by the scoring function we optimize for, with higher values being better.

Equation 4.8

$$\Delta S = S_C - S_P$$

Molecules keep track of which fragments were placed and/or removed from each connector during the operation. For each connector involved in the operation, based on the nature of the operation and its outcome (Table 4.2), the weights array of both the *P* and *C*'s connectors are modified according to Equation 4.9:

Equation 4.9

$$W_F = W_F \cdot (1 + g \cdot l \cdot Tc_{FV})$$

In Equation 4.9 *g* is the reinforcement sign, *l* is a user-specified reinforcement rate and Tc_{FV} is the Tanimoto topological similarity coefficient of fragments *F* and *V* according to ECFP4 fingerprints [34]. For performance reasons, all pairwise fragment similarity coefficients are precalculated and stored as a square symmetrical matrix in a HDF5 file [155].

Table 4.2. Learning rate sign of Equation 4.9 for bond creations (i.e. attaching a fragment to a connector) and destructions (i.e. deleting a fragment from a connector) based on the score change associated with the operation (Equation 4.8).

Operation	**ΔS**	**Learning rate sign (*g*)**
Bond creation	> 0	+ 1
	≤ 0	- 1
Bond destruction	> 0	- 1
	≤ 0	+ 1

Whether the change in weight is positive or negative (*g*) depends on the nature of the operator and the change in score (Table 4.2). **LEADD** maximizes strictly positive scores. The general principle is that if a newly placed fragment at a given connector increased the molecule's score (i.e. improved the score), the weights of similar fragments are increased, whereas if it stayed the same or decreased, the weights of similar fragments are decreased. The opposite paradigm is true for fragments being removed from a given connector.

This guided evolution serves two purposes. On one hand it can accelerate convergence by focusing the sampling on fragments that have been shown to be associated with good scores. On the other hand, since weights of similar fragments are decreased also when the score does not change, given enough time it could help the algorithm in escaping local fitness minima.

One could interpret a molecule's connectors' weights arrays as its reproductive behavior or its memory regarding which chemotypes at which positions are linked to better scores. Parents adapt their reproductive behavior to increase the likelihood of generating fit offspring based on the outcome of their previous reproductive events. Hence, the reproductive behavior is an acquired trait. This, coupled to the fact that the connector arrays are an integral part of the chromosome, and therefore inherited by the offspring, constitutes a Lamarckian evolutionary mechanism.

4.3.6 Evolutionary algorithm

Over the course of a number of generations (or until some convergence criterion is met) the molecules within the population are bred to generate offspring. The user can combine the following termination criteria: (1) top molecule score threshold, (2) maximum number of generations, and (3) maximum number of generations without improvements to the top molecule. Each generation a number of parent molecules is chosen to generate an equal number of child molecules. Parents reproduce asexually, and the same parent may reproduce more than once in the same generation. A copy of the chosen parent is subjected to a genetic operator to yield the child molecule. Molecules are chosen to be parents through fitness proportionate selection, with the weight of a molecule R being given by Equation 4.10. Note that the ζ parameter takes the same value as in Equation 4.2.

Equation 4.10

$$W_R = S_R^{\zeta}$$

Optionally, the user may enforce population topological diversity through means of an internal similarity filter. The topological similarity between two molecules is calculated as the Tanimoto coefficient between their ECFP4 fingerprint [34]. If the similarity of a child molecule to any of the current members of the population surpasses a given threshold, the child is discarded. Otherwise, it's added to the population.

The child molecules are scored, and a specified number of best scoring molecules within the population, including parents, is retained. If guided evolution is enabled the connector weights are adjusted based on the change in score caused by the operation. Lastly, the surviving molecules are fed to the next generation of the algorithm.

While the use of fragments and connection compatibility rules is meant to reduce the likelihood of designing synthetically unfeasible molecules, this may not be sufficient to achieve this goal. For users wishing to consider synthetic accessibility on a higher level a SAScore [124] filter and heuristic score modifier [109] are provided.

A flowchart of the algorithm can be found in Figure 4.12. Note it's very similar to the algorithm described in 3.3.8.1. The most important difference is that **LEADD** lacks the inner loop that previously ensured molecule correctness (Figure 3.7). This makes **LEADD** more computationally efficient.

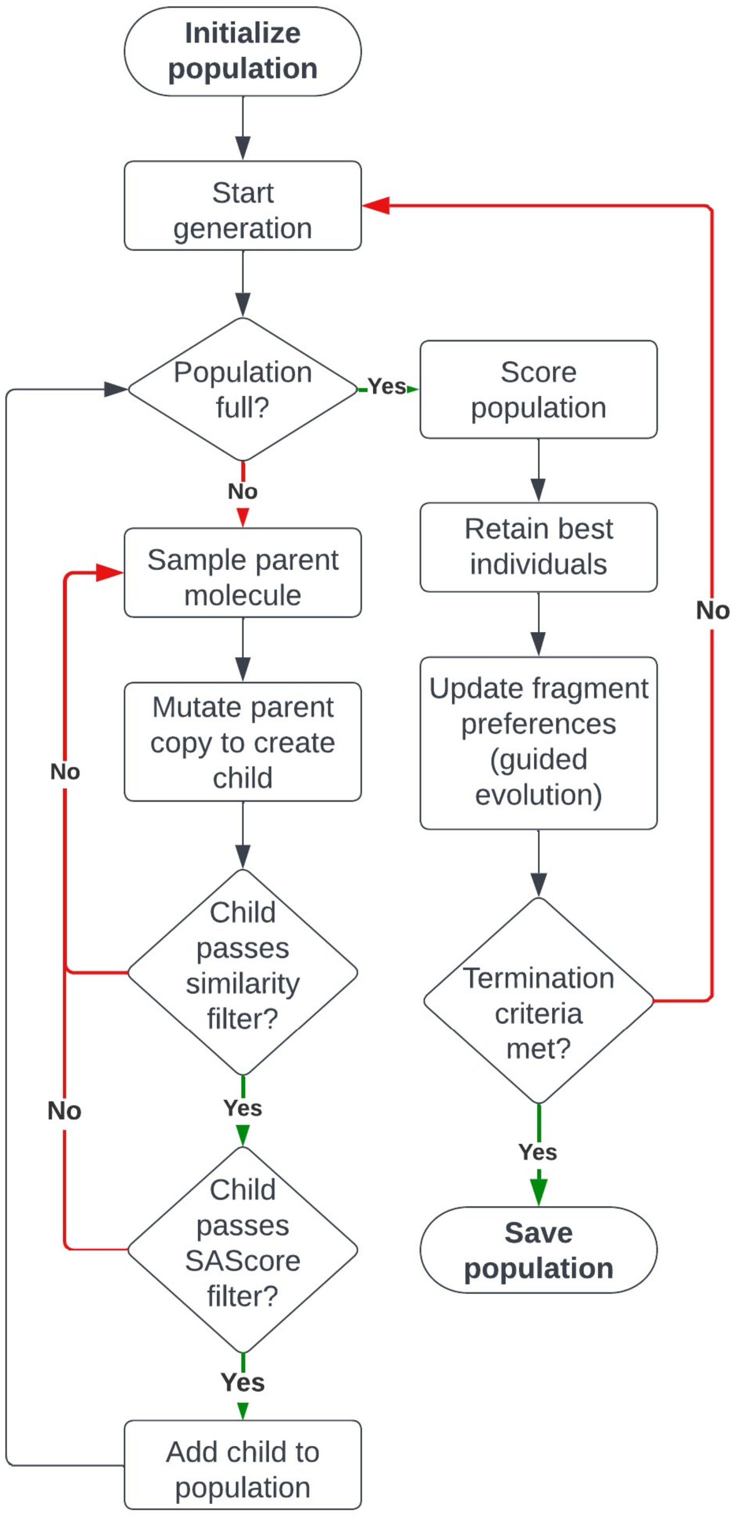


*Figure 4.12. Flowchart of **LEADD**'s main loop. Note that some of the flowchart's steps are optional, including the internal similarity and SAScore filters and the guided evolution.*

4.3.7 Benchmark

LEADD’s performance was evaluated with the goal-directed GuacaMol benchmark suites [108]. Specifically, we used the “trivial” and “version 2” (V2) benchmark suites (1.8.1). We chose to include the trivial benchmarks in our analysis because the majority of the V2 objective functions point towards topologies of known and synthetically feasible drugs. Hence, the objective functions implicitly provide some notions of drug-likeness, potentially occluding some SA issues.

For standardization purposes we used GuacaMol’s training set, which is a subset of ChEMBL [136], as fragmentation input. Fragment databases were created for each investigated combination of fragmentation and atom typing scheme (Table 4.1).

The benchmark suite was used to find a set of reasonable default parameters for **LEADD**. Given the large number of parameters an exhaustive parameter exploration was unfeasible. We resorted largely to a trial-and-error approach. Some parameters, including the population size and convergence criteria were fixed. Additionally, since **LEADD** requires a guess of the number of ring atoms in the ideal solution, where possible, we used the benchmark goals to set reasonable values for these parameters (Table 4.3). The rest of the parameters were sorted according to their perceived importance. For parameters assumed to be uncorrelated we tested multiple values for each one and fixed it to the value that yielded the best results. If this was not the case, we evaluated combinations of the correlated parameters in a multi-factorial design.

*Table 4.3. **LEADD**'s cyclicity control settings used during the GuacaMol benchmark. N_r stands for number of ring atoms.Standard deviations are half the means, and maximums are three times the means.*

GuacaMol benchmark suite	Benchmark name	Mean N_r	Standard deviation N_r	Maximum N_r
Trivial	logP (target: -1.0)	6	3	18
	logP (target: 8.0)	12	6	36
	TPSA (target: 150.0)	18	9	54
	CNS MPO	6	3	18
	QED	18	9	54
	C7H8N2O2	6	3	18
	Pioglitazone MPO	17	8.5	51
V2	Celecoxib rediscovery	17	8.5	51
	Troglitazone rediscovery	21	10.5	63
	Thiothixene rediscovery	20	10	60
	Aripiprazole similarity	22	11	66
	Albuterol similarity	6	3	18
	Mestranol similarity	17	8.5	51
	C11H24	6	3	18
	C9H10N2O2PF2Cl	6	3	18
	Median molecules 1	6	3	18
	Median molecules 2	23	11.5	69
	Osimertinib MPO	21	10.5	63
	Fexofenadine MPO	24	12	72
	Ranolazine MPO	18	9	54
	Perindopril MPO	9	4.5	27
	Amlodipine MPO	12	6	36
	Sitagliptin MPO	15	7.5	45
	Zaleplon MPO	15	7.5	45
	Valsartan SMARTS	17	8.5	51
	Deco Hop	20	10	60
	Scaffold Hop	20	10	60

Ten replicas were ran for each combination of settings. Benchmark scores and SAScores of designed molecules were taken as OP and SA metrics, respectively. ChEMBL [136] feature counts were used for SAScore calculations. For statistical analysis the results of all benchmarks were pooled per setting. Since OP was found to be distributed non-normally, differences in it were evaluated with non-parametric statistical tests: either the Wilcoxon-Mann–Whitney U-test [141] or the Kruskal–Wallis [140] / Schreirer-Ray-Hare [156] H-test followed by pairwise Conover-Iman tests [157] with Šidák correction [142]. SAScores were distributed normally and analyzed with t-tests or one- or two-way analysis of variance (ANOVA) with interaction followed by Tukey's Honestly Significant Differences test. $\alpha = 0.05$ was taken as significance level and family-wise error rate (FWER) for all tests. Most statistical tests and post hoc corrections were performed using the SciPy [143] and statsmodels [144] Python packages, respectively. The Conover-Iman and Schreirer-Ray-Hare tests were performed with the Scikit-learn Python [158] and rcompanion R packages instead [159].

LEADD's performance was compared to that of GB-GA [100], an atom- and graph-based genetic algorithm for molecular design which has previously been shown to be a powerful optimizer [62, 108], and a standard virtual screen of GuacaMol's training set using the benchmark's objective function. GB-GA's mutation rate was set to the default 0.01. Both algorithms used a population size of 100 and were granted a maximum of 10,000 generations. Evolution terminated prematurely after a number of generations without improvements in the population's scores: 1,000 for **LEADD** and 5 for GB-GA. We explored granting GB-GA 1,000 generations without improvement but found that its lack of convergence guards caused the population diversity, and ultimately the benchmark scores, to degrade during long runs.

4.4 Results and discussion

4.4.1 Base parameter exploration

LEADD was found to be quite robust to changes in most of its construction parameters, as different values did not influence its performance greatly. As an exception, **LEADD** was sensitive to the internal similarity threshold since it's the algorithm's main premature convergence guard (data not shown). **LEADD**'s default base parameters can be found in Table 4.4. Fragmentation parameters had larger effects on both OP and SA of designed molecules, and will be discussed in the coming sections.

*Table 4.4. Summary of **LEADD**'s default reconstruction settings. Some settings were condensed or omitted from this table. For a more detailed list of settings, as well as recommended value ranges, we refer readers to the software's documentation.*

Parameter name	Value
Fragment frequency exponent (γ)	1.0
Fragment size exponent (λ)	0.0
Molecule score exponent (ζ)	2.5
Peripheral expansion weight	4.0
Internal expansion weight	4.0
Peripheral deletion weight	4.0
Internal deletion weight	4.0
Substitution weight	56.0
Transfection weight	20.0
Translation weight	4.0
Stereo-flip weight	0.0
Randomize unspecified stereo	False
# seed molecules	100
# children per generation	100
# survivors per generation	100
Maximum child similarity	0.9
Maximum # generations	10,000
Maximum # generations stuck	1,000
SAScore filter	Disabled
SAScore heuristic	Disabled
Lamarckian evolution guidance	Disabled

4.4.2 Effect of atom typing scheme

One of the main questions we wanted to answer was if the knowledge-based atom compatibility rules aided the algorithm in designing SA molecules. To that end, we measured the SAScores of molecules designed using the MMFF and Morgan (r = 1 and r = 2) atom typing schemes. As a control, we included "dummy" atom types (i.e. all atoms have the same atom type), whereby all connections with the same bond order are compatible. All tests used single-atom acyclic fragments (s = 0). Molecules with lower SAScores are predicted to be easier to synthesize. Figure 4.13 shows that molecules designed with Morgan atom types, regardless of the radius, have lower SAScores than those designed with dummy or MMFF atom types. Differences between all other pairs of atom typing schemes were of little practical significance (Table 4.5). It's interesting to note that the mean SAScore values for Morgan atom types fall well below 4.5, which has been suggested as a cut-off for easy to synthesize molecules [125]. By contrast, the mean SAScore values for dummy and MMFF atom types are approximately 4.6.

Unfortunately, we also noted that Morgan atom types were associated with significantly lower OP compared to dummy and MMFF atom types (Figure 4.14). The differences between dummy and MMFF atom types and between Morgan atom types of different radii were not statistically significant (Table 4.6). The results for the Valsartan SMARTS benchmark are very poor, regardless of the chosen atom typing scheme. This poor performance permeates throughout this work, and can be explained by the associated scoring function. Said scoring function has a binary component demanding the presence of a specific substructure. Binary scoring functions respond abruptly to molecular changes, and do not provide fine enough feedback to the optimization algorithm.

Taken together these results suggest that the choice of atom typing scheme defines a trade-off between OP and SA. The chemical diversity of atomic environments is vast, and classifying them into a small number of atom types means that atom typing schemes are degenerate, much like the genetic code. The number of distinct atom types can be taken as an approximate measure of the scheme's degree of degeneracy. **LEADD** tries to replicate the molecular connectivity of molecules seen in a library of drug-like molecules, but if a very degenerate atom typing scheme mischaracterizes this connectivity the algorithm's ability to replicate it falters. In our fragment databases we recorded 64 MMFF, 14,811 Morgan (r = 1) and 381,252 Morgan (r = 2) atom types (Table 4.1). Unique Morgan atom types greatly outnumber their MMFF counterparts, explaining the better SA associated with them.

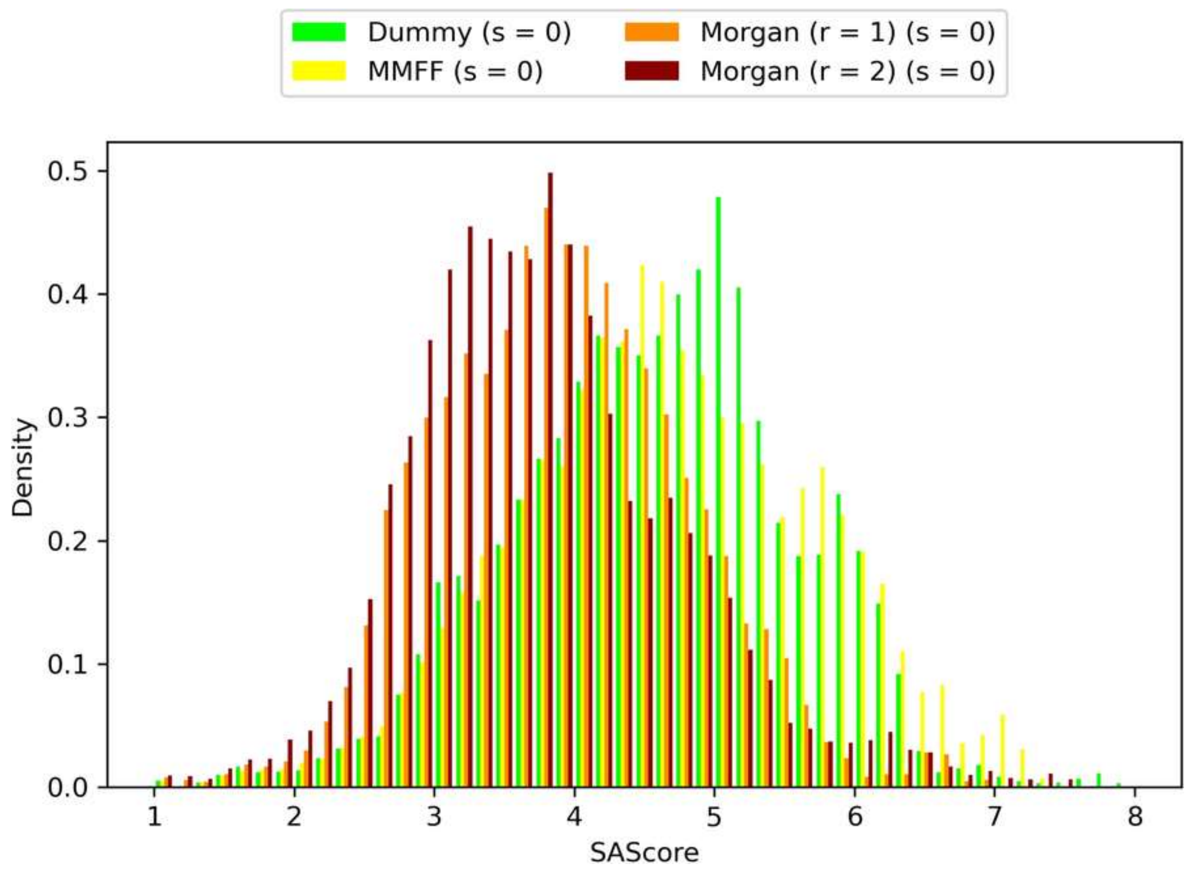


Figure 4.13. Comparison of designed molecules' SAScore distributions using different atom typing schemes. Includes molecules of all benchmarks and replicas. Molecules with lower SAScores are predicted to be easier to synthesize.

Table 4.5. Multiple comparisons of SAScore means using different atom typing schemes with Tukey's HSD post-hoc test (FWER = 0.05). The test was preceded by a one-way ANOVA (F = 5675.82, p < 0.001).

Group 1	**Group 2**	$\overline{SAScore}_{Group2} - \overline{SAScore}_{Group1}$	**Adjusted p-value**
Dummy	MMFF	0.0636	< 0.001
Dummy	Morgan (r = 1)	-0.7070	< 0.001
Dummy	Morgan (r = 2)	-0.817	< 0.001
MMFF	Morgan (r = 1)	-0.7706	< 0.001
MMFF	Morgan (r = 2)	-0.8806	< 0.001
Morgan (r = 1)	Morgan (r = 2)	-0.1101	< 0.001

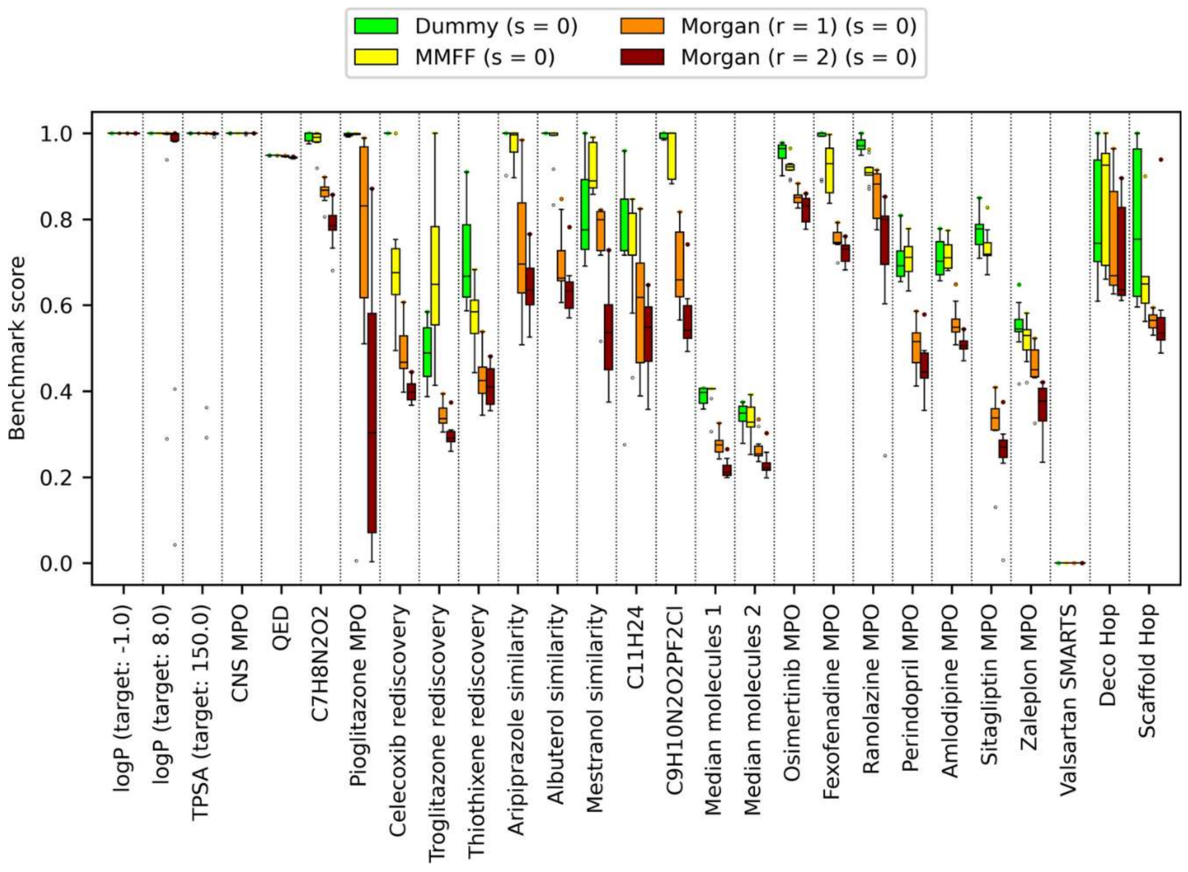


*Figure 4.14. **LEADD** optimization power comparison between atom typing schemes. Benchmark scores range between 0 and 1, with higher scores being better. Boxes represent interquartile ranges (IQR), the black line within them medians and the whiskers Q ± 1.5IQR. Data beyond the whiskers are considered outliers and represented as dots. Colored dots represent maximum benchmark scores.*

Table 4.6. Multiple comparisons of benchmark score distributions´ stochastic dominances using different atom typing schemes with Conover-Iman´s post-hoc test with Šidák correction (FWER = 0.05). The test was preceded by a Kruskal-Wallis test (H = 149.90, p < 0.001).

Group 1	**Group 2**	$\widetilde{Score}_{Group2} - \widetilde{Score}_{Group1}$	**Adjusted p-value**
Dummy	MMFF	-0.066	0.189
Dummy	Morgan (r = 1)	-0.302	< 0.001
Dummy	Morgan (r = 2)	-0.384	< 0.001
MMFF	Morgan (r = 1)	-0.236	< 0.001
MMFF	Morgan (r = 2)	-0.318	< 0.001
Morgan (r = 1)	Morgan (r = 2)	-0.082	0.099

The atom typing scheme's degree of degeneracy also defines the observed OP-SA trade-off. **LEADD** considers two atom types to be compatible, and therefore suitable for bonding, if they have been observed bonded in reference molecules at least once. Given the same set of reference molecules, the probability of observing any specific pair of atom types bonded is larger when the number of distinct atom types is small. Consequently, the more degenerate an atom typing scheme, the more promiscuous its atom types, in the sense that atom types will be deemed compatible with a larger number of other atom types. Ultimately, this also affects the number of fragments that are compatible with each connection. In the case of MMFF atom types, 85.96% of all fragments are compatible with the average connection according to the lax compatibility definition. This number drops to 1.53% and 0.05% for Morgan (r = 1) and Morgan (r = 2) atom types, respectively. Even more dramatic differences are observed when considering the strict compatibility definition (Table 4.1). This highlights that atom type promiscuity enables the algorithm to access a larger number of states (i.e. molecules) from the current state. In other words, promiscuous atom types are associated with a dense chemical transition graph (Figure 3.1). This may aid the algorithm in escaping local fitness minima and explain the associated greater OP.

Out of the tested atom typing schemes, we believe that for most use cases Morgan (r = 1) atom types represent the best OP-SA compromise. Other compromises of interest may be achievable with alternative atom typing schemes. **LEADD** can be readily expanded to use other atom typing schemes. For instance, one could collapse Morgan atom types into a smaller number of atom types with some type of hashing function. However, as this would inevitably cause collisions, the hashing function would need to be locality sensitive to avoid merging completely unrelated atom types. An alternative approach might be to cluster atomic environments and use cluster assignments as atom types. This approach could allow fine control over the OP-SA trade-off by modulating the number of clusters. We would like to remark however that the number of unique atom types is only a good metric for atom typing degeneracy when atomic environments are distributed uniformly across atom types. This is likely to be the case for Morgan atom types since they are calculated using hashing functions, which are designed to distribute inputs uniformly over an integer range, but may not be the case for other schemes. Instead, it would be more appropriate to use metrics that measure the information content of atom types (i.e. within atom type atomic environment similarities).

4.4.3 Implications of compatibility binarization

LEADD's approach to find suitable fragments for genetic operators requires that connection compatibility be expressed as a binary property. However, it may be argued that connection pairs are on a compatibility spectrum based on the observed frequency of said pair: if a pairing is observed thousands of times it's more compatible than if it's observed just once, yet they are deemed equally compatible. Consequently, infrequent connections may misrepresent molecular connectivity. We regularly observed large disparities among compatible connection pairing frequencies and wanted to measure the extent to which this is detrimental to the SA of designed molecules. By default the MBPM approach uses the lax compatibility definition, but this may be changed to the strict definition. Under the strict compatibility definition each connection is compatible with exactly one other connection, eliminating compatible connection pairing frequency imbalances. We found no practically significant differences in SAScore when using the strict compatibility definition for MBPM as opposed to the lax one (Figure 4.15). Considering that a fragment's connectivity is part of its identity, infrequent connections are contained to infrequent fragments. Since **LEADD** samples fragments with a probability proportional to their frequency we hypothesize that, while the binarization of connection compatibility does misrepresent the molecular connectivity of the reference library, this rarely manifests itself in designed molecules.

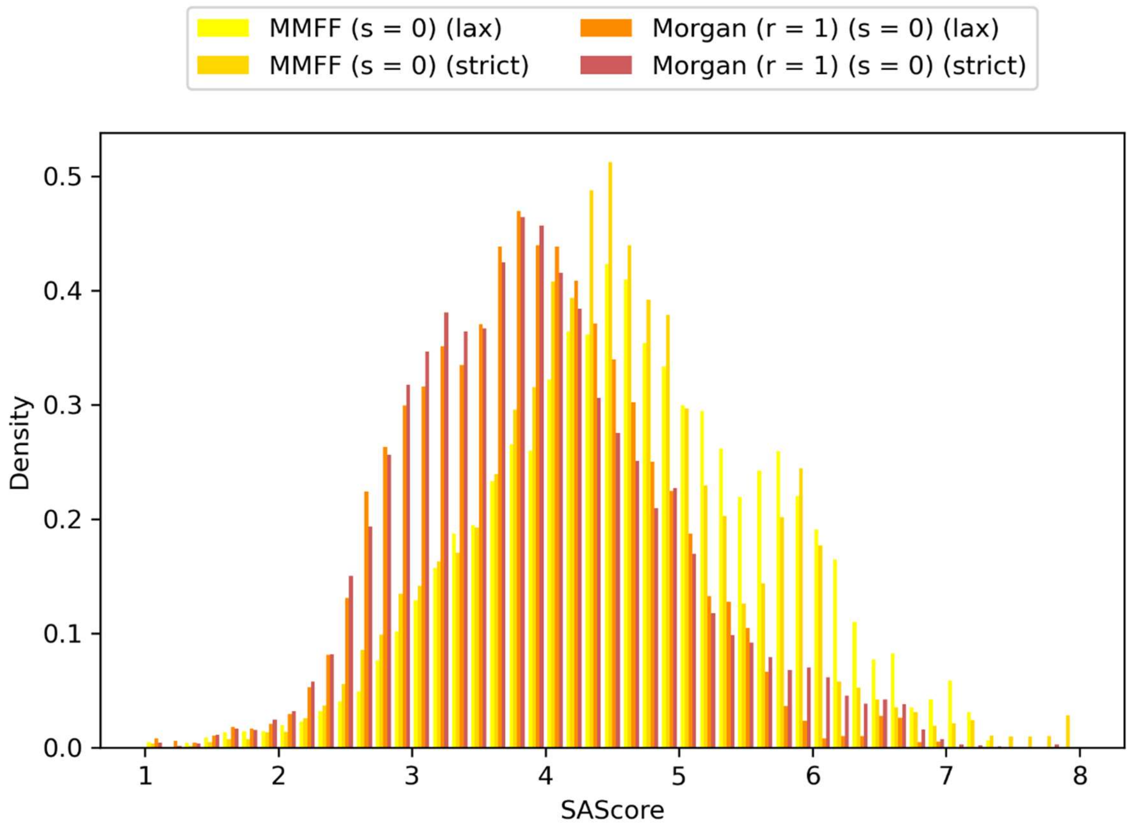


Figure 4.15. Comparison of designed molecules' SAScore distributions using different MBPMB connection compatibility stringencies. Includes molecules of all benchmarks and replicas. Molecules with lower SAScores are predicted to be easier to synthesize.

4.4.4 Effect of fragmentation scheme

The atom typing scheme degeneracy, the binarization of connection compatibility, and other factors such as connection compatibility being expressed only as pairwise relationships, all contribute towards **LEADD**'s description of molecular connectivity being imperfect. Each bond created by the algorithm has a probability of being non-drug-like. While we have discussed approaches to decrease this probability, an alternative approach to improve the drug-likeness of designed molecules is to reduce the number of bonds created by the algorithm. This can be achieved using larger fragments. To prove this we ran the benchmark using different types of acyclic fragments: single-atom fragments (s = 0), fragments with 0 to 2 bonds (s ϵ [0 .. 2]) and whole side chains and linkers resulting from the deletion of ring systems. In general, the SAScores of molecules designed using larger fragments were lower than those designed using smaller fragments (Figure 4.16). While the SAScore differences between s = 0 and s ϵ [0 .. 2] were almost negligible, using monolithic acyclic fragments did lead to substantial improvements in SAScore (Table 4.7, Table 4.8). It's interesting to note that the observed improvements in SAScore were larger for dummy atom types than for Morgan atom types, highlighting that the bonds created when using Morgan atom types are more drug-like.

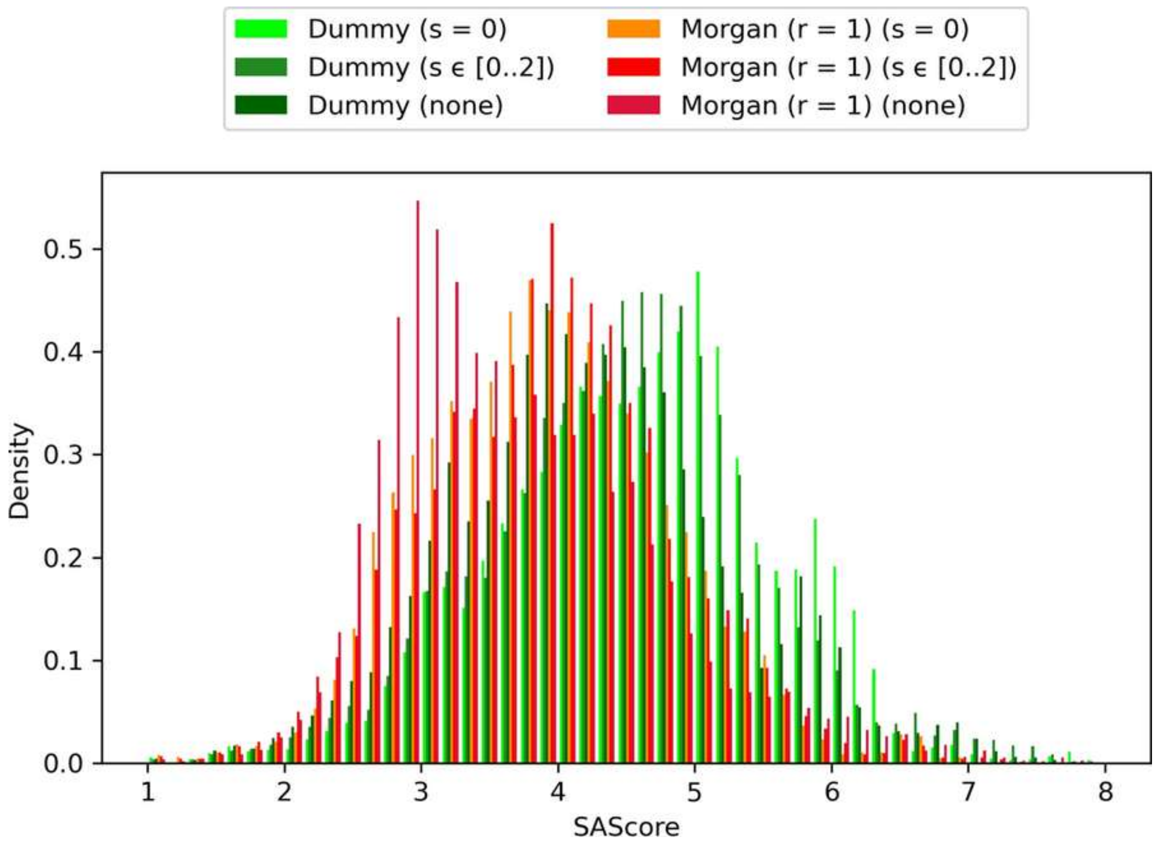


Figure 4.16. Comparison of designed molecules' SAScore distributions using different atom typing schemes. Includes molecules of all benchmarks and replicas. Molecules with lower SAScores are predicted to be easier to synthesize.

Table 4.7. Two-way ANOVA on the effect of atom typing scheme, fragmentation scheme and their interaction on the SAScore of designed molecules.

Source of variation	**df**	**Sum Sq**	**F**	**p-value**
Atom typing	1	15,205.18	16,055.82	< 0.001
Fragmentation	2	1,879.06	992.09	< 0.001
Interaction	2	119.08	62.87	< 0.001
Residual	154,720	146,522.90		

Table 4.8. Multiple comparisons of SAScore means using different combinations of atom typing and fragmentation schemes with Tukey's HSD post-hoc test (FWER = 0.05).

Group 1	**Group 2**	$\overline{SAScore}_{Group2} - \overline{SAScore}_{Group1}$	**Adjusted p-value**
Dummy (s = 0)	Dummy (s ϵ [0 .. 2])	-0.1030	< 0.001
Dummy (s = 0)	Dummy (none)	-0.3111	< 0.001
Dummy (s = 0)	Morgan (r = 1) (s = 0)	-0.7070	< 0.001
Dummy (s = 0)	Morgan (r = 1) (s ϵ [0 .. 2])	-0.6913	< 0.001
Dummy (s = 0)	Morgan (r = 1) (none)	-0.9016	< 0.001
Dummy (s ϵ [0 .. 2])	Dummy (none)	-0.2082	< 0.001
Dummy (s ϵ [0 .. 2])	Morgan (r = 1) (s = 0)	-0.6040	< 0.001
Dummy (s ϵ [0 .. 2])	Morgan (r = 1) (s ϵ [0 .. 2])	-0.5884	< 0.001
Dummy (s ϵ [0 .. 2])	Morgan (r = 1) (none)	-0.7987	< 0.001
Dummy (none)	Morgan (r = 1) (s = 0)	-0.3959	< 0.001
Dummy (none)	Morgan (r = 1) (s ϵ [0 .. 2])	-0.3802	< 0.001
Dummy(none)	Morgan (r = 1) (none)	-0.5905	< 0.001
Morgan (r = 1) (s = 0)	Morgan (r = 1) (s ϵ [0 .. 2])	0.0157	0.4443
Morgan (r = 1) (s = 0)	Morgan (r = 1) (none)	-0.1946	< 0.001
Morgan (r = 1) (s ϵ [0 .. 2])	Morgan (r = 1) (none)	-0.2103	< 0.001

The use of larger fragments did not affect **LEADD**'s OP when using dummy atom types. However, we did observe significant improvements in OP when using large fragments coupled with Morgan (r = 1) atom types (Figure 4.17, Table 4.9, Table 4.10). Genetic operations using larger fragments are associated with bigger step sizes in chemical space, which allows the algorithm to escape local fitness minima. Because the number of chemical states accessible from a given state is much smaller when using Morgan atom types as compared to dummy atom types, the probability of getting stuck in local fitness minima is larger in the former case. This explains why a bigger step size is beneficial for Morgan, but not dummy atom types. It's worth noting that the step size associated with larger fragments is not longer solely because of the bigger number of atoms per fragment, but also due to the greater degree of branching in larger fragments. While we implemented internal operators that attempt to mitigate this, there still is a risk that the algorithm may design certain highly branched topologies that are difficult to modify with genetic operators without unwinding the entire stack of operations. Since large fragments can capture branched motifs as a single unit, the risk of this happening is reduced. Future algorithms could improve upon this by implementing operators that target entire sections or branches of the meta-graph instead of a single vertex.

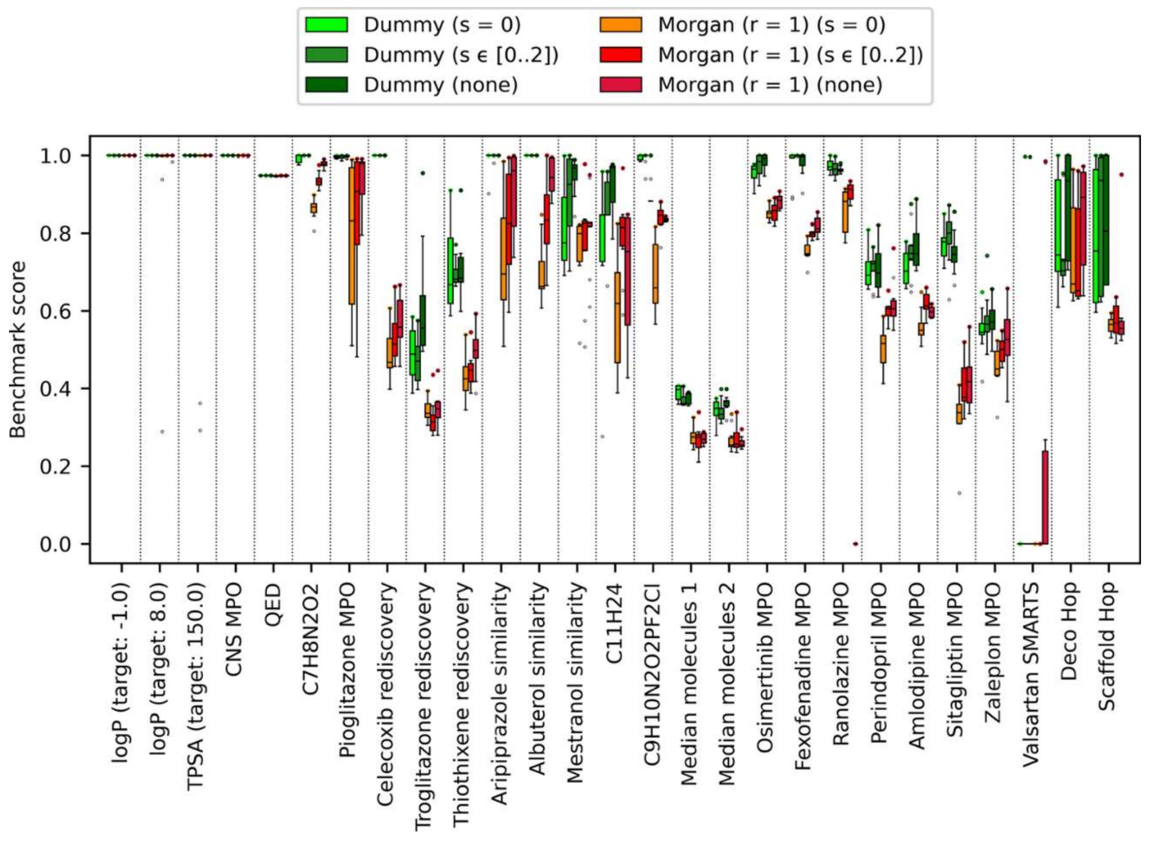


*Figure 4.17. **LEADD** optimization power comparison between different combinations of atom typing and fragmentation schemes. Boxes represent interquartile ranges (IQR), the black line within them medians and the whiskers Q ± 1.5IQR. Data beyond the whiskers are considered outliers and represented as dots. Colored dots represent maximum benchmark scores.*

Table 4.9. Scheirer-Ray-Hare test on the effect of atom typing scheme, fragmentation scheme and their interaction on the GuacaMol benchmark scores.

Source of variation	df	Sum Sq	H	p-value
Atom typing	1	36,511,969	169.401	< 0.001
Fragmentation	2	1,854,912	8.606	0.014
Interaction	2	439,082	2.037	0.361
Residual	1,604	307,991,950		

Table 4.10. Multiple comparisons of benchmark score distributions' stochastic dominances using different combinations of atom typing and fragmentation schemes with Conover-Iman´s post-hoc test with Šidák correction (FWER = 0.05).

Group 1	**Group 2**	$\widetilde{\boldsymbol{Score}}_{\boldsymbol{Group2}} - \widetilde{\boldsymbol{Score}}_{\boldsymbol{Group1}}$	**Adjusted p-value**
Dummy (s = 0)	Dummy (s ϵ [0 .. 2])	0.006	0.949
Dummy (s = 0)	Dummy (none)	0.012	0.900
Dummy (s = 0)	Morgan (r = 1) (s = 0)	-0.302	< 0.001
Dummy (s = 0)	Morgan (r = 1) (s ϵ [0 .. 2])	-0.176	< 0.001
Dummy (s = 0)	Morgan (r = 1) (none)	-0.122	< 0.001
Dummy (s ϵ [0 .. 2])	Dummy (none)	0.006	0.998
Dummy (s ϵ [0 .. 2])	Morgan (r = 1) (s = 0)	-0.308	< 0.001
Dummy (s ϵ [0 .. 2])	Morgan (r = 1) (s ϵ [0 .. 2])	-0.182	< 0.001
Dummy (s ϵ [0 .. 2])	Morgan (r = 1) (none)	-0.128	< 0.001
Dummy (none)	Morgan (r = 1) (s = 0)	-0.315	< 0.001
Dummy (none)	Morgan (r = 1) (s ϵ [0 .. 2])	-0.188	< 0.001
Dummy (none)	Morgan (r = 1) (none)	-0.134	< 0.001
Morgan (r = 1) (s = 0)	Morgan (r = 1) (s ϵ [0 .. 2])	0.127	0.185
Morgan (r = 1) (s = 0)	Morgan (r = 1) (none)	0.180	0.015
Morgan (r = 1) (s ϵ [0 .. 2])	Morgan (r = 1) (none)	0.053	0.932

Observant readers will have noticed the abrupt drop in the Ranolazine MPO benchmark score when using Morgan atom types coupled with monolithic fragments (i.e. Morgan (r = 1) (none)). Said benchmark provides a molecule starting point, which must be converted to a meta-graph through fragmentation. This process is only possible when the connections resulting from the fragmentation are also present in the database and have associated compatibility information. This turns out to not be the case for the monolithic fragment database. As such, the benchmark fails to run and receives a null score.

Given that larger fragments improve SA and either increase OP or do not affect it, it is tempting to conclude that the use of large fragments is always preferable. However, it should be noted that the larger step sizes associated with big fragments also carry the risk of “jumping” over good solutions. This can be partially overcome by mixing fragments of different sizes (e.g. $s \in [0..\ 2]$). A more pressing issue is that the use of large fragments requires a very extensive and diverse library of fragments to adequately represent chemical space. Besides dictating greater amounts of memory to store the pre-calculated compatible fragments, as the number of fragments grows so does the size of the search space, and with it the number of operations and generations necessary to adequately explore it. For Morgan atom types, we believe that the improved SA and OP tied to monolithic fragments justify their use. However, for dummy atom types we think that the minor SA improvements are not sufficient justification.

4.4.5 Handling fragment numerosity

A large number of fragments also poses the question of how to prioritize fragments to explore chemical space efficiently. We opted to use the fragments’ frequencies in drug-like matter as biasing weights to determine the outcomes of genetic operations. In an attempt to improve upon this, we also implemented a Lamarckian evolutionary mechanism that biases the exploration towards certain areas of the search space based on the outcomes of previous operations. A similar concept was explored in the particle swarm optimizer Colibree [98], where each molecule has preferences towards certain fragments, encoded as a floating point number array. In Colibree these preferences apply to the entire molecule, which is computationally more efficient and enables straightforward communication of preferences among molecules within the swarm, but lacks the spatial resolution that one would desire when working with structure-based scoring functions. Our Lamarckian evolutionary mechanism attempts to improve on this by assigning fragment preferences to connectors instead. Unfortunately, with the explored settings, the Lamarckian guided evolution mechanism failed to significantly improve the optimization power of the algorithm (data not shown). One possible explanation for the shortcomings of the approach is that, given the large number of fragments (10^5–10^6 compared to the 7,196 of Colibree), the number of generations for which **LEADD** runs (i.e. 1,000–10,000) is insufficient to resolve connector-fragment preferences. The impermanence of connectors may exacerbate the problem. When a fragment is deleted or substituted the knowledge accumulated in its connector arrays is erased, effectively resetting the progress of the

Lamarckian evolution. A potential solution could be mapping fragment preferences to points in space instead, which also would allow molecules to share their preferences among each other. However, the observed slower runtimes and larger memory footprints discourage us from exploring this approach further.

4.4.6 Comparison of SA improvement approaches

LEADD also ships with more traditional means of improving the SA of designed molecules, namely a simple filter that deletes molecules with SAScores above a given threshold and a SAScore-based heuristic score modifier that biases the objective function towards molecules with lower SAScores, as described by Gao and Coley [109]. As a reminder, the SAScore is a composite metric based on (a) how much the molecular connectivity of a molecule resembles that of reference drug-like molecules (i.e. FeatureScore) and (b) the number of synthetic nuisances within that molecule, for example stereo centers, spiro-, bridged- and macro-cycles (i.e. ComplexityPenalty). Because the atom type approach to increase SA only tries to improve the FeatureScore it can be of interest to combine it with the SAScore filter or heuristic. We were interested in comparing how these different approaches to increase SA fare on their own. The parameters for the SAScore filter (SAScore ≤ 4.5) and heuristic ($\mu = 2.23$, $\sigma = 0.65$) were taken from the literature, where they have been described as effective means to design SA molecules [109, 125]. Our results confirm that all approaches can be used to design more SA molecules (Figure 4.18, Table 4.11) and that, with the exception of the SAScore filter, this was accompanied by a significant loss of optimization power (Figure 4.19, Table 4.12). There appears to be an inverse correlation between SA and OP, and the observed OP-SA compromises seem to define a FeatureScore Pareto front (Figure 4.20). However, it should be noted that each approach has a different SA target. We did not manage to find SAScore filter and heuristic parameters that replicate the SAScore distribution of Morgan ($r = 1$) atom types. Hence, which approach provides the best OP-SA trade-off, if any, is inconclusive.

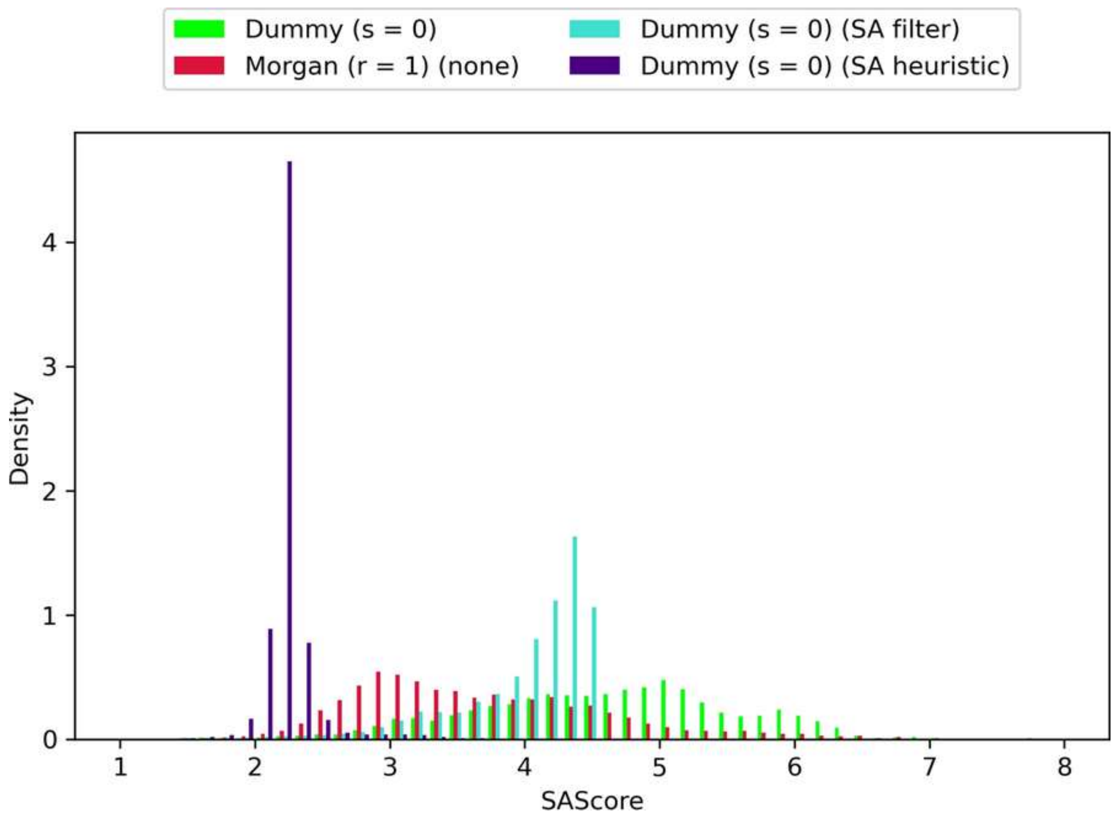


Figure 4.18. Comparison of designed molecules' SAScore distributions using different SA optimization strategies. Includes molecules of all benchmarks and replicas.

Table 4.11. Multiple comparisons of SAScore means using different approaches to improve SA with Tukey's HSD post-hoc test (FWER = 0.05). The test was preceded by a one-way ANOVA (F = 45720.82, p < 0.001).

Group 1	**Group 2**	$\overline{SAScore}_{Group2} - \overline{SAScore}_{Group1}$	**Adjusted p-value**
Dummy	Morgan (r = 1)	-0.9016	< 0.001
Dummy	Dummy (SAScore filter)	-0.5988	< 0.001
Dummy	Dummy (SAScore heuristic)	-2.3578	< 0.001
Morgan (r = 1)	Dummy (SAScore filter)	0.3028	< 0.001
Morgan (r = 1)	Dummy (SAScore heuristic)	-1.4561	< 0.001
Dummy (SAScore filter)	Dummy (SAScore heuristic)	-1.7590	< 0.001

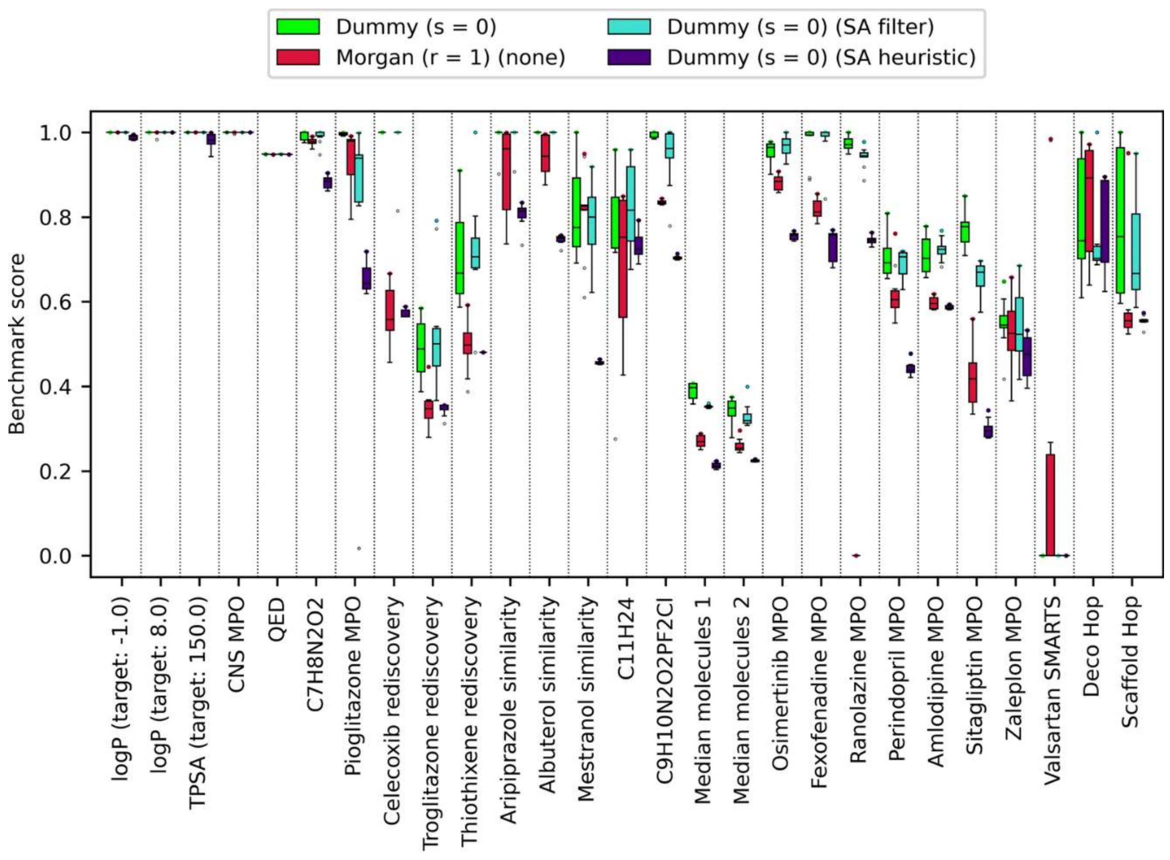


*Figure 4.19. **LEADD** optimization power comparison using different SA optimization strategies. Colored dots represent maximum benchmark scores.*

Table 4.12. Multiple comparisons of benchmark score distributions' stochastic dominances using different approaches to improve SA with Conover-Iman´s post-hoc test with Šidák correction (FWER = 0.05). The test was preceded by a Kruskal-Wallis test (H = 94.69, p < 0.001).

Group 1	**Group 2**	$\widetilde{Score}_{Group2} - \widetilde{Score}_{Group1}$	**Adjusted p-value**
Dummy	Morgan (r = 1)	-0.122	< 0.001
Dummy	Dummy (SAScore filter)	-0.027	0.724
Dummy	Dummy (SAScore heuristic)	-0.253	< 0.001
Morgan (r = 1)	Dummy (SAScore filter)	0.096	< 0.001
Morgan (r = 1)	Dummy (SAScore heuristic)	-0.161	0.006
Dummy (SAScore filter)	Dummy (SAScore heuristic)	-0.226	< 0.001

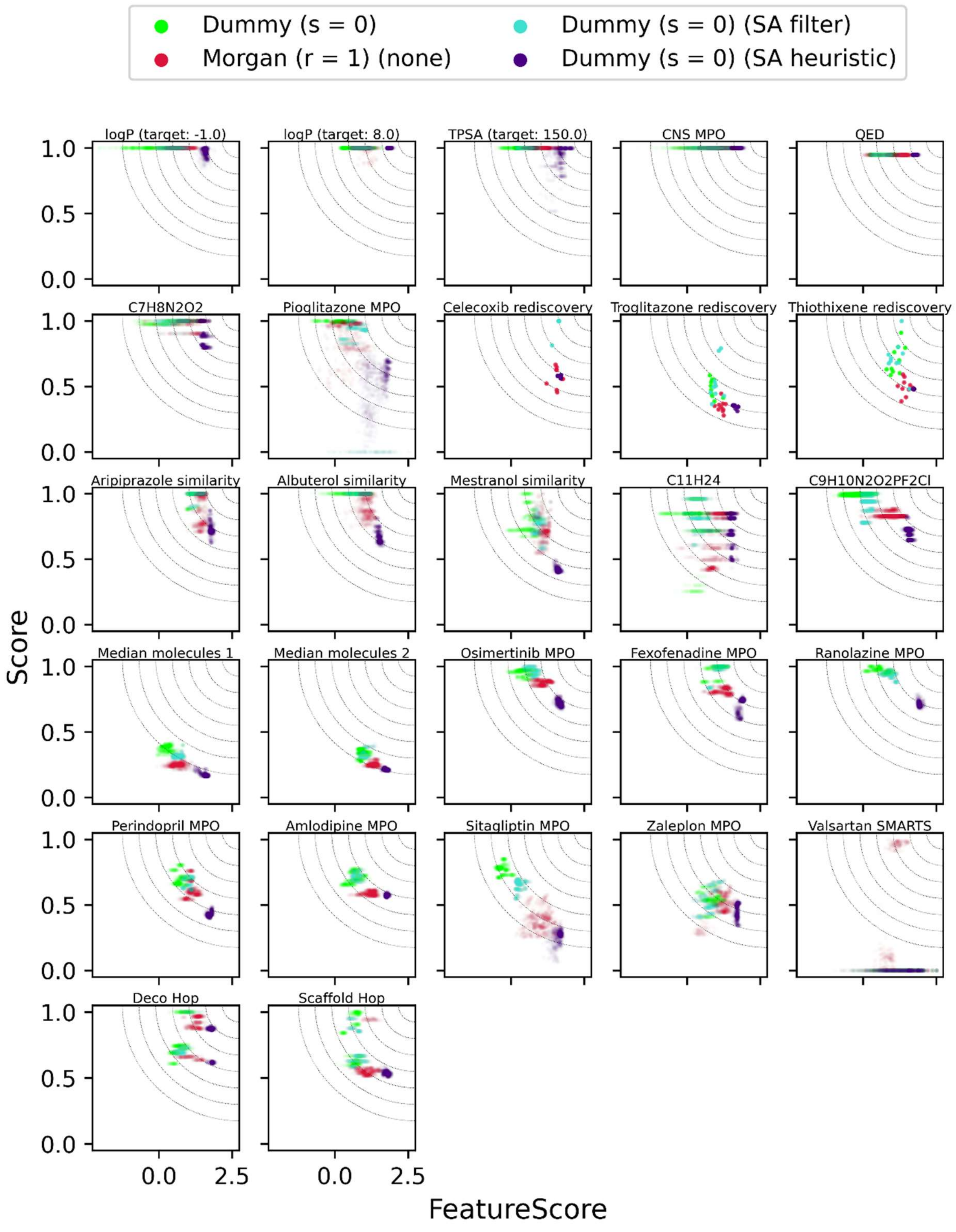


Figure 4.20. Correlation between FeatureScore and benchmark score using different approaches to increase SA. Large feature scores are associated with better SA.

4.4.7 Comparison to other algorithms and virtual screening

Lastly, we wanted to compare **LEADD**’s performance to that of GB-GA [100] and a VS of the GuacaMol virtual library. In terms of OP, **LEADD** with dummy atom types outperformed the VS in 26/27 benchmarks, with the only exception being the Valsartan SMARTS benchmark which uses a binary scoring function ill-suited for goal-directed optimization approaches (Figure 4.25). **LEADD** with the use of dummy atom types is comparable to GB-GA, in the sense that both are graph-based EAs with very few restrictions on how atoms can be connected. Correspondingly, the SA (Figure 4.21, Table 4.13) and OP (Figure 4.25, Table 4.14) of these two systems are comparable. The key difference between both algorithms is that **LEADD** modifies molecules on a fragment level as opposed to the atom level of GB-GA. Although we paired dummy atom types with single-atom acyclic fragments, ring systems are always treated as monolithic fragments. We expected this to yield improved SA and smaller OP, yet found the opposite. **LEADD** has better OP than GB-GA, outperforming it in 18/27 benchmarks and performing equally well or better in 23/27 benchmarks. We attribute this to the bigger step size associated with fragments and the internal topological similarity threshold to enforce population diversity, giving it an edge at escaping local fitness minima. It’s also possible that the same factors explain the better SA of molecules designed by GB-GA. Most GuacaMol benchmarks incorporate topological similarity to a reference drug-like molecule in their objective functions, implicitly capturing some SA notions. Because of **LEADD**’s internal similarity threshold only the best individual within the population can assume the identity of the reference molecule, whereas the rest are forced to diverge from it. In GB-GA all individuals are allowed to approach the target molecule as much as possible, benefitting to a greater extent from the implicit SA target of the objective function. Moreover, GB-GA does not allow the creation of SSSR cycles bigger than six-membered rings whereas some of the fragments used by **LEADD** do include bigger cycles. Since the SAScore incorporates a macrocycle penalty this could account for some of the observed differences. Ultimately, the magnitude of the SA changes associated with the use of fragments, be it cyclic or acyclic, are small (Table 4.10, Table 4.13). This calls into the question the widespread practice of fragment-based molecular construction as a means to improve SA, and we hypothesize that its effectiveness depends on how well *in silico* fragments and their assembly rules correlate with *ex silico* reactants and chemical reactions.

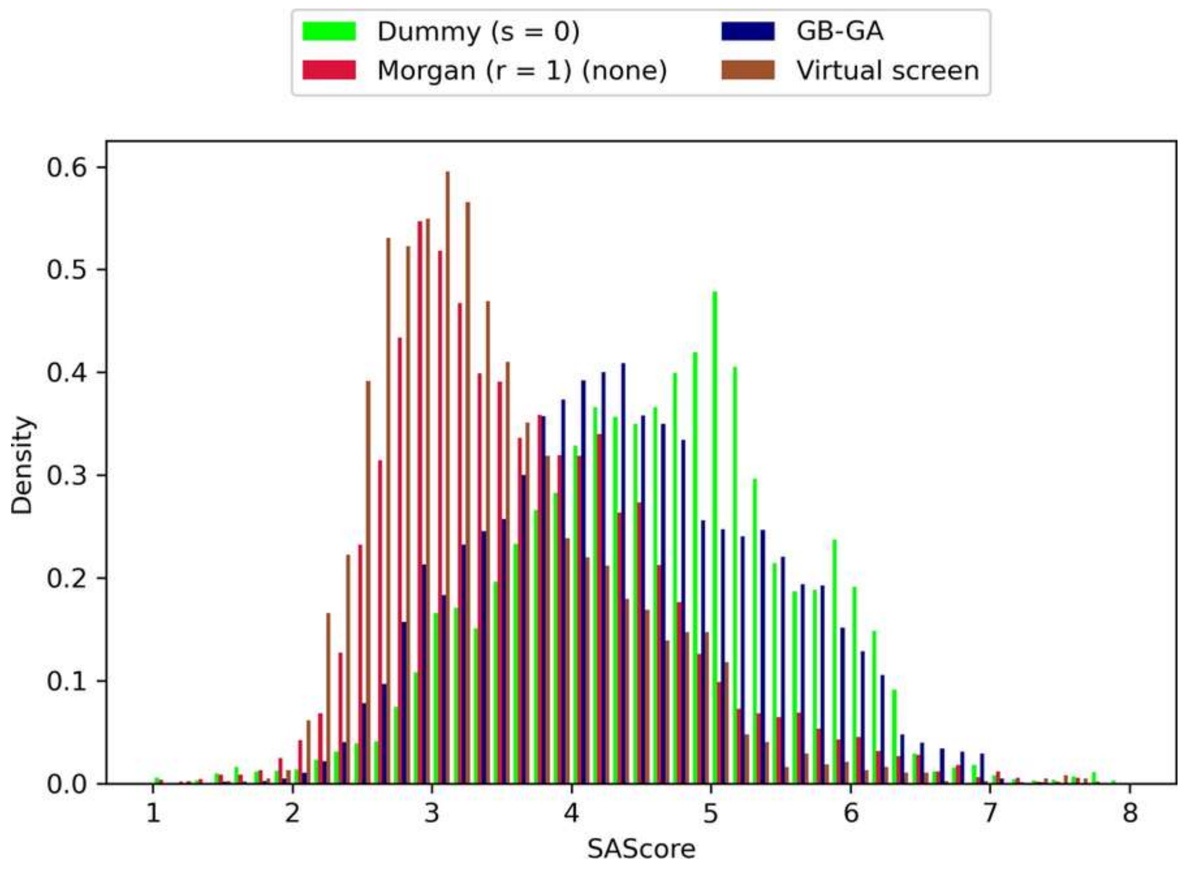


*Figure 4.21. Comparison of SAScore distributions between molecules designed by **LEADD** and GB-GA and those found through a VS. Includes molecules of all benchmarks and replicas. Molecules with lower SAScores are predicted to be easier to synthesize*

*Table 4.13. Multiple comparisons of SAScore means between **LEADD**, GB-GA and a VS with Tukey's HSD post-hoc test (FWER = 0.05). The test was preceded by a one-way ANOVA (F = 5715.55, p < 0.001).*

Group 1	**Group 2**	$\overline{SAScore}_{Group2} - \overline{SAScore}_{Group1}$	**Adjusted p-value**
Dummy	Morgan (r = 1)	-0.902	< 0.001
Dummy	GB-GA	-0.222	< 0.001
Dummy	VS	-1.170	< 0.001
Morgan (r = 1)	GB-GA	0.680	< 0.001
Morgan (r = 1)	VS	-0.268	< 0.001
GB-GA	VS	-0.948	< 0.001

When using Morgan (r = 1) atom types and monolithic acyclic fragments **LEADD** designs molecules with much better SA than GB-GA (Figure 4.21, Table 4.13). This is to be expected since GB-GA does not take SA into account intrinsically. However, it's possible to design SA molecules with GB-GA by using the previously discussed extrinsic SAScore-based heuristic score modifier [62, 109]. Doing so yields a similar OP-SA trade-off to the one observed for **LEADD** and the same heuristic (Figure 4.18, Figure 4.19), strongly favoring SA over OP (Figure 4.22, Figure 4.23). The SA of molecules designed by **LEADD** using Morgan (r = 1) atom types is almost on par with those found by a VS (Figure 4.21, Table 4.13). We would like to remark that the feature set we used to calculate SAScores was extracted from ChEMBL [136], and that the screened GuacaMol library is a subset of ChEMBL [108]. It's therefore to be expected that molecules found through VS have better SAScores. Since SAScores are a rather crude way of assessing SA, to confirm our findings we ran retrosynthetic analyses on the top 10 scoring molecules of each benchmark replica using AiZynthFinder [132] with the ZINC [2] reactant stock and USPTO-derived reaction template policy provided by the authors. Both **LEADD** and GB-GA designed less synthesizable molecules than those found by the VS, but when using Morgan atom types **LEADD** designed considerably more synthesizable molecules than GB-GA (Figure 4.24). It's worth noting that only 60% of the molecules selected by the VS from the ChEMBL subset were deemed synthesizable by the retrosynthetic analyses. If we assume that all molecules in ChEMBL are synthesizable this would suggest that we might be underestimating the SA of molecules, including those designed by the EAs.

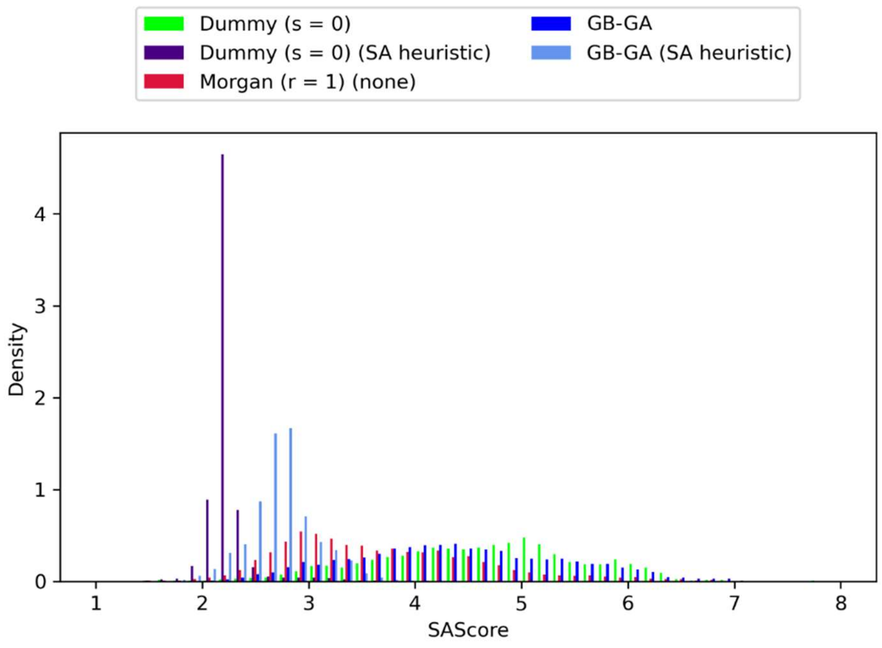


*Figure 4.22. Comparison of SAScore distributions between molecules designed by **LEADD** and GB-GA with or without using the SAScore-based score modifier. Includes molecules of all benchmarks and replicas. Molecules with lower SAScores are predicted to be easier to synthesize.*

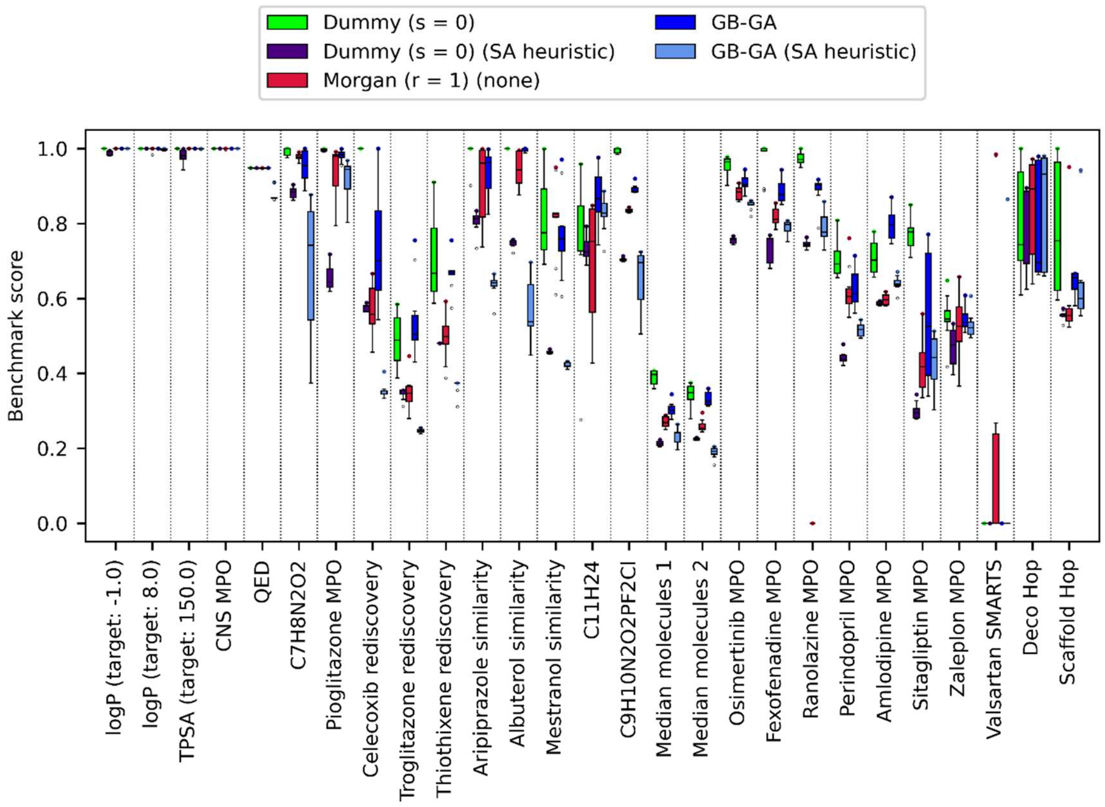


*Figure 4.23. Optimization power comparison between **LEADD** and GB-GA with or without using the SAScore-based score modifier. Colored dots represent maximum benchmark scores.*

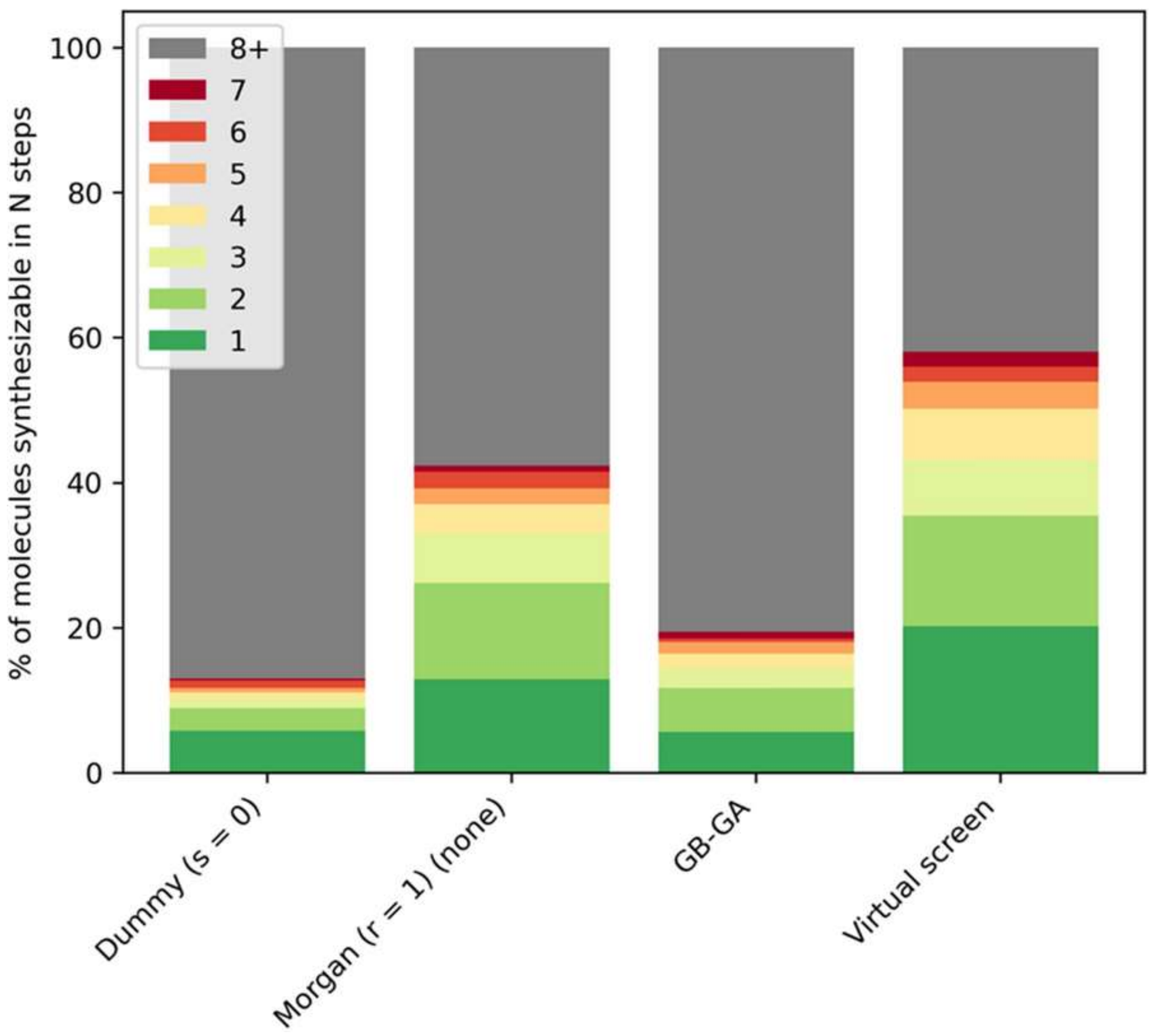


*Figure 4.24. Fraction of top-10 scored molecules per replica synthesizable by **LEADD** (with different settings), GB-GA and VS in N or less steps using ZINC reactants and USPTO reaction templates, as assessed by AiZynthFinder. Molecules requiring more than 8 synthetic steps are considered not synthesizable.*

Interestingly, we did not observe a statistically significant difference in OP stochastic dominance between **LEADD** with Morgan atom types and GB-GA (Table 4.14). Given that EAs are stochastic in nature, one would typically run multiple replicas and keep the best results. This justifies comparing maximum instead of average benchmark scores. In terms of maximum score, **LEADD** with Morgan atom types performed comparably or better than GB-GA in 16/27 benchmarks and comparably or better than the VS in 23/27 benchmarks (Figure 4.25). Crucially, **LEADD** performed better than GB-GA in the Deco Hop and Scaffold Hop benchmarks, which are arguably the most representative of real drug discovery problems.

We would like to note that goal-directed design employing structure-based scoring functions is associated with an additional set of challenges that is not posed by the ligand-based GuacaMol benchmark suite, including the handling of stereochemistry, pose inversion and the typical bias of these scoring functions towards large, hydrophobic and flexible molecules. Indeed, preliminary results using OpenEye ROCS [160] as **LEADD**'s scoring function show a tendency towards designing large and very cyclic molecules. This also makes it challenging to compare 2D molecular design algorithms like **LEADD** and GB-GA to their 3D counterparts [16, 18, 19, 61, 111, 113].

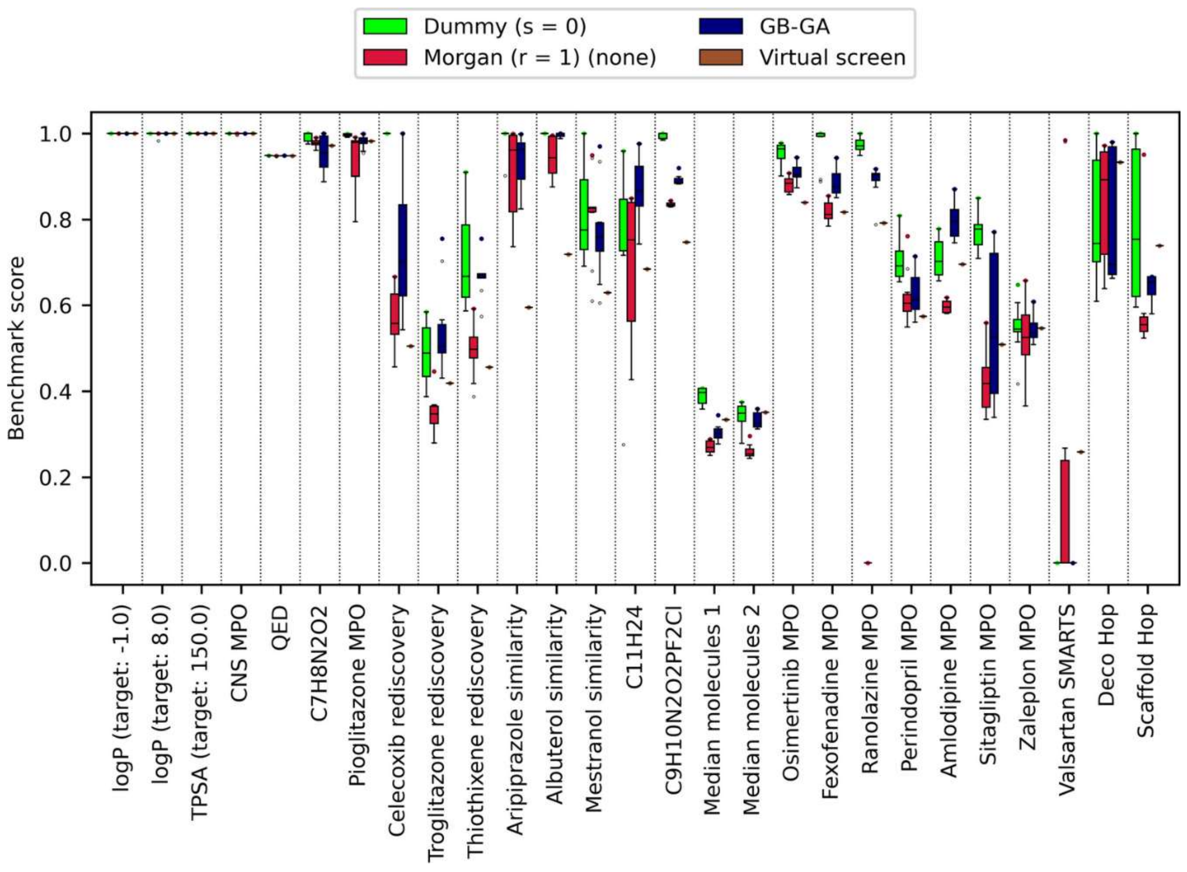


*Figure 4.25. Optimization power comparison between **LEADD**, GB-GA and a VS. Colored dots represent maximum benchmark scores. Note that VS results are deterministic and have null variability.*

Table 4.14. Multiple comparisons of benchmark score distributions' stochastic dominances using different approaches to improve SA with Conover-Iman´s post-hoc test with Šidák correction (FWER = 0.05). The test was preceded by a Kruskal-Wallis test (H = 94.69, p < 0.001).

Group 1	**Group 2**	$\widetilde{Score}_{Group2} - \widetilde{Score}_{Group1}$	**Adjusted p-value**
Dummy	Morgan (r = 1)	-0.122	< 0.001
Dummy	Dummy (SAScore filter)	-0.027	0.724
Dummy	Dummy (SAScore heuristic)	-0.253	< 0.001
Morgan (r = 1)	Dummy (SAScore filter)	0.096	< 0.001
Morgan (r = 1)	Dummy (SAScore heuristic)	-0.161	0.006
Dummy (SAScore filter)	Dummy (SAScore heuristic)	-0.226	< 0.001

It is also important to consider the amount of computational resources spent by each approach to achieve its results. Figure 4.26 shows how an average EA replica finds higher scoring molecules than a VS with a smaller number of scoring function calls. While one should keep in mind that it's generally desirable to run multiple EA replicas, EAs make better use of computational resources than a VS, especially if evaluating the scoring function is expensive. It's also worth noting that **LEADD**, despite its use of fragments, did not find solutions much slower than GB-GA (Figure 4.26). Naturally, there is an overhead associated with the design algorithm. On a single core of a Xeon E5-2680v2 CPU (2.8 GHz), **LEADD** designed on average 272 mol/s. Assuming that about 10^4–10^5 molecules must be designed to find good solutions (Figure 4.26) this corresponds to an overhead of just a couple CPU minutes. For comparison GB-GA designed 98 mol/s. This difference in performance is mostly due to implementation optimizations rather than due to algorithmic differences since **LEADD** is considerably more complex algorithmically. When using fast scoring functions molecule generation can become the rate limiting step. During the GuacaMol benchmark suite **LEADD** generated molecules slower than they were scored in 25/27 benchmarks. On average, molecules were designed eightfold slower than they were scored, with differences exceeding 20-fold in some benchmarks. This showcases the need for fast molecular design algorithms. Note that the reported values are averages, and that execution times depend heavily on the number of possibilities the algorithm has to consider. For instance, when using a smaller database of fragments or smaller population the algorithm is faster. Similarly, the computational resources spent per operation increase with molecular complexity, specifically degree of branching.

If one wishes to achieve even greater OP it's possible to use the results of a VS as the starting population for EAs. While we do not believe this qualifies as *de novo* molecular design, this type of molecular optimization may be interesting when computational resources are abundant. Unsurprisingly, we found that using VS results as starting populations decreased the variability between replicas and increased the mean replica score (Figure 4.28). However, when using Morgan atom types this did not always translate into higher maximum scores, as the starting population may already be close to local fitness minima in which the algorithm might get stuck. It's interesting to note that, while the molecules designed this way have better SA than those in a true *de novo* design setting, it's worse than that of the starting population (Figure 4.27). The SA loss is small for **LEADD** with Morgan atom types, but substantial for GB-GA and **LEADD** with dummy atom types, in the latter case almost reverting to the *de novo* design values. This showcases a tendency to design synthetically complex molecules when algorithms form bonds carelessly.

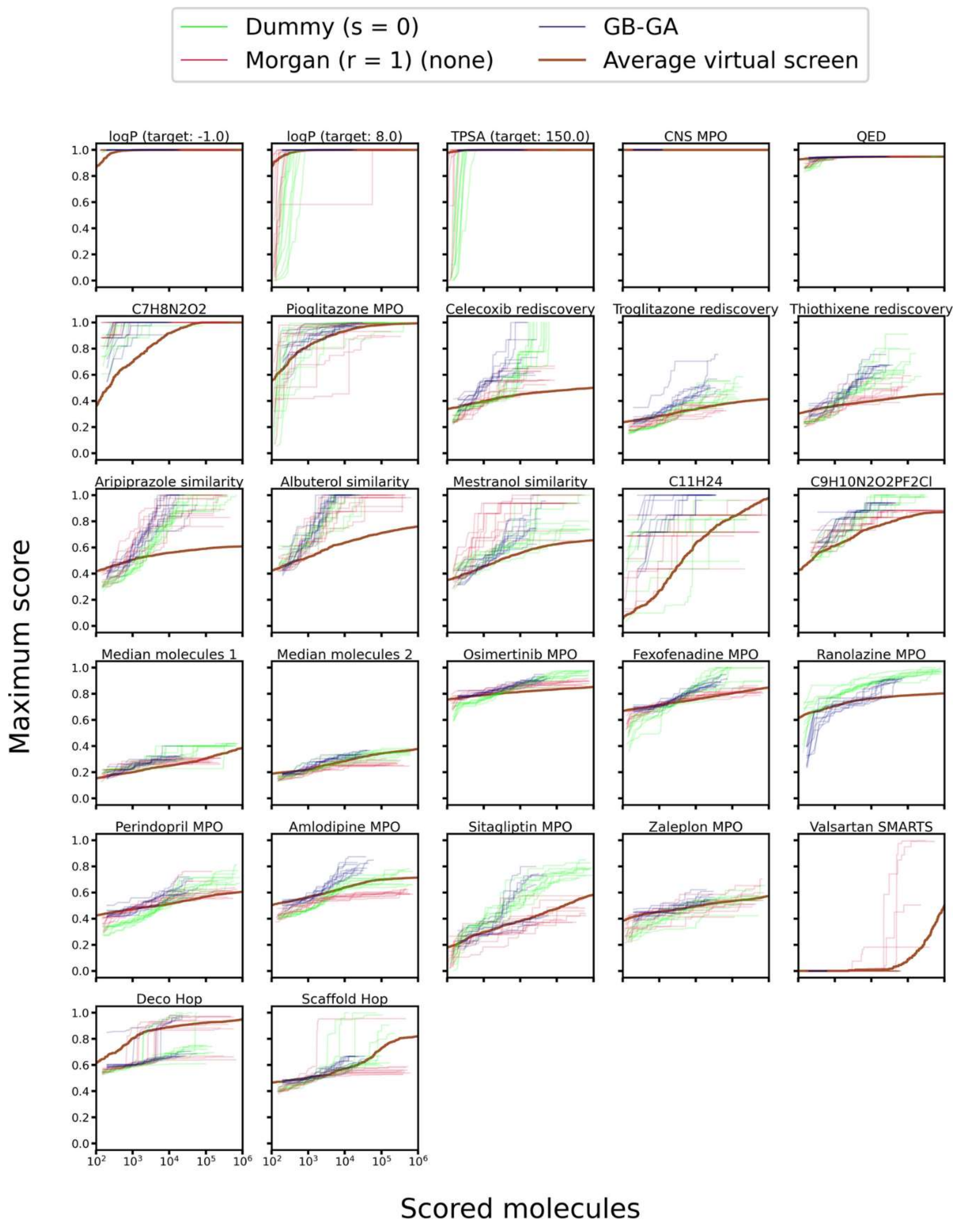


*Figure 4.26. Score of best found molecule as a function of the number of scored molecules. For **LEADD** and GB-GA each line represents a replica. VS results were shuffled 100 times and averaged to account for the effects of molecule screening order. Note that these are individual molecule scores and not population/benchmark scores and therefore do nott correspond to the values in Figure 4.25.*

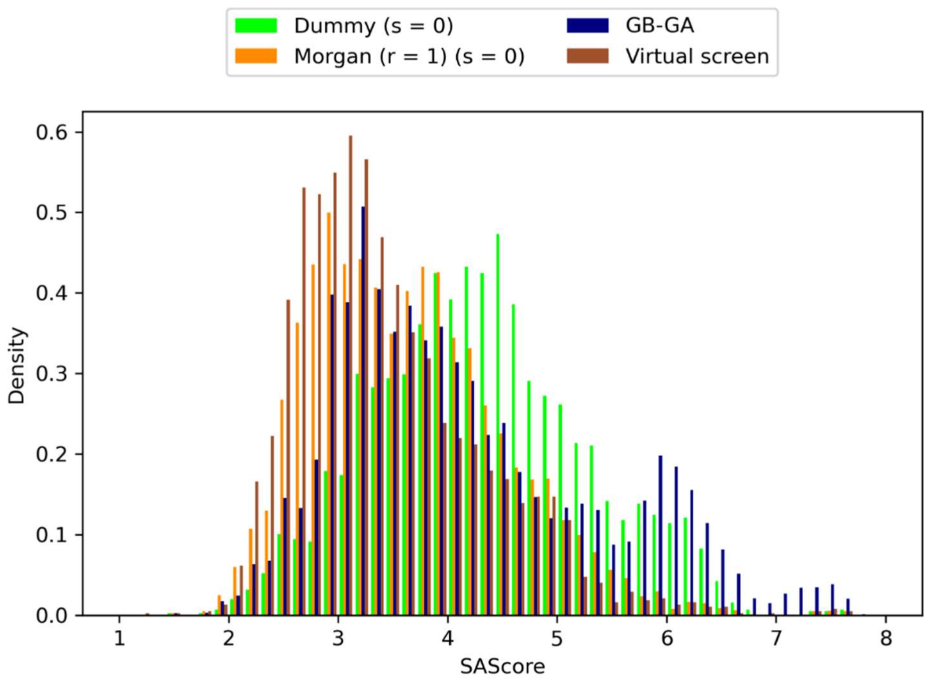


*Figure 4.27. Comparison of SAScore distributions between molecules designed by **LEADD** and GB-GA using VS results as a starting population and said VS results. Includes molecules of all benchmarks and replicas.*

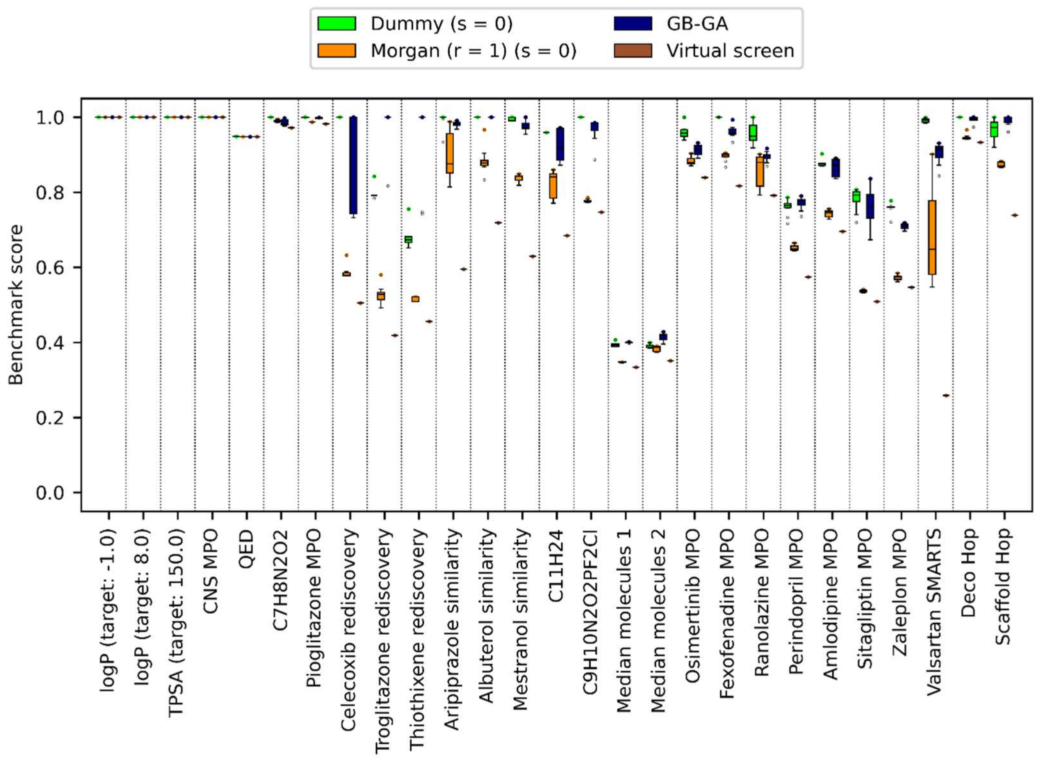


*Figure 4.28. Optimization power comparison between **LEADD**, GB-GA and a VS using the VS results as starting populations for the EAs.Note that VS results are deterministic and have null variability.*

Chapter 5 Easy enforcement of molecular constraints

5.1 Source

A manuscript based on this chapter has been accepted for publication and is in press:

Kerstjens, A., De Winter, H. **Molecule auto-correction to facilitate molecular design**. *J. Comput. Aided Mol. Des.* (2024)

5.2 Problem statement

There are numerous schemes to increase the likelihood of designing chemically appealing molecules, some of which have been previously explored in this work (Chapter 3, Chapter 4). While in principle one could reuse these frameworks in other molecule generators, doing so is not always straightforward. For example, the strategies we presented demand molecule generators that iteratively build up or modify molecules. In the case of **LEADD** (Chapter 4), an additional requirement is that these modifications must be on a fragment basis. Not all molecule generators fulfill these criteria. Some algorithms use different molecular representations. Among the methods using molecular (meta-) graphs some might redefine the units of a molecule that are perturbed [12, 64, 116–118]. Other algorithms forgo graph-like representations altogether, modifying text-based line notations [86, 91, 92] or abstract continuous representations of molecules [67, 89, 90]. Even among the algorithms that do meet the criteria for integrating the presented frameworks, integration might be non-trivial and require significant adjustments to the code.

In this chapter we will describe a more flexible technique to designing chemically desirable molecules, which we hope can be integrated into a wide range of molecule generators and workflows with minimal effort. In layman's terms it could be described as "molecule auto-correction". The algorithm describes a query molecule with local structural features, and compares said features to those found in reference desirable molecules. If the query molecule possesses features that are absent or rare in the reference molecules, the features are deemed "foreign" or incorrect. Otherwise they are deemed "familiar" or correct. Through a tree search algorithm we locally modify foreign features until they are familiar enough. Certain heuristics are used to prioritize modifications that are most likely to yield familiar features. One can draw an analogy between our algorithm and a primitive spell checker, where chemical features are the equivalent of words. Each word is checked against a dictionary of known words. If a word is not present it's deemed incorrect and a heuristic suggests similar correct words.

Our intention is twofold. Firstly, we envision the tool being used to address molecule quality issues that were not caught or covered by a third-party molecule generator (Figure 5.1). Secondly, and perhaps most importantly, we hope that it will enable researchers to divest some of their attention from avoiding non-sense molecule generation to other aspects of molecular design.

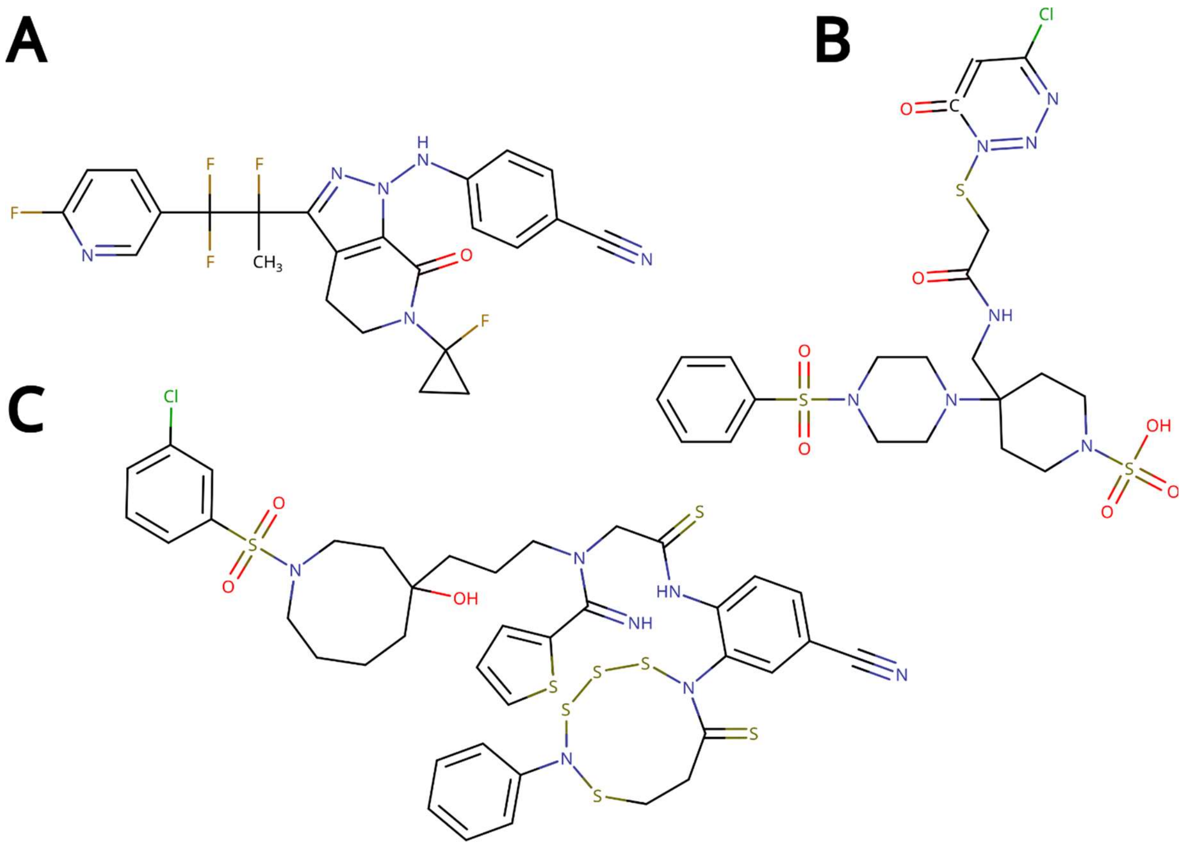


Figure 5.1. Examples of objectionable molecules generated by diverse molecule generators during a JAK2 inhibitor design exercise, as reported by [161]. (A) was generated by a graph-based genetic algorithm [100], (B) was generated by a particle swarm optimizer in an auto-encoder latent space [67] and (C) was generated by a SMILES-based recurrent neural network [91].

5.3 Methodology

5.3.1 Molecule characterization

To identify if a molecule is foreign, and if so, what parts are foreign, we defined some simple localized molecular descriptors. Atoms are characterized with atom keys. Atom keys are integer tuples comprising an atom's degree (D) (i.e. its number of adjacent atoms), valence

(V), atomic number (Z), formal charge (Q) and number of hydrogens (H). These properties were chosen because they are largely independent from the atom's surrounding chemical environment. To avoid cyclic dependencies between properties, in this work valence is defined as the sum of an atom's bonds' orders, without considering the atom's formal charge. The order of the atom key's properties is relevant. We ordered the properties by perceived decreasing significance or importance. For example, we assume that a change in degree, and therefore topology, is more disruptive to a molecule's structure and properties than a change in atomic number. Bonds were characterized as a tuple of the bonded atoms' keys (AK) and an integer representing the bond's type (B), which can be thought of as the bond's order (Figure 5.2).

We also define partial keys of the atom and bond keys. Partial atom keys are constructed by taking the first *j* most significant properties of the atom key, with *j ∈ [1, k-1]*, where *k* is the number of properties in an atom key. Consequently, partial key *j* contains all partial keys with a lower *j*. The same procedure is applied to bond keys but with the range *j ∈ [2, k-1]*. This yields a total of four partial atom keys and one partial bond key (Figure 5.2). Partial keys can be sorted lexicographically, enabling fast key-value store searches.

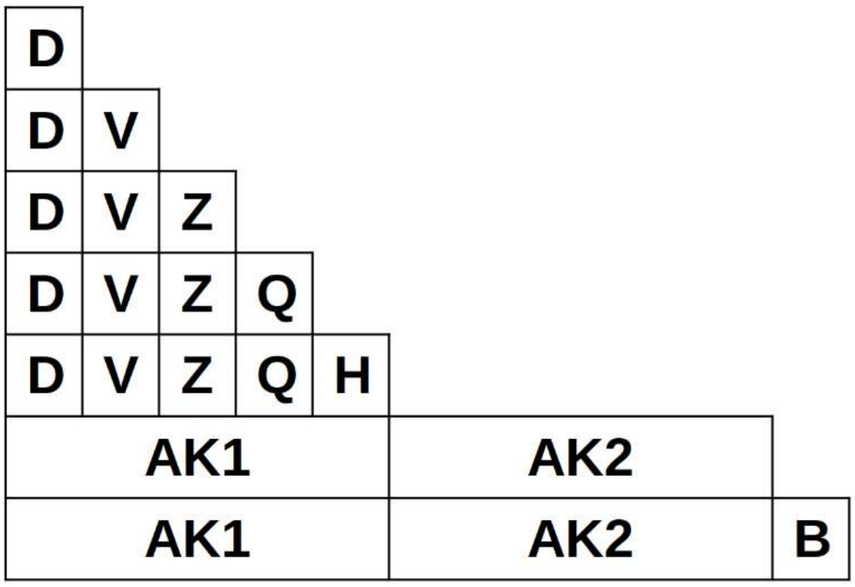


Figure 5.2. Partial atom and bond key pyramid. Higher order keys encompass lower order keys. The (D, V, Z, Q, H) key constitutes the atom key AK, and (AK1, AK2, B) constitutes the bond key.

Lastly, circular atomic environments are defined for all atoms in the molecules. A circular atomic environment comprises a central atom and all surrounding atoms within a given topological distance termed the environment's radius *r*. The resulting atomic environment is hashed to an integer using the Morgan algorithm, like one would when calculating ECFP fingerprints [34, 46].

5.3.2 Reference dictionary

In this work a subset of ChEMBL31 [136] was chosen as the reference library of drug-like molecules. Only small organic molecules were retained. Large biomolecules, natural products and polymers were excluded. For the remaining molecules the unsalted and non-ionized “parent form” was chosen. Molecules in the reference library were characterized using the aforementioned descriptor keys, and the frequency of each key recorded in a “chemical dictionary”. We generated two dictionaries using environment radii of 1 and 2, respectively. If a key’s frequency surpasses a user-specified threshold (by default 0) it’s deemed familiar, and otherwise it is deemed foreign. Owing to the way in which keys are defined, simpler keys are contained by more complex keys. For example, environment keys contain bond keys and bond keys contain atom keys (Figure 5.2). This defines unidirectional dependency relationships between them, meaning that if a key is foreign all dependent keys containing it must also be foreign. The reverse is not necessarily true.

5.3.3 Tree search algorithm

The molecule correction algorithm was implemented as a tree search. An incorrect input molecule serves as the root of the tree. With each iteration a molecule or vertex within the tree is selected and partially expanded. Expansion in this context means enumeration of topologically similar neighboring molecules, and establishment of a parental relationship between the selected predecessor and its neighboring successors. Expansions were performed using the graph-based molecule perturbation library **Molpert** (Chapter 3). Perturbations performed by the library include atom- and bond invariant changes and atom/bond insertions/deletions. To expedite the correction process molecules are sanitized (as described in 3.3.3) after each perturbation by default, but this behavior can be disabled. **Molpert** enables the systematic enumeration of a molecule’s neighbors. Neighbors are enumerated lazily. The enumeration order is optimized to maximize the likelihood of finding a correct molecule with the smallest number of expansions.

As with any tree search algorithm, the search is guided by a search strategy or *policy* that dictates how the tree is expanded with each iteration. For our tree search we distinguish two different types of policies. One policy, which we call the *selection policy*, selects which vertex to expand next. The second policy, termed the *expansion policy*, determines how the selected vertex is expanded.

5.3.3.1 *Selection policy*

To guide the search towards familiar molecules we define the concept of *familiarity*. Every time a vertex is added to the tree it’s featurized into atom, bond and environment keys. Said keys are classified into foreign and familiar by looking them up in the chemical dictionary. Familiarity is calculated as a function of the total number of keys n (Equation 5.1) and the number of familiar keys n^f (Equation 5.2).

Equation 5.1

$$n = n_a + n_b + n_e$$

Equation 5.2

$$n^f = n_a^f + n_b^f + n_e^f$$

In Equation 5.1 n_a, n_b and n_e denote the total number of atom, bond, and environment keys of a given molecule respectively, whereas in Equation 5.2 n^f_a, n^f_b and n^f_e denote their familiar counterparts.

We employ two alternative definitions of familiarity: f_1 (Equation 5.3) and f_2 (Equation 5.4). Both range between 0 and 1, with 1 indicating a familiar or correct molecule, and can mostly be used interchangeably. f_1 can be interpreted as a similarity coefficient between a query molecule and some unknown correct molecule. Conversely, $1 - f_1$ can be interpreted as the distance to a correct molecule. f_1 is therefore well suited for estimating how close to a solution a given molecule is. f_2 provides weaker theoretical guarantees as a similarity coefficient, for its lower boundary is dependent on the molecule's size. f_1's drawback is that it can be maximized trivially by incrementing the numerator and denominator by the same amount, as occurs when adding new familiar environments (e.g. alkane carbons). f_2 cannot be exploited in the same way, and is better suited as an optimization target.

Equation 5.3

$$f_1 = \frac{n^f}{n}$$

Equation 5.4

$$f_2 = \frac{1}{n - n^f + 1}$$

Different selection policies were explored. In all cases the selection is limited to foreign molecules ($f < 1$) that have not been fully expanded yet. As baselines we evaluated Breadth-First Search (BFS), where the shallowest vertices are expanded first, and greedy familiarity selection, where the vertices with the highest f_2 familiarity are expanded first. These correspond to exploration-only and exploitation-only approaches, respectively (Figure 5.3). Note that a deep BFS is computationally intractable since the branching factor of chemical space is very large (Figure 5.4).

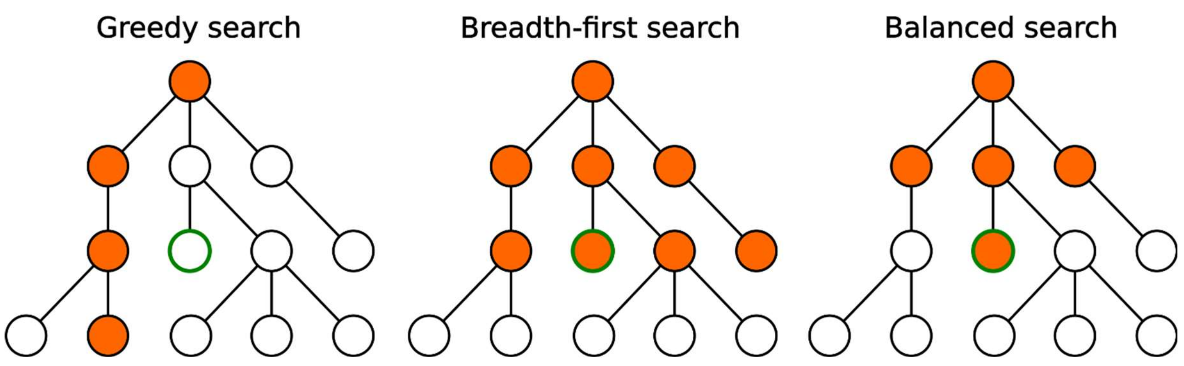


Figure 5.3. Different types of selection policies. Orange vertices represent visited vertices. The goal is to find the optimal green vertex while minimizing the number of visited vertices. Greedy search visits very few vertices but may miss the goal vertex. Breadth-first search is guaranteed to find the goal vertex but visits many other vertices in the process. An ideal selection policy balances exploration and exploitation.

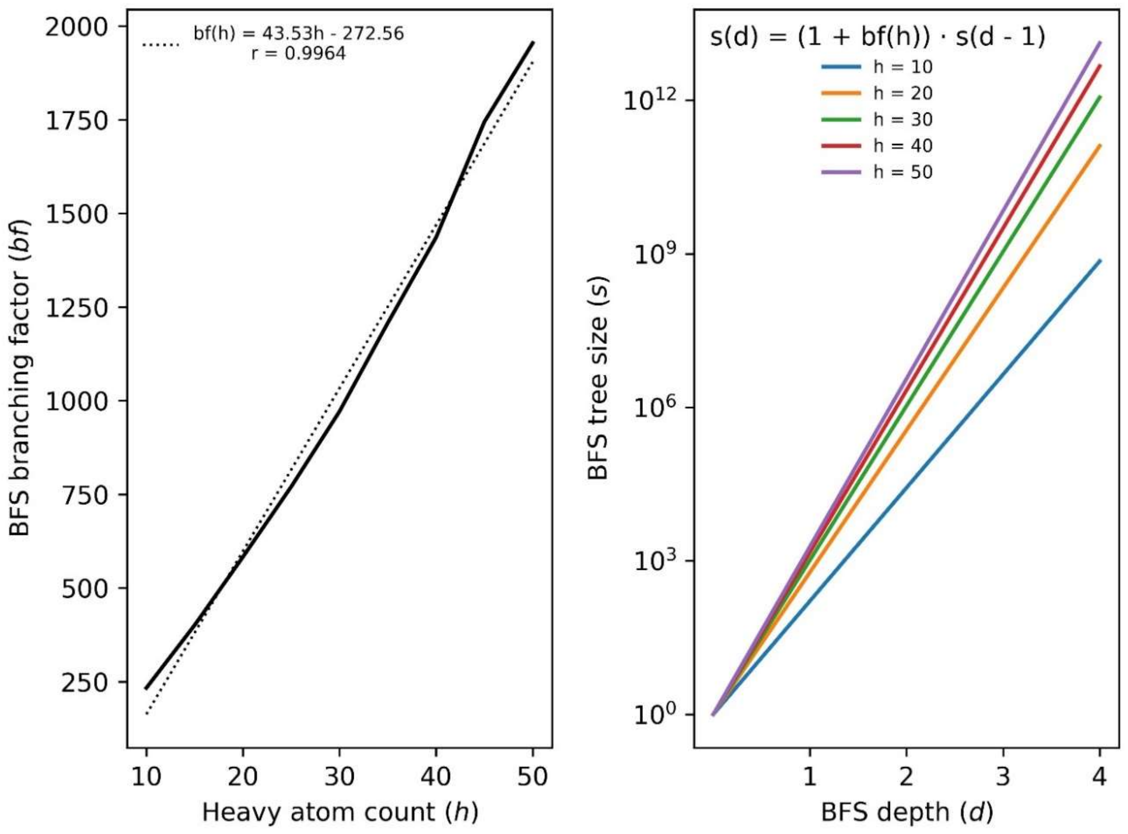


Figure 5.4. Left panel: Branching factor (bf) of a Breadth-First Search (BFS) as a function of the root molecule's number of heavy atoms (h). The branching factor was calculated by enumerating all neighboring molecules using Molpert's "balanced" settings. Right panel: Projection of tree size (s) for a given BFS depth (d) assuming constant molecule size throughout the search. This assumption is reasonable since the average heavy atom count of molecules only increases about 0.25 per BFS search level.

There are many correct molecules and many paths leading to them from the input molecule. We would prefer finding the correct molecule *w* that is most closely related to the input or root molecule *u*, as according to the similar property principle it is the most likely to preserve the properties of the input molecule. The distance between the input molecule *u* and another molecule *v* of the tree is measured as the ECFP4 Tanimoto distance *d(u,v)* between both. We chose this fingerprint and distance metric combination because they have been shown to be good predictors of activity preservation [35, 43, 44].

Some policies to favor shallow tree searches and better balance exploration and exploitation were devised (Figure 5.3). The most naive one is to greedily select vertices with the highest $f_1/d(u,v)$ ratio. More sophisticated policies are described below.

5.3.3.1.1 Upper confidence bounds applied to trees

One can estimate how close a vertex is to a yet to be discovered correct molecule using the familiarity metric. However, it is not always true that the vertex with the highest familiarity is involved in the path to the closest correct molecule. The values (i.e. familiarities) of a parent vertex's children follow an *a priori* unknown distribution. We can get better estimates of the expected child value by sampling or generating more children. As more samples become available the estimate trends towards the true value. Given limited computational resources one must choose between exploring vertices with uncertain distributions or exploiting vertices with the most promising distributions. This is known as a bandit problem, and the Upper Confidence Bound (UCB) strategy can be applied to tackle it [162]. UCB applied to Tree searches (UCT) dictates that at each iteration one should expand the vertex with the highest upper confidence interval bound [163]. In other words, one should expand the vertex for which the potential upside is maximized. Mathematically, this means expanding the vertex *v* maximizing Equation 5.5.

Equation 5.5

$$UCB1 = \overline{f_{1v}} + c\sqrt{\frac{ln(N_v)}{n_v}}$$

In Equation 5.5 $\overline{f_{1v}}$ is the average f_1 familiarity of *v*'s children, n_v is the number of times *v* was expanded, N_v is the number of times *v*'s parent was expanded. The first term of Equation 5.5 is exploitative and the second term is explorative. *c* is a coefficient balancing between exploitation and exploration. In this work we explored *c* values of ½, 1, $\sqrt{2}$ and 2.

UCT was first applied to Monte Carlo Tree Search (MCTS) [163], and is oftentimes discussed in relation to it. The difference between a plain tree search and MCTS is that in the former the value of a vertex is given by a heuristic function (in our case the familiarity) whereas in the latter the value of a vertex is estimated through means of random simulations or "rollouts". We want to clarify that our tree search is not a MCTS despite using the UCT policy, as random simulations did not produce better value estimates than the familiarity heuristic within reasonable time and resource constraints.

5.3.3.1.2 A-star

The A* (pronounced A-star) search algorithm is a path finding algorithm suitable for finding close to optimal shortest paths in a graph within reasonable amounts of time [164]. It selects for exploration/expansion the vertex *v* for which Equation 5.6 is minimized.

Equation 5.6

$$g(v) = m(v) + h(v)$$

In Equation 5.6 *m(v)* is the distance traversed to reach *v*. In our case *m(v)* is the topological distance between vertex *v* and the root vertex *u*, that is, *m(v) = d(u,v)*. *h(v)* is a heuristic estimate of the distance between *v* and an end point *w*, in our case a correct molecule. In other words, *h(v) ~ d(v,w)*. An obvious heuristic candidate is $h(v) = 1 - f_1(v)$ (Equation 5.7).

Equation 5.7

$$g(v) = d(u, v) + 1 - f_1(v)$$

d(v,w) is a Tanimoto distance, which is the complement of the Tanimoto similarity or Jaccard index. If *V* and *W* denote the feature set of molecules *v* and *w*, their Jaccard index is calculated according to Equation 5.8.

Equation 5.8

$$J(v, w) = \frac{|V \cap W|}{|V \cup W|} = \frac{|V \cap W|}{|V| + |W| - |V \cap W|}$$

$f_1(v)$ is a similarity index measuring the similarity to some unknown correct molecule *w*. While not equivalent to the Jaccard index, it's related to it. If *W* denotes the feature set of this hypothetical correct molecule, $f_1(v)$ can be rewritten as shown in Equation 5.9.

Equation 5.9

$$f_1(v) = \frac{|V \cap W|}{|V|}$$

If $f_1(v)$ were calculated using as keys solely ECFP features Equation 5.8 and Equation 5.9 would differ only in their denominator. It's clear that $|V| + |W| - |V \cap W| \geq |V|$. Therefore, $1 - d(v,w) \leq f_1(v)$, or equivalently $1 - f_1(v) \leq d(v,w)$, which would make $1 - f_1(v)$

an admissible heuristic. Moreover, since Jaccard distances are known to satisfy the triangle inequality [165], that is, *d(u,w) ≤ d(u,v) + d(v,w),* the heuristic would also be consistent. Using a consistent heuristic guarantees that the algorithm will find the optimal solution given enough time. We included additional terms in *f_1(v)* besides the environment keys as we believe this additional granularity can provide finer guidance to the tree search. Consequently *1 – f_1(v)* as described in Equation 5.3 is theoretically not an admissible heuristic. Nonetheless in practice it very rarely overestimates the *d(v,w)* distance (Figure 5.5).

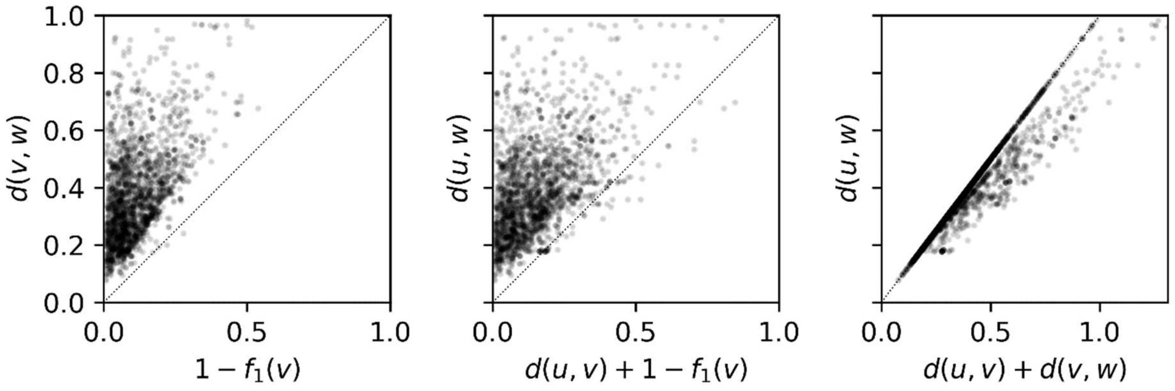


Figure 5.5. Relationships between d(u,v), d(v,w), d(u,w) and 1 – f_1(v). The two leftmost panels show that in practice 1 – f_1(v) is an almost admissible and consistent heuristic, respectively. The rightmost panel is visual proof of Jaccard distances obeying the triangle inequality. Note that the correlation between d(u,v) + d(v,w) and d(u,w) is very high, which is typical of hyper dimensional spaces such as chemical space.

5.3.3.1.3 Multiple linear regression

The A* algorithm was devised for path finding and searches for the shortest path between two vertices. We are interested in finding the closest goal vertex, that is, minimizing the distance to a goal vertex "as the crow flies". Both of these distances are not equivalent (Figure 5.6).

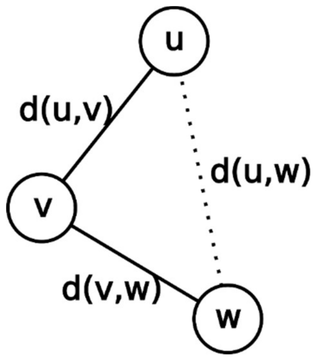


Figure 5.6. Difference between path distance (d(u,v) + d(v,w)) and straight distance (d(u,w)).

To minimize *d(u,w)* we developed a policy that selects the vertex for which the predicted *d(u,w)* is minimal. We wanted to predict *d(u,w)* as a function of *d(u,v)* and *f_1(v)*, which are both known for any vertex. To study the relationships between these metrics we randomly perturbed a sample of 10^3 molecules from ChEMBL [136] by applying between 1 and 10 perturbations to each of them using **Molpert** [166] for a total of 10^4 perturbed and likely incorrect molecules. We then attempted to correct these molecules with BFS as selection policy, which, given sufficient resources, guarantees to find the closest correct molecule. A dictionary containing chemical environments of radius 2 was used. Once a correct molecule had been found the search was allowed to continue until the whole tree level was visited. The maximum tree size was limited to 10^5. Of the 10,000 structures, 1,573 molecules were successfully corrected within these resource constraints, with an average search depth of 2.4 edges. For each vertex along the shortest path between the corrected molecule and the root vertex we measured *d(u,v)*, *f_1(v)* and *d(u,w)* for a total of 3,773 data points which we took as training data. A Multiple Linear Regression (MLR) model was fit on this data (Equation 5.10), resulting in a model with a Root Mean Squared Error (RMSE) of 0.135 (Figure 5.7). As a control we also built the null model $g(v) = \overline{d(u,w)} = 0.383$, with an RMSE of 0.159. Constants can be quite predictive when the response variable has a narrow range. Since our training data is comprised of shallow searches the null model appears unusually predictive. However, constants cannot extrapolate by nature, and therefore the null model won't be predictive for deeper searches. The practical shortcomings of the null model will be showcased later.

Equation 5.10

$$g(v) = 0.42 \cdot d(u,v) - 0.91 \cdot f_1(v) + 1.18$$

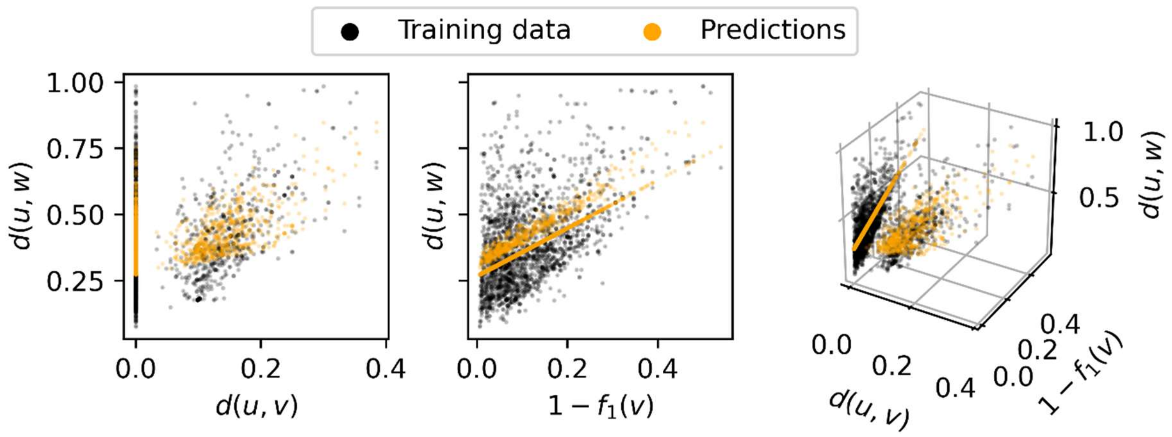


Figure 5.7. MLR model fit to training data. The two leftmost panels show the correlation between each of the model's parameters and the training data/predictions separately, while the rightmost panel aggregates the effects of both parameters.

5.3.3.1.4 Explicit objective preservation

The above-described selection policies try to find correct molecules that are structurally closely related to the input molecule. The primary reason for doing so is that structurally similar molecules are believed to have similar properties [36, 37]. Yet this is not always the case [94, 95]. Two molecules may share a large common substructure and differ in a single atom. While the overall structural similarity between them may be large, if this distinctive atom is key to the molecule's activity their properties may differ significantly.

Given an objective function *o(v)* that evaluates a vertex *v*'s property of interest we can explicitly guide the tree search into preserving this objective as opposed to relying implicitly on the similar property principle [36]. This helps tackle the cases where said principle breaks down. A simple way to do so is selecting for expansion the vertex *v* for which Equation 5.11 is maximal. Note that the objectives are multiplied as opposed to being summed to prevent the search algorithm from sacrificing one objective in favor of the other.

Equation 5.11

$$g(v) = f(v) \cdot o(v)$$

5.3.3.2 *Expansion policy*

A molecule is expanded by applying a perturbation to a copy of itself. Perturbations that are most likely to make the molecule familiar are applied first. Foreign molecular keys are identified and ordered according to their significance. Identifying the most significant foreign key serves as a way of identifying the most pressing problem a molecule has. The location of the problem is given by the location of the key, which is either an atom or a bond. It is this atom or bond that will be targeted by a perturbation.

When it comes to foreign atom and bond keys it's possible to identify not only the location but also the nature of the problem. Partial keys build up on each other by progressively adding properties. Since more significant keys are contained by the less significant ones the latter cannot be familiar if the former are not either. The property differentiating the most significant foreign partial key from its familiar predecessor partial key is responsible for the latter being foreign. For example, the most significant foreign partial atom key may be DVZ = (4, 6, 6), corresponding to a hexavalent carbon. Its predecessor key DV = (4, 6) is necessarily the least significant familiar key. We can then conclude that the atomic number (Z) is not compatible with the atom's degree and valence. Since we deem the atomic number to be less significant than the degree or valence we identify the atomic number as the culprit for the atom key being foreign, meaning perturbations modifying the atomic number will be prioritized.

The predecessor key can also be used to access the chemical dictionary and retrieve acceptable property values for the successor key. These values are sorted according to their frequency in reference molecules in descending order, meaning that the most frequent values are tried first. In the example above we can use the DV key to retrieve elements compatible with an atom of degree 4 and valence 6, which might be sulfur (Z = 16) and selenium (Z = 34). Sulfur is more frequent than selenium, so a perturbation replacing the carbon with sulfur would be prioritized.

Choosing which perturbations to apply to correct Z, Q, H or B is obvious as each of these properties has a corresponding perturbation to change its value. Correcting other properties and keys is less trivial. D is corrected by deleting bonds associated with the atom or deleting adjacent atoms. Depending on the dictionary it may also be possible to correct it by inserting more bonds or atoms, but this is disabled by default, as for organic molecules degrees higher than 6 are exceedingly rare. V is preferably corrected by changing the bond types (i.e. bond orders) of bonds associated with the atom. If this does not succeed it may also be corrected by modifying the topology of the molecule, in the same way one would correct D.

Two atom keys AK may be familiar separately, but their combination in a bond key AK1AK2 may be foreign. If the AK1AK2 partial key is foreign one or both atom keys must be changed. Perturbation types can be ordered by significance similarly to how molecular keys are ordered by significance. The lower the significance of a perturbation the less it will disrupt the molecule when applied. The perturbation significance order matches the atom property significance order (Figure 5.2), being from least to most significant as follows: number of hydrogen changes, formal charge changes, atomic number changes, bond type changes, bond deletions, atom deletions, bond insertions and atom insertions. Less significant perturbations are applied first to disrupt the molecule as little as possible. While deletions do not necessarily disrupt the molecule less than insertions, they typically simplify the molecule. Simple molecules are more likely to be familiar, which is why deletions are prioritized over insertions.

Once all atom and bond keys have been corrected the molecule may still possess foreign atomic environments. Recall that atomic environments are characterized solely by their hash, meaning little information about what makes them foreign is available. Atomic environments overlap, in the sense that the same atom or bond may be a part of multiple environments simultaneously. Knowing the exact boundaries of atomic environments, it is possible to calculate in how many environments a given atom or bond participates (Figure 5.8). We calculate the “foreign environment membership” of atoms and bonds, that is the number of foreign environments they are involved in. Atoms and bonds for which this number is highest are prioritized by perturbations, under the assumption that since they participate in many foreign environments they are likely to be a culprit for the environments being foreign. Ties are broken with the atom- and bond keys’ frequencies, prioritizing least frequent keys. Once a target has been acquired perturbations are executed in order of increasing significance, just like for bond keys.

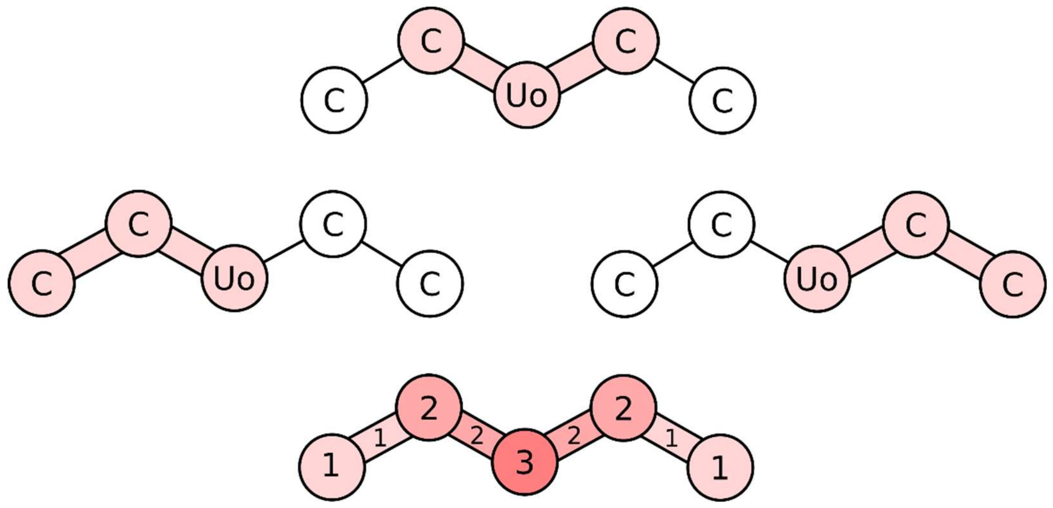


Figure 5.8. Foreign atomic environments and their overlap. The central unobtainium atom (Uo) is foreign. All atomic environments it is a part of are necessarily foreign too. Foreign circular atomic environments of radius 1 are highlighted in pink. The bottom molecule labels each atom and bond based on how many foreign environments they are involved in. The Uo atom is involved in all foreign environments, making it a likely culprit for the environments being foreign.

5.3.4 Constraints

Our molecule auto-correction implementation was developed using the graph-based molecule perturbation library **Molpert** (Chapter 3). One of **Molpert**'s features is the support of user-specified arbitrary constraints perturbed molecules ought to fulfill. This functionality is inherited by the auto-correct implementation, providing the user with fine grained control over the output molecules. Among other things, this allows the user to define properties and/or parts of the molecule that should not be modified by the correction algorithm.

5.3.5 Benchmark

A random sample of 10^3 molecules from ChEMBL31 [136] was taken. **Molpert** (Chapter 3) was used to "break" these molecules by sequentially applying 10 random perturbations to each molecule, resulting in a series of 10 perturbed and likely incorrect molecules. In total 10^4 perturbed molecules were generated. These molecules were sorted by the number of perturbations that gave rise to them. On average, as more random perturbations are applied to a molecule, more foreign keys are generated, decreasing its familiarity. We then attempted to correct these perturbed molecules with our algorithm using the different selection policies described above. A maximum tree depth of 25 and tree size of 25,000 molecules were imposed. A chemical dictionary of circular environments of radius 2 was used for this purpose. The output molecule as well as its familiarity and similarity to the input molecule were recorded. The familiarity provides some measure of how "correct" molecules are. Nonetheless, to better contextualize the quality of the generated molecules we also measured their SAScore [124] and ran retrosynthetic analysis on them with AiZynthFinder [132] using the ZINC [2] reactants stock and USPTO-derived reaction template policy provided by the authors. SAScores were calculated using ChEMBL31 [136] as reference chemistry. Molecules were sanitized prior to calculating their properties.

We investigated two scenarios of how molecule correction may be applied in molecular design (Figure 5.9). In both cases we took the **Molpert** based evolutionary algorithm (as described in Chapter 3), capable of (1) designing molecules without any regard for chemical validity and (2) designing molecules fulfilling specific structural requirements. The algorithm was tasked with designing high-scoring molecules in the goal-directed GuacaMol benchmark suite [108]. As a first scenario (Figure 5.9A) molecules designed without constraints by the algorithm were subjected to auto-correction as a post-processing step using different selection policies, a maximum tree depth of 25 and a maximum tree size of 25,000. For our second scenario (Figure 5.9B), we injected the correction procedure as part of the mutation and recombination operators using the greedy familiarity policy, a maximum tree depth of 10 and a maximum tree size of 100. In both cases we used a chemical dictionary comprised of circular atomic environments of radius 1. 50 replicas were ran for each approach, retaining the best-scoring molecule per replica and benchmark. The different approaches were compared by their designed molecules' benchmark scores and SAScores [124]. Molecules of all 20 benchmarks and 50 replicas were aggregated, for a total of 1000 optimized molecules per approach. Benchmark scores were compared through pairwise Mann-Whitney U-tests [141] with Šidák correction [142]. SAScores were compared with Tukey's Honestly Significant Differences test [167]. $\alpha = 0.05$ was taken as family-wise error rate and significance level for all tests. Statistical tests and post hoc corrections were performed using the SciPy [143] and statsmodels [144] Python packages, respectively.

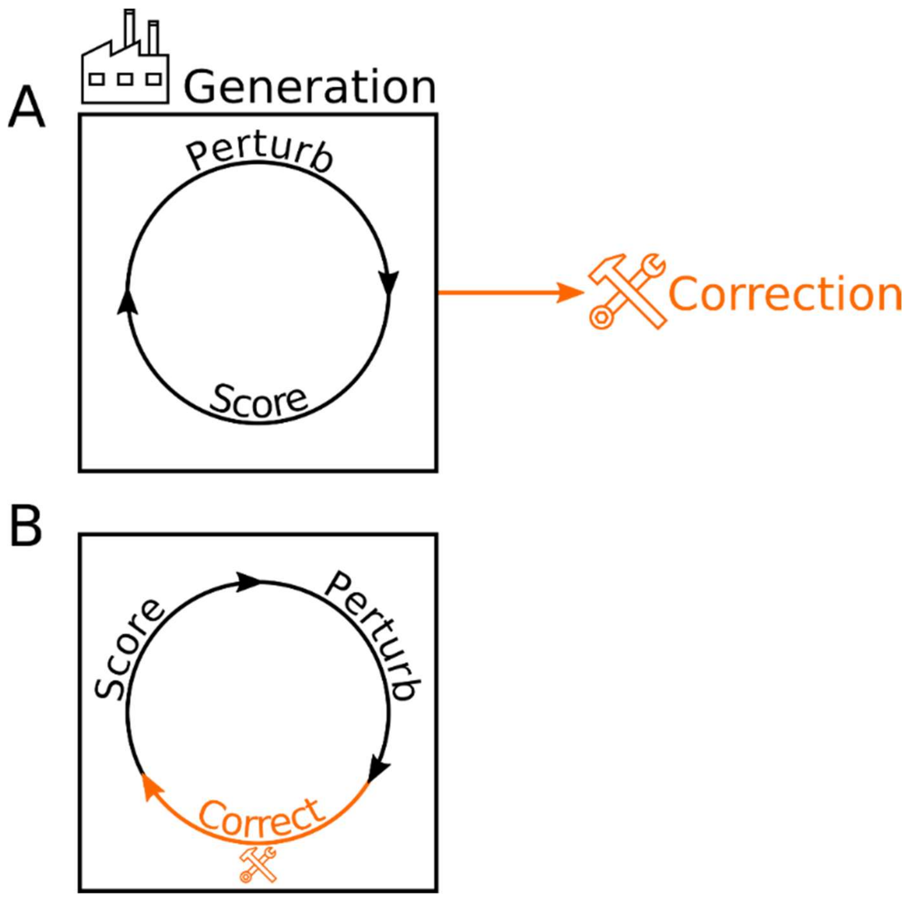


Figure 5.9. Different ways of applying molecule auto-correction in molecular design. It may be used as a final post-processing step of a molecule generator (A) or as an integral part of a molecule generator by injecting it into the molecule construction process (B).

5.4 Results and discussion

5.4.1 Selection policy comparison

Figure 5.10 compares the correction output using different selection policies. We can identify three distinct groups of policies: greedy familiarity, BFS-like policies and MLR. The greedy familiarity policy is very effective at correcting molecules, as virtually all output molecules achieve the maximum f_1 familiarity of 1 and could be considered correct. Moreover, it achieves this with a minimal amount of computational resources. Its biggest drawback, and the reason the other policies were developed, is that it favors deep searches, meaning the corrected molecules may be quite different from the input molecules.

BFS is the benchmark for how close an output molecule can possibly be to an input molecule. Indeed, unless an input molecule is familiar to begin with the output molecule must be different. Greedy distance normalized familiarity, A* and UCT approach this ceiling quite well. Unfortunately, this group of policies also spends more resources on the search, oftentimes to no avail as the output molecule is frequently not entirely familiar.

MLR stands in between the very exploitative greedy familiarity and very explorative BFS-like policies. In our opinion it achieves a good compromise between correcting molecules within reasonable amounts of time while not straying excessively far away from the input molecule.

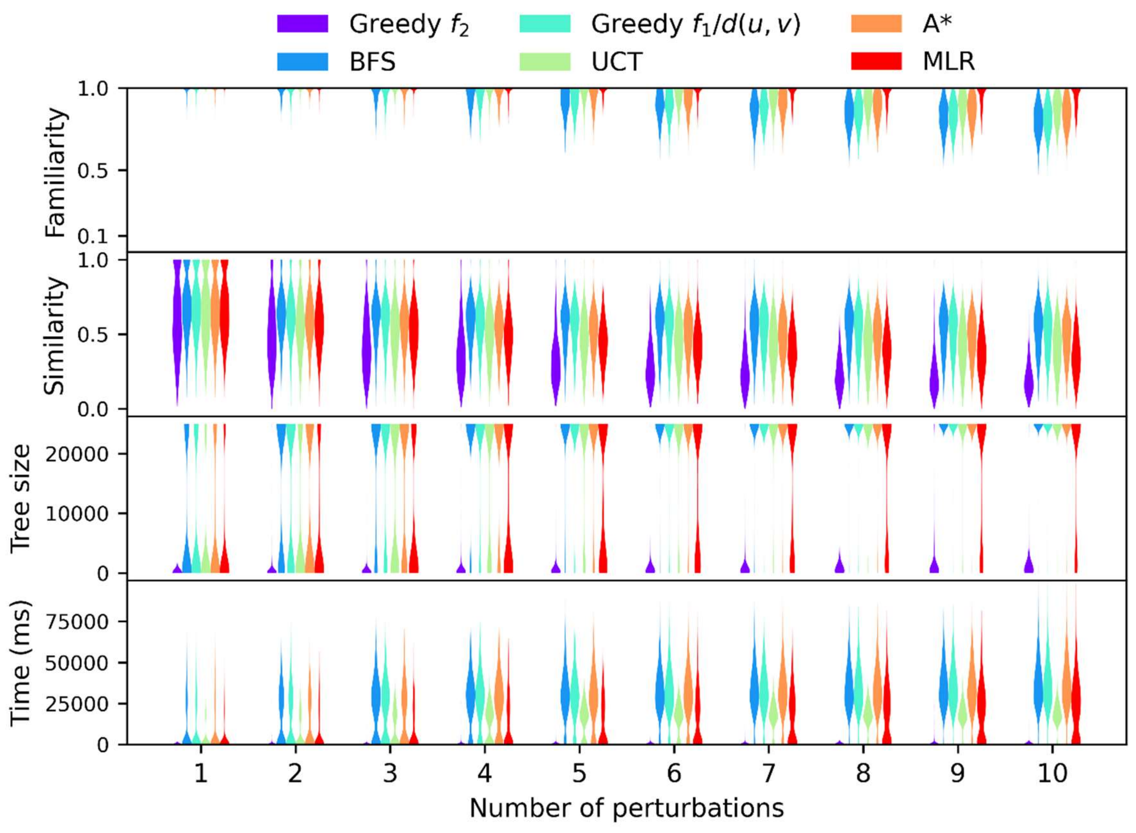


Figure 5.10. Molecule correction benchmark results. The number of perturbations applied to the input molecule is shown on the x axis. The violin plots display the density of output molecules' properties and the cost to generate them. A chemical dictionary with environment radii of 2 was used. For the UCT policy we only display the results of using the optimal coefficient c=0.5. Note that the tree size was limited to a maximum of 25,000. Timings are given for a single-threaded workload on an AMD Epyc 7452 CPU @ 2.35 GHz.

As a control we evaluated replacing the MLR model with a constant null model. Despite the null model fitting the training data well, it cannot extrapolate, leading to poor real world performance (Figure 5.11).

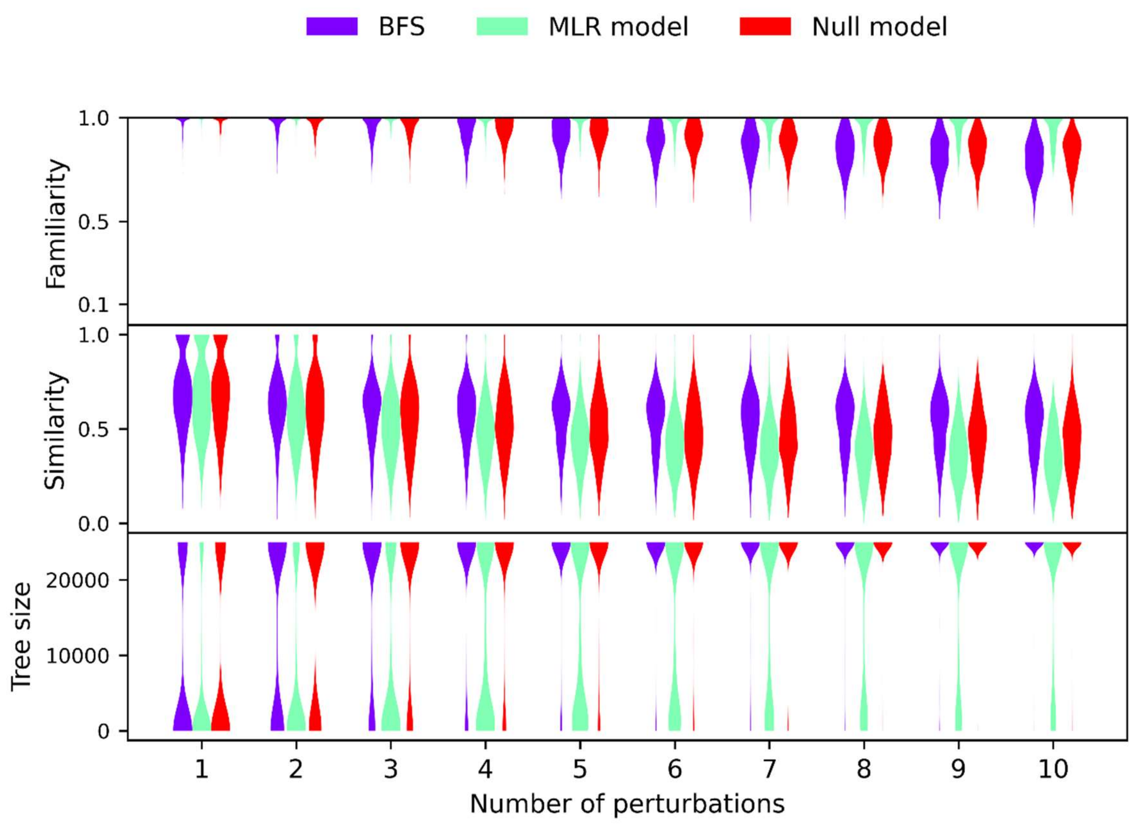


Figure 5.11. Comparison between the MLR model and its null equivalent during the molecule correction benchmark. The number of perturbations applied to the input molecule is shown on the x axis. The violin plots display the density of output molecules' properties and the cost to generate them. A chemical dictionary with environment radii of 2 was used. Note that the tree size was limited to a maximum of 25000.

To further understand the anatomy of the generated trees Figure 5.12 depicts diagrams of the search trees resulting from correcting the same input molecule while using different selection policies. As can be seen the greedy f_2 and MLR policies define narrower and deeper trees than BFS.

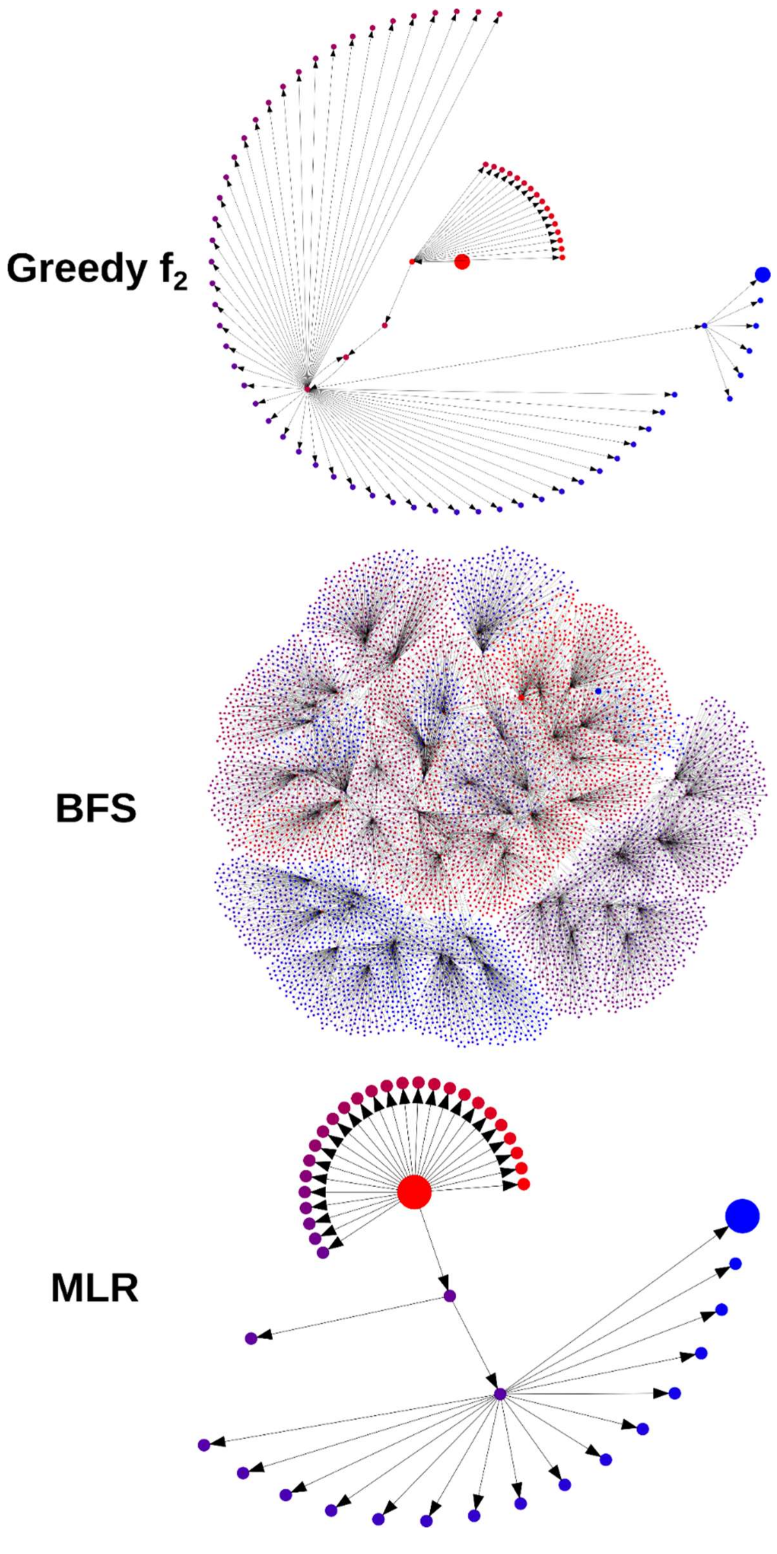


Figure 5.12. Diagrams of search trees resulting from trying to correct the same input molecule (OOC1[C]2#S1C2) using different selection policies and a chemical dictionary with environment radii of 2. Nodes are color coded according to their discovery order, with red and blue being the first and last nodes to be discovered, respectively. The root node is shown as a large red node, and the solution node is shown as a large blue node. The diagrams were created using GraphViz [168].

The chemical quality of the input molecules and the output corrected molecules was assessed using the SAScore [124]. As can be seen in Figure 5.13, applying random perturbations to reasonable molecules makes them progressively harder to synthesize. Encouragingly applying the correction algorithm to these broken molecules largely recovers their synthesizability. As SAScores are rather crudes measures of synthesizability [126] we sought to confirm these findings with retrosynthetic analyses [132]. Figure 5.13 confirms that corrected molecules are indeed easier to synthesize, but for highly perturbed molecules the fraction of synthesizable molecules remains small after correction. The correction algorithm is tasked with finding a molecule that is simultaneously similar to reference chemistry and similar to the input perturbed molecule, which is by design dissimilar to reference chemistry. This is intrinsically a challenging task as both objectives are opposed. Moreover, since the retrosynthesis engine is imperfect, the reported fraction of synthesizable molecules is underestimated, as exemplified by less than 60% of the ChEMBL sample being deemed synthesizable.

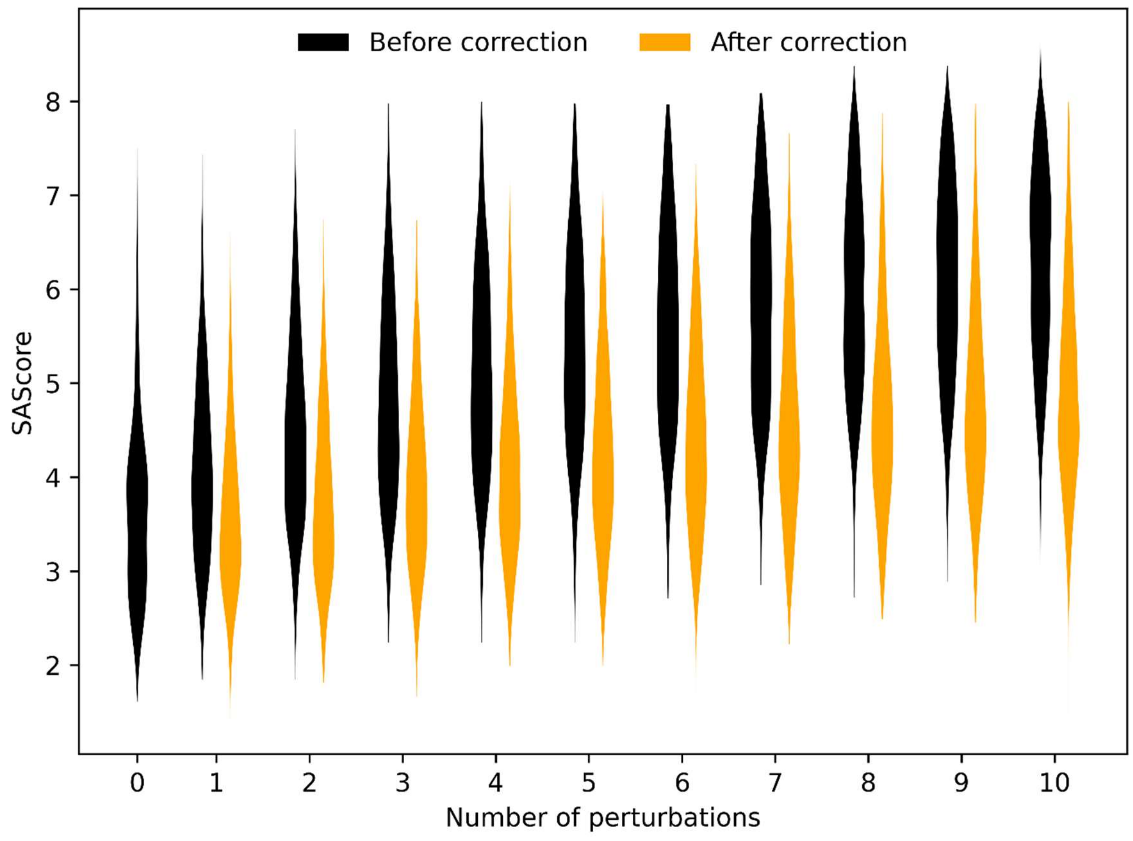


Figure 5.13. Shift in SAScore distributions associated with molecule auto-correction using the MLR selection policy and a chemical dictionary with environment radii of 2. Lower SAScores are indicative of an easier synthesis. The "0 perturbations" distribution corresponds to the non-perturbed ChEMBL subset on which the perturbed molecules were based.

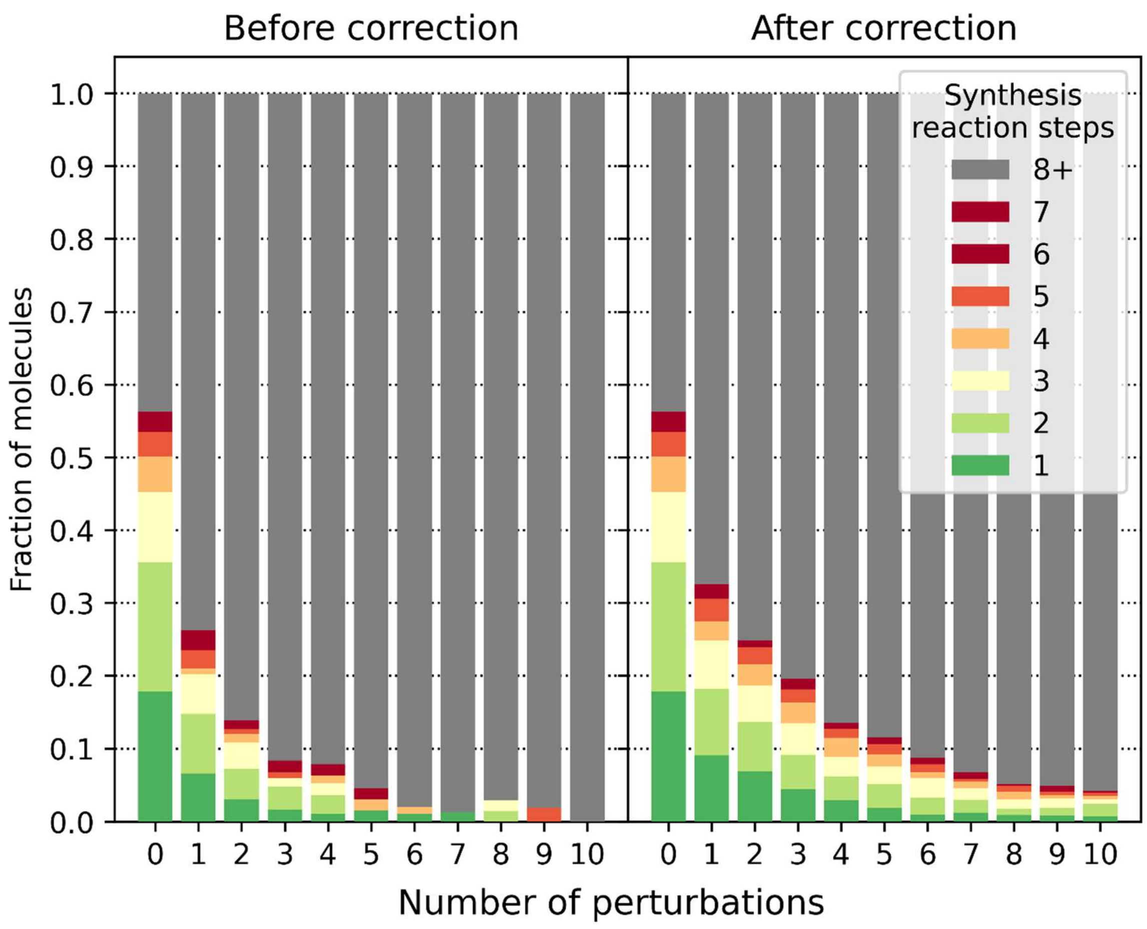


Figure 5.14. Fraction of molecules synthesizable within a certain number of synthetic steps based on the number of random perturbations they were subjected to and whether they were corrected or not. The MLR selection policy and a chemical dictionary with environment radii of 2 were used for the correction process. Retrosynthetic analyses were performed using AiZynthFinder using the ZINC reactant stock and USPTO-derived reaction template policy provided by the authors. "0 perturbations" corresponds to the control ChEMBL sample. Molecules requiring 8 or more synthetic steps are considered unsynthesizable.

5.4.1.1 ***Post-processing applications***

If the user would like to apply the algorithm in a low throughput setting, perhaps as a final sanitization step for the output of a molecule generator (Figure 5.9A), we recommend choosing an explorative policy that yields molecules closely related to the input. If no fitness function is given and resources are infinite, BFS is guaranteed to yield the optimal result, but its cost scales rapidly due to the combinatorial explosion of visited chemical states as the depth of the search increases (Figure 5.4, Figure 5.12). UCT and A* are computationally more reasonable. While both explore approximately the same number of molecules during the tree search, UCT is computationally more efficient as vertices are selected by a fast tree traversal, whereas A* requires a priority queue to be maintained. The MLR policy is a viable alternative on tight budgets. The greedy f_2 policy can be used as fallback should all aforementioned policies fail to find solutions within reasonable amounts of time. We advise raising the ceiling on the maximum tree size as the one we chose for our benchmark is conservative. Since all molecules in the tree are stored in memory in practice the user will likely be limited by the available system memory. Note that memory consumption will be higher when the input molecules are large.

As an example we took molecules designed by a naive evolutionary algorithm during optimization tasks and attempted to correct them using different selection policies. A sample of incorrect molecules designed by the evolutionary algorithm as well as their corrected counterparts are shown in Figure 5.15. Unfortunately the molecules' fitness, as assessed by the optimization task's objective function, was degraded by the correction procedure (Figure 5.16). While all policies performed reasonably well, fitness was preserved best using the explicit objective preservation selection policy. Note however that the objective preservation policy is only applicable when one has access to an objective that ought to be preserved, and when said objective is not excessively expensive to evaluate. Further analysis revealed that fitness degradation was most pronounced in benchmarks whose scores depend on the presence of specific and fragile chemical features (Figure 5.17). As one might expect the correction process can disturb these features which negatively affects the score. For a more hands-on approach to objective preservation one could define molecular constraints to preserve key chemical features. If fitness cannot be preserved during the correction procedure through any means we recommend enforcing molecule validity throughout the construction process instead (Chapter 3, Chapter 4) [74, 75, 166].

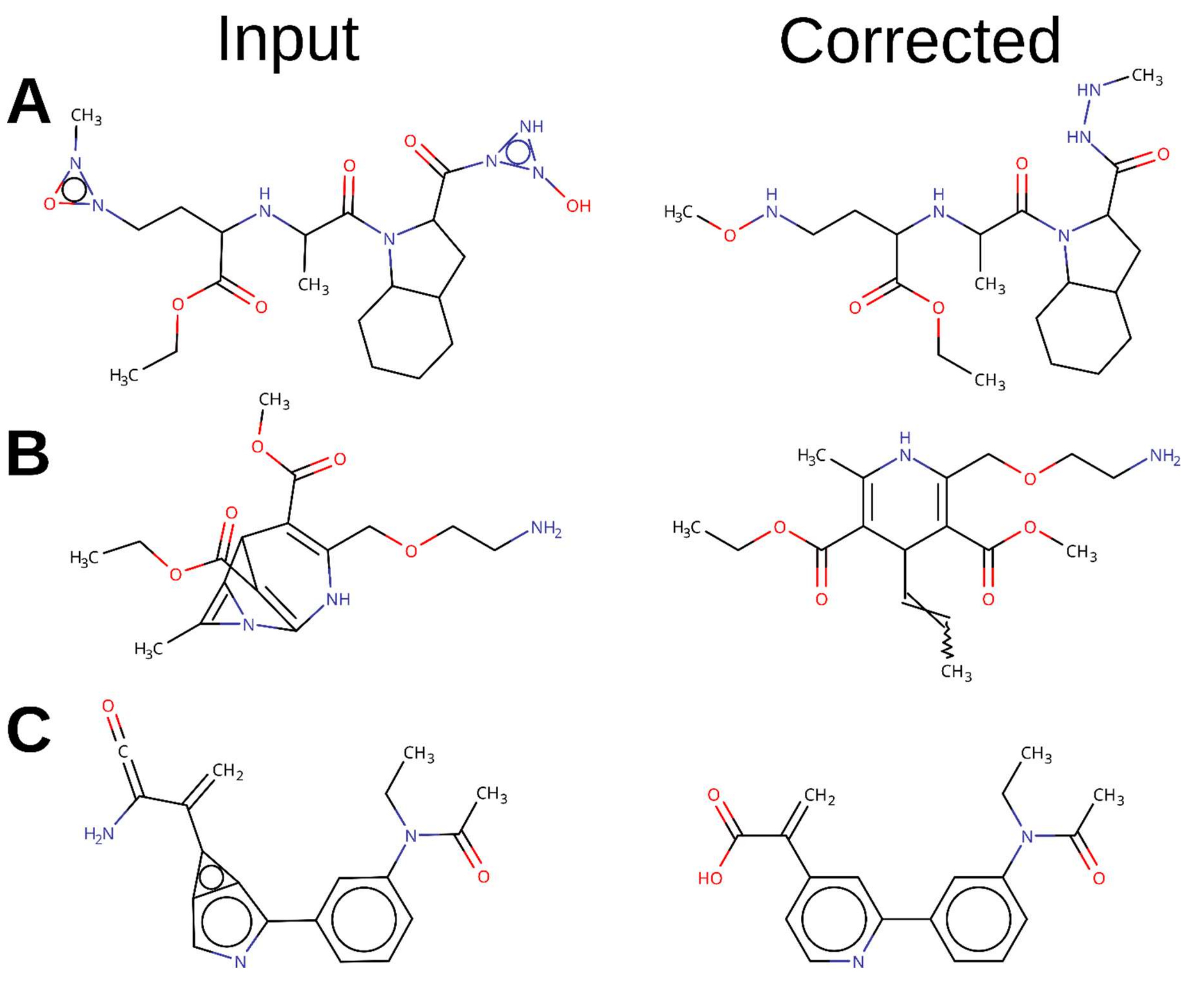


Figure 5.15. Examples of molecules designed by a naïve evolutionary algorithm (left) and their corrected counterparts (right). The MLR selection policy and a chemical dictionary with environment radii of 2 were used for correction. (A) was designed during the Perindopril MPO benchmark, (B) was designed during the Amlodipine MPO benchmark, and (C) was designed during the Sitagliptin MPO benchmark.

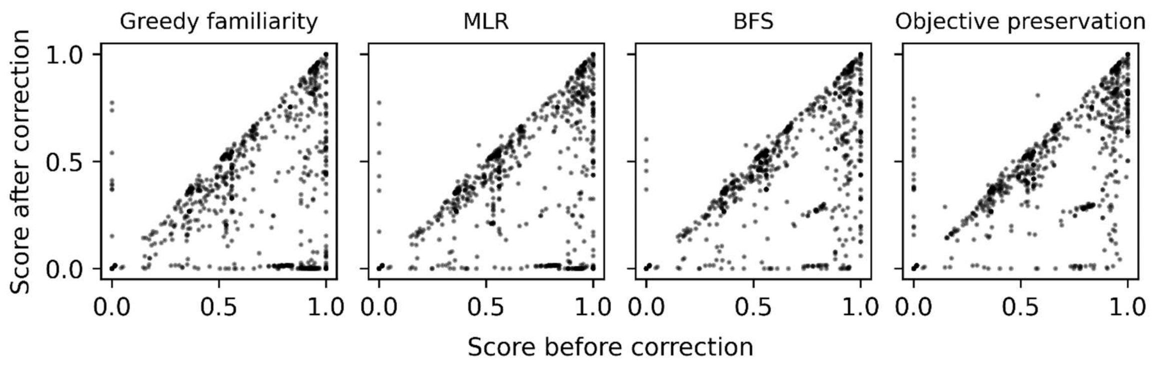


Figure 5.16. Correction algorithm's effect on the GuacaMol benchmark scores using different selection policies and a chemical dictionary with environment radii of 2. Points below the diagonal correspond to molecules becoming less fit. Molecules that were already correct are not included as their score would not change.

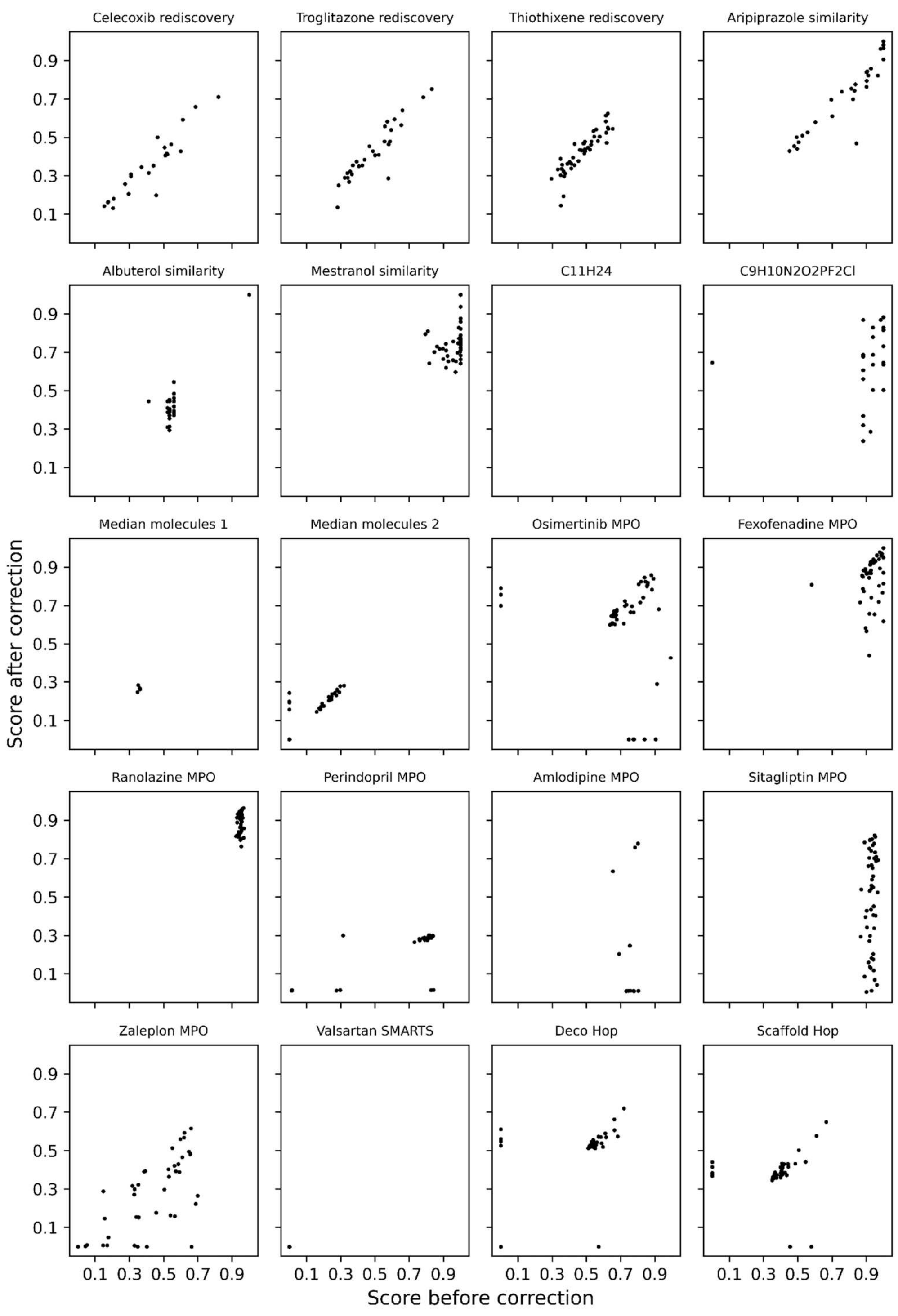


Figure 5.17. GuacaMol benchmark suite score degradation broken down per benchmark. Explicit objective preservation was used as selection policy alongside a chemical dictionary with environment radii of 2. Molecules that were already correct are not included. Benchmarks showing the sharpest score degradation are dependent on specific chemical features and sensitive to molecular modifications. For example, C9H10N2O2PF2Cl, Ranolazine MPO and Sitagliptin MPO require the presence of infrequent elements such as halogens or phosphorus, which may be removed or substituted by the algorithm. Perindopril MPO and Amlodipine MPO require the presence of specific numbers of (aromatic) rings, which are easily broken.

5.4.1.2 *Integrated applications*

If the user intends to apply molecule correction iteratively to very large quantities of molecules, it is advisable to use a cheap and exploitative policy such as the greedy familiarity policy. While output molecules may not closely resemble input molecules, sometimes this is not of great importance, and sometimes it may even be beneficial. Consider a molecular design algorithm that iteratively perturbs molecules to optimize some objective function. One could attempt to correct every intermediate molecule as part of the main loop (Figure 5.9B). In this case the correction would act as an integral part of the perturbation itself, essentially increasing the step size of the perturbation. This may help the algorithm in escaping local fitness minima. Even if the correction process decreases the input molecule’s fitness, the optimization algorithm would presumably correct for this by discarding the molecule, reverting to an earlier stage or focusing its attention elsewhere. It should also be noted that if one were to correct iteratively the distances traversed by correction would match those of input molecules with a single perturbation, which are not as dramatic as those observed for highly perturbed input molecules (Figure 5.10). Occasionally the correction process may effectively undo the effect of the perturbation that preceded it. While we do not anticipate this to be a large concern for most applications one could prevent it from happening using constraints.

To demonstrate the latter approach we injected the correction algorithm into the aforementioned evolutionary algorithm (Figure 5.9B). The greedy familiarity policy with a maximum tree size of merely 100 was chosen to limit computational expenses. Figure 5.18 shows that injecting molecule correction into existing molecule generators is a viable strategy to design molecules that are both fit and easier to synthesize compared to unconstrained molecular design. It should be noted that correction-associated synthesizability improvements are meager due to the GuacaMol benchmarking suites’ scoring functions being biased towards synthesizable molecules [108, 166]. Interestingly, iterative correction yielded better results than attempting to enforce environment correctness through molecular construction constraints (Figure 5.18), and it did so consuming less computational resources (Figure 5.19). We hypothesize that the correction procedure, being unlinked from the objective function, may drag molecules out of local fitness minima aiding the optimization algorithm in the search towards the global minimum.

For completeness’ sake the above experiments and analyses were repeated for atomic environments of radius 2. Under these conditions the correction injection approach failed to improve the synthesizability of the designed molecules, likely because the maximum tree size of 100 is insufficient to find molecules that satisfy the more stringent requirements (Figure 5.20).

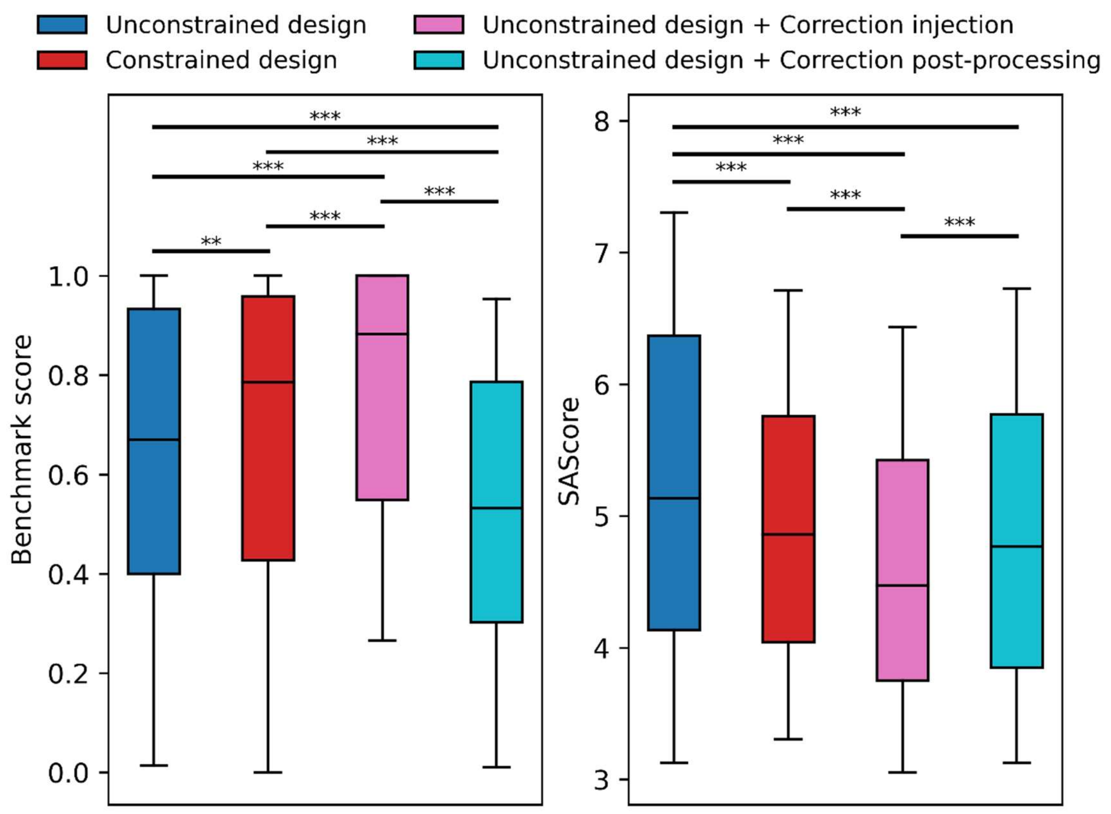


*Figure 5.18. GuacaMol benchmark scores and SAScores of molecules designed by an evolutionary algorithm. Higher benchmark scores and lower SAScores are better. The objective preservation policy was used for post-processing. Unconstrained design refers to liberal modification of the molecular graph and the design of (likely) invalid molecules. All other approaches strive to design molecules with familiar circular atomic environments of topological radius 1, but achieve this goal in different ways. Constrained design refers to the use of molecular construction techniques that prevent the creation of undesirable chemical features. **: $p < 0.01$, ***: $p < 0.001$.*

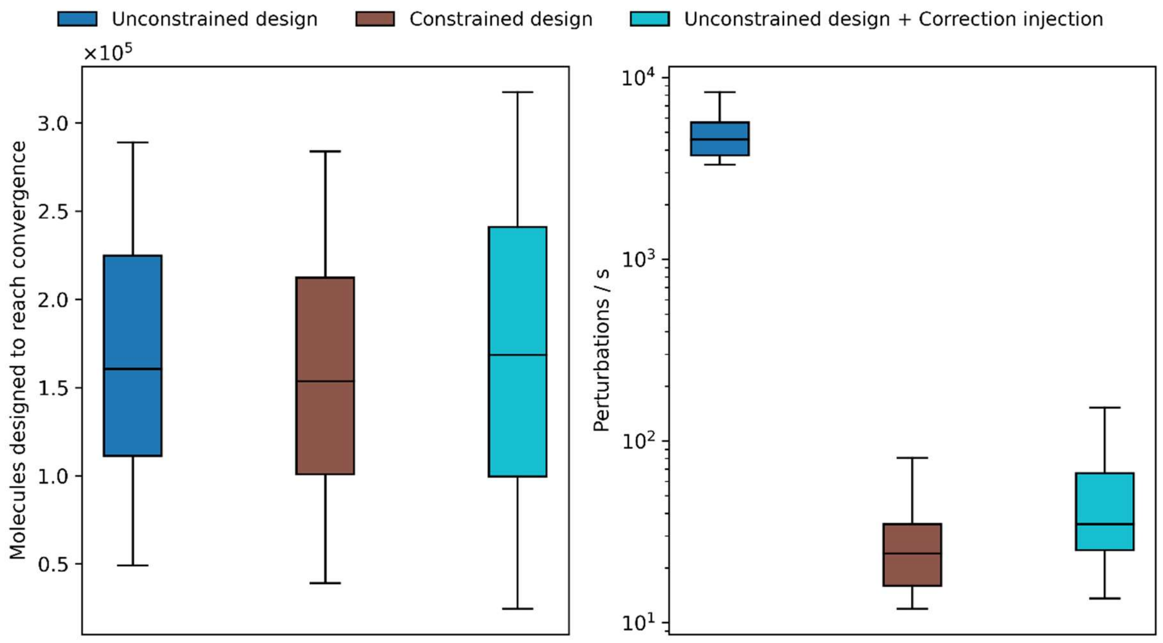


Figure 5.19. Computational cost of designing molecules using different variants of the same evolutionary algorithm. Unconstrained design is the fastest but may result in chemically invalid molecules. The two other approaches both result in molecules with familiar atomic environments of radius 1. Despite achieving this goal in different ways their cost is comparable. Timings are given for a single-threaded workload on an AMD Epyc 7452 CPU @ 2.35 GHz.

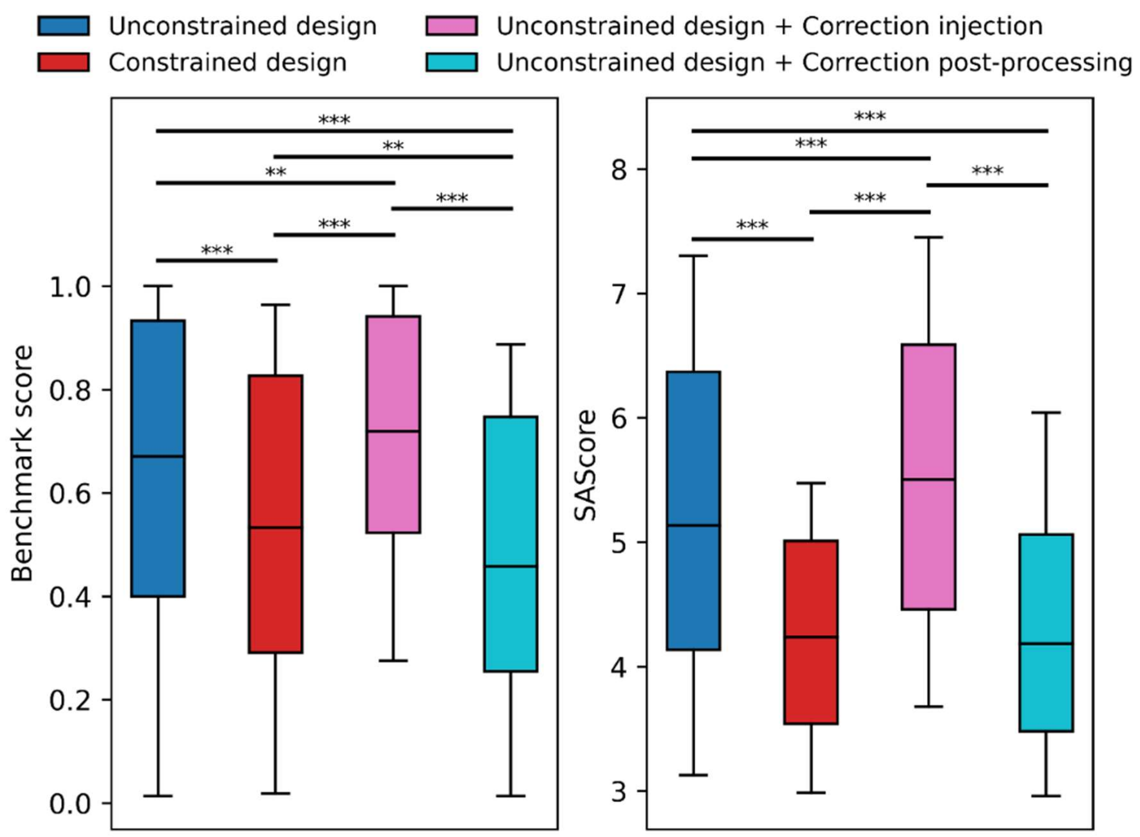


*Figure 5.20. Benchmark scores and SAScores of molecules designed by an evolutionary algorithm. This figure is analogous to Figure 5.18. In both cases the designed molecules were forced to exhibit familiar circular atomic environments, with the key difference being the radii of said environments: 1 for Figure 5.18 and 2 for the present figure. The objective preservation policy was used for post-processing. Unconstrained design refers to liberal modification of the molecular graph and the design of (likely) invalid molecules. Constrained design refers to the use molecular construction techniques that prevent the creation of undesirable chemical features. **: $p < 0.01$, ***: $p < 0.001$.*

It should be stressed that given the same input molecule not all policies will generate the same output molecule (Figure 5.21). It might be of interest to apply the algorithm with different policies and *a posteriori* select the most desirable output.

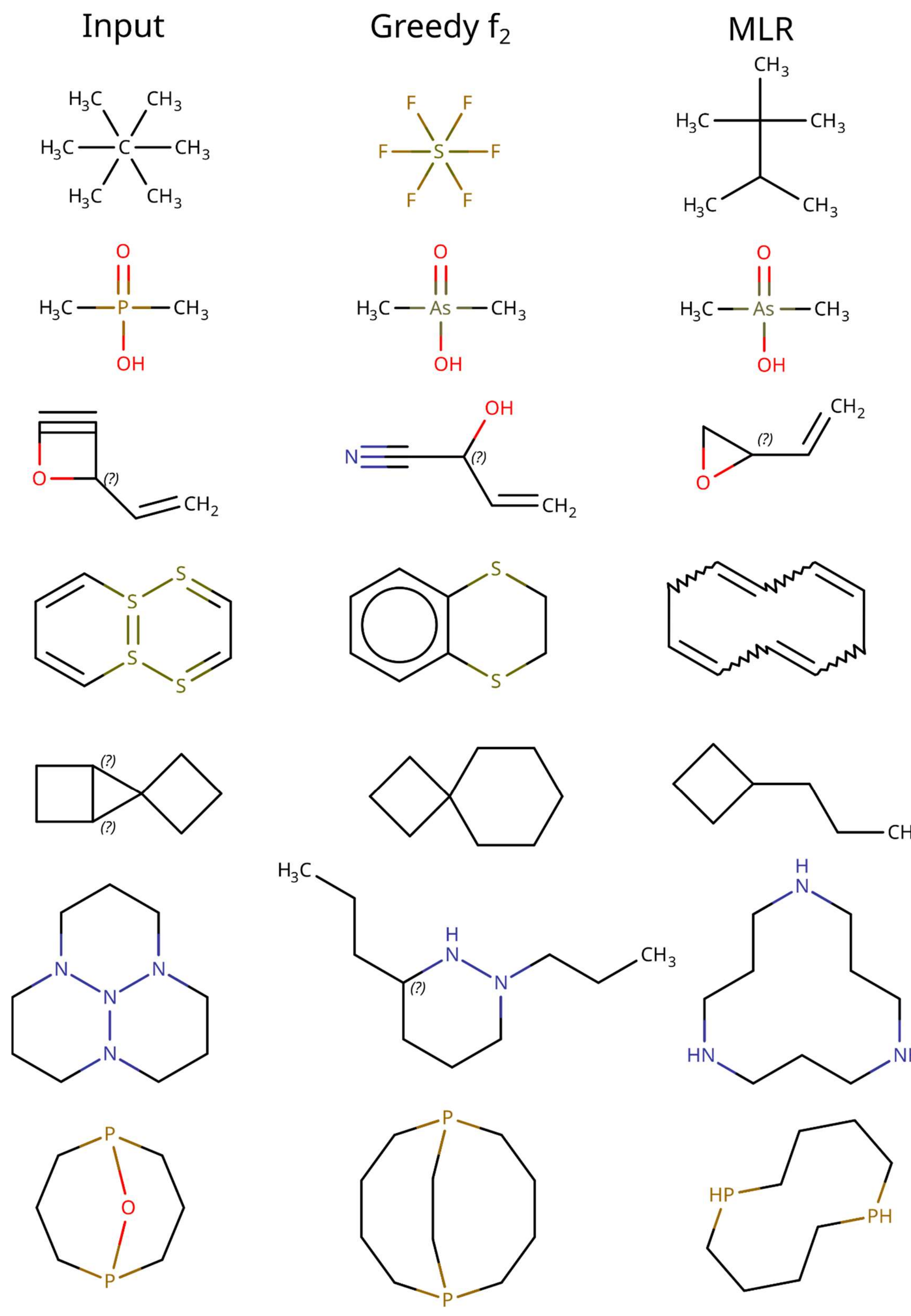


Figure 5.21. Example input molecules and their corrected counterparts using the greedy f₂ and MLR selection policies. Note that molecule correctness is dependent on the reference chemistry library. Some molecules such as dimethylphosphinic acid may be deemed correct or incorrect depending on this context.

5.4.2 Simplification and carbonization of molecules

An unintended consequence of our expansion policy is the "carbonization" of input molecules. Perturbations most likely to increase the familiarity of a molecule are prioritized. As carbon is the backbone of organic chemistry, including our reference library of ChEMBL [136], substituting other elements with carbon is preferred by the algorithm. We also encountered cases where certain selection policies would trigger the growth of long alkane chains, particularly exploitative policies such as the greedy f_1 policy (Figure 5.22, Figure 5.23). We would like the correction process to modify existing chemical features. However, a trivial way of maximizing the f_1 familiarity is by adding new familiar chemical features like alkanes (Equation 5.3). This is a classic case of a search algorithm finding unintended ways to exploit the objective function. Frivolously adding carbons has been described previously as a strategy employed by algorithms to cheat their way to good benchmark results, be it by artificially inflating molecular diversity [161] or reaping low-hanging scoring function rewards [92, 169]. The easiest solution to the issue is to maximize the f_2 familiarity instead (Equation 5.4). While this prevents alkane growth, the search algorithm may occasionally still find it advantageous to introduce extraneous carbons as buffers between heteroatoms (Figure 5.23). Correct heteroatom arrangements are tied to specific functional groups. Given a foreign functional group the path of least resistance may be to break apart said group as opposed to rearranging its atoms. The best carbonization remedy is to choose an explorative selection policy. Should this not be an option the user may choose to disable atom insertions as a perturbation or specify constraints on which parts and/or attributes of the input molecule should be preserved by the correction algorithm.

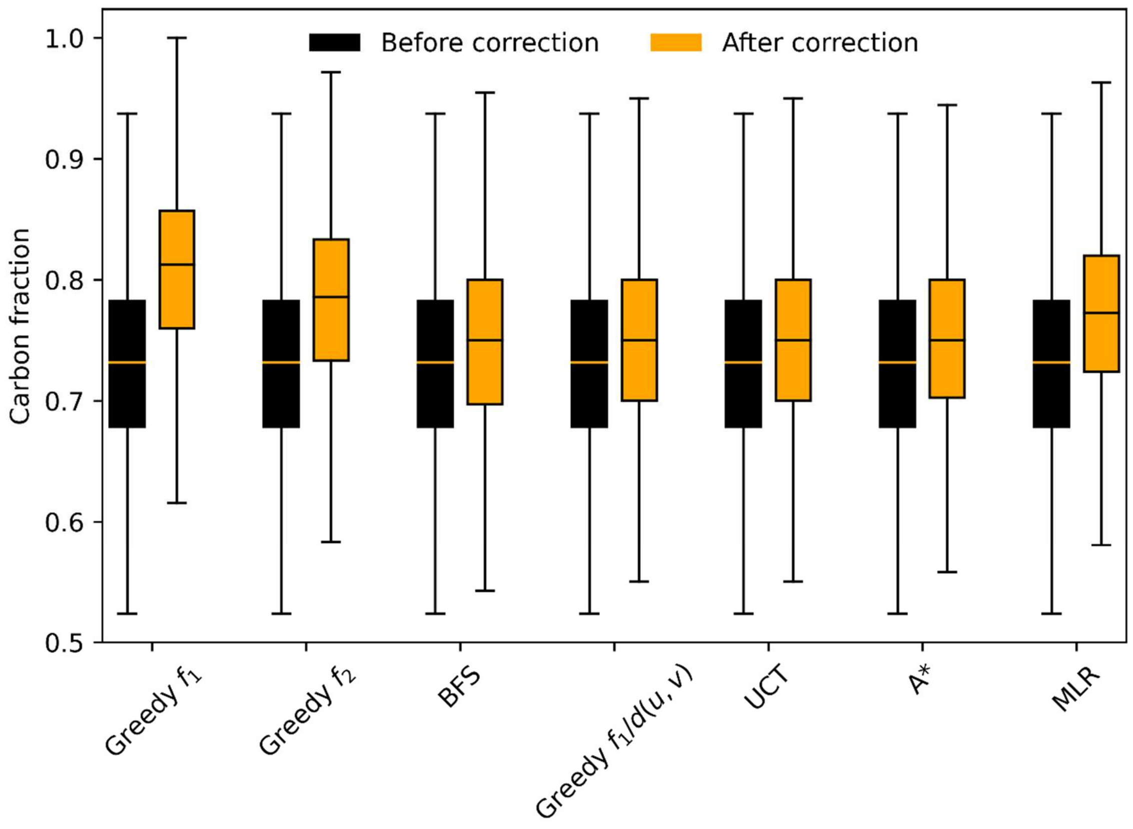


Figure 5.22. Fraction of a molecule's atoms that are carbons, before and after molecule correction using different selection policies. The most exploitative selection policies increase the carbon fraction the most.

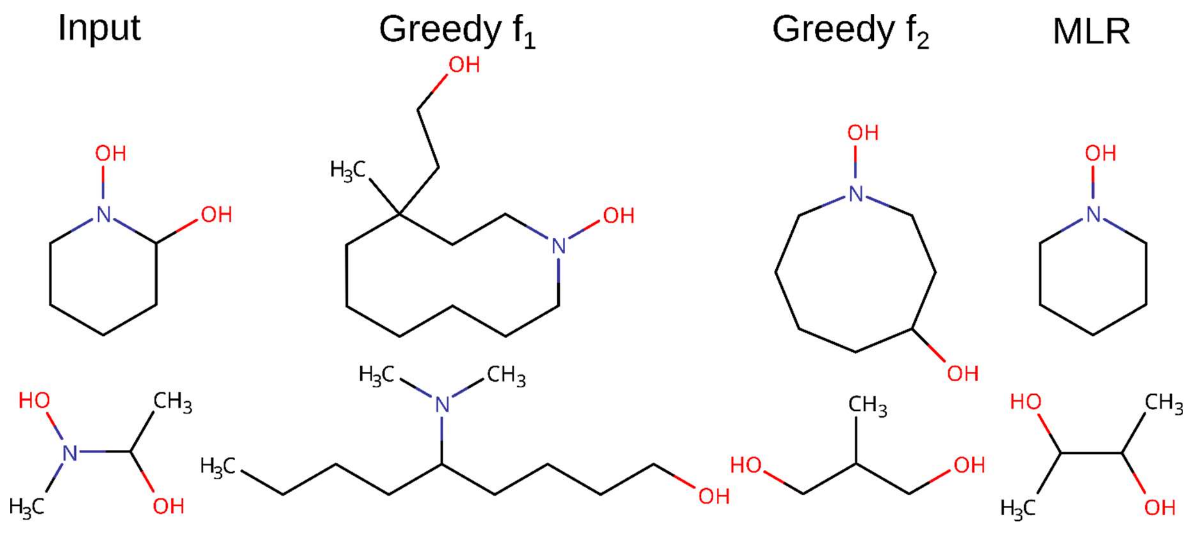


Figure 5.23. Molecule carbonization examples. The greedy f_1 selection policy exploits the scoring function by growing long alkane chains. The other selection policies cannot exploit the scoring function in the same way, but the expansion policy still may opt to substitute heteroatoms with carbons or to separate heteroatoms by inserting carbons between them.

5.4.3 Alternative potential applications

While it is possible to post-process molecules from arbitrary sources, it might not be possible to integrate the correction process into all molecule generators. We have shown how to inject it into a graph-based evolutionary algorithm, and we anticipate equivalent implementations and benefits being achievable for any molecule generators that iteratively modify molecular graphs. Integration opportunities with alternative generators are more nuanced. The algorithm's input is a molecular graph. Our implementation is based on the RDKit [47], which means that molecules must be parsable by the RDKit to be correctable. This precludes the use of ill-formed SMILES [170]. Ill-formed SMILES can be the product of malfunctioning generative models. They may also be an intermediate state of generative recurrent neural networks [92]. In the latter case correction would have to be deferred until the SMILES string has been fully formed, potentially playing a role in sanitizing molecules prior to their objectives being evaluated. Substituting SMILES for a more robust line notation such as SELFIES [171] whose intermediate strings are also valid would enable the "auto-correct" process to behave more as a molecule "auto-complete". In any case the correction process would play a role in steering the chemical space search. Whether this would antagonize or synergize with the model's inherent guidance remains to be explored.

5.4.4 Future perspectives

Caution should be applied when employing molecule generators that rely on the similarity principle, for they amplify existing chemical biases in data due to prior art data conditioning future data collection [75, 172]. This can have detrimental effects on chemical novelty. The problem is compounded by building pipelines of tools relying on the same principle, as we do in this work. We are aware this is suboptimal, but in absence of competing methods grounded on physical first-principles chemical bias amplification postures itself as a necessary evil.

One area worth revisiting in the future is the way in which correctness is assessed. Currently molecular keys are considered either foreign or familiar, depending on their frequency in the chemical dictionary. While the frequency threshold separating both categories can be tweaked, it would be preferable to treat familiarity as a frequency-dependent continuous variable. We also believe there is potential in further development of selection policies. The policies explored herein rely on crude heuristics. We can draw inspiration for policy design from other fields where tree searches are used. Synthesis planning in particular has recently witnessed major breakthroughs thanks to machine learning augmented policies [130, 131]. We believe that similar methods could be applied here to better direct the search, reducing the risk of missing good solutions as well as the cost to find said solutions.

Chapter 6 Discussion & future perspectives

6.1 Imitating reference chemistry

Designing reasonable molecules *in silico* is a non-trivial task. Just attempting to define what constitutes a reasonable molecule is sufficient to trigger heated debates. A chemist will know an undesirable molecule when they see one, yet their perception may differ from that of other chemists [121–123]. Even if some form of consensus could be found, formalizing chemical intuition into logical and mathematical constructs that computers can understand is challenging. This stems from the fact that humans and computers process information in fundamentally different ways.

Human chemical intuition is mostly exclusion based. Throughout their careers chemists will see and make countless molecules, taking mental notes about which functional groups and ring systems have undesirable properties or are difficult to work with. Some have tediously enumerated (incomplete) lists of bothersome chemical motifs which can be used to filter out molecules exhibiting said motifs [173, 174]. Such a workflow may be suitable when working with compound libraries, be they virtual or not, but is of limited use when designing new molecules. How does one generate molecules without objectionable chemical motifs? One could generate large amounts of molecules and discard the unsavory ones. But how many savory molecules would be left? Filtering is a wasteful process resource-wise. Making matters worse features that humans can easily identify visually are expensive to identify for computers [31]. We should strive to bias molecule generation towards desirable molecules, but this requires defining that which is desired, not that which is not.

Humans may struggle formalizing what constitutes desirable chemistry, but computers can systematically identify patterns in molecules that have historically been deemed desirable. The vast majority of approaches towards computational molecular design mimic reference chemistry in some form, and the methods described herein are no exception. Distinctions are found in which patterns are identified and how they are reproduced in generated molecules.

In this work we explored different ways of describing and replicating reference chemistry. In all cases the molecular representation was a graph whose vertices and edges were iteratively modified by heuristic optimization algorithms. The differences reside in the granularity of the vertices as well as the rules governing how these vertices may be connected. A natural question that arises is which molecular description and assembly rules are best suited for molecular design.

6.2 Molecular graph granularity

Let us begin by discussing the granularity of graph-like molecular descriptions. A vertex represents a molecular substructure. These substructures can vary in size, from single atoms (section 3.3.2), to multi atomic fragments (section 4.3.1) such as entire Bemis-Murcko ring systems, linkers and side chains [175]. The larger the granularity of molecular descriptions, the greater the resemblance between designed molecules and reference chemistry. It is widely believed that this helps in the design of synthesizable and drug-like molecules. Indeed, fragment-based design is the go-to strategy for cheminformaticians aiming to design synthesizable molecules [22, 176]. While there is certainly some truth to the underlying belief it paints an incomplete picture. Fragment-based design is almost always coupled to rules governing how said fragments can be assembled. With our experiments we tried to separate the effects of using fragments from the effects of the assembly rules. We found that using fragments on its own has little effect on the synthesizability of designed molecules, and that the bulk of the synthesizability improvements can instead be attributed to molecular assembly rules (section 4.4.4). Constructing molecules as combinations of large fragments reduces the number of bonds that must be formed, as the majority of bonds are pre-formed in the fragments being combined. Hence the use of fragments can occlude some issues underlying bond formation, but it does not solve them. It only takes a few poor bonds for the designed molecule to be non-sensical.

Fragment-based design restricts access to some states in chemical space (section 3.2). This is a double-edged sword, as it offers both benefits and drawbacks. On the bright side, fragment-based design can be a powerful way of steering molecule generators towards relevant regions of chemical space. Theoretical chemical space is vast, with large parts of it being irrelevant for drug design. Using drug-like fragments can help in narrowing down the search space (section 4.4.4). Moreover modifying molecules on a fragment level is associated with bigger steps in chemical space. This larger step size can be beneficial to some optimization algorithms, as it may enable leaping over or escaping local fitness minima.

As for the drawbacks, not being able to access the entirety of chemical space means that some of the best solutions to a problem may not be discoverable. The problem is accentuated when using small fragment libraries. Using very large and diverse fragment libraries could help in this regard, but even if a molecule is theoretically discoverable as a combination of fragments, accessing it might require convoluted sequences of fragment operations that are statistically improbable. One ought to ponder how to prioritize fragments such that the additional data does not pose an obstacle (sections 4.3.4.2 and 4.3.5). Ultimately fragment-based design is likely to be more resource intensive than atom-based design, both during development and run time.

So should one employ fragment-based design? As frustrating of an answer it may be: it depends. Fragment-based design has merit as a part of directed search and/or synthesizability improvement strategies. The price to pay is restricted access to chemical space, inferior chemical novelty and increased computational complexity and development efforts. Personally I deem this price excessive for the purported returns. I believe the ceiling on atom-based design to be higher, but fulfilling its potential will require the development of better molecule characterization and assembly rules. Fragment-based design shines the brightest when the fragments being used *in silico* represent real reactants, and the rules for their combination mimic real reactions. In this case there is a direct correlation between designed molecules and synthetic routes, which enables tight integration with the medicinal chemistry laboratory.

6.3 Molecular graph modification rules

Designed molecules must fulfill certain criteria such as synthesizability. One strategy to enforce these criteria is constraining the way in which molecules are assembled or modified. The way in which molecule generation is constrained has far reaching implications for the output molecules' objective values. This is especially true for molecule generators relying on iterative modification of molecules, as in our case.

In this work we tried to modify molecules in such a way that the resulting molecules resembled reference chemistry. By choosing as reference chemistry a library of synthesizable and drug-like molecules [136] we biased our designed molecules towards being synthesizable and drug-like as well. Despite being distinct objectives there is significant overlap between them as the molecules that are recorded in chemical databases tend to be both.

The success of the mimicry strategy hinges on several factors, with the most important one being the way in which molecules are characterized. The basis for our characterizations were atomic invariants, which were used to describe individual atoms. Characterized atoms were in turn grouped to describe bonds, and the latter were grouped to describe atomic environments (sections 3.3.5, 4.3.1, and 5.3.1). As the coarseness of the molecular characterization increases the corresponding chemical features become more unique. For instance, bonds are bound to be more unique than atoms, since bonds comprise two atoms. With features becoming more distinctive the requirements imposed on designed molecules become stricter, as designed molecules ought to resemble reference chemistry in more concrete ways. Overall this translates into an increased similarity between designed and reference molecules, and with it an increased chemical appeal of the designed molecules (sections 3.4.2 and 4.4.2).

As requirements become stricter, less molecules fulfill them. If all intermediate molecules ought to satisfy these criteria, as is often the case, molecule generators may start to struggle morphing one molecule into another. Restricted chemical space traversal can manifest itself in the deterioration of the designed molecules' objective values. We found this to be the case when enforcing excessively strict requirements. On the other hand, mild requirements did not affect chemical space exploration negatively. On the contrary, they could provide some much needed guidance to search algorithms (section 3.3.8.1).

Another important element to consider is the reference virtual library. It should be representative of the types of molecules one would like to design. The reference library should be chosen in conjunction with a method to assess query molecules' conformity with reference chemistry.

In some instances we treated conformity as a binary property (sections 3.3.5, 4.3.2, and 5.3.2). For example, we deemed chemical features and connections valid if they occurred at least a minimum number of times in the reference library. We chose zero as our default validity frequency threshold. In this case one should ensure that the reference library does not contain a single example of bad chemistry, which may require careful inspection. Raising the frequency threshold is a tempting proposition, but some crucial chemical features and/or connections may be infrequent by nature. Consider methane as an example, whose atomic environment is a carbon of null degree. This atomic environment is exclusive to methane, and will have a maximum frequency of one if methane is included in the reference library. Despite being infrequent, this environment is an essential stepping stone to bootstrap the growth of molecules from vacuum. Failure to include methane in the reference library, or neglecting its environment due to its low frequency, would make the purest form of *de novo* design impossible, and impose the necessity of seeding molecules for optimization.

In other instances we treated chemical conformity as a continuum, with molecules being better conforming when their chemical features are common in reference chemistry (sections 3.3.6, 4.3.4.2, and 5.3.3.2). We used this approach to bias molecule generation towards the most common (and presumably most desirable and/or easiest) types of organic chemistry by sampling molecular structures and properties with probabilities proportional to their frequencies in reference chemistry. Such an approach is arguably more natural as well as more tolerant of small amounts of bad chemistry making their way into the reference library. Unfortunately it is also less straightforward to implement algorithmically. As an additional consideration one should ensure that chemical feature frequency imbalances represent real medicinal chemistry preferences, instead of stemming from systematic synthesis efforts. A database could be flooded with combinatorial chemistry products, which could artificially make other types of chemistry seem undesirable.

6.4 Alternative ways of enforcing molecular requirements

Given a molecule generator that optimizes molecules by iteratively modifying them, one strategy to generate valid optimized molecules is ensuring that all intermediate molecules are valid as well. This approach has been discussed extensively in this work (Chapter 3, Chapter 4). However, it's not the only viable strategy to achieve this goal.

One could explicitly incorporate molecular validity into the objective function(s) the molecule generator optimizes for (section 4.4.6). Ideally, if the generator supports multi-objective optimization, one would do so in form of a separate objective. Unfortunately, real multi-objective optimization is challenging, requiring sophisticated search algorithms. A simpler, and sometimes surprisingly effective approach, is to use single-objective optimization algorithms with composite objective functions. The latter approach is very flexible as one can tweak the objectives being combined, as well as the way in which they are combined, to fine-tune the requirements designed molecules must fulfill. Regardless of the chosen search algorithm it is up to the user to define the relative importance of the different objectives, be it by picking solutions from a Pareto front or by defining coefficients of a composite scoring function. A major drawback of explicit multi-objective optimization is that one must evaluate all objectives for every (intermediate) molecule the generator proposes. This is only computationally tractable for cheap objectives, which may not be significantly more sophisticated than the requirements enforced through construction techniques. We explored the use of composite scoring functions as a control experiment and found both approaches to be competitive (section 4.4.6).

An alternative approach explored herein is the adaptation of molecules to make them conform better with imposed requirements (Chapter 5). A key advantage of this solution is that, once developed, it can integrate into most molecule generation workflows with almost no user effort. The approach was observed to be viable in a post-processing scenario, and to excel when injected into a molecule generator. Encouragingly, despite being one of the lowest effort methods we tested, it performed among the best. To the best of our knowledge this sort of approach had not been explored previously, making this the first proof-of-concept study. We applied the technique to "correct" molecules, but with some tweaks one could potentially apply it to improve other objectives.

Lastly, we would go amiss if we did not mention machine learning-based generative models. Machine learning models can extract and reproduce patterns from reference chemistry. The use of generative models for molecular design is still in its infancy, with the first arguably notable applications dating back to 2017 [92]. Despite being rather immature, it has taken the world by storm and has renewed interest in molecular design.

6.5 Machine learning in generative chemistry

Traditional methods such as evolutionary algorithms have largely been displaced by generative chemistry models. It is easy to understand why: generative models are new and exciting, and they solve the poor synthesizability problem in a relatively simple and elegant way. Why bother with codifying complicated chemistry when a model can autonomously learn what molecules are supposed to look like and reproduce it? Few studies have pitted molecule generators of different classes against each other, but those that did have shown that traditional methods can be competitive with generative models [108].

I must confess I remain a bit skeptical about generative models. For starters, it's well known that machine learning models have an applicability domain beyond which their predictions or inventions are unreliable. If said models are trained on existing chemistry, can we trust them to be able to generate novel chemistry? Many authors report methods capable of designing novel molecules, but the way in which novelty is assessed is often based on chemical identity, which has been shown to be a flawed novelty metric [161]. Some, at first glance promising, molecules produced by generative models [177] have been shown to be concerningly similar to commercialized drugs.

More worryingly, despite progress being made in so-called explainable artificial intelligence [178], as of today the vast majority of deployed models are black boxes. A model might function excellently and propose fabulous molecules, but to an academic the "how" should be equally important as the "what". On the flip side, if a model malfunctions and our understanding of its inner workings is poor, will we be capable of improving it further?

I'm under the impression that, when faced with underperforming models, a significant part of the generative chemistry community sees "more data" as the primary avenue for improvements [179]. I perceive this akin to "kicking the can down the road", and I'm skeptical that the envisioned scenario where "more data" solves our problems will ever materialize. Collecting large quantities of high quality chemical and biological data that can be reliably aggregated and compared is challenging [180–182]. Enabling this type of data collection would require massive changes to existing workflows and standardization efforts that are too disruptive to be adopted. Naturally the present work also relies on data to define the rules of chemistry, and is subject to many of the same caveats generative models are. Yet there are some key differences: (1) we know what sort of patterns are being extracted and reproduced and can therefore guarantee certain degrees of chemical correctness and novelty, (2) the patterns are less abstract and structurally confined, and (3) the number of patterns to collect is finite, placing an upper boundary on the amount of data required for our methods to perform optimally. These differences constitute arguments in favor of not replacing traditional molecular design with generative models entirely.

Another point of concern is the field’s insistence on representing molecules as text strings. In the early days representing molecules as sequences of characters allowed researchers to borrow techniques from natural language processing and make rapid progress [91, 92]. The first, and probably still most commonly used line notation in molecular design, are SMILES [170]. However, SMILES were invented for storage and web transmission purposes, not for molecular design. Accordingly, their manipulation is fragile. Where plain SMILES failed to deliver, alternative more rubust line notations were developed [171, 183]. Despite graph neural networks becoming mature [184–186], line notations continue enjoying widespread use. One can only hope attention will shift towards more natural molecular representations in the near future.

Something most researchers are guilty of is trying to solve problems with tools they are familiar with. For me that may be cheminformatics-powered heuristic optimization algorithms, and for others statistical models and machine learning. As the saying goes, “if all you have is a hammer, everything looks like a nail”. I believe that the greatest future innovations within this space will come from integrating different approaches and harnessing the best of their respective worlds. In fact, the shift towards a hybrid methodology paradigm is underway [90, 187–190]. Basic cheminformatics techniques are easy to understand and tweak. If a flaw is observed one can reason about what went wrong and how to fix it. They may be a suitable solution for critical steps where certain guarantees are necessary, such as constructing molecular graphs. Machine learning models may be harder to understand and modulate, but they are great at finding patterns and predicting properties as a function of said patterns. They may be most suitable for non-critical processes that are not exposed to the user/developer and do not require transparency. For example, they could be used as policies to guide chemical space searches or to augment objective functions.

6.6 Computational resource allocation

The properties of molecules proposed by a molecule generator follow some distribution. We have discussed a variety of techniques to bias these distributions towards desirable values. Regardless of the chosen technique, it is associated with a computational overhead (sections 3.4.4, 4.4.7, and 5.4.1.2). It may be possible to allocate these same computational resources differently for greater payoffs.

An obvious candidate for resource allocation is running more replicas, that is, designing more molecules. Even for unsophisticated molecule generators the output molecules’ property distributions can be wide enough to cover the desired values with significant density (section 3.4). In other words, naive molecule generators may have a small probability of designing a molecule with desired properties. If the cost disparity between the unbiased and biased molecular design is big, or the difference between distributions is small, sampling more times from the worse distribution can be a superior strategy to sampling less times from the better distribution. Some of the biased molecular design strategies we evaluated designed molecules 1,000-fold slower than unbiased design. Even if every molecule proposed by the biased generator were desirable, which is an overtly optimistic scenario, it would suffice for the unbiased generator to have a 1/1,000 chance of proposing a desirable molecule to be competitive with its biased counterpart.

For a fair and resource-aware comparison between approaches one should compare their overall returns on investment. The number of times the objective functions are evaluated as well as the cost of evaluating them ought to be included in the equation.

An accurate scoring function is important to goal-directed *de novo* molecular design, for inaccurate scoring functions may mislead the design process. Generally speaking, the more accurate an objective function is, the higher the cost of evaluating it. For computational inhibitory effect predictions the arguable state of the art methods are physics-based methods involving (quantum) molecular mechanics, such as free energy perturbation calculations. The more affordable end of the objective function spectrum is comprised of QSAR models taking as input simple molecular descriptors such as topological fingerprints.

One should choose the most accurate objective function (or ensemble of objective functions [191]) that the computational budget allows for. Making this decision requires knowing in advance how many times we anticipate to evaluate the objective function. This number is oftentimes obtained empirically, but the relationships between search algorithm, scoring function accuracy and number of scoring function evaluations are poorly understood and should be studied further. We can hypothesize that the more accurate the scoring function the fewer times we must evaluate it to gather the same amount of information, but considering that costs tend to increase with accuracy, is there a point of diminishing returns? If so, what's the optimal accuracy/cost ratio? It is also known that some search strategies are more efficient than others. But what sort of search algorithm is the most efficient for chemical space exploration? How do we best balance exploration and exploitation? Is there a theoretical minimum number of times the objective function ought to be evaluated to achieve a certain degree of exploration?

Our research provides some preliminary insights into these questions. It appears that evolutionary algorithms converge to solutions after scoring 10^4 - 10^5 molecules on average (sections 3.4.4 and 4.4.7). The number of molecules scored to reach convergence appears to be lower when steering the search to specific areas of chemical space through molecule construction constraints (including fragment-based design). As general guidance, if one uses computationally affordable objective functions the cost of scoring molecules is on par or lower than generating them. In such a scenario the aforementioned strategy of "rolling the dice more" may be viable. Conversely, if one uses an expensive objective function it becomes crucial to make the most of each function evaluation. Accordingly it may be worthwhile to invest more computational resources in the molecule generation phase to increase the odds of molecules that make it to the scoring stage receiving favorable scores.

Should reducing the number of objective function evaluations prove too challenging, it is possible to augment the throughput of expensive objective functions with surrogate objective functions, that is, cheap functions predicting the outcome of the more expensive function. The surrogate function is used for fast and broad exploration, with the ground truth objective function being used to confirm or disprove crucial predictions. When the surrogate function is a machine learning model it may be updated as more data points become available, be it through re-training or incremental learning. This has led to a workflow known as “active learning”, where the beliefs and accuracy of the surrogate function are iteratively updated through feedback from the ground truth objective function. This workflow has successfully been applied to virtual screening [192–194], and is beginning to be explored in *de novo* molecular design [195].

6.7 Alternative optimization algorithms

This work relies primarily on evolutionary algorithms for optimization purposes. Evolutionary algorithms are capable of efficiently exploring large and complex fitness landscapes, and do not require a continuous search space. Accordingly, they have a rich history of being applied to molecular design (section 1.7.2). Surprisingly, many other heuristic optimization algorithms that theoretically share these same capabilities have received but a tiny fraction of the attention.

I suspect this favouritism is mostly grounded on historical reasons. Evolutionary algorithms were among the first optimization algorithms to be applied to *de novo* molecular design, specifically to the design of linear macromolecules such as polynucleotides and proteins. The direct parallels between protein design and evolution may have inspired the field’s pioneers. Following their success it must have felt natural to extend evolutionary algorithms to small molecule design. This scientific inertia has led to *de novo* molecular design becoming quasi-synonymous with evolutionary algorithms.

As has been touched upon previously, we should strive to minimize the number of times the objective function is evaluated during the optimization process. There may be limits to how low we can take this number by extending and improving upon evolutionary algorithms, which warrants exploring alternative optimization algorithms more extensively.

Nature can be a good source of inspiration for optimization algorithms [196]. Swarm intelligence optimization algorithms share many characteristics with evolutionary algorithms. Particle swarm optimization has been successfully applied to molecular design, albeit seemingly only twice [98, 99]. Various other swarm intelligence algorithms named after animal behaviour (including ants, fireflies and spider monkeys) remain unexplored within this field. However, beware that most nature-inspired algorithms are population-based. While this contributes to their exploration prowess, it also involves repeatedly evaluating the fitness of every member of the population.

Situations where the fitness function is to be used sparingly call for more directed optimization algorithms. Tree searches and their variants are obvious candidates, but seemingly fail to explore chemical space adequately [100, 108]. This is unsurprising given their exploitative nature. Hybrid algorithms, combining explorative elements for global searches and exploitative elements for local searches, may be able to harness the best of both worlds. We found some circumstantial evidence of this being the case when combining evolutionary algorithms with tree searches (section 5.4.1.2).

One of the primary conclusions of this work is that constraining the way in which molecules are assembled can steer an evolutionary algorithm away from unpromising areas of chemical space. It's reasonable to infer that designing "unconstrained" molecules that do not obey the laws of chemistry serves no practical purpose, and should be avoided beyond academic exercises. However, one should exercise caution when extrapolating our results to unrelated optimization algorithms. Impossible molecular graphs should never be a final result presented to the user, but they may be useful stepping stones in chemical space traversal. One could liken them to reaction intermediates, in that they facilitate or explain the transition between two perfectly sensible molecules. As such, molecular graphs defying the rules of chemistry may be to cheminformatics what complex numbers are to mathematics. They may lack meaning in the physical world, yet have use cases in imaginary worlds. I hypothesize that some optimization algorithms will be capable of realizing their potential. If one could devise a chemical space navigation system [49], alongside a search algorithm to capitalize on it, the denser transition graphs associated with unconstrained design should become a net positive (section 3.2).

It is common and valuable to borrow ideas from unrelated fields, seeing as many scientific problems are somehow related. Nevertheless, we must recall that some methods may be better suited for some purposes than others, and that successes in one field cannot always be translated to another. Even purportedly generic methods are rarely optimal for a specific application, given that they do not harness domain-specific knowledge. The computational molecular design field has embraced the idea of supporting swappable black-box objective functions through the use of generic optimization algorithms, but in the process we often neglect to incorporate domain knowledge. Human chemists are capable of finding desirable molecules by creating relatively small chemical series of $10^2 – 10^3$ molecules. It's important to note that humans tend to perform local optimizations on promising chemical entities [197, 198], whereas computational molecule generators typically aim to perform global optimizations. Nonetheless, humans are orders of magnitude more efficient with their objective functions (i.e. lab experiments) than typical optimization algorithms, and these differences cannot be explained merely by differences in objective function accuracy. If we want computers to reach such degree of efficiency we must expose the objective function's internals to domain-aware optimization algorithms that have been customized to exploit a specific objective. For example, if we fixed our objective function to be ligand-target interaction energies as measured by molecular docking, we could design optimization algorithms that are privy to the target's shape, and harness our knowledge about molecular interactions to achieve shape and electrostatic complementarity between molecules.

6.8 To imitate or not to imitate?

One of the main appeals of *de novo* molecular design is that it can propose novel chemical entities that are dissimilar to previously described molecules. Yet almost all existing molecule generators imitate known chemistry in some shape or form. Molecule quality, including drug-likeness and synthesizability, is almost always assessed based on similarity to known chemistry. How novel can a molecule truly be if its designed to be similar to existing molecules, and its fitness is evaluated based on similarity to existing molecules? A molecule cannot be similar and dissimilar to known chemistry simultaneously. Hence, there is a trade-off between chemical quality and chemical novelty.

We should ask ourselves the question: how novel is novel enough? From an industry perspective, a molecule might be considered novel enough if it can be patented. From an academic perspective, a molecule may be novel enough when it provides new solutions to problems of competing molecules. For example, a molecule might be novel if its mechanism of action or ADMET profile are distinct from other existing compounds with the same indication. On a structural level this usually entails presenting a different scaffold. An avenue to molecular scaffold diversity is the exploration of novel ring systems, but unconventional ring systems can entail complicated syntheses. The flip side to the above question is then: how easy to synthesize is easy enough? Many molecules might be theoretically synthesizable, but the likelihood of success and efforts required to synthesize them might be disproportionate to the potential upside.

Given the large uncertainty surrounding computational molecular property predictions focusing research efforts on familiar chemistry is a pragmatic approach to increase a project's chances of success. Yet we should be aware that at a larger scale we risk creating a self-perpetuating cycle that could lead to academic stagnation. If we are confident enough in the accuracy of our scoring functions perhaps we should not hold historic data in such high regard and occasionally venture into unknown territory.

As of today, the quality-novelty trade-off is unavoidable, but solely due to the way in which quality is assessed. If one were to employ data-independent methods for said assessment the trade-off would cease to exist. It is likely possible to evaluate chemical quality invoking theoretical chemistry principles. I anticipate the practical challenges to be finding ways of performing such an evaluation in an automated way, for a wide enough variety of (potentially non-sensical) molecules, and fast enough to apply it in high-throughput settings.

6.9 Bridging the gap between computer and wet lab

In the late 2000s *de novo* molecular design fell out of favor. Two of the reasons cited for this shift in paradigm were poor designed molecule synthesizability and unreliable scoring functions. Today we have access to algorithms and computing resources that largely solve both problems. So where are the *de novo* designed drugs?

There are many accounts of *de novo* molecular design being employed successfully to design small molecules with experimentally confirmed desirable properties [22, 87, 107, 199]. However, to the best of my knowledge none of these molecules have been approved as drugs. Multiple factors can explain this phenomenon. For starters, *de novo* molecular design, especially the generative chemistry models branch, is a relatively young field, and future drugs that may have been conceived through *de novo* molecular design are yet to see the light of day. Moreover, the origin of a hit that ultimately leads to a drug may be unclear or not disclosed publicly. Lastly, the pharmaceutical industry historically hasn't invested heavily into computational drug discovery as a whole, favoring alternative workflows [197, 198].

Computational workflows rarely integrate well with wet lab workflows, leading to departments working in parallel with insufficient communication between them. In order to achieve tighter integration between both disciplines, software and protocols must be designed around a dialog between user and computer. Experimental data collection in machine-readable format ought to be standardized, and should go beyond electronic lab notebooks. Collected data should be fed continuously to the computer, and the computer should continuously reply with up-to-date insights. Ideally a wet lab scientists should be capable of operating the relevant parts of the software on their own, without the assistance of computational chemists. Software developers ought to make an effort to enhance the user experience. Data should be presented in a graphical and understable way such that it can be acted upon, and the user should be able to interact with the software through graphical user interfaces. Any provided predictions ought to be accompanied by reasoning, enabling the user to scrutinize them. Should the predictions be undesirable there ought to be a mechanism for the user to provide feedback to the software so the latter may it finetune its insights according to the user's expertise [79, 200, 201].

Beyond the general reluctance of using computational techniques, *de novo* molecular design in particular faces additional challenges to adoption. Arguably the most important one is the difficulty of sourcing molecules designed *de novo*. A *de novo* designed molecule is likely to be novel, and even if it is synthesizable its synthesis has not been optimized. Reagents and equipment may have to be acquired, reaction conditions for optimal yield need to be elucidated, and chemists may have to be trained. Similar problems might arise during testing. Other molecular design workflows offer easier ways of sourcing molecules. Traditional human-driven design revolves around creating related chemical entities, which presumably share parts of their synthetic routes. As such, much of the synthetic efforts are amortized. As another example, hits found through virtual screening may be commercially available and purchased, requiring only a small financial commitment.

Future *de novo* molecular design should strive towards easing molecule synthesis. One avenue to do so is designing molecules through virtual chemistry, restricting the algorithm to use building blocks that are available in-house and reactions the chemists are confident with. An even better solution would be designing whole chemical series rather than individual molecules. The chemical series should share a synthetic route, or at the very least a scaffold, and should be diverse enough to obtain SAR data.

Objective function accuracy still leaves some things to be desired. Even the best computational objective functions are noisy, which poses a challenge for goal-directed molecular design. The most accurate objective function, which provides the ground truth, is a wet lab experiment. Lab experiments cannot replace computational objective functions entirely since their throughput is not sufficient to explore the vast chemical space. Funnel strategies, where one uses less accurate but cheap computational objective functions for broad exploration, and more expensive *in vitro* assays for local exploitation, are common place. Automation is likely to increase the throughput of lab experiments, and in the future said experiments will be used for explorative purposes more extensively. In fact, automated “design-make-test” cycles are the poster-child application for *de novo* molecular design, and thanks to advances in robotics and automated synthesis they have been achieved to some extent [202–204].

Not every chemist or biologist will be excited about such a prospect. Some may find computational techniques encroaching upon their domain, or even automating away some of their responsibilities, uncomfortable. New technologies and automation can be perceived as a threat to the scientist’s employment and creativity. I wish to ease the mind of those concerned, for human ingenuity and problem solving are not easily substituted by algorithms. For those developing the technologies of the future, I urge you to design them in a manner that empowers scientists and harnesses their expertise.

6.10 Code speaks louder than words

I hope to have provided a comprehensive overview of the algorithms behind this research in prose. Yet inevitably readers will find that some technical details require further clarification. All of the software used to perform this research is free and open source. For implementation details I invite the readers to inspect the code for themselves:

- **Molpert**: https://github.com/AlanKerstjens/Molpert
- **LEADD**: https://github.com/UAMCAntwerpen/LEADD
- Molecule auto-correct: https://github.com/AlanKerstjens/MoleculeAutoCorrect

List of abbreviations

- ADME(T): Absorption, Distribution, Metabolism, Excretion, (Toxicity)
- ANOVA: ANalysis Of VAriance
- BFS: Breadth-First Search
- DF: Degrees of Freedom
- EA: Evolutionary Algorithm
- ECFP: Extended Connectivity FingerPrint
- FWER: Family-Wise Error Rate
- GA: Genetic Algorithm
- HAC: Heavy Atom Count
- MBPM: Maximum BiPartite Matching
- MCTS: Monte Carlo Tree Search
- MLR: Multiple Linear Regression
- MMFF: Merck Molecular Force Field
- MSI: Multiple Set Intersection
- MPO: Multiple Parameter Optimization
- OP: Optimization Power
- PCA: Principal Component Analysis
- PDF: Probability Density Function
- QED: Quantitative Estimation of Drug-likeness
- (Q)SAR: (Quantitative) Structure-Activity Relationship
- RA: Ring-Aware
- RDM: Randomly Designed Molecules
- RMSE: Root Mean Squared Error
- RNN: Recurrent Neural Network
- SA: Synthetic Accessibility
- UCB: Upper Confidence Bounds
- UCT: Upper Confidence bounds applied to Trees
- VAE: Variational Auto-Encoder

Bibliography

1. Ripphausen P, Nisius B, Peltason L, Bajorath J (2010) Quo Vadis, Virtual Screening? A Comprehensive Survey of Prospective Applications. J Med Chem 53:8461–8467. https://doi.org/10.1021/jm101020z

2. Irwin JJ, Tang KG, Young J, et al (2020) ZINC20-A Free Ultralarge-Scale Chemical Database for Ligand Discovery. J Chem Inf Model 60:6065–6073. https://doi.org/10.1021/acs.jcim.0c00675

3. Hu Q, Peng Z, Sutton SC, et al (2012) Pfizer global virtual library (PGVL): A chemistry design tool powered by experimentally validated parallel synthesis information. ACS Comb Sci 14:579–589. https://doi.org/10.1021/co300096q

4. Chevillard F, Kolb P (2015) SCUBIDOO: A Large yet Screenable and Easily Searchable Database of Computationally Created Chemical Compounds Optimized toward High Likelihood of Synthetic Tractability. J Chem Inf Model 55:1824–1835. https://doi.org/10.1021/acs.jcim.5b00203

5. Nicolaou CA, Watson IA, Hu H, Wang J (2016) The Proximal Lilly Collection: Mapping, Exploring and Exploiting Feasible Chemical Space. J Chem Inf Model 56:1253–1266. https://doi.org/10.1021/acs.jcim.6b00173

6. Ruddigkeit L, Deursen RV, Blum LC, Reymond JL (2012) Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17. J Chem Inf Model 52:2864–2875. https://doi.org/10.1021/ci300415d

7. Bohacek RS, McMartin C, Guida WC (1996) The art and practice of structure-based drug design: A molecular modeling perspective. Med Res Rev 16:3–50. https://doi.org/10.1002/(SICI)1098-1128(199601)16:1<3::AID-MED1>3.0.CO;2-6

8. Ertl P (2003) Cheminformatics Analysis of Organic Substituents: Identification of the Most Common Substituents, Calculation of Substituent Properties, and Automatic Identification of Drug-Like Bioisosteric Groups. J Chem Inf Comput Sci 34:374–380. https://doi.org/10.1002/chin.200321198

9. Polishchuk PG, Madzhidov TI, Varnek A (2013) Estimation of the size of drug-like chemical space based on GDB-17 data. J Comput Aided Mol Des 27:675–679. https://doi.org/10.1007/s10822-013-9672-4

10. Brown DG, Boström J (2016) Analysis of Past and Present Synthetic Methodologies on Medicinal Chemistry: Where Have All the New Reactions Gone? J Med Chem 59:4443–4458. https://doi.org/10.1021/acs.jmedchem.5b01409

11. Schneider N, Lowe DM, Sayle RA, et al (2016) Big Data from Pharmaceutical Patents: A Computational Analysis of Medicinal Chemists Bread and Butter. J Med Chem 59:4385–4402. https://doi.org/10.1021/acs.jmedchem.6b00153

12. Virshup AM, Contreras-García J, Wipf P, et al (2013) Stochastic voyages into uncharted chemical space produce a representative library of all possible drug-like compounds. J Am Chem Soc 135:7296–7303. https://doi.org/10.1021/ja401184g

13. Lin A, Horvath D, Afonina V, et al (2018) Mapping of the Available Chemical Space versus the Chemical Universe of Lead-Like Compounds. ChemMedChem 13:540–554. https://doi.org/10.1002/cmdc.201700561

14. Cho SJ, Zheng W, Tropsha A (1998) Rational combinatorial library design. 2. Rational design of targeted combinatorial peptide libraries using chemical similarity probe and the inverse QSAR approaches. J Chem Inf Comput Sci 38:259–268. https://doi.org/10.1021/ci9700945

15. Brown N, McKay B, Gasteiger J (2006) A novel workflow for the inverse QSPR problem using multiobjective optimization. J Comput Aided Mol Des 20:333–341. https://doi.org/10.1007/s10822-006-9063-1

16. Bohm HJ (1992) The computer program LUDI: a new method for the de novo design of enzyme inhibitors. J Comput Aided Mol Des 6:61–78. https://doi.org/10.1007/BF00124387

17. Gillet V, Johnson AP, Mata P, et al (1993) SPROUT: A program for structure generation. J Comput Aided Mol Des 7:127–153. https://doi.org/10.1007/BF00126441

18. Rotstein SH, Murcko MA (1993) GenStar: A method for de novo drug design. J Comput Aided Mol Des 7:23–43. https://doi.org/10.1007/BF00141573

19. Rotstein SH, Murcko MA (1993) GroupBuild : A Fragment-Based Method for De Novo Drug Design. J Med Chem 36:1700–1710. https://doi.org/10.1021/jm00064a003

20. Pearlman DA, Murcko MA (1993) CONCEPTS: New dynamic algorithm for de novo drug suggestion. J Comput Chem 14:1184–1193. https://doi.org/10.1002/jcc.540141008

21. Clark DE, Frenkel D, Levy SA, et al (1995) PRO_LIGAND: An approach to de novo molecular design. 1. Application to the design of organic molecules. J Comput Aided Mol Des 9:13–32. https://doi.org/10.1007/BF00117275

22. Schneider G (2013) De novo molecular design. John Wiley & Sons

23. Downs GM, Gillet VJ, Holliday JD, Lynch MF (1989) Review of ring perception algorithms for chemical graphs. J Chem Inf Comput Sci 29:172–187. https://doi.org/10.1021/ci00063a007

24. Lipinski CA, Lombardo F, Dominy BW, Feeney PJ (1997) Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. Adv Drug Deliv Rev 23:3–25. https://doi.org/10.1016/S0169-409X(96)00423-1

25. Wiener H (1947) Structural Determination of Paraffin Boiling Points. J Am Chem Soc 69:17–20. https://doi.org/10.1021/ja01193a005

26. Randic M (1975) Characterization of molecular branching. J Am Chem Soc 97:6609–6615. https://doi.org/10.1021/ja00856a001

27. Balaban AT (1982) Highly discriminating distance-based topological index. Chem Phys Lett 89:399–404. https://doi.org/10.1016/0009-2614(82)80009-2

28. Ivanciuc O. BAT (1999) Historical Development of Topological Indices. In: Topological Indices and Related Descriptors in QSAR and QSPR, 1st ed. CRC Press

29. Durant JL, Leland BA, Henry DR, Nourse JG (2002) Reoptimization of MDL Keys for Use in Drug Discovery. J Chem Inf Comput Sci 42:1273–1280. https://doi.org/10.1021/ci010132r

30. Bolton EE, Wang Y, Thiessen PA, Bryant SH (2008) Chapter 12 - PubChem: Integrated Platform of Small Molecules and Biological Activities. In: Annual Reports in Computational Chemistry. Elsevier, pp 217–241

31. Cook SA (1971) The complexity of theorem-proving procedures. In: Proceedings of the third annual ACM symposium on Theory of computing. Association for Computing Machinery, pp 151–158

32. Carhart RE, Smith DH, Venkataraghavan R (1985) Atom pairs as molecular features in structure-activity studies: definition and applications. J Chem Inf Comput Sci 25:64–73. https://doi.org/10.1021/ci00046a002

33. Daylight Theory Manual. In: Daylight Theory Man. https://www.daylight.com/dayhtml/doc/theory/. Accessed 3 Nov 2023

34. Rogers D, Hahn M (2010) Extended-Connectivity Fingerprints. J Chem Inf Model 50:742–754. https://doi.org/10.1021/ci100050t

35. Bajusz D, Rácz A, Héberger K (2015) Why is Tanimoto index an appropriate choice for fingerprint-based similarity calculations? J Cheminformatics 7:1–13. https://doi.org/10.1186/s13321-015-0069-3

36. Johnson MA, Maggiora GM (1991) Concepts and applications of molecular similarity, 1st ed. Wiley

37. Maggiora G, Vogt M, Stumpfe D, Bajorath J (2014) Molecular Similarity in Medicinal Chemistry. J Med Chem 57:3186–3204. https://doi.org/10.1021/jm401411z

38. Willett P, Barnard JM, Downs GM (1998) Chemical Similarity Searching. J Chem Inf Comput Sci 38:983–996. https://doi.org/10.1021/ci9800211

39. Cao Y, Jiang T, Girke T (2010) Accelerated similarity searching and clustering of large compound sets by geometric embedding and locality sensitive hashing. Bioinformatics 26:953–959. https://doi.org/10.1093/bioinformatics/btq067

40. Butina D (1999) Unsupervised data base clustering based on daylight's fingerprint and Tanimoto similarity: A fast and automated way to cluster small and large data sets. J Chem Inf Comput Sci 39:747–750. https://doi.org/10.1021/ci9803381

41. Wigh DS, Goodman JM, Lapkin AA (2022) A review of molecular representation in the age of machine learning. WIREs Comput Mol Sci 12:e1603. https://doi.org/10.1002/wcms.1603

42. Bender A, Jenkins JL, Scheiber J, et al (2009) How Similar Are Similarity Searching Methods? A Principal Component Analysis of Molecular Descriptor Space. J Chem Inf Model 49:108–119. https://doi.org/10.1021/ci800249s

43. Riniker S, Landrum GA (2013) Open-source platform to benchmark fingerprints for ligand-based virtual screening. J Cheminformatics 5:26. https://doi.org/10.1186/1758-2946-5-26

44. O'Boyle NM, Sayle RA (2016) Comparing structural fingerprints using a literature-based similarity benchmark. J Cheminformatics 8:36. https://doi.org/10.1186/s13321-016-0148-0

45. Weininger D, Weininger A, Weininger JL (1989) SMILES. 2. Algorithm for generation of unique SMILES notation. J Chem Inf Comput Sci 29:97–101. https://doi.org/10.1021/ci00062a008

46. Morgan HL (1965) The Generation of a Unique Machine Description for Chemical Structures—A Technique Developed at Chemical Abstracts Service. J Chem Doc 5:107–113. https://doi.org/10.1021/c160017a018

47. Landrum GA RDKit: Open-source cheminformatics. https://www.rdkit.org

48. Drew KLM, Baiman H, Khwaounjoo P, et al (2012) Size estimation of chemical space: How big is it? J Pharm Pharmacol 64:490–495. https://doi.org/10.1111/j.2042-7158.2011.01424.x

49. Oprea TI, Gottfries J (2001) Chemography: The Art of Navigating in Chemical Space. J Comb Chem 3:157–166. https://doi.org/10.1021/cc0000388

50. Renner S, van Otterlo WAL, Dominguez Seoane M, et al (2009) Bioactivity-guided mapping and navigation of chemical space. Nat Chem Biol 5:585–592. https://doi.org/10.1038/nchembio.188

51. Osolodkin DI, Radchenko EV, Orlov AA, et al (2015) Progress in visual representations of chemical space. Expert Opin Drug Discov 10:959–973. https://doi.org/10.1517/17460441.2015.1060216

52. van Deursen R, Reymond J-L (2007) Chemical Space Travel. ChemMedChem 2:636–640. https://doi.org/10.1002/cmdc.200700021

53. Sayle RA, Batista J, Grant JA (2013) Efficient maximum common subgraph (MCS) searching of large chemical databases. J Cheminformatics 5:O15. https://doi.org/10.1186/1758-2946-5-S1-O15

54. Hoksza D, Škoda P, Voršilák M, Svozil D (2014) Molpher: A software framework for systematic chemical space exploration. J Cheminformatics 6:1–13. https://doi.org/10.1186/1758-2946-6-7

55. Kim R, Skolnick J (2008) Assessment of programs for ligand binding affinity prediction. J Comput Chem 29:1316–1331. https://doi.org/10.1002/jcc.20893

56. Ballester PJ, Schreyer A, Blundell TL (2014) Does a More Precise Chemical Description of Protein–Ligand Complexes Lead to More Accurate Prediction of Binding Affinity? J Chem Inf Model 54:944–955. https://doi.org/10.1021/ci500091r

57. Scannell JW, Bosley J (2016) When quality beats quantity: Decision theory, drug discovery, and the reproducibility crisis. PLoS ONE 11:1–21. https://doi.org/10.1371/journal.pone.0147215

58. Pantsar T, Poso A (2018) Binding Affinity via Docking: Fact and Fiction. Molecules 23:1899. https://doi.org/10.3390/molecules23081899

59. Lyu J, Wang S, Balius TE, et al (2019) Ultra-large library docking for discovering new chemotypes. Nature 566:224–229. https://doi.org/10.1038/s41586-019-0917-9

60. Nicolaou CA, Apostolakis J, Pattichis CS (2009) De novo drug design using multiobjective evolutionary graphs. J Chem Inf Model 49:295–307. https://doi.org/10.1021/ci800308h

61. Yuan Y, Pei J, Lai L (2011) LigBuilder 2: A Practical de Novo Drug Design Approach. J Chem Inf Model 51:1083–1091. https://doi.org/dx.doi.org/10.1021/ci100350u

62. Steinmann C, Jensen JH (2021) Using a genetic algorithm to find molecules with good docking scores. PeerJ Phys Chem 3:e18. https://doi.org/10.7717/peerj-pchem.18

63. Hartenfeller M, Schneider G (2011) Enabling future drug discovery by *de novo* design. WIREs Comput Mol Sci 1:742–759. https://doi.org/10.1002/wcms.49

64. Brown N, McKay B, Gilardoni F, Gasteiger J (2004) A Graph-Based Genetic Algorithm and Its Application to the Multiobjective Evolution of Median Molecules. ChemInform 35:1079–1087. https://doi.org/10.1002/chin.200431198

65. Ekins S, Honeycutt JD, Metz JT (2010) Evolving molecules using multi-objective optimization: Applying to ADME/Tox. Drug Discov Today 15:451–460. https://doi.org/10.1016/j.drudis.2010.04.003

66. Daeyaert F, Deem MW (2017) A Pareto Algorithm for Efficient De Novo Design of Multi-functional Molecules. Mol Inform 36:. https://doi.org/10.1002/minf.201600044

67. Winter R, Montanari F, Steffen A, et al (2019) Efficient multi-objective molecular optimization in a continuous latent space. Chem Sci 10:8016–8024. https://doi.org/10.1039/C9SC01928F

68. Verhellen J (2022) Graph-based molecular Pareto optimisation. Chem Sci 13:7526–7535. https://doi.org/10.1039/D2SC00821A

69. Fromer JC, Coley CW (2023) Computer-aided multi-objective optimization in small molecule discovery. Patterns 4:100678. https://doi.org/10.1016/j.patter.2023.100678

70. Fang G, Xue M, Su M, et al (2012) CCLab - A multi-objective genetic algorithm based combinatorial library design software and an application for histone deacetylase inhibitor design. Bioorg Med Chem Lett 22:4540–4545. https://doi.org/10.1016/j.bmcl.2012.05.123

71. Herring RH, Eden MR (2015) Evolutionary algorithm for de novo molecular design with multi-dimensional constraints. Comput Chem Eng 83:267–277. https://doi.org/10.1016/j.compchemeng.2015.06.012

72. Maltese J, Ombuki-Berman BM, Engelbrecht AP (2018) A Scalability Study of Many-Objective Optimization Algorithms. IEEE Trans Evol Comput 22:79–96. https://doi.org/10.1109/TEVC.2016.2639360

73. Kutchukian PS, Lou D, Shakhnovich EI (2009) FOG: Fragment optimized growth algorithm for the de novo generation of molecules occupying druglike chemical space. J Chem Inf Model 49:1630–1642. https://doi.org/10.1021/ci9000458

74. Polishchuk P (2020) CReM: chemically reasonable mutations framework for structure generation. J Cheminformatics 12:28. https://doi.org/10.1186/s13321-020-00431-w

75. Kerstjens A, De Winter H (2022) LEADD: Lamarckian evolutionary algorithm for de novo drug design. J Cheminformatics 14:3. https://doi.org/10.1186/s13321-022-00582-y

76. Lewell XQ, Judd DB, Watson SP, Hann MM (1998) RECAP - Retrosynthetic Combinatorial Analysis Procedure: A powerful new technique for identifying privileged molecular fragments with useful applications in combinatorial chemistry. J Chem Inf Comput Sci 38:511–522. https://doi.org/10.1021/ci970429i

77. Degen J, Wegscheid-Gerlach C, Zaliani A, Rarey M (2008) On the Art of Compiling and Using “Drug-Like” Chemical Fragment Spaces. ChemMedChem 3:1503–1507. https://doi.org/10.1002/cmdc.200800178

78. Ecemis MI, Wikel J, Bingham C, Bonabeau E (2008) A Drug Candidate Design Environment Using Evolutionary Computation. IEEE Trans Evol Comput 12:591–603. https://doi.org/10.1109/TEVC.2007.913131

79. Lameijer EW, Kok JN, Bäck T, Ijzerman AP (2006) The molecule evoluator. An interactive evolutionary algorithm for the design of drug-like molecules. J Chem Inf Model 46:545–552. https://doi.org/10.1021/ci050369d

80. Hartenfeller M, Zettl H, Walter M, et al (2012) Dogs: Reaction-driven de novo design of bioactive compounds. PLoS Comput Biol 8:e1002380. https://doi.org/10.1371/journal.pcbi.1002380

81. Spiegel JO, Durrant JD (2020) AutoGrow4: An open-source genetic algorithm for de novo drug design and lead optimization. J Cheminformatics 12:1–16. https://doi.org/10.1186/s13321-020-00429-4

82. Schneider G, Lee ML, Stahl M, Schneider P (2000) De novo design of molecular architectures by evolutionary assembly of drug-derived building blocks. J Comput Aided Mol Des 14:487–494. https://doi.org/10.1023/A:1008184403558

83. Fechner U, Schneider G (2006) Flux (1): A virtual synthesis scheme for fragment-based de novo design. J Chem Inf Model 46:699–707. https://doi.org/10.1021/ci0503560

84. Ghiandoni GM, Bodkin MJ, Chen B, et al (2021) RENATE: A Pseudo-retrosynthetic Tool for Synthetically Accessible de Novo Design. Mol Inform 2100207:1–8. https://doi.org/10.1002/minf.202100207

85. Masek BB, Baker DS, Dorfman RJ, et al (2016) Multistep Reaction Based de Novo Drug Design: Generating Synthetically Feasible Design Ideas. J Chem Inf Model 56:605–620. https://doi.org/10.1021/acs.jcim.5b00697

86. Zhou Z, Kearnes S, Li L, et al (2019) Optimization of Molecules via Deep Reinforcement Learning. Sci Rep 9:10752. https://doi.org/10.1038/s41598-019-47148-x

87. Mouchlis VD, Afantitis A, Serra A, et al (2021) Advances in De Novo Drug Design: From Conventional to Machine Learning Methods. Int J Mol Sci 22:1676. https://doi.org/10.3390/ijms22041676

88. Bilodeau C, Jin W, Jaakkola T, et al (2022) Generative models for molecular discovery: Recent advances and challenges. WIREs Comput Mol Sci 12:e1608. https://doi.org/10.1002/wcms.1608

89. Gómez-Bombarelli R, Wei JN, Duvenaud D, et al (2018) Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules. ACS Cent Sci 4:268–276. https://doi.org/10.1021/acscentsci.7b00572

90. Sattarov B, Baskin II, Horvath D, et al (2019) De Novo Molecular Design by Combining Deep Autoencoder Recurrent Neural Networks with Generative Topographic Mapping. J Chem Inf Model 59:1182–1196. https://doi.org/10.1021/acs.jcim.8b00751

91. Segler MHS, Kogej T, Tyrchan C, Waller MP (2018) Generating Focused Molecule Libraries for Drug Discovery with Recurrent Neural Networks. ACS Cent Sci 4:120–131. https://doi.org/10.1021/acscentsci.7b00512

92. Olivecrona M, Blaschke T, Engkvist O, Chen H (2017) Molecular de-novo design through deep reinforcement learning. J Cheminformatics 9:48. https://doi.org/10.1186/s13321-017-0235-x

93. Grisoni F, Moret M, Lingwood R, Schneider G (2020) Bidirectional Molecule Generation with Recurrent Neural Networks. J Chem Inf Model 60:1175–1183. https://doi.org/10.1021/acs.jcim.9b00943

94. Maggiora GM (2006) On Outliers and Activity CliffsWhy QSAR Often Disappoints. J Chem Inf Model 46:1535–1535. https://doi.org/10.1021/ci060117s

95. Bajorath J (2017) Representation and identification of activity cliffs. Expert Opin Drug Discov 12:879–883. https://doi.org/10.1080/17460441.2017.1353494

96. Kvasnička V, Pospíchal J (1996) Simulated Annealing Construction of Molecular Graphs with Required Properties. J Chem Inf Comput Sci 36:516–526. https://doi.org/10.1021/ci9500703

97. Ourique JE, Silva Telles A (1998) Computer-aided molecular design with simulated annealing and molecular graphs. Comput Chem Eng 22:S615–S618. https://doi.org/10.1016/S0098-1354(98)00108-2

98. Hartenfeller M, Proschak E, Schüller A, Schneider G (2008) Concept of combinatorial de novo design of drug-like molecules by particle swarm optimization. Chem Biol Drug Des 72:16–26. https://doi.org/10.1111/j.1747-0285.2008.00672.x

99. Fu Y, Chen Z, Sun J (2018) Random drift particle swarm optimisation algorithm for highly flexible protein-ligand docking. J Theor Biol 457:180–189. https://doi.org/10.1016/j.jtbi.2018.08.034

100. Jensen JH (2019) A graph-based genetic algorithm and generative model/Monte Carlo tree search for the exploration of chemical space. Chem Sci 10:3567–3572. https://doi.org/10.1039/c8sc05372c

101. Popova M, Isayev O, Tropsha A (2018) Deep reinforcement learning for de novo drug design. Sci Adv 4:eaap7885. https://doi.org/10.1126/sciadv.aap7885

102. Kamphausen S, Höltge N, Wirsching F, et al (2002) Genetic algorithm for the design of molecules with desired properties. J Comput Aided Mol Des 16:551–567. https://doi.org/10.1023/A:1021928016359

103. Jones DT (1994) De novo protein design using pairwise potentials and a genetic algorithm. Protein Sci 3:567–574. https://doi.org/10.1002/pro.5560030405

104. Sheridan RP, Kearsley SK (1995) Using a Genetic Algorithm To Suggest Combinatorial Libraries. J Chem Inf Comput Sci 35:310–320. https://doi.org/10.1021/ci00024a021

105. Venkatasubramanian V, Chan K, Caruthers JM (1994) Computer-aided molecular design using genetic algorithms. Comput Chem Eng 18:833–844. https://doi.org/10.1016/0098-1354(93)E0023-3

106. Venkatasubramanian V, Chan K, Caruthers JM (1995) Evolutionary Design of Molecules with Desired Properties Using the Genetic Algorithm. J Chem Inf Comput Sci 35:188–195. https://doi.org/10.1021/ci00024a003

107. Meyers J, Fabian B, Brown N (2021) De novo molecular design and generative models. Drug Discov Today 26:2707–2715. https://doi.org/10.1016/j.drudis.2021.05.019

108. Brown N, Fiscato M, Segler MHS, Vaucher AC (2019) GuacaMol: Benchmarking Models for de Novo Molecular Design. J Chem Inf Model 59:1096–1108. https://doi.org/10.1021/acs.jcim.8b00839

109. Gao W, Coley CW (2020) The Synthesizability of Molecules Proposed by Generative Models. J Chem Inf Model 60:5714–5723. https://doi.org/10.1021/acs.jcim.0c00174

110. Douguet D, Thoreau E, Grassy G (2000) A genetic algorithm for the automated generation of small organic molecules: drug design using an evolutionary algorithm. J Comput Aided Mol Des 14:449–66. https://doi.org/10.1023/a:1008108423895

111. Douguet D, Munier-Lehmann H, Labesse G, Pochet S (2005) LEA3D: A computer-aided ligand design for structure-based drug design. J Med Chem 48:2457–2468. https://doi.org/10.1021/jm0492296

112. Leguy J, Cauchy T, Glavatskikh M, et al (2020) EvoMol: a flexible and interpretable evolutionary algorithm for unbiased de novo molecular generation. J Cheminformatics 12:55. https://doi.org/10.1186/s13321-020-00458-z

113. Dey F, Caflisch A (2008) Fragment-based de novo ligand design by multi-objective evolutionary optimization. Supporting Information. J Chem Inf Model 48:679–690. https://doi.org/10.1021/ci700424b

114. Kawai K, Nagata N, Takahashi Y (2014) De novo design of drug-like molecules by a fragment-based molecular evolutionary approach. J Chem Inf Model 54:49–56. https://doi.org/10.1021/ci400418c

115. Pegg SC, Haresco JJ, Kuntz ID (2001) A genetic algorithm for structure-based de novo design. J Comput Aided Mol Des 15:911–33. https://doi.org/10.1023/a:1014389729000

116. Globus AI, Lawton J, Wipke T (1999) Automatic molecular design using evolutionary techniques. Nanotechnology 10:290–299. https://doi.org/10.1088/0957-4484/10/3/312

117. Pierce AC, Rao G, Bemis GW (2004) BREED: Generating novel inhibitors through hybridization of known ligands. Application to CDK2, P38, and HIV protease. J Med Chem 47:2768–2775. https://doi.org/10.1021/jm030543u

118. Lindert S, Durrant JD, Mccammon JA (2012) LigMerge: A Fast Algorithm to Generate Models of Novel Potential Ligands from Sets of Known Binders. Chem Biol Drug Des 80:358–365. https://doi.org/10.1111/j.1747-0285.2012.01414.x

119. Polykovskiy D, Zhebrak A, Sanchez-Lengeling B, et al (2020) Molecular Sets (MOSES): A Benchmarking Platform for Molecular Generation Models. Front Pharmacol 11:. https://doi.org/10.3389/fphar.2020.565644

120. García-Ortegón M, Simm GNC, Tripp AJ, et al (2022) DOCKSTRING: Easy Molecular Docking Yields Better Benchmarks for Ligand Design. J Chem Inf Model 62:3486–3502. https://doi.org/10.1021/acs.jcim.1c01334

121. Kutchukian PS, Vasilyeva NY, Xu J, et al (2012) Inside the Mind of a Medicinal Chemist: The Role of Human Bias in Compound Prioritization during Drug Discovery. PLOS ONE 7:e48476. https://doi.org/10.1371/journal.pone.0048476

122. Gibb BC (2012) Chemical intuition or chemical institution? Nat Chem 4:237–238. https://doi.org/10.1038/nchem.1307

123. Gomez L (2018) Decision Making in Medicinal Chemistry: The Power of Our Intuition. ACS Med Chem Lett 9:956–958. https://doi.org/10.1021/acsmedchemlett.8b00359

124. Ertl P, Schuffenhauer A (2009) Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. J Cheminformatics 1:1–11. https://doi.org/10.1186/1758-2946-1-8

125. Voršilák M, Kolář M, Čmelo I, Svozil D (2020) SYBA: Bayesian estimation of synthetic accessibility of organic compounds. J Cheminformatics 12:35. https://doi.org/10.1186/s13321-020-00439-2

126. Skoraczyński G, Kitlas M, Miasojedow B, Gambin A (2023) Critical assessment of synthetic accessibility scores in computer-assisted synthesis planning. J Cheminformatics 15:6. https://doi.org/10.1186/s13321-023-00678-z

127. Thakkar A, Chadimová V, Bjerrum EJ, et al (2021) Retrosynthetic accessibility score (RAscore)-rapid machine learned synthesizability classification from AI driven retrosynthetic planning. Chem Sci 12:3339–3349. https://doi.org/10.1039/d0sc05401a

128. Gillet VJ, Myatt G, Zsoldos Z, Johnson AP (1995) SPROUT, HIPPO and CAESA: Tools for de novo structure generation and estimation of synthetic accessibility. Perspect Drug Discov Des 3:34–50. https://doi.org/10.1007/BF02174466

129. Law J, Zsoldos Z, Simon A, et al (2009) Route designer: A retrosynthetic analysis tool utilizing automated retrosynthetic rule generation. J Chem Inf Model 49:593–602. https://doi.org/10.1021/ci800228y

130. Segler MHS, Waller MP (2017) Neural-Symbolic Machine Learning for Retrosynthesis and Reaction Prediction. Chem - Eur J 23:5966–5971. https://doi.org/10.1002/chem.201605499

131. Segler MHS, Preuss M, Waller MP (2018) Planning chemical syntheses with deep neural networks and symbolic AI. Nature 555:604–610. https://doi.org/10.1038/nature25978

132. Genheden S, Thakkar A, Chadimová V, et al (2020) AiZynthFinder: a fast, robust and flexible open-source software for retrosynthetic planning. J Cheminformatics 12:1–9. https://doi.org/10.1186/s13321-020-00472-1

133. Schreck JS, Coley CW, Bishop KJM (2019) Learning Retrosynthetic Planning through Simulated Experience. ACS Cent Sci 5:970–981. https://doi.org/10.1021/acscentsci.9b00055

134. Liu C-H, Korablyov M, Jastrzębski S, et al (2022) RetroGNN: Fast Estimation of Synthesizability for Virtual Screening and De Novo Design by Learning from Slow Retrosynthesis Software. J Chem Inf Model 62:2293–2300. https://doi.org/10.1021/acs.jcim.1c01476

135. Degen J, Wegscheid-Gerlach C, Zaliani A, Rarey M (2008) On the art of compiling and using “drug-like” chemical fragment spaces. ChemMedChem 3:1503–1507. https://doi.org/10.1002/cmdc.200800178

136. Gaulton A, Bellis LJ, Bento AP, et al (2012) ChEMBL: A large-scale bioactivity database for drug discovery. Nucleic Acids Res 40:1100–1107. https://doi.org/10.1093/nar/gkr777

137. Kim S, Chen J, Cheng T, et al (2023) PubChem 2023 update. Nucleic Acids Res 51:D1373–D1380. https://doi.org/10.1093/nar/gkac956

138. Reeves S, DiFrancesco B, Shahani V, et al (2020) Assessing methods and obstacles in chemical space exploration. Appl AI Lett 1:e17. https://doi.org/10.1002/ail2.17

139. Efraimidis PS, Spirakis PG (2006) Weighted random sampling with a reservoir. Inf Process Lett 97:181–185. https://doi.org/10.1016/j.ipl.2005.11.003

140. Kruskal WH, Wallis WA (1952) Use of Ranks in One-Criterion Variance Analysis. J Am Stat Assoc 47:583–621. https://doi.org/10.1080/01621459.1952.10483441

141. Mann HB, Whitney DR (1947) On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. Ann Math Stat 18:50–60. https://doi.org/10.1214/aoms/1177730491

142. Šidák Z (1967) Rectangular Confidence Regions for the Means of Multivariate Normal Distributions. J Am Stat Assoc 62:626–633. https://doi.org/10.1080/01621459.1967.10482935

143. Virtanen P, Gommers R, Oliphant TE, et al (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nat Methods 17:261–272. https://doi.org/10.1038/s41592-019-0686-2

144. Seabold S, Perktold J (2010) statsmodels: Econometric and statistical modeling with Python. In: 9th Python in Science Conference

145. Bickerton GR, Paolini GV, Besnard J, et al (2012) Quantifying the chemical beauty of drugs. Nat Chem 4:90–98. https://doi.org/10.1038/nchem.1243

146. Fisher RA (1992) Statistical Methods for Research Workers. In: Breakthroughs in Statistics: Methodology and Distribution. Springer, pp 66–70

147. Dunnett CW (1955) A Multiple Comparison Procedure for Comparing Several Treatments with a Control. J Am Stat Assoc 50:1096–1121. https://doi.org/10.1080/01621459.1955.10501294

148. Pearson K (1901) LIII. On lines and planes of closest fit to systems of points in space. Lond Edinb Dublin Philos Mag J Sci 2:559–572. https://doi.org/10.1080/14786440109462720

149. Hückel E (1937) Grundzüge der Theorie ungesättigter und aromatischer Verbindungen. Z Für Elektrochem Angew Phys Chem 43:752–788. https://doi.org/10.1002/bbpc.19370430907

150. Lepetit C, Chermette H, Gicquel M, et al (2007) Description of Carbo-oxocarbons and Assessment of Exchange-Correlation Functionals for the DFT Description of Carbo-mers. J Phys Chem A 111:136–149. https://doi.org/10.1021/jp064066d

151. Voršilák M, Svozil D (2017) Nonpher: computational method for design of hard-to-synthesize structures. J Cheminformatics 9:1–7. https://doi.org/10.1186/s13321-017-0206-2

152. Halgren TA (1996) Merck Molecular Force Field. J Comput Chem 17:490–519. https://doi.org/10.1002/(SICI)1096-987X(199604)17:5/6<520::AID-JCC2>3.0.CO;2-W

153. ChemAxon Extended SMILES and SMARTS - CXSMILES and CXSMARTS. https://docs.chemaxon.com/display/docs/chemaxon-extended-smiles-and-smarts-cxsmiles-and-cxsmarts.md. Accessed 1 Oct 2023

154. Hopcroft JE, Karp RM (1971) N5/2 Algorithm for Maximum Matchings in Bipartite Graphs. IEEE, pp 122–125

155. The HDF Group (1997) Hierarchical Data Format, version 5. https://www.hdfgroup.org/HDF5/

156. Scheirer CJ, Ray WS, Hare N (1976) The Analysis of Ranked Data Derived from Completely Randomized Factorial Designs. Biometrics 32:429–434. https://doi.org/10.2307/2529511

157. Conover WJ, Iman RL (1981) Rank Transformations as a Bridge Between Parametric and Nonparametric Statistics. Am Stat 35:124–129. https://doi.org/10.2307/2683975

158. Pedregosa F, Varoquaux G, Gramfort A, et al (2011) Scikit-learn: Machine Learning in Python. J Mach Learn Res 12:2825–2830

159. Mangiafico SS (2023) rcompanion: Functions to Support Extension Education Program Evaluation. Rutgers Cooperative Extension, New Brunswick, New Jersey

160. Hawkins PCD, Skillman AG, Nicholls A (2007) Comparison of Shape-Matching and Docking as Virtual Screening Tools. J Med Chem 50:74–82. https://doi.org/10.1021/jm0603365

161. Renz P, Van Rompaey D, Wegner JK, et al (2019) On failure modes in molecule generation and optimization. Drug Discov Today Technol 32–33:55–63. https://doi.org/10.1016/j.ddtec.2020.09.003

162. Auer P, Cesa-Bianchi N, Fischer P (2002) Finite-time Analysis of the Multiarmed Bandit Problem. Mach Learn 47:235–256. https://doi.org/10.1023/A:1013689704352

163. Kocsis L, Szepesvári C (2006) Bandit Based Monte-Carlo Planning. In: Machine Learning: ECML 2006. Springer, pp 282–293

164. Hart P, Nilsson N, Raphael B (1968) A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Trans Syst Sci Cybern 4:100–107. https://doi.org/10.1109/TSSC.1968.300136

165. Grygorian A, Iacob IE (2018) A Concise Proof of the Triangle Inequality for the Jaccard Distance. Coll Math J 49:363–365

166. Kerstjens A, De Winter H (2023) A molecule perturbation software library and its application to study the effects of molecular design constraints. J Cheminformatics 15:89. https://doi.org/10.1186/s13321-023-00761-5

167. Tukey JW (1949) Comparing individual means in the analysis of variance. Biometrics 5:99–114. https://doi.org/10.2307/3001913

168. Gansner ER, North SC (2000) An open graph visualization system and its applications to software engineering. Softw Pract Exp 30:1203–1233. https://doi.org/10.1002/1097-024X(200009)30:11<1203::AID-SPE338>3.0.CO;2-N

169. Guimaraes GL, Sanchez-Lengeling B, Outeiral C, et al (2017) Objective-Reinforced Generative Adversarial Networks (ORGAN) for Sequence Generation Models. arXiv:1705.10843. https://doi.org/10.48550/arXiv.1705.10843

170. Weininger D (1988) SMILES, a Chemical Language and Information System: 1: Introduction to Methodology and Encoding Rules. J Chem Inf Comput Sci 28:31–36. https://doi.org/10.1021/ci00057a005

171. Krenn M, Häse F, Nigam A, et al (2020) Self-referencing embedded strings (SELFIES): A 100% robust molecular string representation. Mach Learn Sci Technol 1:045024. https://doi.org/10.1088/2632-2153/aba947

172. Dost K, Pullar-Strecker Z, Brydon L, et al (2023) Combatting over-specialization bias in growing chemical databases. J Cheminformatics 15:53. https://doi.org/10.1186/s13321-023-00716-w

173. Métivier J-P, Lepailleur A, Buzmakov A, et al (2015) Discovering Structural Alerts for Mutagenicity Using Stable Emerging Molecular Patterns. J Chem Inf Model 55:925–940. https://doi.org/10.1021/ci500611v

174. Limban C, Nuţă DC, Chiriţă C, et al (2018) The use of structural alerts to avoid the toxicity of pharmaceuticals. Toxicol Rep 5:943–953. https://doi.org/10.1016/j.toxrep.2018.08.017

175. Bemis GW, Murcko MA (1996) The properties of known drugs. 1. Molecular frameworks. J Med Chem 39:2887–2893. https://doi.org/10.1021/jm9602928

176. Schneider G, Fechner U (2005) Computer-based de novo design of drug-like molecules. Nat Rev Drug Discov 4:649–663. https://doi.org/10.1038/nrd1799

177. Zhavoronkov A, Ivanenkov YA, Aliper A, et al (2019) Deep learning enables rapid identification of potent DDR1 kinase inhibitors. Nat Biotechnol 37:1038–1040. https://doi.org/10.1038/s41587-019-0224-x

178. Jiménez-Luna J, Grisoni F, Schneider G (2020) Drug discovery with explainable artificial intelligence. Nat Mach Intell 2:573–584. https://doi.org/10.1038/s42256-020-00236-4

179. (2023) For chemists, the AI revolution has yet to happen. Nature 617:438–438. https://doi.org/10.1038/d41586-023-01612-x

180. Waldman M, Fraczkiewicz R, Clark RD (2015) Tales from the war on error: the art and science of curating QSAR data. J Comput Aided Mol Des 29:897–910. https://doi.org/10.1007/s10822-015-9865-0

181. Rodrigues T (2019) The good, the bad, and the ugly in chemical and biological data for machine learning. Drug Discov Today Technol 32:3–8. https://doi.org/10.1016/j.ddtec.2020.07.001

182. Kolmar SS, Grulke CM (2021) The effect of noise on the predictive limit of QSAR models. J Cheminformatics 13:92. https://doi.org/10.1186/s13321-021-00571-7

183. O’Boyle N, Dalke A (2018) DeepSMILES: An Adaptation of SMILES for Use in Machine-Learning of Chemical Structures. ChemRxiv. https://doi.org/10.26434/chemrxiv.7097960.v1

184. Mercado R, Rastemo T, Lindelöf E, et al (2021) Graph networks for molecular design. Mach Learn Sci Technol 2:025023. https://doi.org/10.1088/2632-2153/abcf91

185. Xiong J, Xiong Z, Chen K, et al (2021) Graph neural networks for automated de novo drug design. Drug Discov Today 26:1382–1393. https://doi.org/10.1016/j.drudis.2021.02.011

186. Wang Y, Wang J, Cao Z, Barati Farimani A (2022) Molecular contrastive learning of representations via graph neural networks. Nat Mach Intell 4:279–287. https://doi.org/10.1038/s42256-022-00447-x

187. Wang J, Wang X, Sun H, et al (2022) ChemistGA: A Chemical Synthesizable Accessible Molecular Generation Algorithm for Real-World Drug Discovery. J Med Chem 65:12482–12496. https://doi.org/10.1021/acs.jmedchem.2c01179

188. Nigam A, Pollice R, Aspuru-Guzik A (2022) Parallel tempered genetic algorithm guided by deep neural networks for inverse molecular design. Digit Discov 1:390–404. https://doi.org/10.1039/D2DD00003B

189. Nigam A, Friederich P, Krenn M, Aspuru-Guzik A (2020) Augmenting Genetic Algorithms with Deep Neural Networks for Exploring the Chemical Space. arXiv:1909.11655. http://arxiv.org/abs/1909.11655

190. Grantham K, Mukaidaisi M, Ooi HK, et al (2022) Deep Evolutionary Learning for Molecular Design. IEEE Comput Intell Mag 17:14–28. https://doi.org/10.1109/MCI.2022.3155308

191. Kwon S, Bae H, Jo J, Yoon S (2019) Comprehensive ensemble in QSAR prediction for drug discovery. BMC Bioinformatics 20:521. https://doi.org/10.1186/s12859-019-3135-4

192. Graff DE, Shakhnovich EI, Coley CW (2021) Accelerating high-throughput virtual screening through molecular pool-based active learning. Chem Sci 12:7866–7881. https://doi.org/10.1039/D0SC06805E

193. Khalak Y, Tresadern G, Hahn DF, et al (2022) Chemical Space Exploration with Active Learning and Alchemical Free Energies. J Chem Theory Comput 18:6259–6270. https://doi.org/10.1021/acs.jctc.2c00752

194. Zaverkin V, Holzmüller D, Steinwart I, Kästner J (2022) Exploring chemical and conformational spaces by batch mode deep active learning. Digit Discov 1:605–620. https://doi.org/10.1039/D2DD00034B

195. Dodds M, Guo J, Löhr T, et al (2023) Sample Efficient Reinforcement Learning with Active Learning for Molecular Design. ChemRxiv. https://doi.org/10.26434/chemrxiv-2023-j88dg

196. Yang X-S (2014) Nature-Inspired Optimization Algorithms. Elsevier

197. Brown DG, Boström J (2018) Where Do Recent Small Molecule Clinical Development Candidates Come From? J Med Chem 61:9442–9468. https://doi.org/10.1021/acs.jmedchem.8b00675

198. Dragovich PS, Haap W, Mulvihill MM, et al (2022) Small-Molecule Lead-Finding Trends across the Roche and Genentech Research Organizations. J Med Chem 65:3606–3615. https://doi.org/10.1021/acs.jmedchem.1c02106

199. Rodrigues T, Schneider G (2014) Flashback Forward: Reaction-Driven De Novo Design of Bioactive Compounds. Synlett 25:170–178. https://doi.org/10.1055/s-0033-1340216

200. Sundin I, Voronov A, Xiao H, et al (2022) Human-in-the-loop assisted de novo molecular design. J Cheminformatics 14:1–16. https://doi.org/10.1186/s13321-022-00667-8

201. Choung O-H, Vianello R, Segler M, et al (2023) Learning chemical intuition from humans in the loop. ChemRxiv. https://doi.org/10.26434/chemrxiv-2023-knwnv

202. Steiner S, Wolf J, Glatzel S, et al (2019) Organic synthesis in a modular robotic system driven by a chemical programming language. Science 363:eaav2211. https://doi.org/10.1126/science.aav2211

203. Coley CW, Thomas DA, Lummiss JAM, et al (2019) A robotic platform for flow synthesis of organic compounds informed by AI planning. Science 365:. https://doi.org/10.1126/science.aax1566

204. Rohrbach S, Šiaučiulis M, Chisholm G, et al (2022) Digitization and validation of a chemical synthesis literature database in the ChemPU. Science 377:172–180. https://doi.org/10.1126/science.abo0058

Curriculum vitae

Education

Ph. D. Pharmaceutical Sciences 2019 – 2024, University of Antwerp	**Fields of study:** De novo drug design, Cheminformatics, Molecular Modelling **Thesis:** Computational design of synthesizable molecules by imitating reference chemistry
M. Sc. Pharmaceutical Modelling 2017 – 2019, Uppsala University	**Fields of study:** Molecular Modelling, Bioinformatics, Machine Learning, Pharmacometrics **Thesis:** Optimization of molecular docking protocols using tailor-made datasets
B. Sc. Biochemistry & Molecular Biology 2013 – 2017, University of the Basque Country	**Fields of study:** Biochemistry, Biology, Physiology, Genetics, Pharmacology **Thesis:** Interaction between the eCB and S1P systems in rat brain
Baccalaureate Life Sciences 2011 – 2013, I.E.S. Ricardo Bernardo	**Fields of study:** Biology, Chemistry, Physics, Mathematics

Publications

Kerstjens, A., De Winter, H. Molecule auto-correction to facilitate molecular design. *J. Comput. Aided Mol. Des.* (in press) (2024)

Kerstjens, A., De Winter, H. A molecule perturbation software library and its application to study the effects of molecular design constraints. *J Cheminform* 15, 89 (2023). https://doi.org/10.1186/s13321-023-00761-5

Kerstjens, A., De Winter, H. LEADD: Lamarckian evolutionary algorithm for de novo drug design. *J Cheminform* 14, 3 (2022). https://doi.org/10.1186/s13321-022-00582-y

Conference presentations

Kerstjens, A. De novo design of synthetically accessible molecules using an evolutionary algorithm. 12th International Conference on Chemical Structures (2022).

Acknowledgements

At some point I'll drop off this book in the department, probably in the library buried among a pile of theses. And there it will lay, dormant for possibly decades. Yet at some point a curious last year PhD student will open it looking for inspiration. And they will uncover horrors that should've remained hidden. Stories of a guy who spend 4 years designing molecules straight out of a chemist's worst nightmare, and somehow got paid to do so.

In my defense, such feat wouldn't have been possible without the support of my supervisor, Hans De Winter. The only thing I knew about Hans when I applied for this position was that he was somewhat of a local TV celebrity. During our first online interview my potato notebook overheated and the camera died. But while he couldn't see me, I could hear him laugh. My camera dying probably didn't make a good first impression (I made sure to buy a new one the next day), but his attitude towards problems sure made a good impression on me. I came to find out that Hans has a friendly and relaxed personality.

One of the things I appreciate most about his attitude towards supervision is that he isn't controlling. My research project was originally meant to revolve around the development of antibiotics. Much to the dismay of the poor microbiologist in my jury, that didn't end up happening. I quickly realized that I was more passionate about method development than applying existing methods, and I was allowed to steer the research into the directions that excited me the most. Of course with great power comes great responsibility, and we encountered more than one setback. But I would like to think that I made the most out of bad situations. Another of his virtues is openness. Naturally a professor is more experienced than a student, and when faced with a student's proposal they could very easily dismiss it. But Hans didn't dismiss my ideas. He was open to them, and encouraged me to "Just try it" or "Just do it" (Nike please don't sue). Naturally not everything winds up working as intended, but self-driven failure is an important component of personal growth. I don't take the opportunities I was given for granted. Thank you Hans.

My studies wouldn't have been nearly as enjoyable without the company of all my colleagues. When I first joined the lab in 2019 it was rather empty. We even fit around the table in the kitchen. Then around 2020-2021 the population exploded. The influx of medicinal chemists led to the banishment of computational chemists to office A2.16. To add insult to injury they even misspelled my surname on the door. I would hold a grudge against them, if it weren't for the fact that I got the big table next to the window. I can't possibly thank all of you individually since you are way too many. So thank you to all chemists that roam (and have roamed) the hallways of the lab, from seniors to insolent students.

I would like to acknowledge some notorious characters individually. Let me tell you about my office mates.

Olivier used to quietly slouch in his chair, but later on he became more talkative. He would occasionally stand up, draw something on the whiteboard with a dried out marker that you could barely see, and rope you into hour long conversations. During these totally scientific conversations I found out that his true passions were luxury hand bags and lobbying for the oil industry. He would tell you with a straight face that energy providers weren’t greedy. Then when you turned around he would pull up the Exxon Mobil stock chart with the 1 minute candles. I’d like to think that he wound up seeing the error in his ways, as he’d eventually relocate to Antwerpen to save on gas.

Joep (a.k.a. DJ Wals) joined the lab recently, but quickly became the soul of the group. He is a teaching assistant on a 6-year PhD trajectory, so he has lots of time to be a nuisance. To keep him busy we put him in charge of absolutely everything in the lab. He organizes the group meetings, orders/buys sandwiches and drinks, and harasses you via e-mail to upload the slides to Teams. Somehow he still found time for two walks around the pond per day. The rest of us joined under duress, for if you didn’t he would play high-energy electronic music in the office. Now with deadlines coming up he must make up for lost time, and he is under strict orders to work during the weekends.

Roy is one of the latest additions to the computational roster. I was super excited when he first joined because he was allegedly going to do *de novo* molecular design, but alas I was deceived. What he lacks in research taste he makes up for in attitude (in a good way). He is always keen on participating in various activities, and legend says he doesn’t know how to say no. This, coupled with his politeness, makes him the perfect victim for distracting conversations when your calculations are running or you don’t feel like working.

To my office mates, Kenneth, Olivier, Joep, Roy and Stijn, thank you for all the discussions (scientific or not), brainstorming and fun moments we had together. You made the atmosphere in the office not only sweaty and full of carbon dioxide, but also fun and vibrant. At least you now know how to open the crusty windows. Unfortunately I didn’t get to know all of you equally well, but such is the nature of ephemeral academic stays, working from home, and the virus that shall not be named.

I also wish to thank all the VrijMiBo attendees, which was always one of the highlights of the week and good motivation to come to the university. Special shout-out to the VrijMiBo “big four” Joep, Philipp and Nicolò. I thoroughly enjoyed the silly conversations, jokes and games. Feasting on bears and snakes was a plus.

I extend my gratitude to all the members of my doctoral jury. To the members of the Individual Doctoral Committee: Paul Cos, Yann Sterckx and Wouter Herrebout. The subject matter of this research may have drifted away from what was originally foreseen. I greatly appreciate the effort of critically evaluating research outside one’s domain of expertise. To the external jury members: Greg Landrum and Mazen Ahmad. Having you as opponents is a privilege, despite being a bit daunting. Thank you for finding time in your schedules to not only read the thesis and examine me, but also to travel to Antwerp for the occasion. Hopefully the food and drinks are worthwhile compensation.

Many thanks to the the Fonds Wetenschappelijk Onderzoek (FWO) for generously funding this research, and to the Vlaams Supercomputing Centrum (VSC) for providing many of the computational resources that powered this research. Special thanks to the CalcUA support staff for their trainings and personalized help in setting up and troubleshooting my software on the computer cluster.

This thesis may be the conclusion of an academic trajectory that spans far beyond the last 4 years. I feel it adequate to also acknowledge the entities that made the journey up to this point possible. Thank you to the Dr. Jost Henkel Stiftung for supporting me with a scholarship during my bachelors and masters studies. Thank you to the Svensk-Spanska Stiftelsen for contributing to my master expenses. And of course, thank you to my parents for being the most generous funding agency in the world. Lastly, I wish to acknowledge all European taxpayers, for you are the real unsung heroes that enable academic research like this. Hopefully it will make an impact in your lives sometime in the future.

A good support network extends beyond the academic environment. I would like to thank all my friends for keeping me entertained outside working hours, be it offline or online. Thank you to my girlfriend for providing emotional support and listening to all my nagging and crazy ideas. She has kept my morale up throughout these years with excursions, cultural activities, nights out, boiled eggs and chamomile.

And of course, I owe arguably the most to my parents. When I finished high school I wanted to study law to become a notary. All the forms had been filled, but before I had the chance to hand them in my mother found out, and unilaterally decided that science was the way to go. Now I have more fun but less job security and money. So thanks mom? In all seriousness my parents have been nothing but supportive. Beyond feeding me and putting up with me for many years, they have helped me financially, helped me look for accommodation, helped me move, driven me around to do paperwork (or even done it themselves), and of course provided lots of encouragement. So from the bottom of my heart, thank you very much to both of you.