4 The Molecule Evoluator. An interactive evolutionary algorithm for the design of drug-like molecules

Eric-Wubbo Lameijer[1], Joost N. Kok[2], Thomas Bäck[2] and Ad P. IJzerman[1]
[1]Leiden/Amsterdam Center for Drug Research, Division of Medicinal Chemistry, PO Box 9502, 2300RA Leiden, The Netherlands
[2]Leiden Institute of Advanced Computer Science, Leiden University, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands

This chapter was first published in the Journal of Chemical Information and Modeling, reference: Lameijer, E.W.; Kok, J.N.; Bäck, T.; IJzerman, A.P. The Molecule Evoluator. An Interactive Evolutionary Algorithm for the Design of Drug-Like Molecules. *J. Chem. Inf. Model.*, **2006**, *46*(2), 545-552.

Abstract

We developed a software tool to design drug-like molecules, the "Molecule Evoluator", which we introduce and describe here. An atom-based evolutionary approach was used allowing both several types of mutation and crossover to occur. The novelty we claim is the unprecedented *interactive* evolution, in which the user acts as a fitness function. This brings a human being's creativity, implicit knowledge and imagination into the design process, next to the more standard chemical rules. Proof-of-concept was demonstrated in a number of ways, both computational and in-the-lab. Thus, we synthesized a number of compounds designed with the aid of the Molecule Evoluator. One of these is described here, a new chemical entity with activity on α-adrenergic receptors.

Introduction

It has been estimated that the combination of the four elements C, N, O and S only may lead to the design of 10^{60} molecules with maximally 30 non-hydrogen atoms[1]. Currently, only over 26 million (in between 10^7 and 10^8) organic and inorganic compounds have been synthesized since the foundation of organic chemistry in the 19th century[2]. Obviously, designing new molecules (*de novo* design) is crucial to cover more of the "chemical space". Computational methods have been employed to this end, among which so-called evolutionary algorithms (reviewed in ref. 3). These algorithms are based on principles from biology, such as natural selection and Darwinian evolution through mutation and crossover.

There are at least two problems with the application of evolutionary algorithms in drug design. The first is that linear 'genes' as used in biological evolution are ill-suited to represent molecules. Molecules are graphs that have to obey certain chemical rules. Converting a molecule into a bit string or fixed-size vector of numbers as used in conventional evolutionary algorithms will therefore often result in mutation and crossover operations that produce invalid molecules.

The second problem relates to natural selection and a useful fitness function. While currently the most common methods use a similarity index to a reference compound[4-7] or calculations of the binding strength to a target[8,9], only a few examples have been published of successful applications of evolutionary algorithms to find new, biologically active structures. Only Schneider et al[7] claimed success since their algorithm found a ligand with a similar kind of activity as the lead compound, be it approximately 1000 times less potent.

In this study we propose a new approach for the use of evolutionary algorithms in *de novo* drug design, called the "Molecule Evoluator", to address these problems. We introduce an atom-based method with a set of mutation operators that contains all one-atom and one-bond mutations. This will enable a fuller search of the chemical space and finer optimization of the molecular structure than is possible with most other published methods. Secondly, we use another approach to natural selection that is new in evolutionary algorithms in drug design: we make the medicinal chemist the "fitness function" of the proposed structures. This would largely eliminate structures that are difficult to make in the laboratory and enable the program to optimize the molecular structure by using the chemist's knowledge about structure-activity relationships.

We will first introduce a new representation of molecules, the so-called TreeSMILES notation, and the evolutionary operators we used. Subsequently we will discuss how the user's choices affect the fitness of the molecules. We will then shortly describe the graphical user interface of the Molecule Evoluator, together with the results of some of our experiments in interactive drug design.

Methods

Molecule representation

A molecule can be considered to be a connected graph consisting of one or more atoms (nodes) connected by bonds (undirected edges). One of the main constraining rules of chemistry is the valence of each type of atom. Moreover, atoms such as sulfur can have several valence states, of which bivalent sulfur and hexavalent sulfur are the most common. In our genotype, the starting point in each evolutionary algorithm, we handle this by creating a separate symbol for each valence state, so that divalent sulfur atoms are encoded by "S" and hexavalent sulfur atoms by "Sh". Thus, whether a graph represents a valid molecule does not merely depend on the structure of the graph, but also on the identity of each particular node.

Molecules are graphs that can be of different sizes; they may also contain branches and cycles. Since they too are subject to the laws of chemistry they additionally have to obey certain rules that usually prohibit changing one node/edge without changing its neighbors. To develop a computer representation of a molecule that can store these properties and that is convenient to mutate and cross over, we looked at some common molecular data formats.

There are various ways to represent molecules on the computer. Nowadays, the two most common representations are the MOL-file format[10], which is a list of atoms and bonds in the molecule, and the SMILES-notation[11], a line notation/string that is human-readable and that can easily be transformed into a 2D-structure. In the SMILES notation the nonlinear parts of a molecule are encoded with brackets to indicate branches and numbers to label rings, as is illustrated in Figure 4.1A. By default the hydrogen atoms are not incorporated into the SMILES notation, since their presence can be deduced from the rules of chemical valence.

Neither of these representations is ideal, however, when molecules are mutated and crossed over. Since chemical valence rules explicitly state how many bonds any

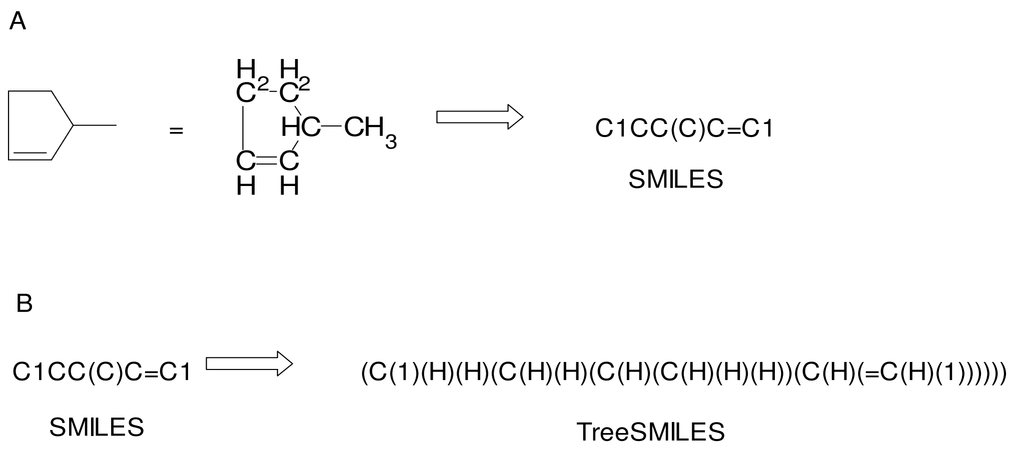


Figure 4.1:
A) Example of a SMILES representation of a molecule. One bond of each ring is chosen at random (in this case, the bond from the top left to the bottom left atom) and is designated by a unique number. Both atoms participating in a ring bond get the bond number(s) immediately after them in the line notation. Branches are indicated by brackets.
B) SMILES versus TreeSMILES. By making the hydrogen atoms explicit going from SMILES to TreeSMILES the notation becomes less compact and less human-readable. However, the positions available for substitution are now readily interpreted by the computer.

atom has, both the most common 2D-MOL-file format and the SMILES notation take the hydrogen atoms and the bonds to which they are attached for granted. We should therefore calculate for each atom whether it has the right number of hydrogen atoms to be mutated in some way. To avoid these recalculations, we decided to explicitly add the hydrogen atoms to the representation. Also, we decided to use a SMILES-like structure as our main molecule representation, particularly for reasons of computational efficiency. Making the hydrogen atoms explicit we obtain a bracket-rich, expanded SMILES representation (Figure 1B) to which we can apply mutation by relatively simple algorithms. We coin this chemical notation “TreeSMILES”.

Crossover and mutation

Crossover is implemented like crossover in standard genetic programming[12]: subtrees of two different molecules are selected and swapped. The only complication in the present study is that the trees represent graphs and can contain cycles, while subgraphs are not allowed to contain incomplete cycles. So when a subtree at a random "root" atom is selected, the subtree is first checked for unmatched ring bonds, and if these are present the current root is discarded and another subtree is selected for crossover.

Mutation, however, is the most important variance operator in the Molecule Evoluator. Theoretically, changing a graph into any other graph can be performed by a limited set of operations: adding nodes, adding edges, deleting nodes, deleting edges. While this would be sufficient from a graph-theoretical point of view, these operations are more complicated in a chemical system since the valence rules must be obeyed, and the graph must remain connected: deleting the indicated carbon atom in Figure 4.2 would also require deleting its three attached hydrogen atoms and attachment of a hydrogen that replaces the carbon. In the Molecule Evoluator, deleting an atom involves removal of the hydrogen atoms attached to it and renaming the atom itself into a hydrogen atom.

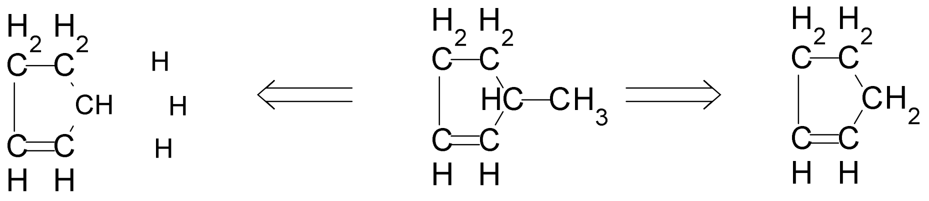


Figure 4.2. Why simple mutations are often wrong mutations. When removing the rightmost carbon from the middle molecule, only removing the carbon (left picture) results in a wrong molecule: due to the loose hydrogen atoms the structure is unconnected (and therefore not a valid molecule), also the carbon atom to which it was connected has three bonds instead of four, violating chemical valence rules and also making the left molecule invalid. The molecule on the right shows the valid version of this mutation: the to-be-deleted atom is changed into a hydrogen and its own hydrogen atoms are removed.

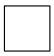
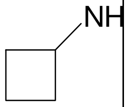
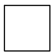
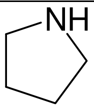
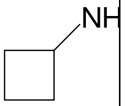
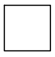
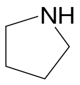
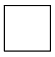
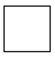
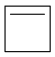
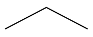
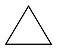
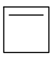
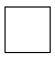
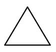
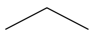
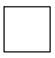
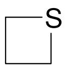
The implemented mutations, graphically shown in Table 4.1, are as follows:

1) *Add atom/group*: this replaces a hydrogen atom in the molecule with a non-hydrogen atom or a larger chemical group such as a phenyl group. In the case of atom addition, the remaining bonds of the added atom are filled with hydrogen atoms.
2) *Insert atom*: also adds an atom, but does this by inserting the new atom (which should have a valence of two or higher) into a bond. The remaining bonds of the new atom are completed by adding hydrogen atoms to it.
3) *Delete atom*: this removes an atom that is attached to only one non-hydrogen atom (with a single bond) by first deleting the hydrogen atoms attached to it, and renaming the atom to a hydrogen atom.
4) *Uninsert atom*: This removes an atom that has exactly two non-hydrogen neighbors. It removes the atom and its hydrogen atoms and subsequently creates a bond between its two neighboring non-hydrogen atoms.
5) *Increase bond order*: if two atoms that are bonded to each other both have at least one hydrogen atom, those hydrogen atoms are removed and an extra bond is created between the atoms (the bond order is increased from single to double or from double to triple).
6) *Create ring*: similar to *increase bond order*, but works between two atoms that are not bonded to each other. These atoms are connected using a single bond (the two hydrogen atoms are changed into ring indices).
7) *Decrease bond order*: if there is a double or triple bond between two atoms, its bond order is decreased by one and a hydrogen atom is attached to each of the two atoms.
8) *Break ring*: this mutation chooses a single bond in a ring, breaks that bond and adds hydrogen atoms to correct the valences. The algorithm should be able to break any bond in the ring, which is not easy to do with a tree structure[13]. To solve this problem, we converted the TreeSMILES into an adjacency list. In the adjacency list, any ring bond can be broken easily and afterwards a new TreeSMILES representation is built. This is the only mutation where we found it necessary to temporarily convert the TreeSMILES representation of the molecule into an adjacency list format for easier modification. Since other operators such as crossover are more easily implemented for a TreeSMILES string, we decided not to use the adjacency list for the other mutations. Changing representations is probably a convenient way to accommodate different mutations. To our knowledge, this technique has not been used before in evolutionary algorithms in *de novo* design, and is probably also rare in evolutionary algorithms in general. However,

we think that it might also have applications outside the Molecule Evoluator.

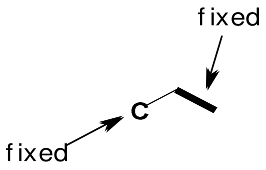
9) *Mutate atom*: a non-hydrogen atom is changed into another non-hydrogen atom which has a valence of at least the number of bonds of the original atom with other non-hydrogen atoms.

Table 4.1: Schematic overview of the different mutations in the Molecule Evoluator. Most leave the TreeSMILES string fairly intact and can be performed by string editing. The only exception is the "break ring" mutation, which can substantially rearrange the TreeSMILES.

Mutation name	**Initial structure**	**Final structure**	**Initial TreeSMILES**	**Final TreeSMILES**
Add atom			...(C(H)(*H*)(C…	...(C(H)(**N(H)(H)**)(C…
Insert atom			...(C(H)(H)(C...)...	...(C(H)(H)(**N(H)**(C...))...
Delete atom			..(C(H)(*N(H)(H)*)(C...	...(C(H)(**H**)(C...
Uninsert atom			...(C(H)(H)*(N(H)*(C...*))*...	...(C(H)(H)(C...)...
Increase bond order			...(C*(H)*(H)(C*(H)*(H)(...	...(C(H)(**=**C(H)(…
Create ring			(C(*H*)(H)(H)(C(H) (H)(C(*H*)(H)(H)))	(C(**1**)(H)(H)(C(H)(H)(C(**1**)(H)(H)))
Decrease bond order			...(C(H)(=C(H)(…	...(C(**H**)(H)(C(**H**)(H)(...
Break ring			(C(1)(H)(H)(C(H)(H)(C(1)(H)(H)))	(C(C(H)(H)(H))(H)(H)(C (H)(H)(H)))
Mutate atom			...(C(H)(H)(*C(H)(H)*(C...	…(C(H)(H)(**S**(C...

We also allow the user to select atoms and bonds that will remain unaltered by crossover and mutation (‘fix’ option). Therefore we have further modified the data structure of the TreeSMILES by using an array of character pairs instead of a normal string/array of characters, in which the first character of the pair is the normal TreeSMILES character and the second character is a flag, which indicates whether the atom or bond designated by the first character can be modified (Figure 4.3).

Figure 4.3. The TreeSMILES notation of n-propane (C_3H_8) that incorporates fixable characters ("charf"s). In this example, one atom and one bond are fixed.



(	C	(	H	)	(	H	)	(	H	)	(	C	(	H	)	(	H	)	(	C	(	H	)
ø	F	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	F	ø	ø	ø	ø

(	H	)	(	H	)	)	)	)
ø	ø	ø	ø	ø	ø	ø	ø	ø

Fitness

The final component of the evolutionary algorithm is the fitness function. So far, investigations on *de novo* design with evolutionary algorithms have used several types of a fitness function, including:

- similarity to a target molecule[4,5]
- QSAR-functions[13]
- ligand-protein docking[8,9]
- experiment[14]

For drug design, each of these methods has its advantages and disadvantages. Similarity approaches often result in molecules very similar to the target molecule, in many cases even the target molecule itself, which is not useful for designing truly novel molecules. QSAR functions should be able to optimize activity, yet have important disadvantages. First they require quite a lot of reprogramming for each new class of molecules that is investigated. Secondly, they are generally difficult to use for finding very active compounds since they grow less and less reliable as the molecular structures deviate more from the average structure (and thereby activity) of the training dataset[15]. Unfortunately, this is exactly what would happen during optimization. The last theoretical method ("docking") is still too inaccurate for optimization, and may yield molecules with an activity that is orders of magnitude lower than the calculated value[9]. Experiments as fitness function, finally, are generally slow and expensive, and so far have only been performed for a class of molecules that was particularly easy to synthesize[14]. Thus experimental fitness has yet to prove to be practical in a more realistic drug design scenario.

As an entirely different approach, we decided to use the *user* as a fitness function. This concept has been recently employed in other application areas and is often called interactive evolutionary computing[16-18]. While a user cannot know the binding strength of a given molecule, this defect may not be much worse than the inaccuracy of scoring functions. A definite advantage in letting the user choose would be that intensive feedback from a medicinal chemist would make the compounds easier to synthesize, and steer the evolution away from areas which have already been explored. Furthermore, the algorithm could still be easily coupled to experimental results or advanced computed fitness functions if so desired. In Algorithm 4.1 the general workflow within the Molecule Evoluator is detailed, from the creation of the initial population to the interactive evolution of new molecules.

Algorithm 4.1: General workflow of the Molecule Evoluator

```
Create initial generation of 1..generation_size molecules
      // ( generation_size set by user, by default 12)
   For each molecule, get molecule structures from one of
   these sources:
   a. MOL or SD-files
   b. Sketches by user in editor of Molecule Evoluator
   c. Random generation of molecules (see algorithm 4.2)
```

```
Evolve:
      Repeat while the user wants to continue:
          -The user may set or change ratios between
          modification, combination and de novo molecule
          generation.
          -The user selects m most interesting molecules of
          the current generation.
          Copy the m selected molecules to the next
          generation.
          -For each of the ( generation_size - m ) molecule
          slots left in the next generation, do the
          following:
                   -generate a new molecule with one of the
                   three methods in ratios set by the user:
                   -modification: create a new molecule by
                   mutating the old molecule
                   -combination: create a new molecule by
                   crossing over two selected molecules (can
                   be the same molecule - this is allowed in
                   tree-crossover)
                   -de novo generation: create a new molecule
                   by de novo generation, see algorithm 2.
                   -add the new molecule to the next
                   generation.
          -The user may change molecules by fixing (as
          explained in the text) or editing, or replace
          molecules by loading or drawing new molecules.
          -Make the next generation the current generation.
```

As a further example, Algorithm 4.2 performs the random generation of molecules as mentioned in Algorithm 4.1.

Algorithm 4.2: Random generation of a molecule

```
set the current molecule to a methane molecule

pick a random number x from 1 to MaxNumberOfAtoms
repeat x times:
       replace a hydrogen atom of the current molecule
       by a random atom or fragment from the
       atom/fragment database

pick a random number y from 0 to MaxNumberOfRings
repeat y times or until no more rings can be made:
make a ring in the current molecule

pick a random number z from 0 to
MaxNumberOfDoubleBonds
repeat z times or until no more bonds can be oxidized:
       oxidize a random bond in the current molecule,
       so increment the bond order
```

Since a user cannot evaluate as many structures as a computer program and preliminary experiments have shown that users only want to see “good” structures, we added several descriptor calculations to the Molecule Evoluator. The selected descriptors are (physico-)chemical parameters of a compound, such as the number of hydrogen bond donors/acceptors, molecular weight, logP (lipophilicity), Polar Surface Area, the number of rotatable bonds, and the number of aromatic systems and substituents. Physico-chemical parameters of a compound, either calculated or experimentally determined, can also constitute a useful fitness function, for instance in a certain weighed combination. It is increasingly realized that they determine e.g., aspects of absorption and blood-brain barrier passage[19], although they usually are of lesser importance for the interaction with the target protein. Upper and lower bounds for all these descriptors can be set by the user as a filter to create more ‘realistic’ molecules. Additionally, we implemented some ‘chemical’ filters. The aim was to eliminate molecules with undesirable (sub)structures such as strained paracyclophanes. Similarly, molecules not allowed because of Bredt’s rule[20] are also automatically discarded. Both the physical and the chemical filters can be switched on and off.

Results

The graphical user interface of the Molecule Evoluator

When the program is started, a ‘seed’ molecule (indole in this case) can be either loaded or drawn and used for a first generation of derivatives (Figure 4.4). Alternatively, the Molecule Evoluator itself can initialize the population with random molecules. The latter is done by taking the computer’s clock time and using it as a seed in the *rand* function in the C programming language.

As an example one of the suggested molecules (3-methylindole) was selected for yet another round of evolution. Again by pressing the "Go" button a window appears that contains the selected molecule (this so-called elitism is on by default) together with the newly generated analogs (Figure 4.5). The user can again select the most attractive molecules, press “Go”, and this process is repeated until the user has gathered enough ideas.

Comments from medicinal chemists on previous test versions of the program have led us to include three extra features that allow the user more control over the evolution, i) editing the molecules directly, ii) fixing parts of the molecule, and iii) using filters to prevent that certain unrealistic molecules are shown to the user. We next discuss these features in more detail.

Editing molecules is useful when the user wants to start the evolution with a molecule that has not been stored yet in the computer and must be drawn. Additionally, if during the evolution a given molecule suggests a more interesting structure, editing the molecule into the desired structure can be done ‘on-the-fly’. This will allow the user to evolve molecules immediately from the desired structure, rather than wait until it will be generated by the program. Editing the molecules is performed in the "Molecule Edit" window (not shown), which pops up when the user clicks on a molecule in the main window. Editing is quite similar to that in normal chemical drawing programs. After the popup window has been closed, the drawn structure is converted into TreeSMILES-format.

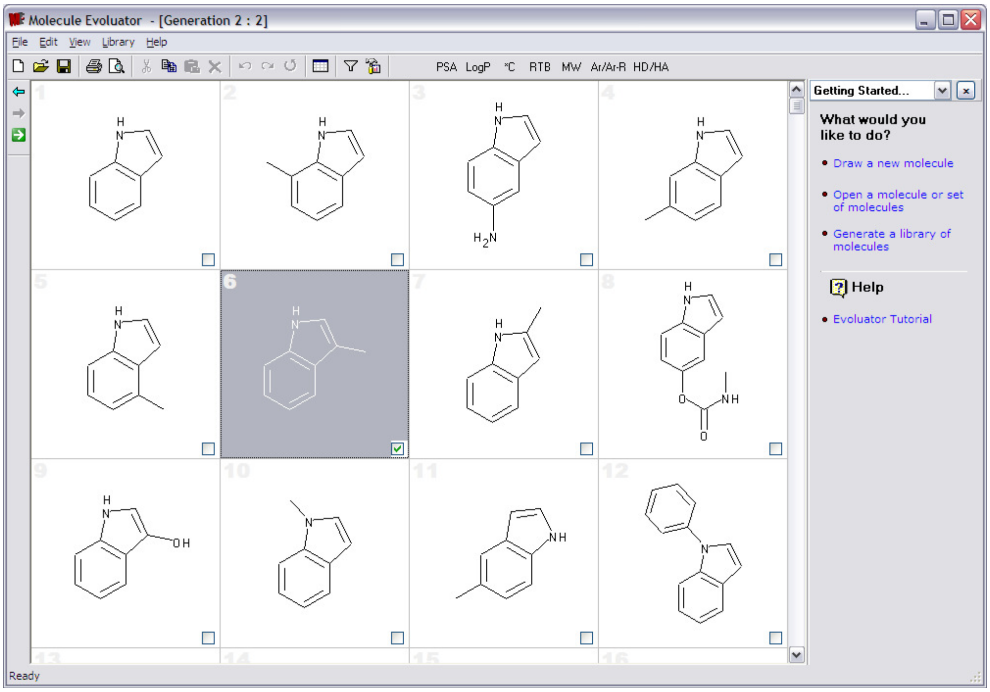
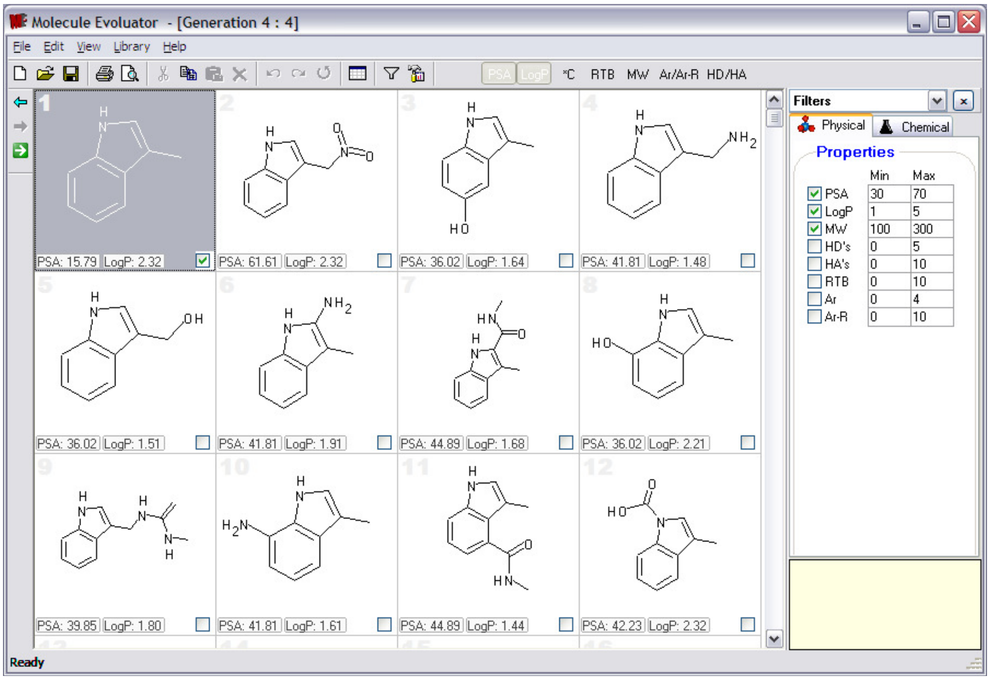

Molecule Evoluator - [Generation 2 : 2]
File Edit View Library Help
PSA LogP °C RTB MW Ar/Ar-R HD/HA
Getting Started...
What would you like to do?
Draw a new molecule
Open a molecule or set of molecules
Generate a library of molecules
Help
Evoluator Tutorial
Ready

Figure 4.4: A molecule, in this case indole (#1), is drawn or loaded as a 'seed' for a population of 12 derivatives generated by pushing the "Go" button (green arrow in left-hand toolbar). In this simple example with atom additions only the methylated analog #6 was selected for further evolution (see Figure 5).

Fixing part(s) of the molecule can be useful in cases where structure-activity relationships demand that a particular, necessary part of the molecule is present in all its descendants. The Molecule Evoluator allows this conservation with the "fix atoms or bonds" option in the "Molecule Edit" window, which enables the user to generate new molecules with the conserved part remaining intact, and only variation on the "free" atoms.



1 PSA: 15.79 LogP: 2.32 ✓	2 PSA: 61.61 LogP: 2.32	3 PSA: 36.02 LogP: 1.64	4 PSA: 41.81 LogP: 1.48
5 PSA: 36.02 LogP: 1.51	6 PSA: 41.81 LogP: 1.91	7 PSA: 44.89 LogP: 1.68	8 PSA: 36.02 LogP: 2.21
9 PSA: 39.85 LogP: 1.80	10 PSA: 41.81 LogP: 1.61	11 PSA: 44.89 LogP: 1.44	12 PSA: 42.23 LogP: 2.32

	Min	Max
✓ PSA	30	70
✓ LogP	1	5
✓ MW	100	300
HD's	0	5
HA's	0	10
RTB	0	10
Ar	0	4
Ar-R	0	10

Figure 4.5: Pressing "Go" (green arrow in the left-hand toolbar) generates mutants of methylindole (see Figure 4) according to preset filters (PSA, logP and MW in right-hand panel). PSA and logP values are displayed in the individual boxes, according to buttons pushed in the upper toolbar.

The filters are the third extra feature. In the "Physical Filters" panel (see also Figure 4.5) the ranges are set within which the physicochemical properties of a molecule must lie for the molecule to be incorporated into the population (for example: molecular weight between 100 and 300). Molecules outside these boundaries imposed by the user will be automatically eliminated by the Molecule Evoluator and will therefore not be shown to the user. Additionally, some chemical structures which are usually undesirable, such as hemiketals, can be forbidden in the "Chemical Filters" panel. The Molecule Evoluator creates offspring molecules using mutation and crossover until feasible molecules – fulfilling all filter conditions – have been found.

In addition to these three main control features, there is a "Parameters" panel (not shown), in which the user can influence the evolutionary process itself. The user is allowed to steer evolution by, amongst other things, enabling/disabling certain kinds of mutations. For example, the “decrease bond order” mutation tends to partially reduce phenyl rings, which is chemically undesirable. As an alternative to fixing the phenyl bonds explicitly, disabling this mutation will protect the bonds from being reduced. This will however also prevent useful mutations, such as those which reduce a ketone (C=O) to an alcohol (CHOH). Another option would be a special version of the "decrease bond order" mutation that does not reduce aromatic rings; this is something we are currently considering. Other mutation types (eight in total) can also be toggled ‘on’ or ‘off’. By default, each of these eight categories of mutations is applied with the same frequency (0.125). When mutation types are disabled, the remaining active mutation types are still applied with identical frequency, so if for example only three mutations are allowed, the probability of each of them is 0.33. Another option is to change the ratio of mutation/crossover. Since we learned that most medicinal chemists prefer mutation over crossover, the default settings are 90% mutation and 10% crossover (and so are applied with probabilities of 0.9 and 0.1, respectively). A third option is to allow the Molecule Evoluator to occasionally add random molecules to the population. The relative amount of random molecules is approximately 16% (so 1-2 new random molecules in a new population). This option is off by default. Lastly, the user can toggle elitism on and off. Elitism conserves the selected molecules and makes them also the first molecules on the screen in the next generation.

Experiments

We conducted a computational experiment first for a simple proof-of-concept. For this we used a dataset of biological activities of neuramidase inhibitors[21]. Using the measured activities as input for the evolutionary algorithm we located the experimental minimum (IC_{50} = 1 nM, a 6300-fold improvement over the original structure) within four generations.

To test whether we could use the Molecule Evoluator to discover interesting new molecules with possible biological activity, we performed an experiment using its option for random molecule generation.

First we generated a library of 10,000 small molecules (150D < MW <250D) with drug-like features: either one or two aromatic rings, 5 or fewer rotatable bonds, 2 or fewer hydrogen donors, 4 or fewer hydrogen acceptors, and a polar surface area of at most $70Å^2$. Out of this library, three sublibraries of 100 compounds were chosen randomly. Each of these sublibraries was presented to a different chemist, who could choose and modify the molecules created by the program. Out of the 300 compounds, 35 were chosen for further investigation.

Checking the latter molecules in the SciFinder[2] and Beilstein databases[22] (in April 2003) we found that six structures represented chemical classes yet unknown in literature. Based on these six core structures ten molecules were designed, of which eight compounds were synthesized successfully. One typical compound with a calculated log P value of 1.65, a calculated polar surface area of 32 Å^2 and a molecular weight of 191D, is shown in Figure 6. It proved active on both α_1- and α_2-adrenergic receptors, i.e. at 10 μM it displaced more than 50% of the radioligand used in binding assays on the two receptor subtypes. The full results of this experimental proof-of-concept are described in chapter 7.

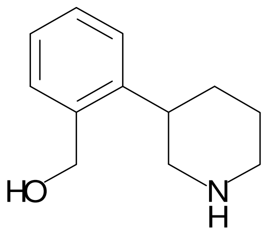


Figure 6. One of the compounds synthesized on the basis of a random generation of virtual molecules by the Molecule Evoluator.

Discussion

In this paper we have presented an evolutionary algorithm to help design new molecules. As reviewed very recently, evolutionary algorithms form one of the approaches to computer-based *de novo* design of drug-like molecules[23]. A large variety of methods is being used to this end, even when confined to evolutionary concepts only. The two most important components of all evolutionary algorithms used in chemistry/drug design are the molecule representation and the fitness function.

The molecule representation

One of the main choices when creating a de novo design algorithm is whether to make the algorithm atom- or fragment-based. Atom-based algorithms work by mutating atoms, and can therefore fine-tune each structure optimally[4,5,13,24]. On the other hand, several investigators construct molecules using larger fragments[7,9,24]. This has the advantage that the representation can be simpler, since there is generally no need for the genome to contain cycles (for these are incorporated into the fragments). Also fragment-based methods have the potential benefit of easier synthesis. While the current version of the Molecule Evoluator uses both atoms and fragments to construct molecules, its mutations are atom-based. We believe that atom-based evolution is superior to fragment-based evolution for adapting the molecular structure. The main disadvantage of using fragments instead of atoms is that most mutations in fragment-based evolution are “macromutations” that change the molecule into something completely different, with a vastly different fitness value. In most cases, it is not clear whether fragment-based evolution is an improvement over random search, unless the fitness function is fragment-based. However, this is certainly not the case in drug design where biological activity is subtly dependent on the molecular structure. We have observed that making an atom-based algorithm interactive, as in the Molecule Evoluator, partially compensates for the disadvantage that molecules generated on the basis of atoms are generally more difficult to synthesize. This is because the medicinal chemist has the option to immediately discard or modify structures at will.

Using the user as a fitness function

The most important problem of the *de novo* design programs which have been described in literature is the difficulty of creating a fitness function that is relevant to drug design. In this work, we propose to use an evolutionary algorithm not as a black box that will give the user the right answer when given the right question, but as a means of supporting the creativity and imagination of the user by interactive evolution, thereby automatically incorporating the user's explicit as well as implicit (subconscious) knowledge of the problem domain.

Using user feedback as fitness function has several advantages and disadvantages, and some consequences that require special adaptations and modifications of the software.

One disadvantage of user interaction is that the population must be small. It is unlikely that any chemist would want to see 50 to 100 molecules before pressing “Go” again. However, small population sizes (12 as a default in the current version of the program) may lead to premature convergence, i.e. tempt the user to stop (too) early. Another hurdle relates to programming the software: the more the user can interact with the program, the more is required from the user interface. In this project, more time was spent on constructing the user interface than on creating and fine-tuning the evolutionary algorithm. Modifications of the evolutionary algorithm should in many cases be reflected by changes in the user interface, and this makes programming and testing new ideas more time-consuming than in a non-interactive system. Software validation is also more difficult – one cannot well run hundreds of tests automatically to objectively verify whether the algorithm outperforms other algorithms. A user is not an objective function that can be easily shared with others. Finally, the user may see the computer program as a competitor, not as a tool, and might not like to work with it since “someone else” has thought of the molecule. While the computer has so little knowledge that one should say that the user has invented the molecule (and recognized it as something interesting), the user may perceive his or her position differently.

There are however also many advantages to user interaction. One attractive advantage is that the feedback from the user can produce molecules which can be synthesized more easily in the laboratory than is possible with computer-generated, random molecules. The difficulty of synthesis would also be automatically adapted to the user’s level of knowledge and experience.

A second advantage is that the program can use all kinds of rules and problem domain knowledge that the user has. The alternatives, expert systems and flexible input, have distinct disadvantages in this case. Creating an expert-system is time-consuming and must be done anew for each optimization project. Flexible input would require the domain expert, namely the chemist, to learn a complicated language or user interface which would definitely diminish accessibility of the software and thereby its use greatly. The program can even benefit from the user’s subconscious rules, which cannot be programmed since they are unknown and may be very difficult to derive. Furthermore, as the user's problem knowledge grows, this knowledge is automatically updated and applied to the process without time-consuming intervention by programmers. In experimental sciences, seldom all required knowledge is known beforehand, and allowing experiments with the computer can also lead to finding new rules and discarding obsolete ones.

A third advantage is that the software can stimulate computer use by medicinal chemists. Far too often, compounds suggested by the "computational department" are rejected by medicinal chemists for reasons of synthesis, and collaboration between the departments is hampered by busy schedules and the necessity to have meetings for feedback. This makes collaboration slow and difficult, and probably results in chemists mainly designing their own compounds without the help the computer could give. We believe that creating a program for the problem domain experts instead of for computer experts can lead to better use of the help that the computer could give in the drug design process, and perhaps even to increased understanding and a more fruitful collaboration between medicinal chemists and computer scientists.

Finally, a program such as the Molecule Evoluator may make a chemist more conscious of his/her own design process, i.e. which rules he or she follows. Consciousness of the rules and methods can lead people to experiment with them and occasionally break them for enhanced creativity.

Adding extra user control to the evolution

We found that when we added interactivity to the evolutionary algorithm, it was not enough to restrict the user's influence to selection only. The users were generally quite "impatient" and wanted more control to accelerate or even directly manipulate the evolution, so we added features to enable this. First, we incorporated edit functions to directly modify the molecular structure. Second, we added an option for selecting a part of the molecule to remain constant. Third, we allowed the settings (which mutations are allowed, what range a property may have) to change interactively. We think that these options will make the Molecule Evoluator more attractive for drug design since they give the user more control over the evolution. We must however beware of the complication that having a feature is not enough if the user does not know that the feature is there or how to use it. Good user interface design is necessary for users to learn to use the multiple filters without having to read the manual. The second danger is perhaps more serious: by discarding "bad molecules" one may at the same time eliminate paths to escape from local optima. Secondly, if all molecules shown are good according to a specific user's criteria, it may be exactly what the user had designed him/herself anyway, thus eliminating the added value of the program. However, lack of control may frustrate and bad structures may irritate the user, so we aimed for a middle road between control and creativity.

Conclusions and future perspectives

In this report we have described the "Molecule Evoluator". It is a software program based on evolutionary algorithms and has been created to aid chemists in designing new drug-like molecules. With this program all relevant chemical mutations are possible. The most distinguishing feature of the Molecule Evoluator relative to other *de novo* design programs is that it has the user as fitness function. In this way it can combine the domain knowledge of the chemist with the memory and processing speed of the computer. We therefore added a graphical user interface for the evolution and extended the program with options for directly editing the molecule, marking part of a molecule as conserved, and calculating relevant physicochemical parameters.

Considering the algorithms used and the feedback from users so far, there are several directions open for future investigation. First, many molecules generated by the program seem difficult to synthesize. Perhaps encoding explicit chemical knowledge in the program or using chemical databases could help improve this. A second direction would be to create a command-line version which links to other software such as docking programs, since the “high-resolution” optimization resulting from our atom-based model might be very useful for optimizing lead compounds. Third, more selection criteria could be added such as additional physicochemical properties or an input method for QSAR-formulas.

Acknowledgements
We thank Hans Brussee, Lisa Chang and Reynier Tromp of the division of Medicinal Chemistry at Leiden University for their feedback during the first stages of development of the Molecule Evoluator, and Siegfried Nijssen for his helpful discussions and providing the first "charf" class implementation to conserve atoms/bonds. Biopartner First Stage Grant and Gruenenthal GmbH are thanked for financial support.

References

[1] Bohacek, R.S.; McMartin, C; Guida, W.C. The art and practice of structure-based drug design: a molecular modeling perspective. *Med. Res. Rev.* **1996**, *16*, 3-50.

[2] As taken from www.cas.org/scifinder/ataglance.html, as accessed on August 18, 2005.

[3] Gillet, V.J. In Evolutionary algorithms in molecular design; Clark, D.E., Ed.; Wiley-VCH: Weinheim, Germany, 2000; Chapter 4, pp 49-69.

[4] Douguet, D.; Thoreau, E.; Grassy, G. A genetic algorithm for the automated generation of small organic molecules: Drug design using an evolutionary algorithm. *J. Comput.-Aided Mol. Des.* **2000**, *14*, 449-466.

[5] Globus, A.; Lawton, J.; Wipke, T. Automated molecular design using evolutionary techniques. *Nanotechnology* **1999**, *10*, 290-299.

[6] Glen, R.C., and Payne, A.W.R. A genetic algorithm for the automated generation of molecules within constraints. *J. Comput.-Aided Mol. Des.* **1995**, *9*, 181-202.

[7] Schneider, G.; Lee, M-L.; Stahl, M.; Schneider, P. De novo design of molecular architectures by evolutionary assembly of drug-derived building blocks. *J. Comput.-Aided Mol. Des.* **2000**, *14*, 487-494.

[8] Pegg, S.C.-H.; Haresco, J.J.; Kuntz, I.D. A genetic algorithm for structure-based de novo design. *J. Comput.-Aided Mol. Des.* **2001**, 15, 911-933.

[9] Vinkers, M.H.; De Jonge, M.R.; Daeyaert, F.F.D.; Heeres, J.; Koymans, L.M.H.; Van Lenthe, J.H.; Lewi, P.J.; Timmerman, H.; Van Aken, K.; Janssen, P.A.J. SYNOPSIS: SYNthesize and Optimize System in Silico. *J. Med. Chem.* **2003**, 46, 2765-2773.

[10] MDL-file format http://www.mdli.com/downloads/public/ctfile/ctfile.jsp as accessed on August 18, 2005. See also: Dalby, A.; Nourse, J.G.; Hounshell, W.D.; Gushurst, A.K.I.; Grier, D.L.; Leland, B.A.; Laufer, J. Description of several chemical structure file formats used by computer programs developed at Molecular Design Limited. *J. Chem. Inf. Comput. Sci.* **1992**, *32*, 244-255.

[11] Weininger D. SMILES: a chemical language and information system. 1. Introduction to methodology and encoding rules. *J. Chem. Inf. Comput. Sci.* **1988**, *28*, 31-36.

[12] Banzhaf, W.; Nordin P.; Keller, R.E.; and Francone, F.D. *Genetic Programming - An Introduction.* Morgan-Kaufmann, San Francisco CA, 1998.

[13] Nachbar, R.B. Molecular Evolution: A Hierarchical Representation for Chemical Topology and Its Automated Manipulation. In *Genetic Programming 1998: Proceedings of the Third Annual Conferenc*e, (University of Wisconsin, Madison, Wisconsin, July 22-25, 1998). Morgan Kaufmann, San Francisco, CA, 1998, 246-253.

[14] Kamphausen, S.; Höltge, N.; Wirsching, F.; Morys-Wortmann, C.; Riester, D.; Goetz, R.; Thürk, M; Schwienhorst, A. Genetic algorithm for the design of molecules with desired properties. *J. Comput.-Aided Mol. Des.* 2002, *16*, 551-567.

[15] Sheridan, R.P.; Feuston, B.P.; Maiorov, V.N.; Kearsley, S.K. Similarity to molecules in the training set is a good discriminator for prediction accuracy in QSAR. *J. Chem. Inf. Comput. Sci.* **2004**, *44*, 1912-1928.

[16] Banzhaf, W. In *Handbook of Evolutionary Computation;* Bäck, T.; Fogel D.B.; Michalewicz, Z., Eds.; Oxford University Press, New York, and Institute of Physics Publishing, Bristol, 1997.

[17] Bentley, P.J. *Evolutionary Design by Computers;* Morgan Kaufmann Publishers, San Francisco, 1999.

[18] Takagi, H. *Interactive Evolutionary Computing: Fusion of the capacities of EC optimization and human evaluation*. Proceedings of the IEEE **2001**, *89*, 1275-1296.

[19] Lipinski, C.A., Lombardo, F., Dominy, B.W., and Feeney, P.J. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Adv. Drug Delivery Rev.* **1997**, *23,* 3-25.

[20] See: www.iupac.org/goldbook/B00732.pdf, as accessed on November 21, 2005.

[21] Kim, C.U.; Lew, W.; Williams, M.A.; Liu, H.; Zhang, L.; Swaminathan, S.; Bischofberger, N.; Chen, M.S.; Mendel, D.B.; Tai, C.Y.; Laver, W.G.; Stevens, R.C.; Influenza neuraminidase inhibitors possessing a novel hydrophobic interaction in the enzyme active site: design, synthesis, and structural analysis of carbocyclic sialic acid analogues with potent anti-influenza activity. *J. Am. Chem. Soc.* **1997**,*119*, 681-690.

[22] http://www.beilstein-institut.de/englisch/1024/chemie/index.php3 , as accessed on August 18, 2005.

[23] Schneider, G.; Fechner, U. Computer-based *de novo* design of drug-like molecules. *Nature Rev. Drug Disc*. **2005**, *4*, 649-663.

[24] Brown, N.; McKay, B.; Gilardoni, F.; Gasteiger, J. A graph-based genetic algorithm and its application to the multiobjective evolution of median molecules. *J. Chem. Inf. Comput. Sci*. **2004**, *44*, 1079-1087.