Evolutionary Algorithms for Automated Drug Design Towards Target Molecule Properties

[Real-World Applications]

Johannes W. Kruisselbrink
LIACS, Leiden University
Niels Bohrweg 1, Leiden
The Netherlands
jkruisse@liacs.nl

Thomas Bäck*
LIACS, Leiden University
Niels Bohrweg 1, Leiden
The Netherlands
baeck@liacs.nl

Ad P. IJzerman
Leiden/Amsterdam Center for
Drug Research, Leiden University
Einsteinweg 55, Leiden
The Netherlands
ijzerman@lacdr.leidenuniv.nl

Eelke van der Horst
Leiden/Amsterdam Center for
Drug Research, Leiden University
Einsteinweg 55, Leiden
The Netherlands
e.van.der.horst@chem.leidenuniv.nl

ABSTRACT

This paper presents an evolutionary algorithm for the automated design of molecules that could be used as drugs. It is designed to provide the medicinal chemist with a number of candidate molecules that comply to pre-defined properties. These candidate molecules can be promising for further evaluation.

The proposed algorithm is implemented as an extension to the so-called Molecule Evoluator [3] which implements an interactive evolutionary algorithm. The Molecule Evoluator is extended with an automated evolutionary algorithm that implements a variable sized population and bases its search on target-bounds that are set for a number of molecule properties. Moreover, the algorithm uses a selection procedure based on the notion of Pareto domination.

The results show that it is indeed possible to apply the concept of evolutionary computation on automated molecule design using target-bounds for molecule properties as optimization goals. For practical usage, the presented algorithm could serve as a starting point, but should be further improved with respect to diversity within the generated set of molecules.

Categories and Subject Descriptors

J.2 [**Physical Sciences and Engineering**]: Chemistry

*Nutech Solutions, Inc., Charlotte, NC, USA

General Terms

Algorithms, Design, Experimentation

Keywords

Evolutionary algorithms, molecules, drug design

1. INTRODUCTION

De novo drug design and development is the process of finding and creating molecules which have a specific activity on a biological organism.

Generally, drugs are formed around small key molecules that can bind to a target, which, in most cases, is a protein such as a receptor, enzyme, transport protein, or antibody. These proteins can be found on the cell surface, in the cell, or in plasma (except for receptors). By exhibiting or inhibiting the activity of these critical components, the drug molecules can change the behavior of the entire cell. Target cells can be either cells of the disease causing organisms or of the diseased patients themselves.

To aid the development of new drugs, computational methods can be used to generate molecules that could be used as lead compounds for possible new drugs. By automatically generating libraries of molecules that comply to certain predefined physicochemical properties, the medicinal chemist can be provided with sets of promising molecules that are good candidates for further investigation. The particular use of automated methods is that they can speed up the search and cover a larger part of the search space because they are fast and unbiased.

The so-called Molecule Evoluator [3] is such a method. However, until now working in a semi-automated fashion. It implements an interactive evolutionary algorithm which can be used by the medicinal chemist to 'evolve' populations of molecules. By letting the user guide the evolution process, the Molecule Evoluator aims to provide inspiration for possibly new molecules. The Molecule Evoluator can be used for both lead generation (generating new molecules with certain desired properties from scratch) and lead optimization (start from one or more lead molecules and optimize the molecule properties or generate derivatives).

The major strength of the Molecule Evoluator is the use of the expert knowledge of the chemist in the selection procedure of the evolutionary algorithm. The expert user is responsible for selecting the molecules that will be used as parents for the creation of each next generation in the evolutionary process.

In this paper we will propose an evolutionary algorithm that will be an extension to the Molecule Evoluator. In addition to the interactive evolutionary algorithm used by the Molecule Evoluator, we have developed an automated evolutionary algorithm intended to be used for the generation of a diverse set of molecules with very specific physicochemical properties.

Based on a set of strict targets that can be specified by the medicinal chemist, the algorithm will search for molecules that comply to these targets and therefore could be promising candidates for further investigation. We will focus on a method that can incorporate multiple objectives and that can generate a diverse set of molecules as output to offer the medicinal chemist a broad set of promising molecules.

We will start in section 2 with a brief outline of the scope of this paper and the motivation of extending the Molecule Evoluator with an automated evolutionary algorithm. Section 3 will give an overview of the interactive evolutionary algorithm that is currently implemented by the Molecule Evoluator. In section 4 we will describe the automated extension that we have developed for fully automated molecule generation. Section 5 shows the setup and outcomes of the experimental runs. Section 6 will close with conclusions and future recommendations.

2. AUTOMATED EA FOR DRUG DESIGN

The size of the search space together with its complexity make it very hard, if not impossible, to find a classical computational approach to search for molecules. Evolutionary algorithms are specially suited for such large search spaces, and can also deal with the complexity of the problem, as they do not (strictly) require any knowledge about the search space. Hence, evolutionary algorithms are a good choice to use for searching for molecules.

A major problem is finding a good fitness function. Usually there is a clear view on the desired behavior which the molecules should exhibit, but actually capturing this desired behavior in a mathematical scoring function is very difficult. Also, the fitness functions that are used in automated methods are usually poor approximations of the real structure-activity relationships.

The power of the Molecule Evoluator lies in the fact that it uses the expert user as a fitness function. Experts are very well capable of determining whether molecules are ”good” in the scope of what they need and can make decisions much faster than is possible with doing real-life experiments. By letting the expert guide the search process, his/her knowledge and intuition can be used to get quickly to ”good” parts of the search space.

However, the use of expert knowledge also has its downsides. First, using expert knowledge to guide the search process brings along the danger of neglecting parts of the search space that might prove to be very promising. Especially such parts of the search space would be very interesting, but are now missed. Also, interactive methods are limited when it comes to the number of molecules that can be considered in the search.

With the downsides of interactive evolutionary search in mind, we present another attempt to come up with an automated method for molecular design.

2.1 Multi-objective search

We focus our method on the cases where it is possible for the medicinal chemist to determine a number of properties (which can be obtained by calculation, simulation or in other computational ways) to which the desired molecule(s) need to comply. In this case, we use these desired properties as targets for an automated search process by trying to minimize the difference between the property values of the candidate solutions and the desired targets. With this, we get a multi-objective optimization problem.

Taking the multiple objective point of view [1], we optimize over a number of numerically expressable molecule properties with target intervals set for each property. Each property i can be expressed numerically by a function $g_i(x)$ and x being any instance from the set of all possible molecules. With that, for each molecule property i with a target interval $[a_i, b_i]$, we define the objective function f_i as:

$$f_i(x) = \begin{cases} |g_i(x) - a_i| & \text{if } g_i(x) < a_i \\ |g_i(x) - b_i| & \text{if } g_i(x) > b_i \\ 0 & \textit{otherwise} \end{cases} \tag{1}$$

Of course, for future research, we could try to find more complex functions that could improve the algorithm. In any case, minimizing over all objective functions f_i is then the goal of the optimization process.

2.2 Diverse set of molecules

When making use of calculation or simulation based fitness functions, one should have to be very aware of the fact that, as mentioned, these are often very noisy and the application of these scoring functions often results in molecules that are difficult to synthesize. Also, molecular structures are very complex, and in the end, the process of drug design and development will probably always stay a job for the medicinal chemist. The best that automated methods can do is to provide the medicinal chemist with possible candidates.

The awereness of the limitations of automated methods changes the aim of the search algorithm to a great extent. Instead of trying to offer only one solution, the search algorithm should try to provide the medicinal chemist with a broad set (preferably as diverse as possible) of molecules.

Hence, the focus lies on the generation of a diverse set of good candidate molecules. Especially incorporating this into an interactive tool such as the Molecule Evoluator is very useful, because this allows the medicinal chemist to use both methods interchangeably in a combined interactive and automated approach.

2.3 Motivation

The intent of this automated method is to be an extension of the Molecule Evoluator. We do not propose to replace the interactive evolutionary algorithm of the Molecule

Evoluator, but rather to offer it as another option. The brief comparison shown in table 1 shows that both methods have their advantages, so why not use both?

By extending the Molecule Evoluator also with an automated search algorithm, the user is provided with two distinct ways of searching for new lead compounds. As both methods have their advantages, the user is now allowed to choose which of the two methods is most suitable for the problem at hand.

Expert interaction	**Automated**
Expert is able to filter out obviously bad solutions.	Automated selection have difficulties with this.
Expert is able to use heuristics to guide the search.	Automated selection is unbiased. Less obvious parts of the search space will also be considered.
Expert is limited when it comes to the number of evaluations that can be performed.	Automated procedures can do many evaluations and thus consider much more solutions.
Expert is able to the compare different solutions and assigning a fitness order.	Constructing a good fitness measure for automated purposes is very difficult.

Table 1: Expert interaction versus automated methods

3. MOLECULE EVOLUATOR DETAILS

In section we will give a brief overview of the evolutionary algorithm implemented by the Molecule Evoluator.

The Molecule Evoluator uses the knowledge of the expert in a direct way by letting the user guide the evolution process towards desired molecules. Moreover, the involvement of the user also allows the user to test every inspiration perceived during the evolution process directly which makes the Molecule Evoluator especially powerful as an interactive design tool.

3.1 Molecule representation

The representation used by the evolutionary algorithm of the Molecule Evoluator is the so-called TreeSmiles representation [3]. The TreeSmiles notation is an extended version of the SMILES notation [7] commonly used by chemists.

With SMILES and TreeSmiles, the molecules are represented by a string (similar to the Lisp-notation used by genetic programming [2]) which is human-readable and can easily be transformed to a 2D-structure by a chemist. The TreeSmiles extension to SMILES pertains to being more strict and more suited for easy implementation of automated operations (i.e. the genetic operators).

3.2 Population initialization

The Molecule Evoluator also allows the user to provide the initial population of molecules from which the evolution can start. If no initial population is provided by the user, an initial population is generated at random from scratch.

For the creation of an initial population from scratch, two different molecule generation methods are implemented. The first method to generate molecules takes for each generation a simple Methyl molecule and applies a number of mutations on it. The second method is a bit more sophisticated and implements a fragment-based generation of molecular. The balance between the number of molecules generated by both methods can be controlled by the user.

3.3 Genetic operators

The Molecule Evoluator provides one crossover operator and eleven mutation operators for the evolutionary algorithm. To allow for a certain measure of control to the structures of the mutated molecules, the user is allowed to decide which of the eleven mutation operators are actually used (i.e. each mutation operator can be turned on and off). Also, the user is allowed to decide which genetic operator is the primary operator as the Molecule Evoluator provides the option to adapt the balance between crossover and mutation.

Table 2 shows the eleven mutation operators together with their effect. As can be seen, except for the "mutate atom" mutation, for every mutation there is an inverse mutation operator.

The implementation of the crossover operator is inspired by the subtree crossover as used in genetic programming. However, as molecules are graphs and not trees, the recombination operator is restricted to be only applied to subtrees which are connected at exactly one point to the base molecule.

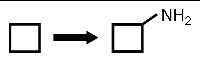
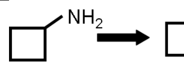
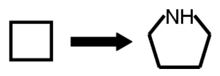
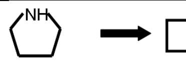
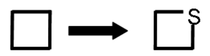
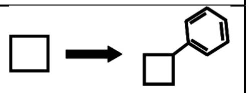
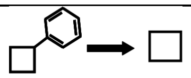
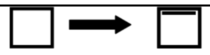
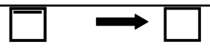
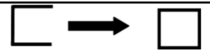
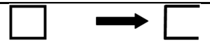
Mutation name	Effect
Add atom	
Remove atom	
Insert atom	
Uninsert atom	
Mutate atom	
Add group	
Remove group	
Increase bond order	
Decrease bond order	
Make ring	
Break ring	

Table 2: Mutation operators of the Molecule Evoluator

3.4 Evaluation and selection

The Molecule Evoluator uses the expert user as a fitness function. The user decides at each iteration which molecules can be used as parents for the next generation. Hence, the user is at the same time the fitness function and the selection function, and the performance of the evolutionary algorithm depends very much on the intuition, knowledge and luck of the user.

To assist the users in their choice, the Molecule Evoluator also supplies several chemical descriptors for each molecule. These descriptors are especially useful for the user to estimate drug-likeness (Lipinski's rule of five makes use of these properties to estimate drug-likeness [4]) or the way that the drug molecule is to be taken in and can be used in addition to the visual representation to estimate how good each molecule is. The descriptors are:

- The number of hydrogen donors (HD)
- The number of hydrogen acceptors (HA)
- The molecular weight (MW)
- The Logarithmic value of the octanol/water partition coefficient (logP)
- The Logarithmic value of the aqueous solubility in mol/l at $25°C$ (logS)
- The polar surface area (PSA)
- The number of rotatable bonds (RB)
- The number of aromatic systems (AR)
- The number of aromatic substituents (AS)

In order to allow the user to force the evolution towards molecules with certain descriptor values, the Molecule Evoluator provides a filter mechanism (chemical filters). When one ore more filters are set, the Molecule Evoluator only returns offspring that have their property values within these filter bounds.

In addition to the chemical filters, the Molecule Evoluator also allows the user to set constraints and enforce the molecules to have certain structures (called physical filters). If these filters are set, then the Molecule Evoluator will reject molecules that do not comply to these constraints. These constraints are listed below:

- Bredtś rule: no double bond with one end at a bridgehead of a bridged ring system.
- Acetals: allow molecules to contain a functional group of a Carbon bonded to two ï£¡OR groups.
- CH2-Imines: allow molecules to contain Imine groups (R-N=CR2) or CH2-Imine groups (R-N=CH2).
- Ortho-, meta-, or paracyclophane: ortho / meta / paracyclophanes are benzene rings that have a short bridge forming a second ring over, for example, the meta-atoms.
- Common Ring System: Filters out all ring systems that do not occur in the NCI. By checking the "Also Consider Atoms" option, the comparison will also consider the atoms in the ring besides comparing the bonds.

4. AUTOMATED EVOLUTION

The procedure `evoluate(`$P'(t)$`, filters, constraints,` λ`)`, which is outlined in algorithm 1, shows the main evolution loop of the automated evolutionary algorithm. This automated evolution loop is motivated by the base algorithm VV of Rudolph and Agapie [6].

The evolution loop starts with an initial population `P` selected by the user or generated randomly if the user did not select any initial population. Also, the user is expected to set the filter bounds, the constraints and the number of desired offspring (λ). Then, the evolution loops a number of times until there are λ offspring that comply to the user-set filter-bounds and constraints.

For each iteration, k offspring are generated and added to the population. Note that the algorithm distinguishes between constraints and filters. Constraints are hard constraints which every offspring needs to satisfy at all times. The filters will be used for the determination of the fitness of each molecule. A new offspring o is accepted if it passes the `Accept(o)` test. The function `Accept(o)` accepts molecules that comply to the constraints.

After the generation of the offspring, the property values of the offspring are evaluated and compared with the user-set filter values. Based on this evaluation, the parents for the next generation are selected by the multi-objective selection procedure `select(`$P(t+1)$`)` that will be described in section 4.2.

Algorithm 1 Automated evolution cycle

```
1: procedure EVOLUATE($P$, filters, constraints, $\lambda$)
2:     $t := 0$;
3:     $P(0) := P$;
4:     while not terminate do
5:         $n := 0$;
6:         while $n < k$ do
7:             $o :=$ generate offspring from $P(t)$;
8:             if Accept($o$) AND $o \notin P(t+1)$ then
9:                 $P(t+1) := P(t+1) \bigcup \{o\}$;
10:                $n := n + 1$
11:            end if
12:        end while
13:        evaluate($P(t+1)$);
14:        $P(t+1) :=$ select($P(t+1)$);
15:        $t := t + 1$;
16:    end while
17:    return $P(t)$
18: end procedure
```

4.1 Genetic operators

For the creation of the offspring, the same genetic operators were chosen as for the Molecule Evoluator. As this set of genetic operators allows the algorithm to cover the whole search space, this set of genetic operators seems to be very fit for our purpose. Especially the broad set of mutation operators is very helpful in this case, as we want to preserve diversity in the population.

4.2 Evaluation and selection

The selection procedure `select(`$P(t+1)$`)` outlined in algorithm 2 bases its determination of the fitness on the difference between the molecules and the filter bounds. For each

Algorithm 2 Automated multi-objective selection

```
 1: procedure SELECT($P$, filters, constraints, $\mu$)
 2:     $Q := \emptyset$;
 3:     $l := 0$;
 4:     while $k < \mu$ do
 5:         for each $p \in P$ do
 6:             $d_p$ := number of solutions that dominate $p$;
 7:             if $d_p == l$ then
 8:                 $Q := Q \bigcup \{p\}$;
 9:                 $k := k + 1$;
10:             end if
11:         end for
12:         $l := l + 1$
13:     end while
14:     return($Q$);
15: end procedure
```

property, we use the objective of minimizing the function as given by equation 1, and use the notion of Pareto domination to assign an ordering.

For completeness; a solution x_1 is said to dominate x_2 *iff*:

$$\forall i \in 1, \ldots, n : f_i(x_1) \leq f_i(x_2)$$
$$\wedge\ \exists i \in 1, \ldots, n : f_i(x_1) < f_i(x_2)$$

The procedure `select(`$P(t{+}1)$`)` selects at least μ individuals from the population P based on the number of individuals that dominate it (i.e. good individuals are dominated by few/zero others, so the less individuals dominate it, the better an individual is). By keeping at least a population size of μ, the selection procedure should prevent the population from collapsing to one or a few individuals (which would disrupt diversity).

Besides that using the notion of Pareto dominance is probably fairer when it comes to the determination of a performance order, it also is in this case a way to maintain a certain level of diversity. Note that with Pareto domination, two distinct solutions have a higher chance of being incomparable than similar solutions have. With this, it is therefore more likely for similar solutions that they will be dominated (by other similar solutions) than distinct solutions. And thus we hope that this selection method will select a more diverse set than using a weighted sum as fitness function. For further research, it would be insightful to test if this effect indeed occurs.

4.3 Variable sized populations

What makes this implementation different from other evolutionary algorithms is the fact that this allows the population to grow without any limits. No matter how many non-dominated solutions there are, the algorithm used here will not choose between any non-dominated solutions, but keep them all.

As it is up to the user to select the filters that should be taken into account for the fitness, the search can be an optimization of only one objective function, but it could also be more. With such a varying number of objectives, it is impossible to determine a good population size yourself. By letting the algorithm grow the population size, it can grow to the number of solutions needed for covering the Pareto front. The advantage is therefore that the algorithm can keep a nice broad Pareto front and thus there is a big chance that the algorithm ends up with a diverse set of molecules with the desired properties.

One might argue that this method will eventually result in huge populations, but we know from experience that this is not the case. In the first few iterations, the population will indeed grow rapidly, but after a while, offspring will be created that are fit on a multiple of the objectives. It is the creation of this kind of offspring that reduces the number of solutions in the population, as they are likely to dominate multiple solutions of the population.

5. EXPERIMENTAL RESULTS

For the experiments, the property values of already existing and well known drugs were used to construct the targets for the test-runs. The goal of the test-runs was of course not only to find the original molecules, but also to find alternative molecules having about the same property values as the drug molecules. Table 3 shows the molecules used for testing.

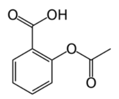
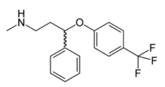
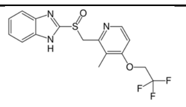
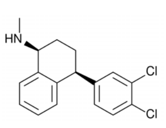
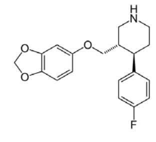
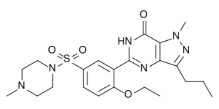
 1. Aspirin (Acetosal)	 2. Prozac (Fluoxetine)	 3. Prevacid (Lansoprazole)
 4. Zoloft (Sertraline)	 5. Paxil (Paroxetine)	 6. Viagra (Sildenafil)

Table 3: Set of test molecules

5.1 Setup

The experiments were run on a Pentium 1.73GHz computer. For each of the test molecules, the algorithm was tested with 10 runs. Each of these runs, the targeted number of molecules was set to 5, the minimal population size was set to 40, and in each generation 10 offspring were produced. The evolution process stopped when the desired number of molecules was found, or at generation 1000.

The relatively small number of molecules in the output set was chosen to save time. For the creation of larger output sets, the algorithm would need a larger minimal population size and more generations which would take longer. With the capacity at hand, taking 5 would give a good view of the performance in relatively little time.

For the experimental runs, tables 4 to 9 show the filter bounds that were used. The filter bounds were set up not to be too strict to allow also for molecules other than the target molecules to be found. Also, not all filter bounds were set in a similar way for the different molecules, e.g. the bounds of the MW-filter for the Aspirin test runs were set much stricter than for the test runs of the other molecules.

	Aspirin	Lower bound	Upper Bound
HD	1	1	1
HA	4	4	4
MW	180.16	175.16	185.16
logP	1.31	0.81	1.81
logS	-2.14	-2.64	-1.64
PSA	63.60	58.60	68.60
RB	3	3	3
AR	1	1	1
AS	2	2	2

Table 4: Filter settings for Aspirin-like molecules

	Prozac	Lower bound	Upper Bound
HD	1	1	1
HA	2	2	2
MW	309.33	304.33	314.33
logP	4.44	3.94	4.94
logS	-4.43	-4.93	-3.93
PSA	21.26	16.26	26.26
RB	6	6	6
AR	2	2	2
AS	2	2	2

Table 5: Filter settings for Prozac-like molecules

	Prevacid	Lower bound	Upper Bound
HD	1	1	1
HA	5	5	5
MW	383.39	378.39	388.39
logP	3.61	3.11	4.11
logS	-4.03	-4.53	-3.53
PSA	76.25	71.25	81.25
RB	6	6	6
AR	2	2	2
AS	4	4	4

Table 6: Filter settings for Prevacid-like molecules

	Zoloft	Lower bound	Upper Bound
HD	1	1	1
HA	1	1	1
MW	306.23	301.23	311.23
logP	5.18	4.68	5.68
logS	-5.83	-6.33	-5.33
PSA	12.03	11.03	13.03
RB	2	2	2
AR	2	2	2
AS	5	5	5

Table 7: Filter settings for Zoloft-like molecules

5.2 Results

Table 10 shows a summary of the outcomes of the test runs. From this table it can be seen that only one test run managed to find the target molecule (Aspirin). Although this is somewhat disappointing, on a positive note we can say that of the in total 60 test runs in total, only 7 runs failed to find 5 molecules complying to the target filter bounds. Moreover, of those failed runs, most did find at least one solution. Thus, the algorithm is indeed capable of finding molecules that have property values similar to the target molecules.

Also from the time-perspective, the algorithm does not perform bad at all. Ranging from an average running time of a little more than 7 minutes (Aspirin) to an hour (Viagra) is not a bad point to start with. Especially when considering the enormous size of the search space. Also from a practical point of view, such a running time is acceptable. However, one should keep in mind that these runs only returned 5 molecules. Further work to speed up the algorithm up some more would still be advisable.

	Paxil	Lower bound	Upper Bound
HD	1	1	1
HA	4	4	4
MW	329.37	324.37	334.37
logP	3.33	2.83	4.83
logS	-3.34	-3.83	-4.84
PSA	39.72	34.72	44.72
RB	4	4	4
AR	2	2	2
AS	5	5	5

Table 8: Filter settings for Paxil-like molecules

	Viagra	Lower bound	Upper Bound
HD	1	1	1
HA	10	10	10
MW	460.55	450.55	470.55
logP	1.98	1.48	2.48
logS	-3.58	-4.08	-3.08
PSA	126.12	121.12	131.26
RB	6	6	6
AR	2	2	2
AS	7	7	7

Table 9: Filter settings for Viagra-like molecules

There is a clear relation between the complexity/size of the targeted molecule and the running time, maximum population size and the number of generations of the evolutionary algorithm. This is not really very surprising, as the search space (and with that the complexity) grows exponentially with the molecule size. The results of Viagra give however a nice indication of the upper bound of the running time of the algorithm as this molecule has a molecular weight very near to the maximum of 500.

Another interesting notion is that the dynamic population size method works and that the population size stays relatively small. Figure 1 and figure 2 show the population size development of a successful run and of a failed run (both are runs of the Aspirin test). In both cases, the population size does not grow beyond control. The big drop of the graph of figure 1 is the point where the desired number of solutions is found and the evolution cycle is terminated.

Lastly there is the population diversity and the molecules that were returned. Figure 3 to 8 show the outcomes of three of the test runs. For simple molecules like Aspirin, the algorithm is capable of finding good and diverse solutions. When more complex molecules (like Viagra) are used, this becomes harder. The algorithm tends very much to converge the population to a particular area in the search space, and diversity is lost. Also with more complex structures, the amount of unsynthesizable solutions increases.

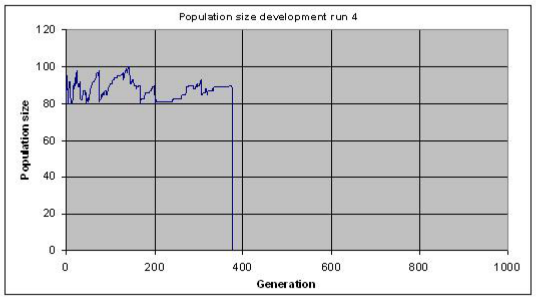


Figure 1: Population size development of a successful run.

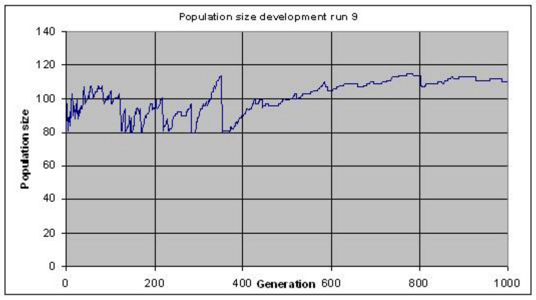


Figure 2: Population size development of an unsuccessful run.

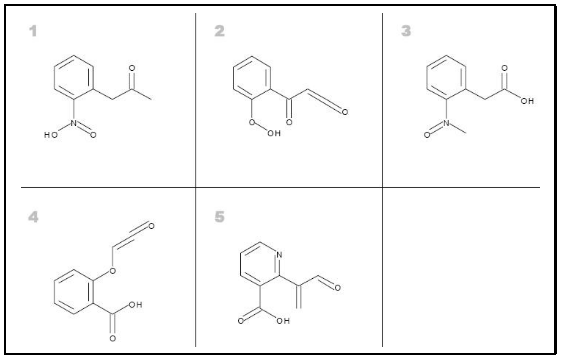


Figure 3: Molecules found in run 1 of the Aspirin tests.

6. CONCLUSIONS AND OUTLOOK

The results show that it is indeed possible to use evolutionary algorithms to find molecules with specific properties. Moreover, extending the Molecule Evoluator with this automated evolutionary algorithm makes the Molecule Evoluator applicable for a broad range of purposes. The chemist can exploit the advantages of both interactive evolution and automated evolution, and choose the method that is most suitable for the problem at hand.

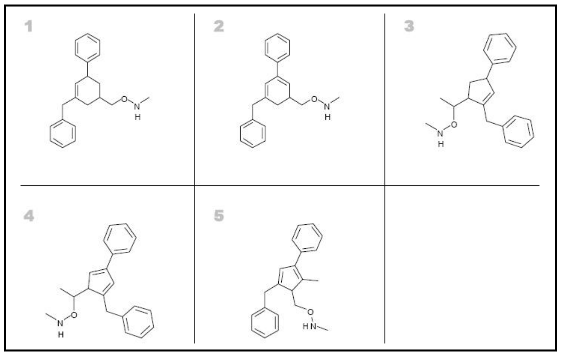


Figure 4: Molecules found in run 1 of the Prozac tests.

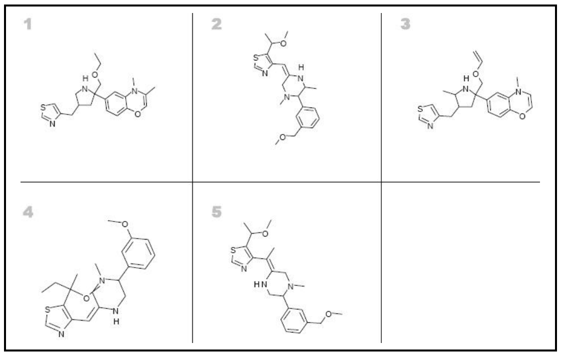


Figure 5: Molecules found in run 1 of the Prevacid tests.

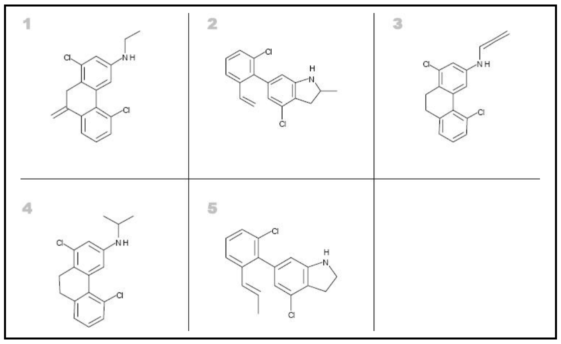


Figure 6: Molecules found in run 1 of the Zoloft tests.

6.1 Multi-objective approach

By taking the multi-objective approach, the algorithm is capable to also implement more or other fitness functions. It is therefore very flexible and especially from a practical point of view, this could prove to be very useful as it allows the algorithm also to be used by pharmaceutical companies that implement their own molecule property functions and simulation programs.

Basing the fitness determination on the notion of Pareto domination is also a first attempt to generate population diversity. By focusing on population diversity, the hope is that a broader part of the search space is covered, and it provides the medicinal chemist with more possible solutions in the ultimate output.

	Aspirin	Prozac	Prevacid	Zoloft	Paxil	Viagra
Failed runs	1	2	1	0	2	1
Runs found target	1	0	0	0	0	0
Average number of generations	331	362	366	315	428	574
Average time of the run (h:mm:ss)	0:07:30	0:30:17	0:25:20	0:14:06	0:28:58	1:09:24
Average maximum population size	108	167	207	145	243	463

Table 10: Results

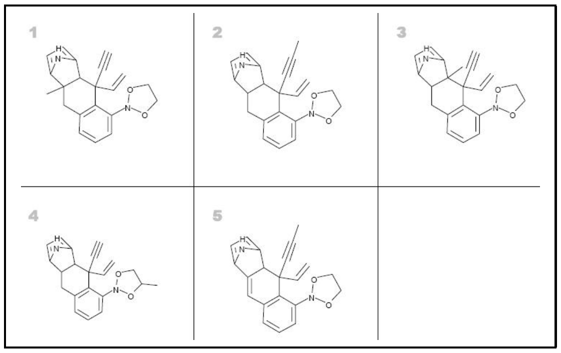


Figure 7: Molecules found in run 1 of the Paxil tests.

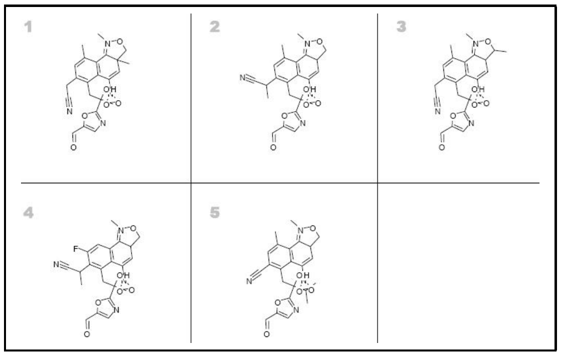


Figure 8: Molecules found in run 1 of the Viagra tests.

6.2 Molecule diversity

Although the proposed algorithm is very well capable of finding molecules that comply to pre-defined properties, the population diversity is still problematic. The population diversity decreases especially in the cases where more complex molecules are sought.

It seems inevitable to resort to niching techniques [5] to improve the population diversity. This will probably also have effects on the performance of the algorithm (both time and quality). We are planning to investigate niching as a further topic of research.

6.3 Variable sized populations

A matter that makes the proposed algorithm different from the common evolutionary algorithms is the fact that it implements a variable sized population. It is remarkable to see that the population size does not grow beyond control which is very useful as no choices have to be made between (from the point of view of Pareto domination) equal solutions in the selection procedure. The cause of this is probably that the fitness landscape is very rugged.

6.4 Genetic operators and fine-tuning

For further research, it would also be very worthwhile to further investigate the genetic operators, and their effect on the fitness landscape of different property functions. The algorithm presented here has simply adapted the mutation operators of the Molecule Evoluator, but although the genetic operators serve their purpose very well for the Molecule Evoluator, there might be more suitable genetic operators for an automated evolutionary algorithm. Testing various combinations of genetic operators could be very interesting.

It would also be interesting to see the effect of different minimal population sizes and different numbers of offspring created every iteration (respectively μ and `k` in algorithm 2). The numbers that were chosen here were picked with only a few tests and heuristical knowledge. A deeper investigation could prove very beneficial.

7. REFERENCES

[1] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms.* John Wiley & Sons, Inc., New York, NY, USA, 2001.

[2] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA, 1992.

[3] E.-W. Lameijer, J. Kok, T. Bäck, and A. IJzerman. The molecule evoluator: An interactive evolutionary algorithm for the design of drug-like molecules. *Journal of Chemical Information and Modeling*, 46(2):545 – 552, 2006.

[4] C. Lipinski, F. Lombardo, B. Dominy, and P. Feeney. Experimental and computational approaches to estimate solubility and permeability in drug discovery and developments settings. *Advanced Drug Delivery Reviews*, 46(1-3):3–26, 2001.

[5] S. W. Mahfoud. *Niching methods for genetic algorithms.* PhD thesis, University of Illinois at Urbana Champaign, Urbana, IL, USA, 1995.

[6] G. Rudolph and A. Agapie. Convergence properties of some multi-objective evolutionary algorithms. In *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*, pages 1010–1016, California, USA, 2000. IEEE Press.

[7] D. Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Science*, 28(1):31–36, 1988.