J. Mol. Model. **2000**, *6*, 498 – 516

Journal of Molecular Modeling

Full Paper

LigBuilder: A Multi-Purpose Program for Structure-Based Drug Design

Renxiao Wang, Ying Gao, and Luhua Lai

Institute of Physical Chemistry, Peking University, Beijing 100871, P.R.China. E-mail: lai@mdl.ipc.pku.edu.cn

Received: 6 December 1999/ Accepted: 28 February 2000/ Published: 16 August 2000

Abstract We have developed a new multi-purpose program, LigBuilder, for structure-based drug design. Within the structural constraints of the target protein, LigBuilder builds up ligands step by step using a library of organic fragments. Various operations, such as growing, linking, and mutation, have been implemented to manipulate molecular structures. The user can choose either growing or linking strategies for ligand construction and a genetic algorithm is adopted to control the whole construction process. Binding affinities of the ligands are estimated by an empirical scoring function and the bioavailabilities are evaluated by a set of chemical rules. Using thrombin and dihydrofolate reductase as examples, we have demonstrated that LigBuilder is able to generate chemical structures similar to the known ligands.

Keywords Structure-based drug design, Automatic ligand construction, Fragment-based approach, Genetic algorithm

Introduction

Since the 1980s, the process of drug discovery and design has been profoundly affected by the emergence of new methods and technologies. With the improvements in experimental techniques of X-ray crystallography and NMR, the amount of information concerning 3D structures of biomolecular targets has increased dramatically. At the time of writing this paper, the number of 3D structures in the Protein Data Bank (PDB) has exceeded 10,000.[1] This vast body of knowledge has led to considerable effort in exploiting structural information in order to design novel ligands that bind tightly and selectively to the target protein, a process which is usually referred to as structure-based drug design. A number of such approaches have already been reported.[2-9] Several compounds designed in this manner are now in clinical trials.

The success of structure-based drug design methodology has encouraged the development of various computational methods that can make use of structural information to suggest novel structures, which may either prove to be useful lead compounds or act as a stimulus to the creativity of designers. Ideally, these methods should be fast, objective and produce a set of diverse yet chemically reasonable structures. This field continues to receive intense interest and has been reviewed from time to time.[10-14]

Current methods for structure-based drug design can be divided roughly into two categories. The first category is about "finding" ligands for a given receptor, which is usually referred as database searching. In this case, a large number of molecules are screened to find those fitting the binding pocket of the receptor. Some researchers call this "virtual screening" in analogy to the bioassay screening procedure employed in the traditional drug discovery. The key advantage of database searching is that it saves synthetic

Correspondence to: L. Lai

effort to obtain new lead compounds. One of the earliest programs for performing 3D database searching is DOCK.[15-19] Another category of structure-based drug design methods is about "building" ligands, which is usually referred as *de novo* design. In this case, ligand molecules are built up within the constraints of the binding pocket by assembling small pieces in a stepwise manner. These pieces can be either atoms or fragments. The key advantage of such a method is that novel structures, not contained in any database, can be suggested. The earliest *de novo* design method began with GRID[20] and many groups have been actively involved in this field since then. Current popular *de novo* design programs include GROW,[21] LUDI,[22-25] LEAPFROG,[26] SPROUT,[27] PROLIGAND[28-33] etc. These techniques are raising much excitement to the drug design community.

In this paper, we will describe a new program, LigBuilder, which has been developed for structure-based drug design. Based on the structural constraints of the target protein, LigBuilder builds up ligands iteratively by using a library of organic fragments. The program provides growing and linking strategies to build up ligands and the whole construction process is controlled by a genetic algorithm. The protein-ligand binding affinity is evaluated by using an empirical scoring function instead of force field energies. Besides binding affinity, biological availability of the ligand is also taken into account by applying certain chemical rules. We have tested the program on two well-characterized enzymes, thrombin and dihydrofolate reductase. In both cases LigBuilder has reproduced the known binding mode found in the crystal structure. The potential application of LigBuilder to drug design and discovery is also discussed.

We will give a brief overview of LigBuilder first. A more detailed description of this program is divided into the following sections: *binding pocket analysis*, *building-up method*, *scoring method* and *genetic algorithm procedure*.

Overview of the program

LigBuilder is designed for structure-based drug design approaches. Therefore, the basic input for the program is the 3D structure of the target protein and the output will be a number of ligands that fit into the binding pocket sterically and chemically. It is composed of three major modules, i.e. POCKET, GROW, and LINK, each of which has a unique function in the designing procedure. The relationships between three modules are illustrated in Figure 1. All the source codes are written in C++ language..

POCKET is the first module to be run. It reads the target protein, analyzes the binding pocket, and then prepares the necessary data for GROW and LINK. POCKET also derives key interaction sites within the binding pocket. Such information could be used as the pharmacophore query for 3D database searching.

LigBuilder provides two strategies to build up ligand molecules: one is *growing strategy* while the other is *linking strategy*. GROW and LINK are designed to carry out them, respectively. The concept of these two strategies is illustrated in Figure 2. With the *growing strategy*, the building-up process starts from a "seed" structure that has been pre-placed in the binding pocket. The user can assign certain "growing sites" on the seed structure and then the program will try to replace each growing site with a candidate fragment. The newly formed structure will serve as the seed structure for the next growing cycle. With the *linking strategy*, the building-up process also starts from a pre-placed seed structure. However, in this case the seed structure consists of several separated pieces that have been positioned to maximize the interactions with the target protein. The growing of fragments happens simultaneously on each piece and the program will always try to link these pieces in an acceptable way. This process continues until all the pieces have been integrated into one molecule.

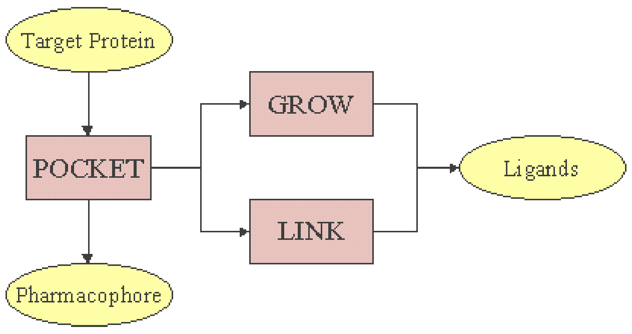


Figure 1 *Overall structure of LigBuilder*

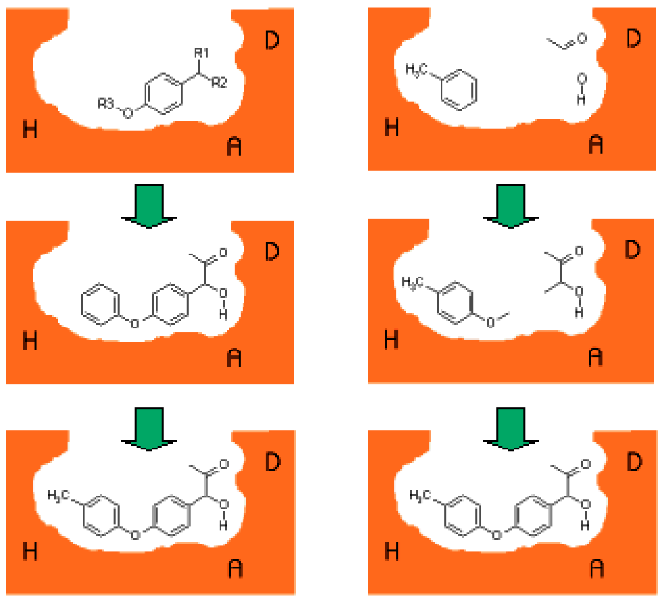


Figure 2 *Growing strategy (the left) and linking strategy (the right) for building up ligands*

As implied above, LigBuilder constructs molecules step by step. The possible solution space for a given design problem could be extremely large due to the combinatorial nature of the construction process. Although in practice the constraints of the binding pocket will reduce the solution space largely, it is still impossible to perform a systematic sampling. For such a complex, large-scale problem, genetic algorithms have proved to be very effective for finding optimal solutions within a reasonable amount of time.[34] Therefore, we have adopted a genetic algorithm to control the whole ligand building-up process in GROW and LINK. The molecules selected as the final outputs are usually ”winners” among thousands of candidates.

Binding pocket analysis

The main function of POCKET is analyzing the binding pocket. Since we treat the target protein as rigid throughout the ligand construction process, this step is necessary only once for a given protein.

The basic input for POCKET is the 3D structure of the protein that is represented in PDB format. The user may also include metal ions and water molecules if they are an important part of the binding pocket. A pre-docked ligand is required to help the program to locate the binding pocket. The program will define a box to cover the ligand and all the surrounding residues and create regular-spaced grids within the box. The grid spacing is 0.5Å by default. Then the program places a hydrogen atom as a probe on each grid to check its accessibility. If the probe bumps into the protein, that grid will be labeled as ”excluded”. A bump is counted when the interatom distance is less than the sum of van der Waals radii minus 0.5Å. The van der Waals radii of all atom types are taken from the Tripos force field.[35] If the probe does not bump into the protein, that grid will be labeled as ”vacant”. Note that, if a grid is farther than 5Å from any atom on the protein, it will be labeled as ”outside” rather than ”vacant”.

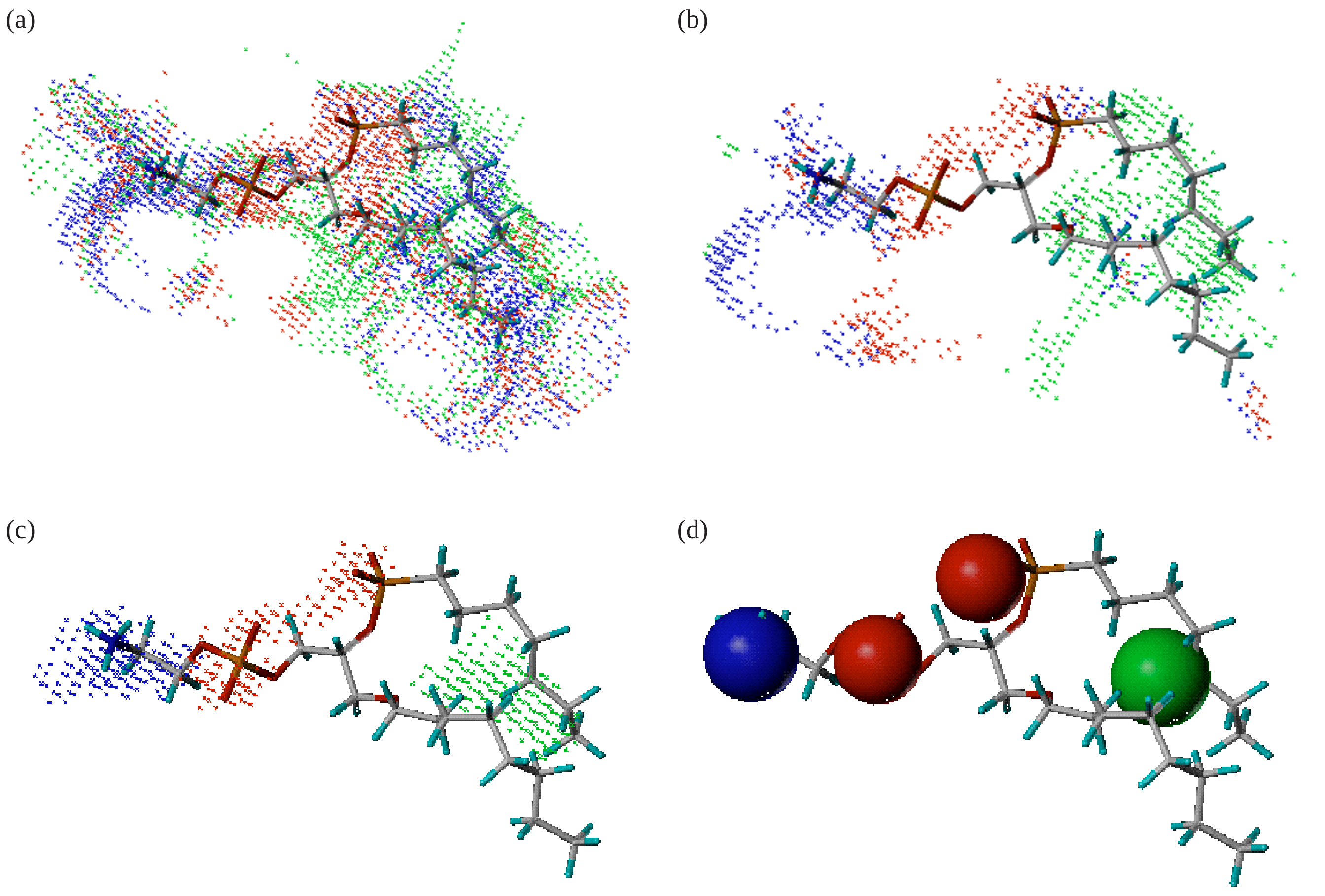


Figure 3 *Deriving key interaction sites of phospholipase A2 (PDB entry 1POE) with the POCKET program. (a) After screening all the grids with three types of probe atoms. (b) After filtering out unimportant grids. (c) After filtering out isolated grids. (d) Pharmacophore model obtained. (In all figures, hydrogen bond donor grids are colored in blue, hydrogen bond acceptor grids in red, and hydrophobic grids in green. The protein is hidden for the sake of a clear representation. The ligand is shown here for instead to illustrate the shape of the binding pocket)*

The assembly of ”vacant” grids forms the body of the binding pocket within which ligands will be built up.

As the next step, the program will derive key interaction sites within the binding pocket. Such information is necessary for the subsequent ligand construction process. The program uses three different types of probe atoms to screen the binding pocket, which include (1) a positively charged sp^3 nitrogen atom (ammonium cation), representing a hydrogen bond donor; (2) a negatively charged sp^2 oxygen atom (as in a carboxyl group), representing a hydrogen bond acceptor, and (3) a sp^3 carbon atom (methane), representing a hydrophobic group. For each ”vacant” grid, all the three probe atoms are applied and the binding energies between the probe atoms and the protein are calculated by using an empirical scoring function we have developed (see the *Scoring method* section below). Each grid will be labeled as ”donor”, ”acceptor”, or ”hydrophobic” according to the highest score on this grid. At this stage, however, we still do not have a clear image of where the key interaction sites are located (see Figure 3a). The program will then ”filter” all the grids to derive the key interaction sites in a two-step process. At the first step, the program calculates the average score of all ”donor” grids. Then it will figure out the grids which score lower than the average and label them back to ”vacant”. The same process is also repeated for the ”acceptor” grids and the ”hydrophobic” grids. After this step, only those grids with significant contributions to the ligand-protein binding process will survive (see Figure 3b). At the second step, the program checks each survived ”donor” grid and counts the number of its neighbors. Here ”neighbor” refers to the same type of grid within 2Å. The program will calculate the average number of neighbors for all the ”donor” grids. Those grids with less neighbors than the average will be filtered out and labeled back to ”vacant”. The same process is also repeated for the ”acceptor” grids and the ”hydrophobic” grids. After this step, only those grids in aggregation will survive and now they represent the key interaction sites within the binding pocket clearly (see Figure 3c). Finally, the program will locate the geometric center of each aggregation and use it to suggest a pharmacophore model (see Figure 3d). This binding-site-derived pharmacophore model could be used as the query structure to perform 3D database searching, which provides an additional way to find novel ligand molecules that fit to the target protein.

POCKET will also generate two other output files: one file stores all the atoms forming the binding pocket; the other file stores all the grids within the binding pocket. These two files will be used by GROW and LINK in the subsequent ligand construction process.

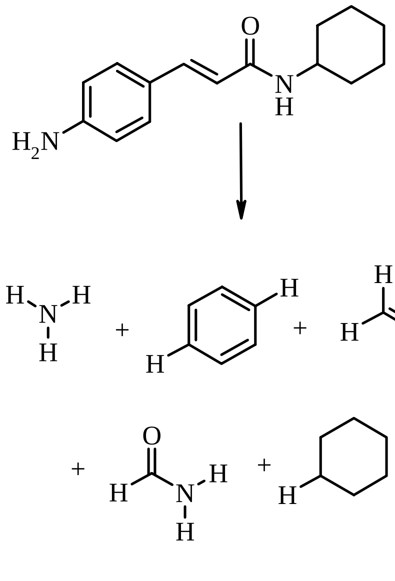


Figure 4 *Decomposing organic structures into basic fragments*

Building-up method

Building block library

LigBuilder uses a fragment-based algorithm to construct molecules. The term ”fragment” is used here to describe the building blocks used in the construction process. The rationale of this algorithm lies in the fact that organic structures can be decomposed into basic chemical fragments (see Figure 4). Although the diversity of organic structures is infinite, the number of basic fragments is rather limited. The fragments used by LigBuilder are listed in Scheme 1. All of them can be classified into two categories: chemical groups and rings. In this library, there are also some complex fragments, such as acetone, which can be decomposed into more elementary fragments. The purpose of including these ”redundant” fragments is to speed up the ligand construction process.

All the fragments are stored in SYBYL MOL2 format and their structures are minimized. If a fragment could take different conformations, then the favorite one is chosen. For example, cyclohexane is represented in the chair conformation. The user is allowed to edit the building-block library to determine which fragments will be used in the ligand construction process. The user is also allowed to add new fragments to this library to meet his special purpose.

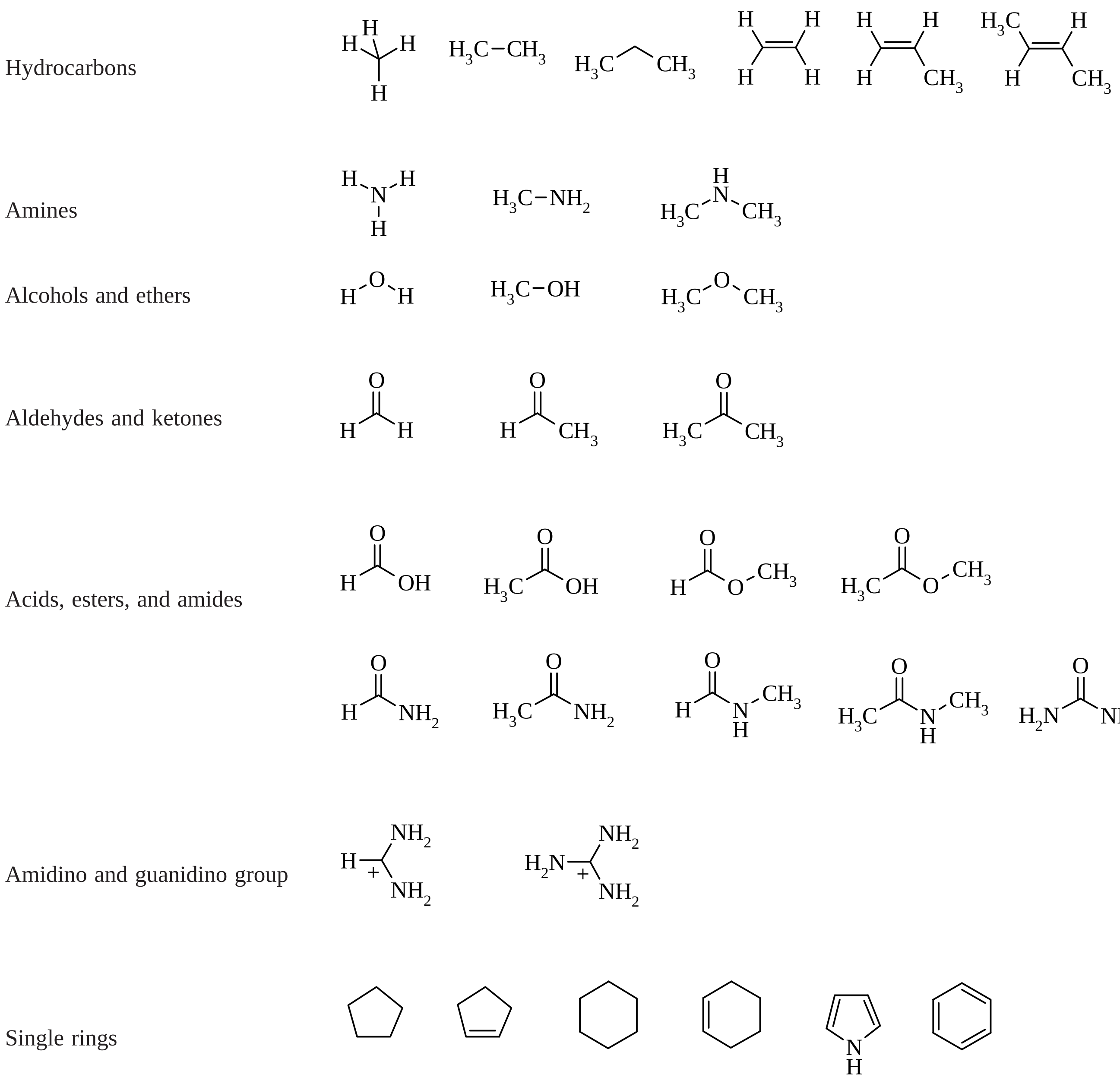
Structural operation

In LigBuilder, the basic structural operation for building up a molecule is adding a fragment to an existing molecule. A molecule can be finally constructed by repeating this operation.

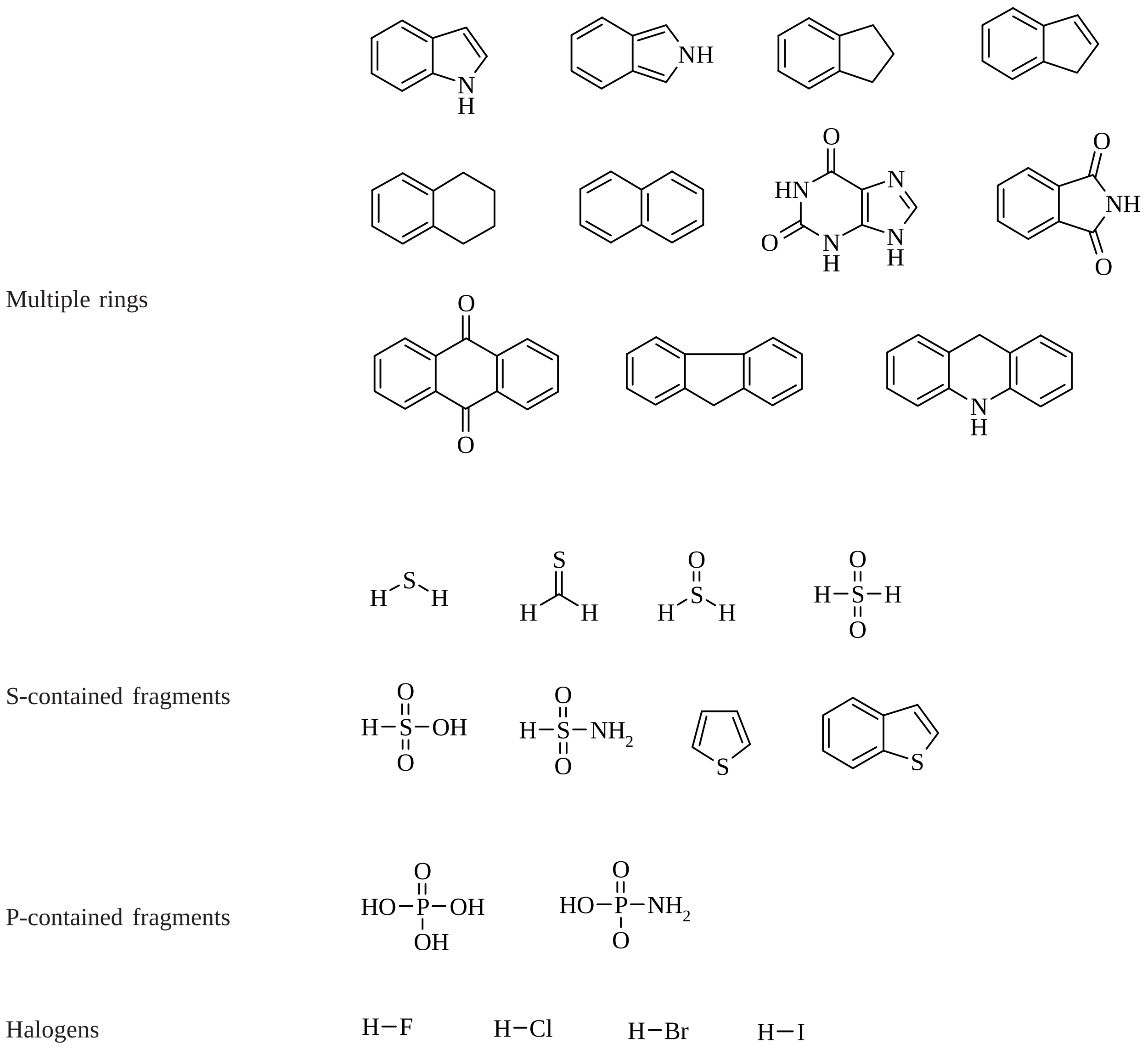
This process starts with *growing operation*. In order to add a fragment onto an existing molecule (referred as ”core” in the following text), the program selects a hydrogen atom on the core and a hydrogen atom on the fragment. The bonds connecting the hydrogen atoms and the corresponding heavy atoms are used to orient the fragment to the core. Once the fragment has been oriented correctly, the two hydrogen atoms are deleted and a new single bond is created to connect the core and the fragment (see Figure 5).

In principle the newly formed single bond is rotatable. Therefore the next step is to determine the dihedral angle between the fragment and the core. To do so, the fragment is rotated along the new bond thoroughly in an increment of 15° and the program will calculate the steric energy for each resultant conformation (24 conformations in total). Based on the energy profile obtained, the program will pick out the conformation(s) corresponding to the energy minima. Since a dihedral angle is usually multiple-folded, growing a fragment on the core structure with the above algorithm often results in more than one conformation. The program will consider them all and treat them as different molecules. In this way, the flexibility of the ligand molecule is taken into account during the ligand construction process.

During the systematic rotation, the fragment may collide with the core. Of course such a colliding conformation will



Scheme 1 (continues next page) *The building block library used in LigBuilder*



Scheme 1 (continued) *The building block library used in LigBuilder*

not be a minimum on the energy profile. However, if the fragment collides with the core in such a way that they can be linked together reasonably, it will be an alternative way to generate structures. We call this a *linking operation*. We have designed three types of linking algorithms (see Figure 6). Which algorithm will be applied basically depends on the distance between the two bumped structures. The first algorithm is applied when the two structures bump only by a pair of hydrogen atoms. In this case, the program will delete the colliding hydrogen atoms and use a methylene group to bridge these two structures. The second algorithm is applied when the two structures bump by a pair of heavy atoms. In this case, the program will create a new single bond to connect the two heavy atoms directly. If the two structures are so close that a pair of heavy atoms overlap, the third algorithm will be applied. In this case, the program will delete one of the overlapped atoms and connect the remained structures. Note that a linking operation will be accepted only when geometry and chemistry of the linkage are both proper. If a linking operation results in reluctant conformations or unreasonable chemical structures, it will be rejected.

The next step is to make necessary modifications on the structure. We call this a *mutation operation*. Our program allows carbon, nitrogen, and oxygen atoms with the same hybridization state to mutate to each other (see Scheme 2). We assume that the bond angles and bond lengths are unchanged during the mutation operation because such changes are actually neglectable comparing to other inaccuracies embedded in the ligand construction process.

We do not allow the mutation to happen randomly on the structure. On the contrary, we adopt a ”smart mutation” strategy. Before performing mutation, the program will check each heavy atom of the molecule to see whether it fits the grid on which it lies. For example, it is all right if a carbon atom lies on a ”hydrophobic” grid. But if it lies on a ”donor” grid, the program will try to mutate it to a hydrogen bonding donor atom, such as nitrogen. The information of the grids is given by POCKET in advance. With this ”smart mutation” strategy, our program improves the ligand binding affinity with a minimal computational cost.

After all the above operations have finished, the program will check the newly generated molecule to see whether it bumps to the target protein. If so, the molecule will be rejected. By default, all the hydrogen atoms on the fragment will be labeled as growing sites for further development of the molecule. However, there could be some hydrogen atoms that are already so close to the protein that no more fragments can be added. The program will label them as dead ends. No attempt will be made to grow fragments from them.

All the above operations are implemented both in GROW and LINK. All the parameters for bond length, torsion angle potential, and van der Waals radii are taken from the Tripos force field.[35]

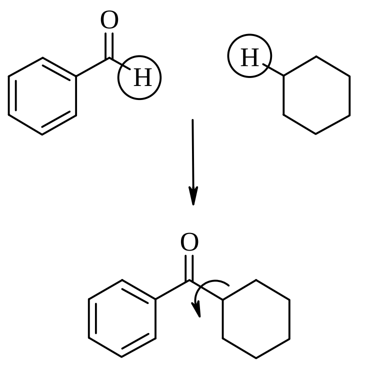
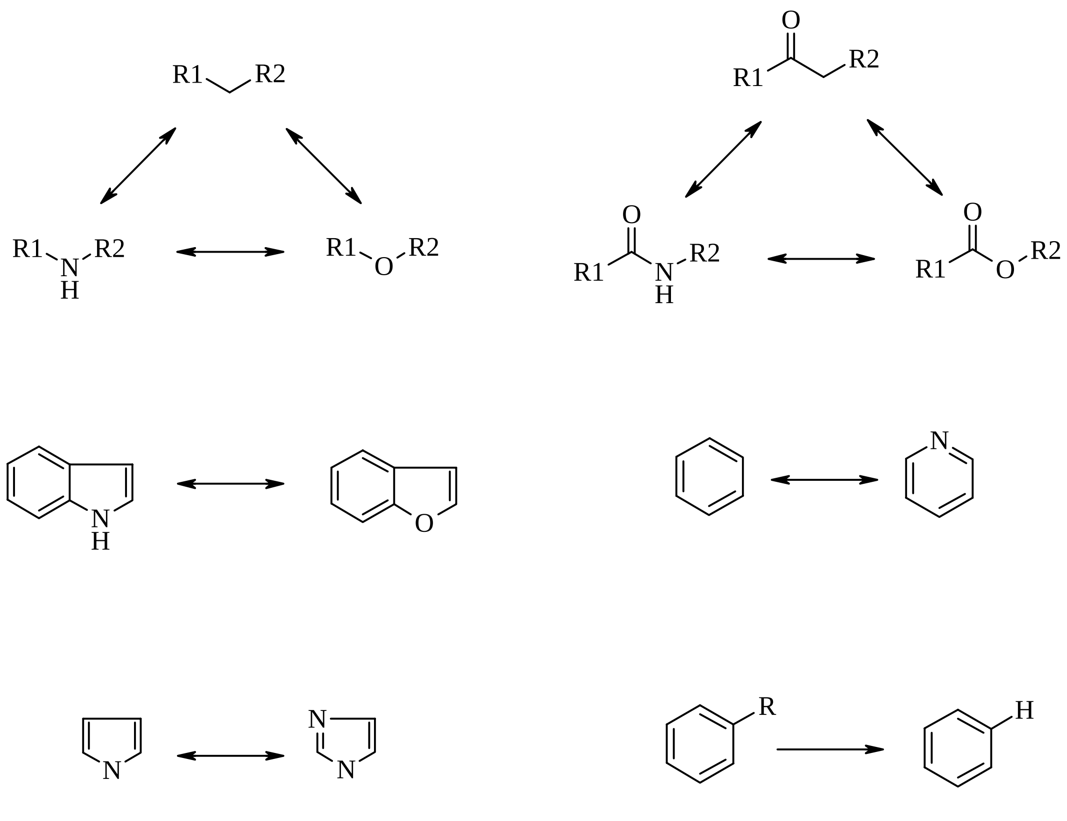


Figure 5 *Illustration of growing operation*

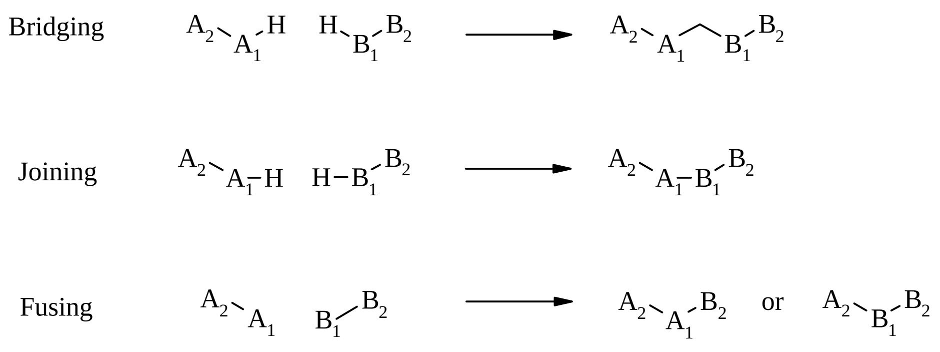
Chemical rules for preventing unreasonable structures

Our program also checks each newly generated molecule in two ways to ensure its chemistry is reasonable. First, we have built a set of rules in the program to define unacceptable structures. Such structures mainly include those in which hetero-atoms bond to each other, e.g. O-O, N-N, and N-O, and those in which too many hetero-atoms bond to the same carbon atom. These rules are derived from the analysis of current drug molecules and the knowledge of our best or-



Scheme 2 *Mutation operations allowed in LigBuilder*

Figure 6 *Illustration of linking operation (Here A and B represent heavy atoms)*



ganic chemists. If a molecule violates any of the rules, it will be rejected. Secondly, we allow the user to supply an external fragment library to filter the molecules generated further. This library is called the ”forbidden substructure library”. The user can build and deposit any chemical structure that is not desired in the resultant molecules into this library. The program will check each molecule with a substructure mapping algorithm. If a molecule contains any of the forbidden fragments, it will be rejected too.

Scoring method

In LigBuilder, a ligand molecule has two kinds of score. One is about its binding affinity to the target protein while the other is about its bioavailability.

To calculate the binding affinity score, our program uses the SCORE algorithm we have described before.[36] SCORE is an empirical procedure developed to estimate the binding free energy of a ligand molecule to its receptor protein when the 3D structure of the complex is known. The basic idea of SCORE is to dissect the binding free energy into some components. It uses the following linear scoring equation.

$$\Delta G_{bind} = \Delta G_{vdw} + \Delta G_{H-bond} + \Delta G_{hydrophobic} + \Delta G_{rotor} + \Delta G_0 \quad (1)$$

Here, ΔG_{vdw} refers to the contribution of van der Waals interaction; $\Delta G_{H\text{-}bond}$ refers to the contribution of hydrogen bonding; $\Delta G_{hydrophobic}$ refers to the contribution of hydrophobic interaction; ΔG_{rotor} refers to the entropy loss due to the freezing of rotatable bonds in the ligand; ΔG_0 is a constant. The coefficients of each term are determined by multivariate regression analysis of 170 protein-ligand complexes with known binding free energies and crystalline structures. This scoring function reproduces the absolute binding free energies of the whole training set with a standard deviation of 1.6 kcal·mol^{-1}. Such a scoring function is especially suitable for a drug design program because it gives an estimation of the binding free energy and makes a good comprise between speed and accuracy.

Good binding affinity is only part of the story for a successful drug molecule. In recent years, more and more attention has been paid to predicting the bioavailability of a molecule.[37-39] Maybe the most popular approach is the so-called ”Lipinski rules”.[37] According to the Lipinski rules, poor absorption or permeation is more likely when (i) molecular weight is over 500, (ii) logP is over 5, (iii) there are more than 5 H-bond donors (the sum of OHs and NHs), or (iv) there are more than 10 H-bond acceptors (the sum of Os and Ns). We have incorporated these rules in the program to evaluate the bioavailability of the designed molecules. If a molecule violates any of these rules, it will be penalized in its bioavailability score and therefore become less competitive in the genetic algorithm procedure. The more it violates a certain rule, the more it will be penalized. By default we use the criteria given by the Lipinski rules. However, the user is allowed to change the criteria according to his own purpose.

While the molecular weight and the number of H-bond donors or acceptors can be easily obtained, the logP value is calculated by using the XLOGP algorithm we have described before.[40]

Genetic algorithm procedure

Due to the combinatorial nature of the ligand building-up process, the possible solution space for a given design problem is extremely large. For example, assuming that you have

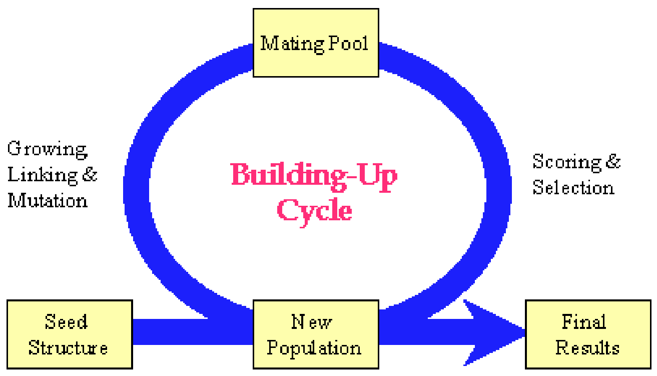


Figure 7 *Flowchart of the genetic algorithm implemented in LigBuilder*

a seed structure with three growing sites and there are 60 fragments in the building-block library, you may get as many as 60*60*60=216,000 derivative molecules. In the next growing round, these 216,000 molecules will generate countless new molecules. For such a complex, large-scale problem, it is simply impossible to perform a systematic sampling. Therefore, we have implemented a genetic algorithm (GA)[34] to control the ligand building-up process in GROW and LINK.

The flow chart of the GA procedure is illustrated in Figure 7, which follows the typical ”generational-replacement” strategy. The whole procedure is basically the same for GROW and LINK. The user is required to give a few necessary parameters, including the population size, the mating pool size, the number of GA generations, and the maximum number of resultant molecules. The following sections will explain the whole GA procedure in detail.

Generating the initial population

The whole procedure starts from a seed structure that is provided by the user. All the resultant molecules given by LigBuilder can be considered as its derivatives. Although there is no problem to develop a program which can suggest a seed structure automatically, we have decided not to do so. It is because we want the user to apply his expertise to the design procedure rather than just use the whole program as a black box. For a drug design project, it is usually not demanding to propose such a seed structure. Typically the seed structure may come from a part of a known inhibitor or any other structure of interest.

The seed structure also needs to be pre-docked into the binding pocket by the user. It will be ideal that the seed structure forms some specific interactions with the target protein. Another thing that the user is supposed to do is to assign growing sites on the seed structure. In LigBuilder, a growing site refers to a certain hydrogen atom of the molecule onto which a fragment could be added. By assigning the growing sites, the user can control where the fragment growing will happen. The user is also encouraged to incorporate his knowledge of organic synthesis by choosing the proper growing sites.

Once the seed structure is ready, the program begins to generate the initial population. Unlike the traditional genetic algorithm, we have decided not to use a binary encoding but rather carry out operations upon the chemical structures themselves. It is more straightforward and intuitive to handle a molecule in this way. Each member in the initial population is generated by adding a certain fragment to a certain growing site on the seed structure with the algorithm described in the *Building-up method* section (growing, linking, and mutation). Therefore, there is no combinatorial problem at this step. The program will try all the available fragments in the building-block library on each growing site on the seed structure and record all the resultant structures. According to our experience, the size of the initial population is approximately proportional to the number of growing sites on the seed structure multiplying the number of fragments in the building-block library. Usually the program will generate several hundreds of structures for the initial population.

Fitness function and selection method

The fitness value of a molecule is given by combining its binding affinity score and bioavailability score. To avoid a subjective selection of weight factors, we have adopted the ”tournament strategy”.[34] The first step is to rank all the members of current population in an increasing order according to the binding affinity score. Then the program counts the ”wins” of each member, which usually equals to its rank minus 1. The second step is to re-rank all the members in an increasing order according to the bioavailability score and, again, count the ”wins” of each member. We define that the fitness value of a molecule is the sum of its two ”wins”. Besides avoiding weight factors, another advantage of this tournament strategy is that it still works well when the difference between the best score and the worst score in the population is too large or too small since it only cares about the relative ranking.

We use the ”roulette-wheel” algorithm to select a certain number of members into the mating pool. Each member in the population will be attributed a slice on the wheel that is proportional to its fitness value. Each time when the wheel spins, a member will be piceked out randomly. Thus, the members with higher fitness values are more likely to enter the mating pool and breed offspring thereafter. The selection process is repeated till the mating pool is full.

While selecting the parent molecules, we allow the user to exert a forced molecular diversity by setting a maximum similarity criterion. In our program, the 3D similarity between two molecules, e.g. A and B, is given by:

$$S_{AB} = \frac{num\_match}{num\_A + num\_B - num\_match} \tag{2}$$

Here ”num_A” refers to the number of heavy atoms in A; ”num_B” refers to the number of heavy atoms in B; while ”num_match” refers to the number of matched atom pairs. Two atoms are considered to be matched if they are of the same atom type (as defined in the Tripos force field) and overlap each other (the distance between them is shorter than 0.5Å). According to the above equation, the maximum similarity could be 1 while the minimum could be 0. While adding a new molecule to the mating pool, the program will calculate the similarity values between this molecule and all the members existing in the mating pool. If any similarity value thus obtained exceeds the user-defined criterion, the molecule will not be allowed to enter the mating pool. By doing so, the similarity between any two members in the mating pool will be below that similarity criterion and therefore the desired diversity is achieved.

Generating new population

The first step to generate a new population is using the elitism algorithm.[34] The user can define a certain elitism ratio, for example 0.10. This means that the top 10% of the old population will be copied directly into the new population. Elitism ensures that the best members in the old population will not lose unless they are replaced by better candidates.

Then the new population is filled by using the molecules in the mating pool as seed structures. Each member in the mating pool will be picked out in turn. The program randomly selects a growing site on it and randomly selects a fragment from the building-block library. Then the fragment will be added onto that growing site by using the algorithm described in the *Building-up method* section (growing, linking, and mutation). Resultant molecules will be added to the new population. This process is repeated till the population is full.

Duplicate checking is performed while adding new molecules to the population. A molecule will be checked against the existing members. If it is found to be a duplicate, it will be discarded. Since we allow flexibility of the ligand molecules, different conformations of the same molecule are not treated as duplicates.

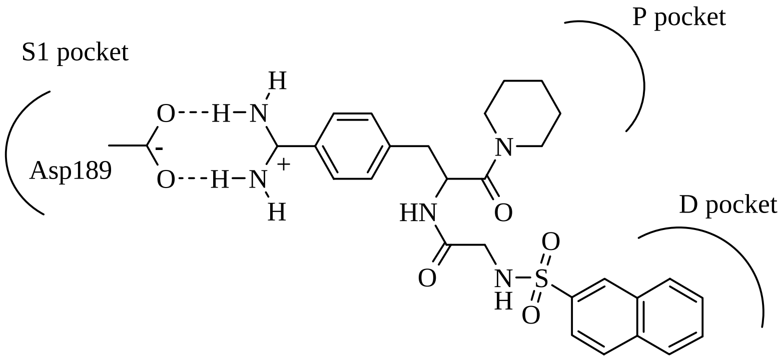
Processing the final results

The GA procedure moves from generation to generation and the average fitness of the population increases steadily. This procedure will stop when the user-defined limit of generations is reached. Then the program will rank all the molecules in the final population in decreasing order in terms of their fitness values. A user-defined number of molecules at the top will be selected as the final results. Each of them will be output in a SYBYL MOL2 file. The program will also give a log file tabulating the file name, molecular weight, logP value, binding affinity and bioavailability score of each molecule.

To help the user to analyze the results, our program will also group the resultant molecules into clusters. We have designed a simple clustering algorithm to do this. First, the program calculates the similarity value (as defined in Equation 2) between every two molecules. All the results form a similarity matrix and the program will calculate the average value of this matrix. As a starting point, the program assumes that each molecule belongs to a different cluster. Then a multistep clustering process is launched, which can be briefly described as:

(i) Find the largest element in the current matrix.

(a)



(b)

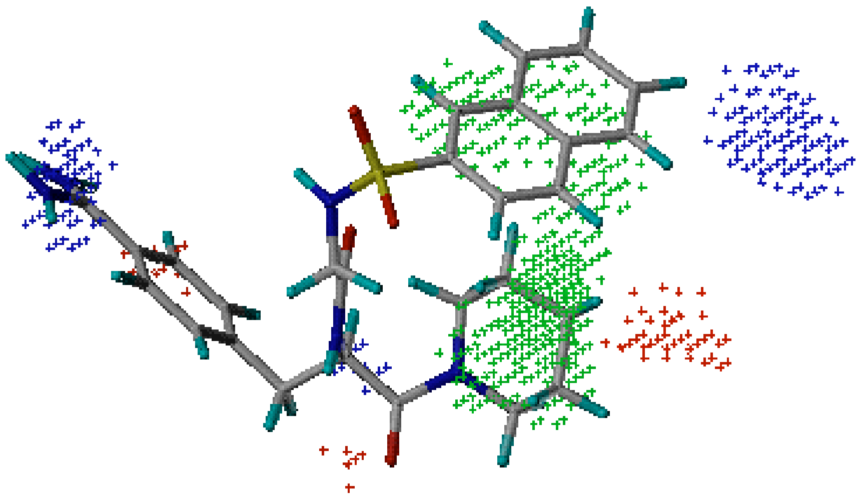


Figure 8 *Key interaction sites of thrombin (PDB entry 1DWD). (**a**) NAPAP and the known interaction sites. (**b**) The results given by POCKET (Hydrogen bond donor grids in blue, hydrogen bond acceptor grids in red, and hydrophobic grids in green. Thrombin is hidden for the sake of a clear representation)*

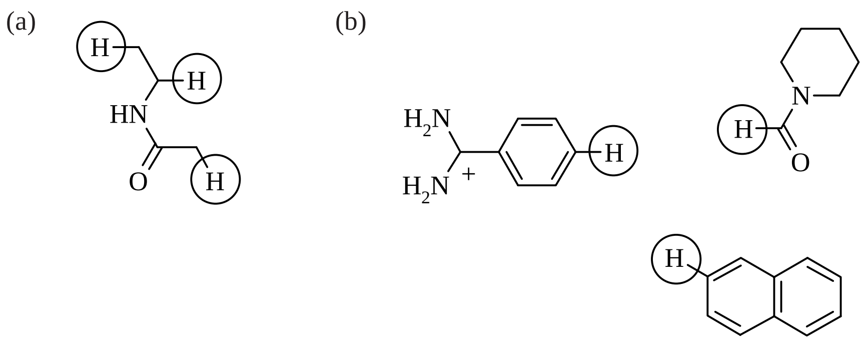


Figure 9 *(**a**) The seed structure used by GROW. (**b**) The seed structure used by LINK. (Growing sites are labeled with circles)*

(ii) Find out the two molecules related to this element.

(iii) If these two molecules belong to the same cluster, go to step (v).

(iv) If these two molecules belong to two different clusters, judge whether these two clusters can be merged together or not. If the similarity between any two molecules in these two clusters is always larger than the average value of the matrix, they will be merged into one cluster. If not so, they are kept unchanged.

(v) Erase the current element and go back to step (i). This cycle is repeated until there is no element larger than the average value.

The above clustering algorithm is very simple and effective. Compared to other clustering algorithms,[41] the key advantage of our algorithm is that the user does not have to supply any parameter for this process.

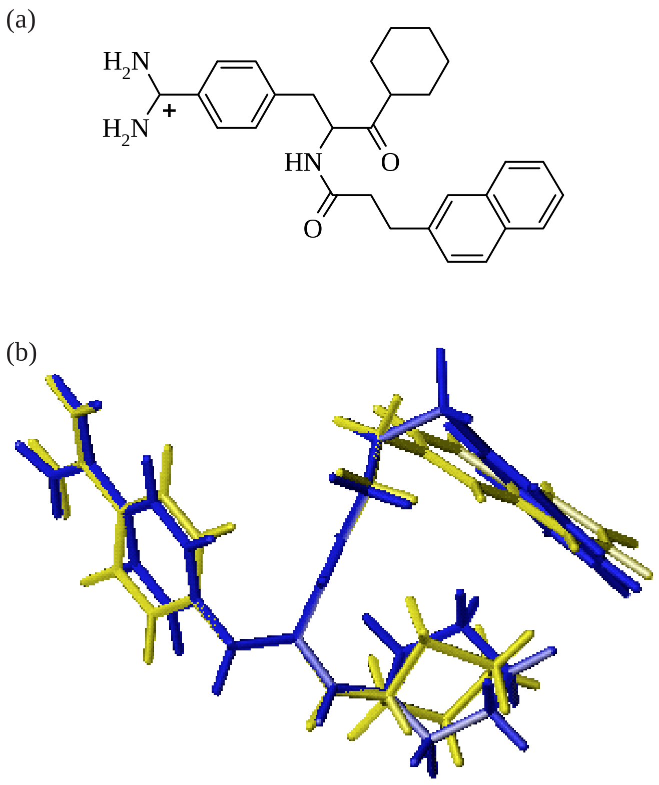


Figure 10 *(**a**) The most similar molecule given by GROW. (**b**) Superimposed with NAPAP (this molecule in yellow while NAPAP in blue)*

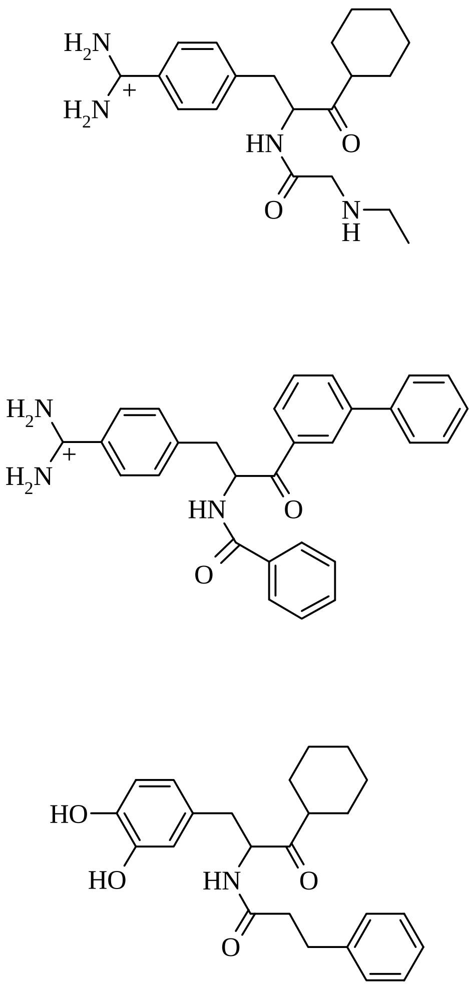


Figure 11 *Some interesting ligands for thrombin given by GROW*

Validation

We have tested LigBuilder on two well-characterized enzymes, thrombin and dihydrofolate reductase. In the following text, we will describe briefly the procedure of running a LigBuilder job for a given target, from analyzing the binding pocket to building up ligand molecules with growing strategy and linking strategy. Rather than suggesting fancy ”novel” structures, we focus on reproducing known ligand molecules. We believe this is the right way for testing a drug design program.

Thrombin

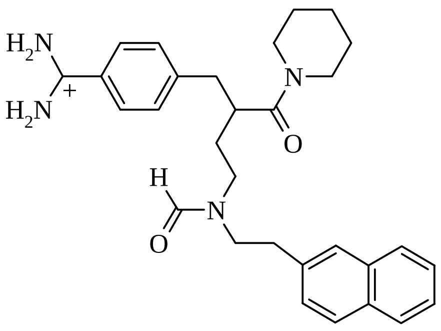
The final step in the process of blood clot formation is the hydrolysis of fibrinogen to fibrin by the serine protease thrombin. This enzyme thus constitutes a good target for the development of antithrombotic agents and has been well studied.

The crystal structure of alpha-thrombin complex used in our test was drawn from the Protein Data Bank (entry 1DWD). The binding pocket of thrombin contains three principal interaction sites, which, according to the literature,[42] are denoted as S1, D, and P (see Figure 8a). The S1 site contains an Asp residue which interacts with a positively charged counter part on the ligand molecule. The D site (denoting its distal relation to the catalytic site) is a hydrophobic pocket which is a favorable binding site for aromatic rings. The P site (denoting its proximal relation to the catalytic site) is also hydrophobic in nature and is important for thrombin specificity. In this complex, the ligand molecule, NAPAP, fits these interaction sites well and exhibits a high binding affinity to thrombin (K_i=10^{-8}M).

To design ligands for thrombin, the first step was to use POCKET to analyze the binding pocket. The thrombin complex structure was used as the input for POCKET and the program reproduced the key interaction sites faithfully. In Figure 8b, we can clearly see the S1 site (the blue aggregation on the left) which overlaps the amidine group of NAPAP. We can also see the D and P site (the green aggregations in the middle) which overlap the hydrophobic rings of NAPAP. It is interesting to notice that POCKET has figured out some other interaction sites which might have been neglected before.

As the next step, we ran GROW by using part of NAPAP as the seed structure. Since NAPAP is a good inhibitor for thrombin, we expect the program to yield some molecules similar to NAPAP. The seed structure was extracted from the

(a)



(b)

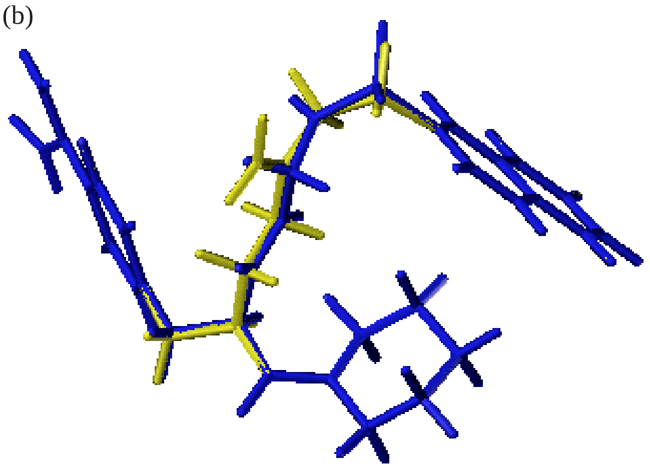


Figure 12 (**a**) *The most similar molecule given by LINK.* (**b**) *Superimposed with NAPAP (this molecule in yellow while NAPAP in blue)*

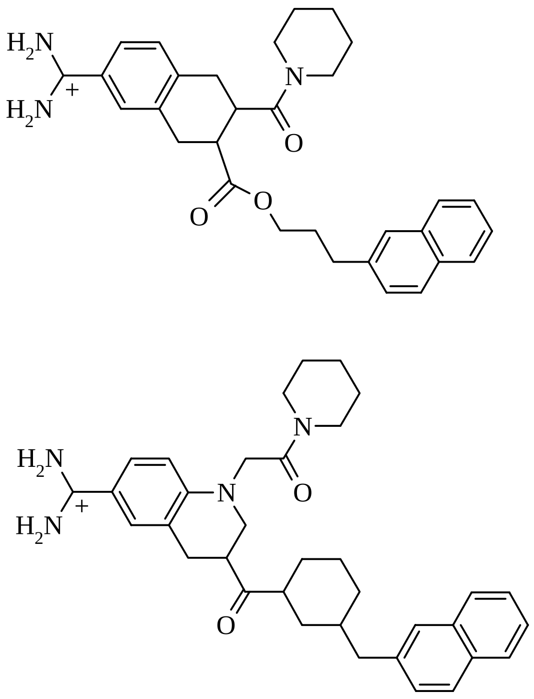
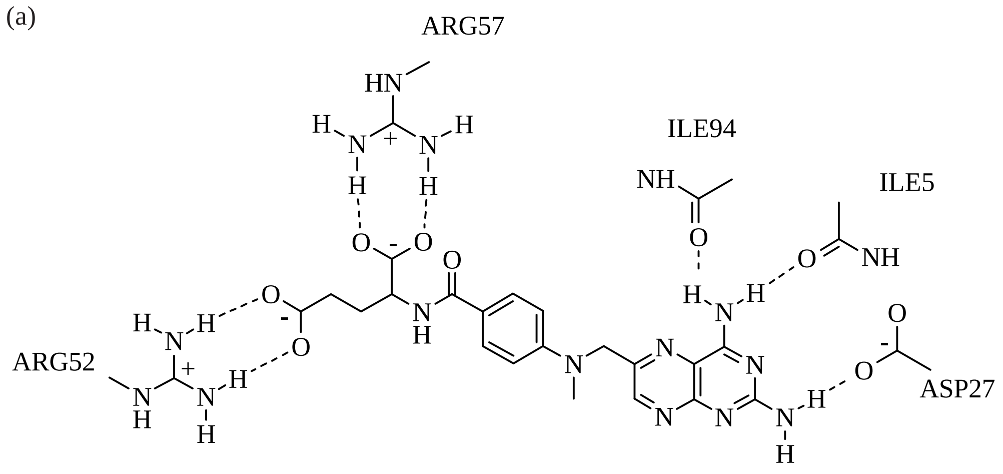


Figure 13 *Some interesting ligands for thrombin given by LINK*



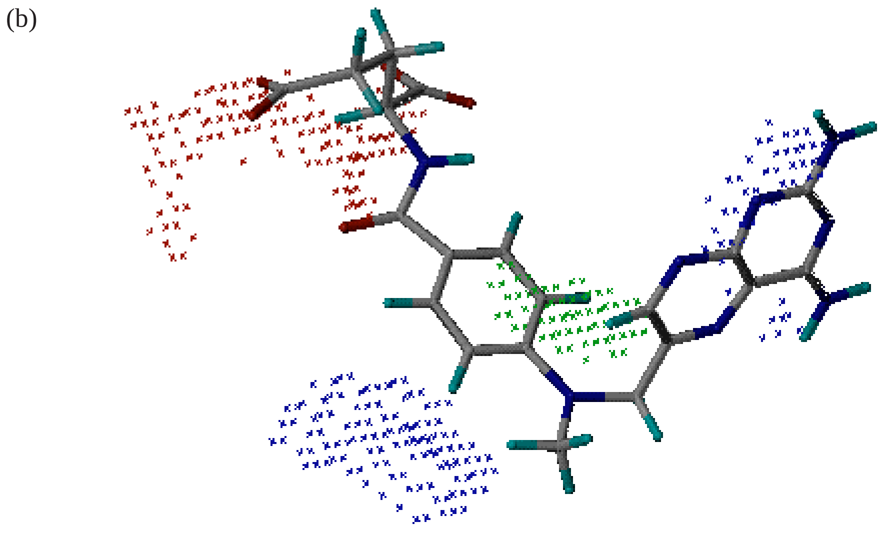
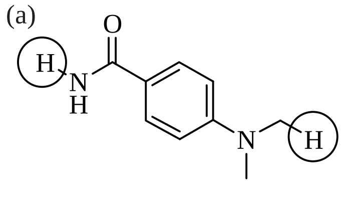


Figure 14 *Key interaction sites of DHFR. (**a**) MTX and the known interaction sites. (**b**) Results given by POCKET (Hydrogen bond donor grids in blue, hydrogen bond acceptor grids in red, and hydrophobic grids in green. DHFR is hidden for the sake of a clear representation)*

central part of NAPAP and three growing sites were assigned on it (see Figure 9a). The bioavailability rules were correspondingly modified to allow structures like NAPAP to occur. GROW was submitted with a population size of 3000, a generation limit of 10, and a maximum output of 100. Due to the stochastic nature of genetic algorithm, we have run the program five times to sample the solution space adequately. Each run took about 5 hours on a SGI O2/R10000 workstation.

Among the results, we did find a molecule extremely similar to NAPAP (see Figure 10a). For the S1 site, this molecule is exactly the same as NAPAP. For the P site, this molecule has a more hydrophobic cyclohexane ring instead of the piperidinium ring of NAPAP. For the D site, this molecule is also more hydrophobic than the sulfonamide counter part of NAPAP. As a whole, this molecule simulates NAPAP well both in structure and conformation (see Figure 10b). The K_i value of this molecule is predicted to be 10^{-9}M, which is also close to the known data of NAPAP. Besides this molecule, there are still some other ones which are similar to NAPAP (see Figure 11).



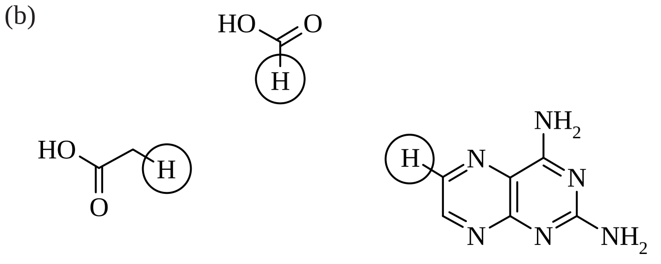


Figure 15 *(**a**) The seed structure used by GROW. (**b**) The seed structure used by LINK. (Growing sites are labeled with circles)*

We also tested LINK on thrombin. In this case, the seed structure was extracted from the ”ends” of NAPAP, which included three separated pieces (see Figure 9b). The three growing sites on this seed structure were assigned according to the structure of NAPAP. The purpose of this job is to test whether LINK can link these three pieces in a reasonable way within the constraints of the binding pocket. LINK was submitted with a population size of 1000 and a generation limit of 10. This process was also repeated for five times. Each run took about 6 hours on a SGI O2/R10000 workstation.

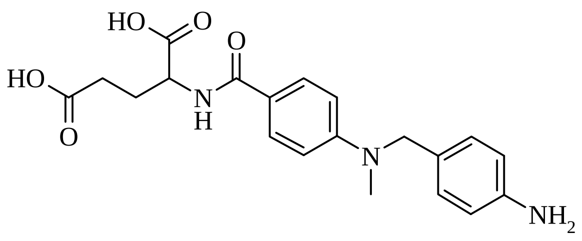
Among the results, we also found a molecule very similar to NAPAP (see Figure 12a). Although the framework of this molecule is slightly different from NAPAP, the ”style” is basically the same (see Figure 12b). LINK also suggested some other frameworks to assemble the given seed structure, which contain fused rings (see Figure 13). This has demonstrated that LigBuilder can generate rings rather than simply use the pre-made rings in the building-block library.

Dihydrofolate reductase

Dihydrofolate reductase (DHFR) is also a well-known target for structure-based drug design. It catalyses the NADPH-dependent reduction of dihydrofolate (FH2) to tetrahydrofolate (FH4). Inhibition of DHFR interrupts the supply of FH4, causing the disruption in the synthesis of purine and pyrimidine bases and eventually the death of the cell. The discovery of inhibitors of DHFR has led to several useful drugs for the treatment of cancer, bacterial infections and malaria.[43]

As a further validation, we have used the crystal structure of DHFR complex from the Protein Data Bank (entry 4DFR). In this complex structure, there is a ligand molecule MTX (see Figure 14a). This ligand fits the binding pocket well and shows a high binding affinity to DHFR ($K_i=10^{-8}$M). The well-characterized protein-ligand interactions in this case include the hydrogen bondings between Arg 52, Arg 57 and the glutamate moiety of MTX as well as the hydrogen bondings between Ile5, Asp27, Ile94 and the pteridine ring of MTX. In addition, the central part of the binding pocket is hydrophobic, which matches the benzene ring in MTX.

(a)



(b)

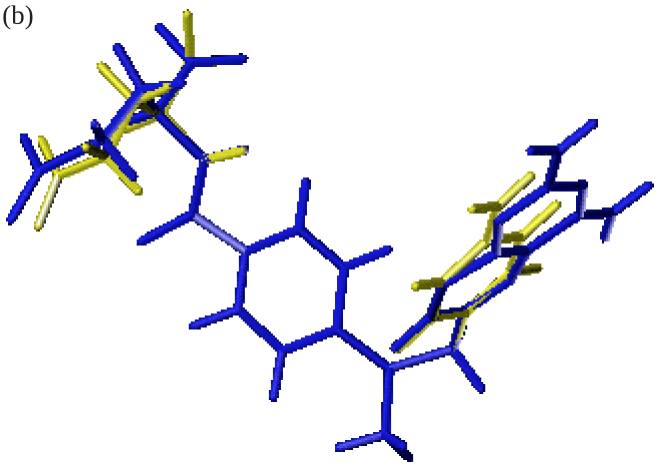


Figure 16 *(**a**) The most similar molecule given by GROW. (**b**) Superimposed with MTX (this molecule in yellow while MTX in blue)*

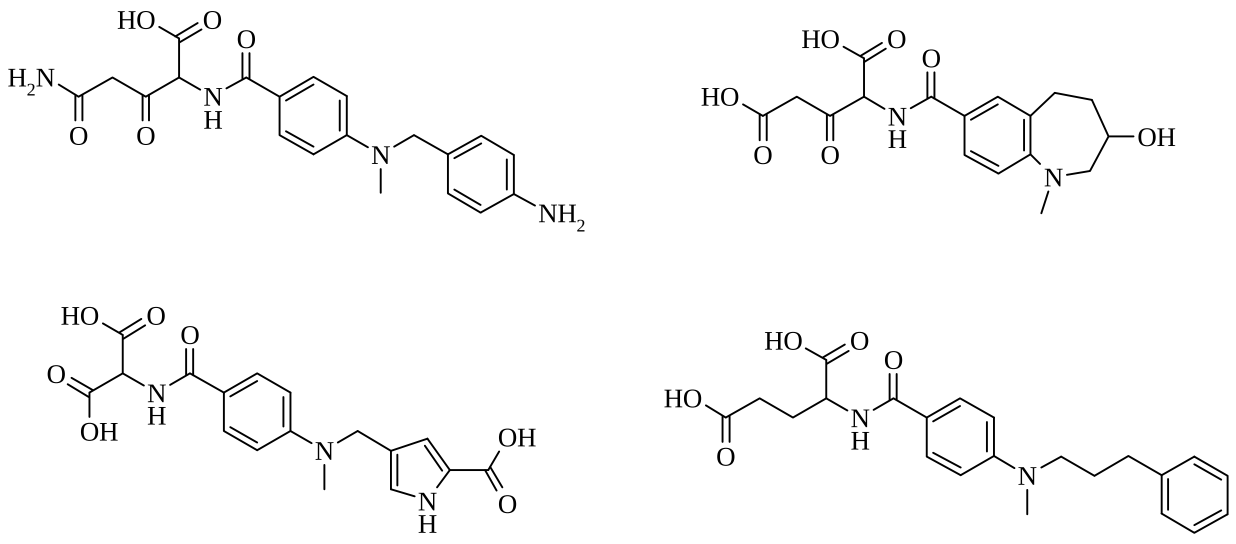
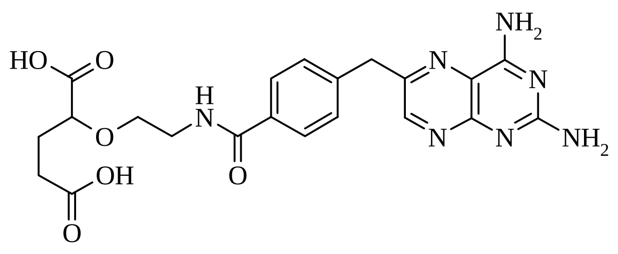


Figure 17 *Some interesting ligands for dihydrofolate reductase given by GROW*

(a)



(b)

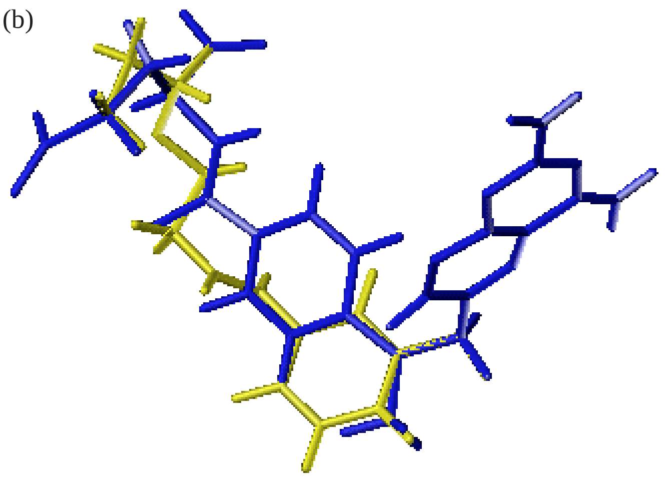


Figure 18 *(**a**) The most similar molecule given by LINK. (**b**) Superimposed with MTX (this molecule in yellow while MTX in blue)*

Again, POCKET reproduced the known interaction sites correctly based on the crystal structure of DHFR. In Figure 14b, we can find all the three major interaction sites mentioned above. Besides these, POCKET also pointed out that there is also a remarkable cavity on the side that favors hydrogen donor groups. We found later that, during the ligand construction process, some of the molecules given by the program do fill in that cavity.

The seed structure to run GROW was extracted from the central part of MTX (see Figure 15a) and two growing sites were assigned on it according to the structure of MTX. By doing so, we also expected to obtain some molecules similar to MTX itself. GROW was submitted with a population size of 3000, a generation limit of 10 and a maximum output of 100. The program was also repeated five times to sample the possible solutions adequately.

Again we found a molecule extremely similar to MTX among the results (see Figure 16a). Its conformation overlaps the one of MTX very well (see Figure 16b). It is remarkable because the program has precisely reproduced the flexible glutamate moiety. The only major difference is that this molecule has a substituted benzene ring instead of the original pteridine ring. But this benzene ring is also placed and substituted correctly to form hydrogen bondings with the adjacent residues. Besides this molecules, GROW also suggested some other structures that are also similar to MTX (see Figure 17).

We have also run LINK for DHFR. The seed structure is extracted from the ”ends” of MTX, which includes three separated pieces (see Figure 15b). The growing sites on this seed structure are also assigned according to the structure of MTX. LINK was submitted with a population size of 1000 and a generation limit of 10. This process was also repeated five times.

Unlike the example of thrombin we have described above, in this case the three pieces in the seed structure are considerably apart from each other. Therefore, LINK has figured out much more structures to assemble the given seed structure. Among them, the most similar molecule to MTX is

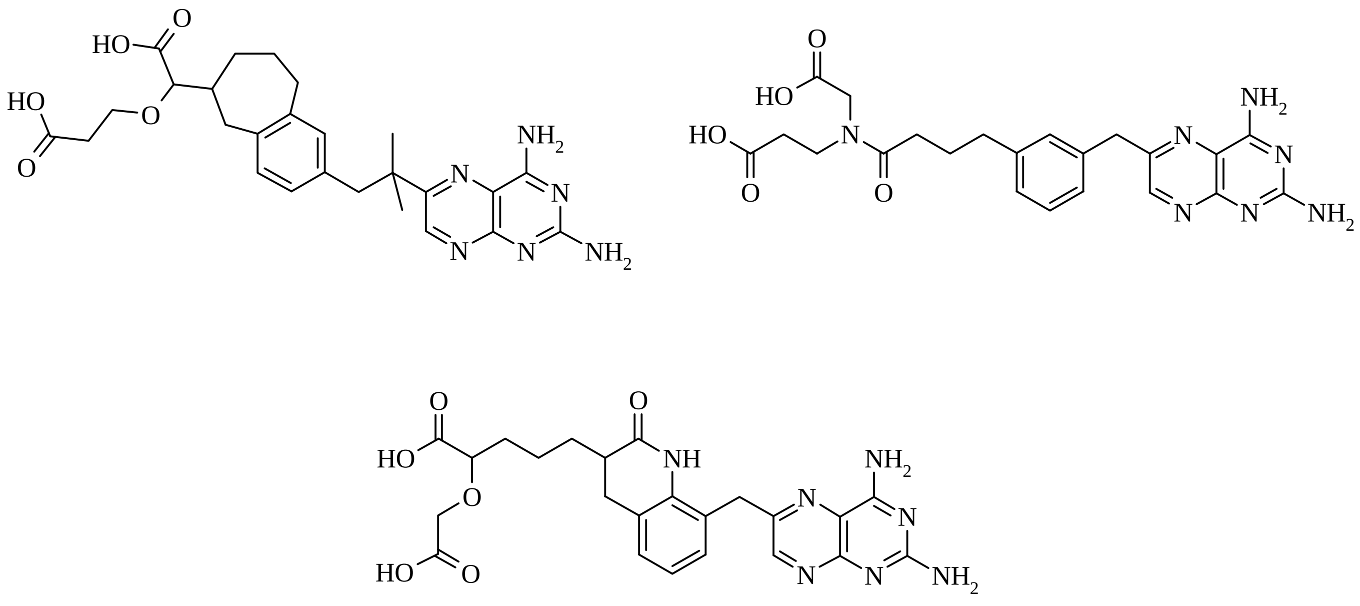


Figure 19 *Some interesting ligands for dihydrofolate reductase given by LINK*

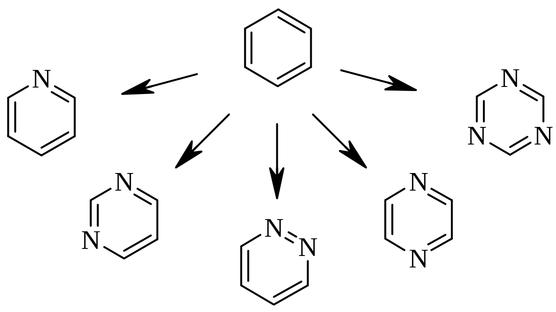


Figure 20 *Each fragment in the building block library is actually a template*

shown in Figure 18a. Although its structure looks similar to MTX, its conformation is somewhat different from MTX (see Figure 18b). This is simply because the central benzene ring of this molecule has ”moved” one unit toward the pteridine ring compared to MTX. But this may not a serious problem because the hydrophobic interaction occurred there is not so sensitive to the position of the benzene ring. Therefore, such a change in structure does not affect the binding affinity considerably. As a proof, the predicted K_i value of this molecule is 10^{-8} M, which is close to the one of MTX. LINK has also suggested many other ways to assemble the seed structure into one integrated molecule (see Figure 19). They all have reasonable conformations.

Discussion

LigBuilder has several remarkable features worthwhile further discussion. These features make it a practical tool for structure-based drug design approaches.

Fragment-based construction of molecules

As mentioned in the *Introduction* section, there have been a whole bunch of programs developed for de novo ligand construction. These programs try to assemble molecules by using some basic pieces, which could be either atoms or chemical fragments. Atom-based approaches include, for example, LEGEND,[44] GenStar[45] and GROWMOL[46] while fragment-based approaches include GROW [21], LUDI,[22-25] LEAPFROG,[26] SPROUT,[27] PROLIGAND,[28-33] NEWLEAD[47] and GROUPBUILD[48]. Clearly, using single atoms as the building blocks will give the maximum diversity because in principle all organic structures can be generated by assembling atoms. However, according to our own experience in developing atom-based program,[49] it is simply not easy to do so. Pure atom-based construction suffers from generating unreasonable structures and it is also inefficient in handling ring systems. In addition, since it needs more steps to build up the whole molecule atom by atom, the combinatorial problem is severe. These probably explain why the atom-based approaches are not so popular at present.

We have adopted the fragment-based algorithm in LigBuilder. The basic idea embedded in our approach is that organic compounds can be dissected into some elementary fragments. If we focus on constructing the molecules that might be valuable for a drug design purpose, the number of such elementary fragments is, fortunately, rather limited. Actually we have used a small building-block library that only contains approximately 60 fragments. If the user has some special requests, he is allowed to extend the building block library by adding new fragments. This can be easily done because that library is organized in a simple and open form.

A special note should be addressed here on how LigBuilder handles ring systems. The reader may have noticed that most common rings observed in drug molecules have been already included in the building-block library. Because mutation operation is implemented in our program, each fragment in the building-block library actually serves as a template rather than a simple structure (see Figure 20). This method is especially useful for generating hetero-substituted rings because with this strategy we do not have to include all the possible hetero-substituted rings in the building-block library (which might be as many as hundreds!). Besides using pre-defined ring templates, LigBuilder is also capable to generate rings efficiently with the linking operation. This is a supplementary way to add structural diversity to the resultant molecules.

Using fragmental building blocks reduces the combinatorial problem largely during the construction process. Another potential advantage concerns the assessment of the synthetic accessibility of the generated molecules. The connection of fragments during a construction process simulates the formation of bonds in a chemical reaction. We encourage the user to incorporate his chemical knowledge by choosing the growing sites on the building block fragments.

Growing strategy and linking strategy

Two major strategies for constructing molecules, i.e. growing strategy and linking strategy, exist for the current fragment-based approaches. The first one starts from a small chemical moiety and then adds fragments to build the molecule step by step. The alternative one places several fragments independently and then searches for a suitable framework that connects all fragments into one molecule.

Both strategies have advantages and disadvantages. The advantage of growing strategy is that chemical knowledge can be easily introduced by choosing proper sites on the seed to add fragments. Therefore, synthetically accessible structures are more likely to be obtained with this strategy. However, growing strategy may run into difficulties if the seed is too small compared to the binding pocket. Due to the combinatorial nature of the building-up process, no algorithm can explore the solution space completely. Therefore in such a situation, it will be very lucky if growing strategy can suggest a molecule which fits to each part of the binding pocket.

The advantage of linking strategy is that you can maximize the interactions between the ligand and the protein at the very beginning by placing proper chemical fragments at the optimal position. However, linking different fragments together is not easy since the position and the orientation of fragments must be maintained while the linker must be chemically feasible at the same time.

However, if one realizes that drug design and discovery is typically an iterative process rather than a single run, he will not be confused by choosing the growing strategy or the linking strategy. Actually these two strategies are suitable for different phases during a drug design process and that is why we have implemented both of them in our program.

The growing strategy is more suitable for lead optimization. In this case, a known compound has exhibited promising binding affinity to the target and now the task is to improve the binding affinity to the nano-mole level so that it is worth further trials. The most common strategy for this job is to make derivative compounds while keeping the framework basically unchanged. By using GROW, one can take the framework of the known lead compound as the seed structure and let the program to build derivatives. Since the framework has occupied the major part of the binding pocket, the problem of insufficient sampling will be much less severe. If the synthetic feasibility is also properly considered, this process can be considered as a virtual combinatorial chemistry within the structural constraints of the biological target.

While the growing strategy is more suitable for lead optimization, the linking strategy is more suitable for suggesting ligands from scratch, i.e. lead discovery. With the help of POCKET, the user can figure out the key interaction sites within the binding pocket. Such information is so straightforward that the user can easily propose certain chemical groups to fit those interaction sites. Then LINK will turn the idea into real molecules. Another possible way to suggest a lead compound is to utilize the pharmacophore model derived by POCKET. Such model can be used as the structural query to perform a 3D database searching. The lead compound discovered in either way can be fed into GROW further to be optimized. Therefore, LigBuilder is a multiple-purposed program that can aid the whole process of structure-based drug design.

Better scoring method

An automatic drug design program like LigBuilder needs to screen a large number of molecules to select out the successful candidates. Therefore, choosing a good scoring method is crucial for this process. Traditionally, drug design programs ”borrow” force field energies to rank the generated molecules. However, this idea does not always work since a force field energy could be related to enthalpy but generally it is quite different from binding free energy. So far there is no force field parameterized specifically to reproduce the interactions between organic molecules with their macromolecular receptors. Pioneered by the LUDI program,[22-25] empirical scoring functions have provided a way to estimate binding free energy. The average error of using such a scoring function could be between 1~2 kcal·mol^{-1}. Although this may not be a very accurate estimation, it has already been a considerable progress for drug design programs. As mentioned in the *Methods* section, we have developed a new empirical scoring function, SCORE, which is at least comparable to other similar approaches. This scoring function has been applied to our program for estimating the binding free energies of the generated molecules and, as we have demonstrated in the *Validation* section, it works well.

One should never forget, however, that there is still a long road between the discovery of a tightly-bound ligand for a target protein and the commercial availability of a drug. A successful lead compound may be rejected later in the clinical trials simply because it is too toxic, too rapidly cleared, too quickly metabolized, or unable to reach the target enzyme in sufficient concentration. Therefore, if one can predict these properties as early as possible, the whole drug discovery procedure could be more efficient. Unfortunately, to predict how a new compound will affect the delicate balances of metabolic, transport, and signaling pathways in the human body is simply impossible at this time. The state-of-the-art approach is to use knowledge-based rules to eliminate the ”obvious” outliers. As a meaningful attempt, we have implemented such rules in LigBuilder to evaluate the bioavailabilies of the generated molecules. Another way to incorporate chemical knowledge into LigBuilder is using the forbidden substructure library. The user is encouraged to add any undesired substructure, which proves to be either toxic or unstable, to that library. The program will take care to reject these structures if they do occur during the ligand construction process.

A practical drug design program

A program will be useless if nobody is willing to use it. Therefore, we have paid special attention to making a user-friendly program. In fact, LigBuilder is very easy to use. All the user needs to do is to prepare the inputs, set a minimal number of parameters in an index file and run the program. We have adopted popular file formats to represent molecules, i.e. PDB format for the protein and SYBYL MOL2 format for the ligands. Therefore, the user will not have problem in preparing the inputs or processing the outputs with his favorite molecular modeling program. We also allow much flexibility in using LigBuilder. For example, the building block library and the forbidden substructure library are all stored in an open manner. The user can view and edit them easily to control the ligand construction process.

When we develop LigBuilder, another thing on the top of our mind is the desire for a unified, extensible system for structure-based drug design. This is reflected in many aspects of our program. First, we have designed LigBuilder in a modularized manner. Each module of the program is only responsible for a certain task of structure-based drug design and they are highly independent to each other. Therefore, if a module needs to be updated to keep up with the latest science, we do not have to rewrite the other parts of the program. And, if a new function is desired, we can simply insert a new module correspondingly. For example, some users of LigBuilder have suggested us to introduce ”similar design” method into LigBuilder (this is about designing mimics for a given set of compounds with known bioassay data). Within the framework of LigBuilder, we only need to add a module, something like POCKET, to analyze the superimposed input molecules, find out the key features, and define the grids outside and inside the molecular aggregation. Then, GROW or LINK can be used to build up molecules based on such information. Secondly, LigBuilder is written in C++ language. By adopting the object-oriented programming techniques, it is very natural to define and manipulate objects like atom, bond, molecule, and force field. The source codes are highly re-usable and extensible. This feature enables that new programs, which offer new functions, can be assembled quickly from existing programs. For example, we developed GROW prior to LINK. The development of LINK has been accelerated considerably since approximately 80% of its source codes was inherited from GROW.

LigBuilder has been released to the public (see the *supplementary material available statement*). At the time of writing this paper, we have registered nearly one hundred users all over the world. From the discussion with the users, we have learned a lot of interesting ideas of how to make a better drug design program. Currently we are working on the updated version of LigBuilder.

Limitations

Here we should also remind the reader that there are two limitations rooted in the ligand construction process implemented in LigBuilder. Knowing about these limitations will help you to use the program correctly. The first limitation is the rigid protein approximation. Considering the flexibility of the protein is not easy for structure-based drug design approaches. Some proteins, such as trypsin and thrombin, have fairly rigid binding sites and do not exhibit large conformational changes upon ligand binding. While in other cases, a part of the protein, typically a loop, moves as a consequence of the ligand binding. However, generally this movement is very similar for different ligands. Therefore, using the known 3D structure of a protein-ligand complex to run LigBuilder will reduce the uncertainty to the minimum. Another limitation is that our program does not minimize the molecule after each growing cycle. We believe that performing minimization for the intermediates is questionable. During the ligand construction, the intermediates are usually too small to fill out the binding pocket. Therefore, full minimization of them will probably lead to unexpect drift inside the binding pocket, which betrays the logic of using a pre-docked seed structure. But we do recommend the user, if necessary, minimize the final output ligands within the constraints of the binding pocket. Since almost all commercially available molecular modeling software can do this, we do not have to implement such a function in our program.

Conclusions

We have described a new program, LigBuilder, for structure-based drug design. This program has implemented many state-of-the-art techniques and could be of great interests for drug designers. Using two well-known examples, we have demonstrated that LigBuilder is able to generate chemical structures similar to the known inhibitors. Expanding LigBuilder to a more powerful system for drug design remains as our active research at present.

Supplementary material available statement The LigBuilder program is available by contacting the authors.

Acknowledgments The authors thank the Science and Technology Ministry of China and the National Natural Science Foundation of China for their financial support.

References

1. Protein Data Bank, Brookhaven National Laboratory, USA, http://www.pdb.bnl.gov/.
2. Montgomery, J. A.; Niwas, S.; Rose, J. D.; Secrist III, J. A.; Babu, S.; Bugg, C. E.; Erion, M. D.; Guida, W. C.; Ealick, S. E. *J.Med.Chem.* **1993**, *36*, 55-69.
3. Webber, S. E.; Bleckman, E. M.; Attard, J.; Deal, J. G.; Kathardekar, V.; Welsh, K. M.; Webber, S.; Janson, C. A.; Matthews, D. A.; Smith, W. M.; Freer, S. T.; Jordan, S. R.; Bacquet, R. J.; Howlan, R. F.; Booth, C. L. J.; Ward, R. W.; Hermann, S. M.; White, J.; Morse, C. A.; Hilliard, J. A.; Bartlett, C. A. *J.Med.Chem.* **1993**, *36*, 733-746.
4. Von Itzstein, M.; Wu, W. Y., Kok, G. B.; Pegg, M. S.; Dyason, J. C.; Jin, B.; Pham, T. V.; Symthe, M. L.; White, H. F.; Oliver, S. W.; Colman, P. M.; Varghese, J. N.; Ryan, D. M.; Woods, J. M.; Bethell, R. C.; Hotham, C. J.; Cameron, J. M.; Penn, C. R., *Nature*, **1993**, *263*, 418-423.
5. Greer, J.; Erickson, J. W.; Baldwin, J. J.; Varney, M. D. *J.Med.Chem.* **1994**, *37*, 1035-1054.
6. Lam, P. Y. S.; Jadhav, P. K.; Eyermann, C. J.; Hodge, C. N.; Ru, Y.; Bachelar, L. T.; Meek, J. L.; Otto, M. J.; Rayner, M. M.; Wong, Y. N.; Chang, C. H.; Weber, P. C.; Jackson, D. A.; Sharpe, T. R.; Erickson, S. E. *Science* **1994**, *263*, 380-384.
7. Hilpert, K.; Ackermann, J.; Banner, D. W.; Gast, A.; Gubernator, K.; Hadvary, P.; Labler, L.; Muller, K.; Schmid, G.; Tschopp, T. B.; Van de Waterbeemd, H. *J.Med.Chem.* **1994**, *37*, 3889-3901.
8. Edwards, P. D.; Andisik, D. W.; Strimpler, A. M.; Gomes, B.; Tuthill, P. A. *J.Med.Chem.* **1996**, *39*, 1112-1124.
9. Anonymous, *Drugs, News, & Perspectives,* **1995**, *8*, 237.
10. Martin, Y. C. *J.Med.Chem.* **1992**, *35*, 2145-2154.
11. Kuntz, I. D.; Meng, E. C.; Shoichet, B. K. *Acc.Chem.Res.* **1994**, *27*, 117-123.
12. Verlinde, C. L. M. J.; Hol, W. G. J. *Structure*, **1994**, *2*, 577-587.

13. Lewis, R. A.; Leach, A. R. *J.Comput.-Aided Mol.Des.* **1994**, *8*, 467-475.
14. Bohm, H. J. *Curr.Opin.Biotech.* **1996**, *7*, 433-436.
15. Kuntz, I. D.; Blaney, J. M.; Oatley, S. J. *J.Mol.Biol.* **1982**, *161*, 269-288.
16. Meng, E. C.; Schoichet, B. K.; Kuntz, I. D. *J. Comput. Chem.* **1992**, *13*, 505-524.
17. Schoichet, B. K.; Kuntz, I. D. *Protein Eng.* **1993**, *6*, 723-732.
18. Meng, E. C.; Kuntz, I. D.; Abraham, D. J. *J.Comput.-Aided Mol.Des.* **1994**, *8*, 299-306.
19. Gschwend, D. A.; Good, A. C.; Kuntz, I. D. *J.Mol.Recog.* **1996**, *9*, 175-186.
20. Goodford, P. J. *J.Med.Chem.* **1985**, *28*, 849-857.
21. Moon, J. B.; Howe, J. W. *Proteins.* **1991**, *11*, 314-328.
22. Bohm, H. J. *J.Comput.-Aided. Mol.Des.* **1992**, *6*, 61-78.
23. Bohm, H. J. *J.Comput.-Aided Mol.Des.* **1992**, *6*, 593-606.
24. Bohm, H. J. *J.Comput.-Aided Mol.Des.* **1994**, *8*, 243-256.
25. Bohm, H. J. *J.Comput.-Aided Mol.Des.* **1994**, *8*, 623-632.
26. Cramer, R. D.; DePriest, S. Implemented in the SYBYL program, 1996, Tripos Associates, St.Louis, MO, USA.
27. Gillet, V. J.; Newell, W.; Mata, P. *J.Chem.Inf.Comput.Sci.* **1994**, *34*, 207-217.
28. Clark, D. E.; Frenkel, D.; Levy, S. A. *J.Comput.-Aided Mol.Des.* **1995**, *9*, 13-32.
29. Waszkowycz, B.; Clark, D. E.; Frenkel, D. *J.Med.Chem.* **1994**, *37*, 3994-4002.
30. Westhead, D.R.; Clark, D.E.; Frenkel, D. *J.Comput.-Aided Mol.Des.* **1995**, *9*, 139-148.
31. Frenkel, D.; Clark, D.E.; Li, J. *J.Comput.-Aided Mol.Des.* **1995**, *9*, 213-225.
32. Clark, D. E.; Murray, C. W. *J.Chem.Inf.Comput.Sci.* **1995**, *35*, 914-923.
33. Murray, C. W.; Clark, D. E.; Byrne, D. G. *J.Comput.-Aided Mol.Des.* **1995**, *9*, 381-395.
34. Judson, R. *Reviews in Computational Chemistry*; Lipkowitz, K. B. and Boyd, D. B. Eds. VCH Publishers: New York, 1997, Vol.10, pp 1-73.
35. Implemented in the SYBYL program, 1996, Tripos Associates, St.Louis, MO, USA.
36. Wang, R.; Liu, L.; Lai, L.; Tang, Y. *J.Mol.Modeling.* **1998**, *4*, 379-394.
37. Lipinski, C. A.; Lombardo, F.; Dominy, B. W.; Feeney, P. J. *Adv. Drug Delivery Rev.* **1997**, *23*, 3-25.
38. Ajay; Walters, W. P.; Murcko, M. A. *J.Med.Chem.* **1998**, *41*, 3314-3324.
39. Sadowski, J.; Kubinyi, H. *J.Med.Chem*. **1998**, *41*, 3325-3329.
40. Wang, R.; Gao, Y.; Lai, L. *Perspectives in Drug Discovery & Design* **2000**, *19*, 47-66.
41. Willett, P. *Concepts and Applications of Molecular Similarity*; Johnson, M. A., Maggiora, G. M. Eds. Wiley-Interscience: New York, 1990, pp 43-64.
42. Banner, D. W.; Hadvary, P., *J.Biol.Chem*. **1991**, *266*, 20085-20093.
43. Kuyper, L. F. *Computer-Aided Drug Design*; Perun, T. J.; Propst, C. L. Eds. Marcel Dekker: New York, 1989, pp.327-369.
44. Nishibata, Y.; Itai, A. *J.Med.Chem.* **1993**, *36*, 2921-2928.
45. Rotstein, S. H.; Murcko, M. A. *J.Comput.-Aided Mol.Des.* **1993**, *7*, 23-43.
46. Bohacek, R. S.; McMartin, C. *J.Am.Chem.Soc.* **1994**, *116*, 5560-5571.
47. Tschinke, V.; Cohen, N. C. *J.Med.Chem.* **1993**, *36*, 3863-3870.
48. Rotstein, S. H.; Murcko, M. A. *J.Med.Chem.* **1993**, *36*, 1700-1710.
49. Luo, Z.; Wang, R.; Lai, L. *J.Chem.Inf.Comput.Sci.* **1996**, *36*, 1187-1194.