Journal of Computer-Aided Molecular Design, 9 (1995) 139–148
ESCOM

J-CAMD 276

PRO_LIGAND: An approach to de novo molecular design. 3. A genetic algorithm for structure refinement

David R. Westhead, David E. Clark, David Frenkel, Jin Li, Christopher W. Murray, Barry Robson* and Bohdan Waszkowycz

Proteus Molecular Design Ltd., Proteus House, Lyme Green Business Park, Macclesfield, Cheshire SK11 0JL, U.K.

Received 4 July 1994
Accepted 28 August 1994

Keywords: Drug design; De novo design; Genetic algorithms

Summary

Recently, the development of computer programs which permit the de novo design of molecular structures satisfying a set of steric and chemical constraints has become a burgeoning area of research and many operational systems have been reported in the literature. Experience with PRO_LIGAND – the de novo design methodology embodied in our in-house molecular design and simulation system PROMETHEUS – has suggested that the addition of a genetic algorithm (GA) structure refinement procedure can 'add value' to an already useful tool. Starting with the set of designed molecules as an initial population, the GA can combine features from both high- and low-scoring structures and, over a number of generations, produce individuals of better score than any of the starting structures. This paper describes how we have implemented such a procedure and demonstrates its efficacy in improving two sets of molecules generated by different de novo design projects.

Introduction

The application of computational techniques in the process of rational molecular design is now pervasive [1,2] and new or improved methodologies are reported regularly. One of the most recent developments in the Computer-Aided Molecular Design (CAMD) field is the emergence of computer programs for de novo molecular design. The purpose of such programs is to automate the generation of novel, yet chemically reasonable structures that meet a particular set of steric and/or chemical constraints. The constraints may be derived from a knowledge of a receptor active site or from a pharmacophore representing the features of a series of known active molecules.

While such de novo design can be carried out manually by an experienced molecular modeller painstakingly piecing together suitable molecules at a graphics terminal, there are several advantages to be gained by automating the process. For instance, automated de novo design is more *efficient* – especially with ever-improving hardware platforms and software algorithms. Also, it is more *systematic* – many more of the combinatorial possibilities can be explored. Finally, it is more *objective* – the computer has no inherent biases about what the right answers are going to be. This also produces more diversity in the structures generated.

These persuasive arguments for automation have resulted in a flurry of activity from which a variety of de novo design programs have resulted [3–22]. We have recently described PRO_LIGAND, our in-house approach for computational de novo design, and shown its efficacy in the generation of novel structures satisfying constraints derived from a variety of targets [23,24]. For the purposes of this paper, a brief outline of the methodology is given below.

PRO_LIGAND

The first module of PRO_LIGAND to operate in the design process is **Design-base Generation**. The purpose of this module is to take the input information (i.e., molecular structure and command files) and to generate the features important for the design process, e.g., the atoms comprising the active site of a receptor or a pharmacophore from a set of active structures. These features are termed the *design base* and are translated into a consistent

*To whom correspondence should be addressed.

internal format (the Generalised Molecular Structure Descriptor (GMSD) file), to be passed on to the subsequent modules.

Next, the **Design-model Generation** module employs a rule-based algorithm to construct a *design model* whose physico-chemical characteristics are either *similar* or *complementary* to those of the design base according to the user's choice. The design model is a template that describes the idealised steric and hydrogen-bonding features of the chemical structures to be designed. These features are represented as *interaction sites* [9,10,25]. Hydrogen-bond acceptors and donors are represented by **A-Y** and **D-X** vectors, respectively, while lipophilic regions are characterised by L or R sites according to whether the region is aliphatic or aromatic in nature.

Once the design model has been produced, the next module to be invoked is **Structure Generation** which produces molecular structures consistent with the design model by assembling small 3D molecular fragments from preconstructed libraries. These library fragments are labelled to indicate the type of interaction site they may match and a rapid graph-theoretical algorithm is used to seek fits of the fragments to the design model. The fitting procedure is subject to user-specified tolerances and also insists that bad inter- or intramolecular van der Waals clashes are avoided. A great variety of modes of fragment assembly are available to the user. At one end of the spectrum are strategies with an exhaustive placing phase, followed by a separate bridging phase. At the other extreme, it is possible to invoke a sequential place-and-join mode where each fragment placed is constrained to join to the growing ligand. Between these two an almost limitless variety of 'hybrid' strategies is permitted, due to the generalised nature of the module. Once each structure is completed, it is assigned a score based upon how many of the design-model features it has succeeded in fulfilling and upon certain of its intrinsic characteristics, such as the number of rings or asymmetric carbon atoms.

Utility of a Structure Refinement procedure

The possible solution space for a given PRO_LIGAND design problem is extremely large, due to the combinatorial nature of fragment assembly. It is thus very difficult to find the optimal solutions within such a space in a reasonable amount of time. In PRO_LIGAND, we have adopted a depth-first strategy to find solutions, that is, the first fit for any fragment is immediately accepted and the algorithm then proceeds to search for a fit for the next randomly selected fragment. Such a strategy invariably results in the rapid generation of a set of solutions of differing qualities.

Thus, in our experience with PRO_LIGAND, what we frequently observe at the end of a run of the Structure Generation module is a set of structures whose scores range from excellent to poor. Nonetheless, the low-scoring structures are not always completely devoid of useful information or structural novelty. It would be interesting to find a method that could draw on information from the *ensemble* of generated structures to suggest further novel entities satisfying the design model. In this manner, value can be added to the results already gained. One such approach, based on the generation of a molecular lattice, has been described by Lewis et al. [12] and used in practice by Rotstein and Murcko [13].

The nature of the problem suggested to us that a *genetic algorithm* approach would be applicable. As a directed global optimisation technique, the GA should be able to make the best use of the information gleaned by the Structure Generation run to generate improved structures. Before detailing our method, some background relating to the theory and application of genetic algorithms will be presented.

Genetic algorithms

During the late 1950s and early 1960s, several workers experimented with the application of evolutionary principles to problem solving. One result of this research was a class of non-deterministic search algorithms called genetic algorithms (GAs), originally developed by Holland [26,27]. In recent years, there has been an upsurge of interest in GAs and they have been applied with success in many varied fields. In what follows, the basic concepts and techniques of GAs, as applied to this work, will be introduced. For a more complete treatment of the subject, the reader is referred to the works of Goldberg [28], Davis [29] and reviews by Lucasius and Kateman [30] and South et al. [31].

Genetic algorithms are based on the theory of Darwinian evolution and, as such, seek to employ processes based on Natural Selection to find (near) optimal solutions for complex problems. A GA works with a *population* of individuals, each of which represents an attempt at a solution to the problem in hand. These population members interact with one another via a set of *genetic operators* which seek to guide the population towards a (near) optimal solution. The proximity of each member of the population to the desired solution is measured by a *fitness function* which assigns a *fitness* to each individual.

Starting with an initial population, a GA works by selecting a pair of individuals (*parents*) for breeding. The *selection* operator chooses parents in a manner which is weighted according to their fitness values; the fitter an individual, the more chance it has of being selected. If the *crossover probability* (Pcross) exceeds a randomly generated number, the selected pair of parents are then bred using the *crossover* operator. This operator divides both parents at a specified point and then joins the pieces together to form a pair of children different from the

parents but maintaining some of their characteristics. A further operator, *mutation*, may also be applied during the breeding process if the *mutation probability* (Pmutate) exceeds another random number. By analogy with nature, mutation is generally invoked with a low frequency and only occasionally produces a beneficial result. However, it does serve to introduce some random variation into the children which helps to maintain the diversity of the population and thus reduces the likelihood of the GA converging on a sub-optimal minimum. The children thus generated then replace the least-fit members of the current population and the breeding process is repeated with a new pair of parents. Over a series of generations, the average fitness of the population is observed to increase and the population eventually converges on a (hopefully good) solution to the problem.

This is the basic modus operandi of a GA. As can be seen, the fundamental concepts and operations are not complicated and yet, GAs have proved to be very effective as search/optimisation algorithms, particularly for complex, large-scale problems [30]. In the last few years, GAs have been applied to several areas of computational chemistry and biology such as protein modelling [32–37], molecular docking [38], conformational search of both small and macromolecules [39–45], energy minimisation [46,47], protein structure comparison [48], molecular fitting and pharmacophore elucidation [49], QSAR and receptor modelling [50–52], and de novo drug design [53–55]. Here, we present a further application of GAs in the area of de novo design, specifically for the refinement of a set of solutions from a de novo design program.

A genetic algorithm for structure refinement

In the **Structure Refinement** module of PRO_LIGAND, our intention is to 'mix and match' the structures produced during Structure Generation by taking them as the initial population for a GA and then breeding successive generations of structures to increase the average fitness. Following the work of Unger and Moult [36], we have decided not to use a binary encoding of the population but rather to carry out the genetic operations upon the chemical structures themselves. It has been suggested that there is no inherent merit in binary encoding [56] and certainly, for our application, it is more straightforward and intuitive to operate with the structures (i.e., the *phenotype*) directly.

A flow chart of the algorithm is depicted in Fig. 1 and a full description of its components is given in what follows.

Genetic operators

We have implemented the three basic genetic operators of selection, crossover and mutation as described below.

Selection

Two parent structures are selected for mating using 'roulette-wheel' selection as described by Goldberg [28]. The use of the roulette wheel ensures that parents are selected with a probability proportional to their fitness scores. The fitness function used to score the members of the population is described in a later section.

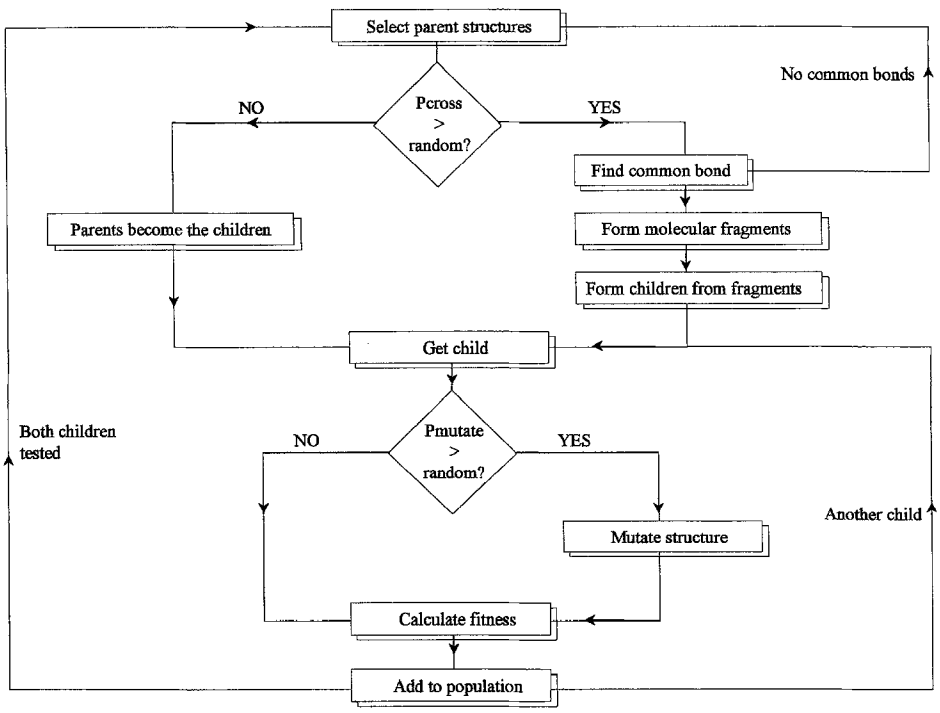

Fig. 1. Flow diagram of the Structure Refinement algorithm.

Crossover

Having selected two parent structures, the GA then proceeds to seek a suitable point on each structure where crossover may occur. For our purposes, this is defined as any single bond (which must be non-terminal and acyclic) which is 'common' to both structures, i.e., lies within the same space on the design model. We note in passing that this is similar to the approach employed by Ho and Marshall in their SPLICE program [19]. The tolerance within which two bonds are deemed to be common is under the control of the user.

Having identified such a common bond, the algorithm then cleaves both the parents at that site to form four molecular fragments. These fragments are then recombined to create two child structures. For each pair of fragments, one is selected at random and this is placed first upon the design model. The second fragment of the pair is then also placed upon the design model in such a way that it also forms a bond to the first fragment. If this process fails at any stage, i.e., if one of the fragments cannot be matched to the design model, then the (potential) child is discarded.

If, as is often the case, several common bonds are identified between two parent structures, the algorithm selects one at random to be the cleavage site.

Mutation

The children thus generated may also undergo mutation. At present, the mutation operator involves the randomisation of one or more torsion angles in the structure. The intent of this operator is to examine whether a given structure can meet a set of interaction sites different from those to which it is currently mapped by altering its conformation. A further mutation operator to mutate the atom types within the structure in accordance with a user-definable rule base is currently being tested.

Fitness function

Having produced two children by means of the three genetic operators, the algorithm must assign them a fitness value. The fitness function employed by the GA to score the members of the population is similar to that employed by the Structure Generation module to score the structures it generates. The fitness of a structure reflects how well it fulfils the constraints and features of the design model and also other intrinsic structural requirements specified by the user. Thus, the fitness is a weighted sum of the number of interaction sites hit, together with terms for the numbers of rings and rotatable bonds.

In more detail, the fitness of a population member may be calculated from the following equation:

$$F = \sum_{1}^{NA} W_A + \sum_{1}^{ND} W_D + \sum_{1}^{NAl} W_{Al} + \sum_{1}^{NAr} W_{Ar} + \sum_{1}^{NRot} W_{Rot} + \sum_{1}^{NRing} W_{Ring}$$

where F is the structure's fitness, NA and ND are the numbers of hydrogen-bond acceptor and donor sites hit by the structure, NAl and NAr are the numbers of lipophilic aliphatic and lipophilic aromatic interaction sites hit by the structure and NRot and NRing are the numbers of rotatable bonds and rings in the structure, respectively. W_A is the contribution to the score for each hit hydrogen-bond acceptor site and the other weights refer to their respective features.

All the weights above may be specified by the user so that those structures which best meet the user's requirements, both in terms of the design-model constraints and intrinsic structural features, will be designated as the most fit. The precise values for each of the weights will obviously vary from one design problem to another, but in general, one would want to weight hydrogen-bonding features more strongly than lipophilic features and to give a small penalty (i.e., negative weight) to each rotatable bond. Rings may be given a positive weight if their presence is desired. In the absence of user-specified values, the program assigns defaults of –0.1 to W_{Rot}, +1.0 to W_A and W_D, +0.25 to W_{Al} and W_{Ar} and 0.0 to W_{Ring}.

The calculation of the fitness value for any structure thus simply involves using the graph theory routines described in Ref. 23 to seek to place the structure on the design model. If a structure cannot be replaced, it is discarded. Once a child has been refitted and assigned a fitness value, it either takes its place in the new population (Generational Replacement mode) or replaces the least fit member of the mating pool (Steady State mode). These two modes of operation of the GA are discussed in the following section.

All fitness values are subject to the linear scaling procedure described by Goldberg [28]. In brief, linear scaling transforms a 'raw' fitness value, F, to a 'scaled' fitness value, F', by means of the following equation:

$$F' = aF + b$$

where the coefficients a and b are chosen, in our implementation, so that the average scaled fitness of the population, F'_{av}, equals the average raw fitness of the population, F_{av}, and the maximum scaled fitness of the population, F'_{max}, is equal to *twice* the average raw fitness of the population. If this latter criterion ever forces the lowest scaled fitness value, F'_{min}, to become negative, the scaling coefficients are instead calculated based upon the equality of the average scaled and raw fitness values and the criterion $F'_{min} = 0.0$.

Early on in a GA run, when the maximum and average fitness values of the population are widely separated, fitness scaling helps to prevent the population being dominated by a strong individual by reducing the fitness value of such a population member. Later on, when the average and maximum fitness values of the population are closer,

fitness scaling helps to maintain competition between near equals by accentuating the differences in their fitness values.

Modes of operation

As suggested in the previous section, we have implemented two different modes of operation for our GA.

Generational Replacement mode

In the Generational Replacement (GR) mode, two populations of structures are maintained, the *old population* and the *new population*. Child structures are bred from the old population and are placed in the new population if their fitness exceeds that of the least fit member of the previous population. When the new population is full, it becomes the old population (generational replacement) and the process is repeated until a maximum number of generations has passed or the population is deemed to have converged. The 'elitist' strategy is also implemented, i.e., the fittest member of the old population is automatically copied into the new one before mating commences [29].

Steady State mode

In Steady State (SS) mode, only one population is maintained. In this instance, if the bred children are fitter than the least fit members of the population, they replace the latter and so become immediately available for mating. This process is continued until a specified maximum number of genetic operations has been performed or until any convergence criteria have been satisfied.

Duplicate checking

In both the above modes, it is desirable to avoid the presence of duplicates in the breeding pool since this reduces the amount of genetic diversity present and often results in premature convergence. To permit a rapid check for population members having the same 2D structure, each structure is assigned a unique identifier calculated by a procedure due to Herndon [57], which follows a partitioning of the atoms by means of a modified Morgan algorithm [58,59]. Before adding a child to a population, its unique identifier is checked against the stored identifiers of the existing population members. If the child is discovered to be a duplicate, it is discarded. It should be noted that, as this check is based upon connectivity and atom-type information only, two different conformations of the same structure will be considered as duplicates. This has the consequence that in Steady State mode, a child structure produced solely by torsional mutation of a parent will be rejected as a duplicate, even if it is scoring higher. However, for our application, the crossover operator is the crucial component for structure generation and so we feel that the limitation mentioned above is not serious and should be lessened considerably if the atom-type mutation operator is also invoked. Furthermore, some kind of 3D duplicate check would probably involve rms comparisons between existing and potential population members which would be far more computationally expensive than the approach currently adopted.

Convergence criteria

At specified intervals, the population is checked to see if the specified convergence criteria have been met. We have implemented two such measures to detect both *inter-population* and *intra-population* convergence.

The inter-population convergence criterion monitors how much the average fitness of the population has changed since the last check. If it has not changed by a user-defined amount then the GA is deemed to have converged. The intra-population convergence criterion monitors the percentage difference between the fittest and least fit members of the population. If this falls below a specified threshold, then the population is judged to have converged. By means of these two measures, the user has a flexible way of monitoring the progress of the algorithm and controlling its termination point.

Results

Having described the implementation of the GA for structure refinement, we now proceed to illustrate its utility by presenting some test cases.

Similar designs to distamycin

Distamycin is a naturally occurring antibiotic which binds to the minor groove of DNA. As part of an ongoing project for the study of such DNA-binding drugs, we have used PRO_LIGAND to generate structures similar to distamycin. In particular, the object was to seek novel designs to replace the pyrrole–carboxamide ring system (Fig. 2). From one such run of the Structure Generation module, 99 structures were taken as the initial population for runs of the Structure Refinement module in Steady State mode. Figure 3 shows the PRO_LIGAND

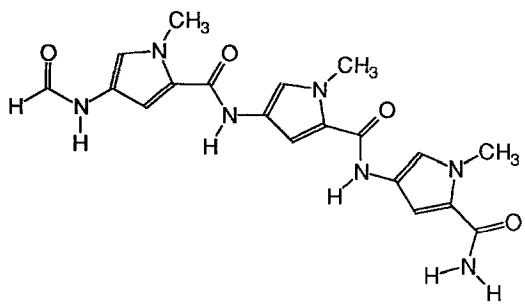

Fig. 2. The pyrrole–carboxamide ring system of distamycin.

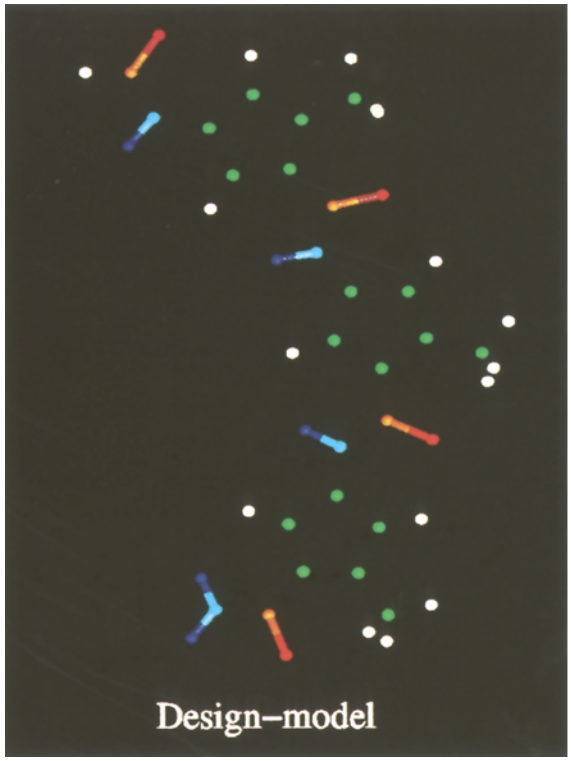

Fig. 3. PRO_LIGAND design model for similar design to the pyrrole–carboxamide ring system.

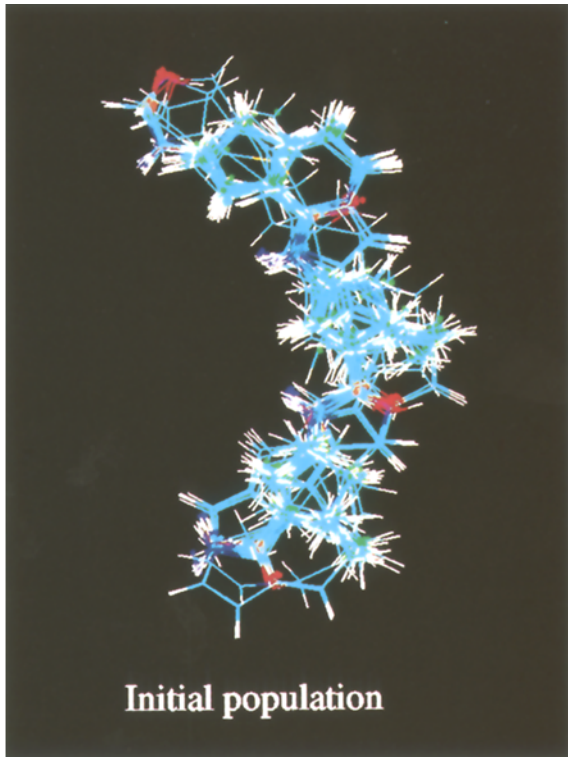

Fig. 4. Overlay of structures generated by the Structure Generation module.

design model for structure generation and structure refinement and Fig. 4 shows all the structures of the initial population overlaid. As can be seen, the structures more than adequately span the space of the design model and give the GA plenty of information with which to work. Since the GA has a stochastic element, five runs were performed with different seeds for the random number generator.

The parameter values employed for this series of experiments are presented in Table 1. As can be seen, the probability of crossover occurring is set much higher than that for mutation – this is customary in GA applications.

TABLE 1
GA PARAMETERS FOR REFINEMENT OF DISTAMYCIN MIMICS

Parameter	Value
Population size	99
Crossover probability	0.95
Mutation probability	0.05
Maximum genetic operations (SS mode)	7500
Operations increment (SS mode)	500
Maximum generations (GR mode)	12
Intra-population convergence (%)	3.0
Inter-population convergence (%)	1.0
H-bond donor score	1.0
H-bond acceptor score	1.0
Lipophilic score	0.5
Ring score	0.5
Common-bond tolerance (Å)	1.0

The maximum number of genetic operations is set to 7500 and the 'operations increment' command specifies that the population is to be examined for convergence and reported on following every 500 operations. (Each invocation of selection, crossover or mutation is deemed to be a genetic operation.) The population is considered to have converged if its fittest member is 1.0% or less fitter than the least fit *and* if the difference in mean fitness between the current population and the last check is 3.0% or less. The various scores shown in the table describe the weights given in the fitness function to various structural attributes of the designed molecules. Finally, two bonds in the parent structures (A-B and A'-B') are considered 'common' if the distances A-A' and B-B' are both 1.0 Å or less.

A further set of five runs was performed on the same set of structures using the Generational Replacement mode. The run parameters were identical, except that the 'maximum genetic operations' parameter was replaced by its

TABLE 2
RESULTS OF STRUCTURE REFINEMENT OF DISTAMYCIN MIMICS

	Initial value	Final value (SS mode)	Final value (GR mode)
Mean fitness	4.83	10.93	10.82
Minimum fitness	2.50	10.20	8.50
Maximum fitness	10.50	12.20	13.30
CPU time (s)	0.00	550.50	540.40

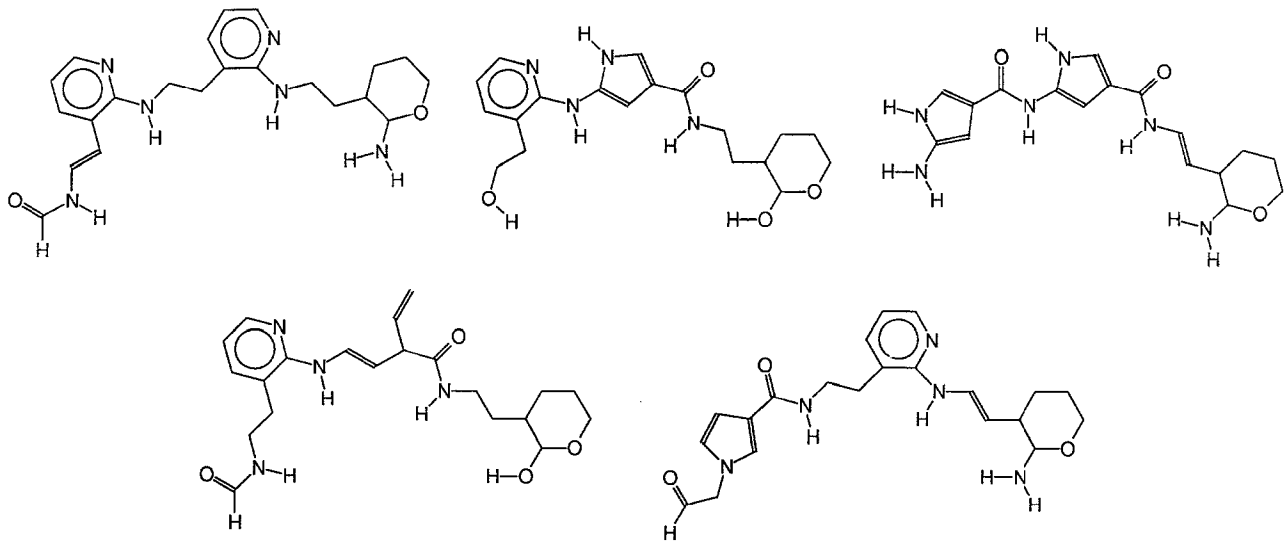

Fig. 5. Mimics of the pyrrole–carboxamide ring system produced by the Structure Refinement module.

GR counterpart 'maximum number of generations', which in this case was set to 12. The 'operations increment' parameter has no counterpart in GR; the convergence tests and reporting are carried out after each generation.

It should be noted that no particular effort was expended to optimise the GA parameters used in the experiments reported in this work. In particular, the maximum numbers of operations and generations were simply chosen for the fact that they were found to be sufficient to allow satisfactory improvement in the population rather than convergence.

The fitness statistics for the population at the beginning of the runs are shown in Table 2, together with the means of the values taken at the end of the five runs for both the SS and GR modes. Also shown are the mean CPU times for the runs as measured on an entry-level SGI Indigo workstation. In this and the following experiment, the CPU timings include the time required to read

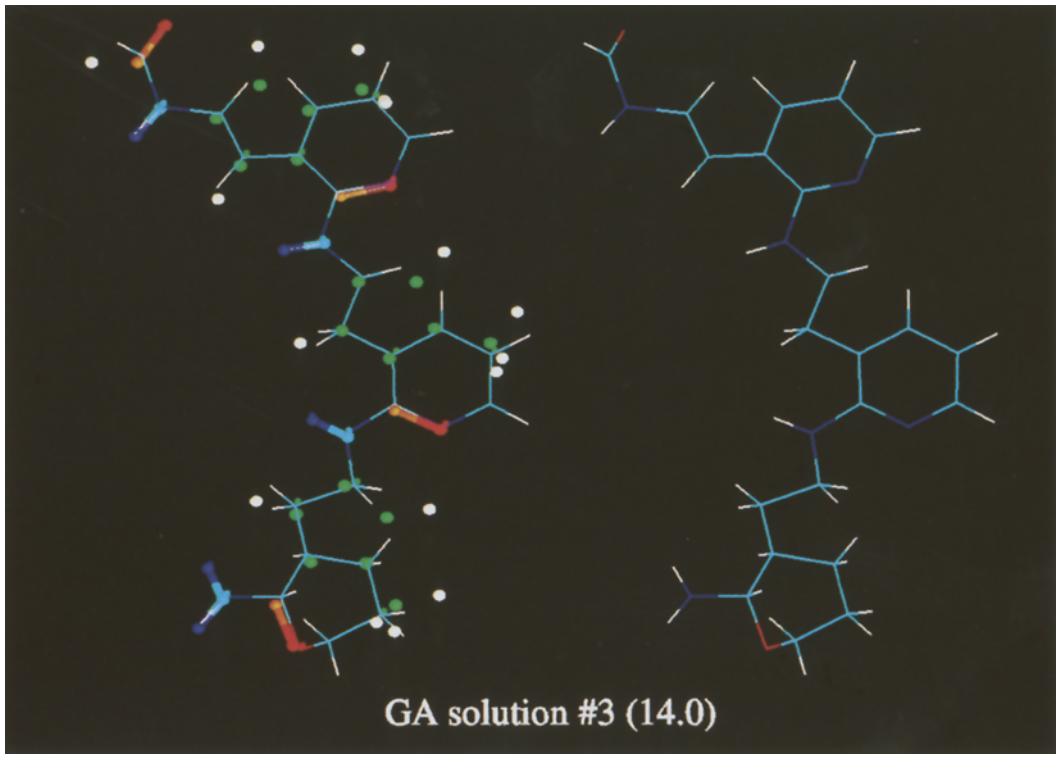

Fig. 6. Top-scoring mimic superimposed on the design model (left) and on its own. Its fitness (14.0) is equal to that of the target pyrrole–carboxamide ring system.

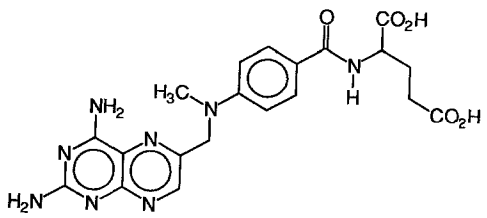

Fig. 7. The structure of methotrexate.

TABLE 4
RESULTS OF STRUCTURE REFINEMENT OF METHOTREXATE MIMICS

	Initial value	Final value (SS mode)	Final value (GR mode)
Mean fitness	6.65	13.16	13.63
Minimum fitness	1.50	11.20	9.00
Maximum fitness	14.50	16.70	17.10
CPU time (s)	0.00	517.40	716.80

in the initial population of structures from disk. Clearly, the GA has succeeded in significantly improving the scores of the initial populations in both SS and GR modes and in doing so in a very reasonable amount of CPU time. Note, in particular, that the final minimum fitness for the Steady State mode is almost as great as the initial maximum. It is also interesting to observe that, while the mean fitness values for the SS and GR modes are almost identical, the difference between the maximum and minimum values is significantly larger for the GR results. This indicates that the GR population is less converged and is most likely a consequence of the separation of the new and old populations in the GR mode. By contrast, in the SS mode, the children enter the breeding pool straightaway and so their new information can be immediately shared with the rest of the population.

Some of the top scoring structures from the runs described above are shown in Fig. 5 and the best is shown superimposed upon the design model in Fig. 6. As can be seen, the structures produced by the GA show interesting variations on the original distamycin structure while maintaining many of its features which are crucial for binding, including the overall molecular shape and the pattern of inward-pointing hydrogen-bond-donating groups. A less desirable feature of the set is the presence of the unstable cyclic hemiacetal or hemiaminal functionality which has been carried through from the initial population generated by the Structure Generation module. This indicates the need for more sophisticated checks during the fragment-joining process and these are currently being implemented. In this particular instance, the problem is not too serious since by simply replacing the oxygen atom of the cyclic ether by an sp^3 carbon atom, the problem of instability can be removed without affecting the features that are important for binding.

TABLE 3
GA PARAMETERS FOR REFINEMENT OF METHOTREXATE MIMICS

Parameter	Value
Population size	125
Crossover probability	0.95
Mutation probability	0.05
Maximum genetic operations (SS mode)	7500
Operations increment (SS mode)	500
Maximum generations (GR mode)	12
Intra-population convergence (%)	3.0
Inter-population convergence (%)	1.0
H-bond donor score	1.0
H-bond acceptor score	1.0
Lipophilic score	0.5
Common-bond tolerance (Å)	1.0

Similar designs to methotrexate

Elsewhere [23], we have reported the application of PRO_LIGAND to the design of mimics of the anti-cancer drug methotrexate (Fig. 7). As a further test of the GA, we have taken 125 structures from various PRO_LIGAND runs on this problem and used them as the initial population for refinement by the GA. The structures were particularly chosen so that, while some of them contained either the α-γ dicarboxylate chain (Fig. 8A) *or* the pteridine moiety (Fig. 8B), none contained both. The test for the GA was to combine these two types of structure to produce true methotrexate mimics. It is worth noting that a detailed modelling study of this system would need to take into account the protonation of the pteridine ring which has not been considered in this simple test case.

As in the distamycin example, five runs of both the Steady State and Generational Replacement modes were run. The parameters employed are given in Table 3 and are similar to those of the previous run. The results are presented in Table 4. As in the previous example, it can be seen that the GA has successfully improved the average fitness of the population and as the structures in Fig. 9 demonstrate, combined the two types of structure mentioned above. It is noticeable in this example that the GR mode has consumed more CPU time than the SS mode. This could be the consequence of a weakness of the GR mode which we have observed in our application and

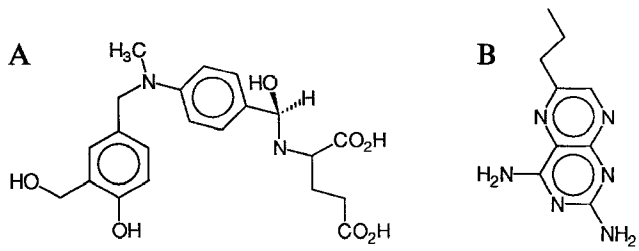

Fig. 8. Partial solutions generated by the Structure Generation module.

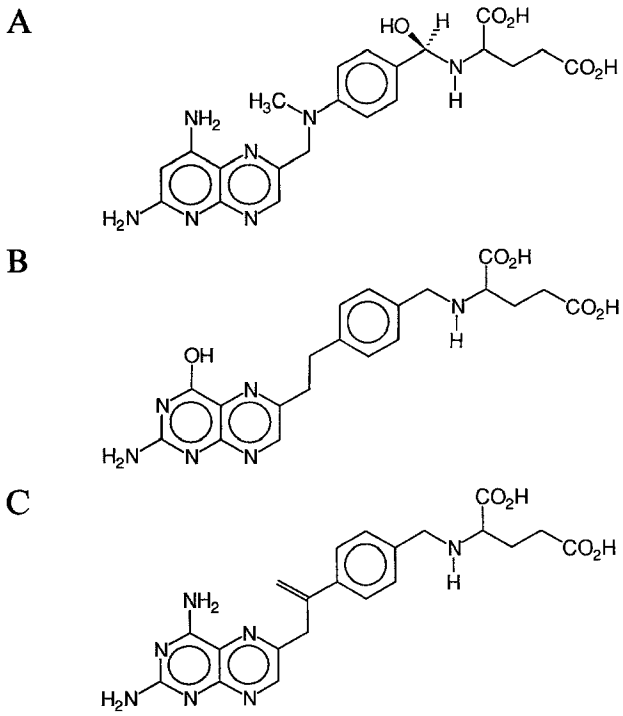

Fig. 9. Methotrexate mimics generated by the Structure Refinement module.

which we have termed 'bottlenecking'. What happens is that after a while, the GA begins to struggle to fill up the new population (remembering that duplicates are not permitted). Thus, for every new structure found, many duplicates must be generated and discarded and this can slow down the operation of the GR mode. With the SS mode, as mentioned before, the children are added immediately to the breeding pool and their new features and improved fitness can therefore be taken advantage of immediately. In addition, the SS mode's progress is unhindered by the need to fill up a new population – it simply counts the number of genetic operations performed as a measure of its progress.

Looking more closely at the structures produced by the GA, it can be seen that they tend to fall into five classes:

(1) Reproduction of methotrexate itself.
(2) Variation within the pteridine ring system (e.g. Fig. 9A).
(3) Variation on substituents of the pteridine ring system (e.g. Fig. 9B).
(4) Variation on the bridge between the pteridine ring system and the *para*-amino benzoic acid unit (e.g. Fig. 9C).
(5) Reduction of the backbone amide (e.g. Figs. 9A and B).

While these structures may not be tremendously exciting in terms of their therapeutic potential, they do illustrate the ability of the GA to build a variety of structures on a given 'theme' from a set of initial structures of lower quality. All the structures shown also retain the essential features for methotrexate binding to dihydrofolate reductase described by Kuyper [60].

Discussion

A growing number of de novo design programs are being reported, all of which have the ability to generate a variety of structures to satisfy a particular set of steric and/or chemical constraints. It is likely that the output from any given run will contain structures that satisfy all the constraints well, if not optimally, but also those which are less good and may only meet a few of the imposed criteria.

It is a reasonable hypothesis that the 'ensemble' of structures so generated may be capable of yielding further and better solutions to the design problem – the whole being greater than the sum of its parts. The work of Lewis et al. [12] has already recognised this and produced a method by which the total information contained in the set of generated structures can be exploited to produce new solutions. In a slightly different context, Ho and Marshall [19] have described a method by which a number of 'partial' solutions from a pharmacophoric pattern search of a 3D database can be 'spliced' together to yield novel structures which satisfy all the constraints of the pharmacophore.

The Structure Refinement module of PRO_LIGAND described here represents a further novel approach to the exploitation of the information generated by a de novo design program. The use of a genetic algorithm-based approach means that the final set of structures evolved can be guaranteed to be superior to the initial set, the crossover operator combining beneficial features from fit individuals and the mutation operator occasionally nudging the evolutionary process into new areas with unexpected rewards. Moreover, through the fitness function, the user is able to direct this evolution towards the goal that is desired.

The examples presented in this paper illustrate the effectiveness of this GA-based approach and also its efficiency. As in most GA applications, the rate-limiting step is the evaluation of the fitness function; the actual genetic operations themselves are generally very fast. We also believe that such an approach could be applied to the results of other de novo design programs – all that is required is the formulation of an appropriate fitness function to guide the GA. The function could take many forms, perhaps including terms for molecular mechanics energy, molecular volume, number of atoms and so forth. It is also possible that the results of 3D database searches could be subject to a similar process to generate further solutions. In this instance, the fitness function could simply be a measure of the structure's deviation from the constraints of the search query.

In continuing the development of the structure refinement GA, we hope to investigate further the application of the atom-type mutation operator and also the feasibility of allowing C-H bonds on rings to be cleavage sites

for crossover. This latter feature would permit the GA to create many more novel structures. There would also be merit in incorporating some limited force-field calculation into the fitness evaluation. As currently implemented, the torsional mutation operator is capable of driving the structure into high-energy conformations without penalty.

Conclusions

We have devised and implemented a novel algorithm for the refinement of molecular structures generated by a de novo design program. The method is based upon a genetic algorithm and has been shown to be both effective and efficient at improving structures designed to be mimics of distamycin and methotrexate. We believe that such an approach could form a useful adjunct to other de novo design programs and perhaps also to 3D database searching systems.

References

1 Martin, Y.C., Methods Enzymol., 203 (1991) 587.
2 Dixon, J.S., Trends Biotechnol., 10 (1992) 357.
3 Moon, J.B. and Howe, W.J., Protein Struct. Funct. Genet., 11 (1991) 314.
4 Moon, J.B. and Howe, W.J., In Wermuth, C.G. (Ed.) Trends in QSAR and Molecular Modelling 92 (Proceedings of the 9th European Symposium on Structure–Activity Relationships: QSAR and Molecular Modelling), ESCOM, Leiden, 1993, pp. 11–19.
5 Miranker, A. and Karplus, M., Protein Struct. Funct. Genet., 11 (1991) 29.
6 Caflisch, A., Miranker, A. and Karplus, M., J. Med. Chem., 36 (1993) 2142.
7 Nishibata, Y. and Itai, A., Tetrahedron, 47 (1991) 8985.
8 Nishibata, Y. and Itai, A., J. Med. Chem., 36 (1993) 2921.
9 Böhm, H.-J., J. Comput.-Aided Mol. Design, 6 (1992) 61.
10 Böhm, H.-J., J. Comput.-Aided Mol. Design, 6 (1992) 593.
11 Böhm, H.-J., In Kubinyi, H. (Ed.) 3D QSAR in Drug Design: Theory, Methods and Applications, ESCOM, Leiden, 1993, pp. 386–405.
12 Lewis, R.A., Roe, D.C., Huang, C., Ferrin, T.E., Langridge, R. and Kuntz, I.D., J. Mol. Graph., 10 (1992) 66.
13 Rotstein, S.H. and Murcko, M.A., J. Comput.-Aided Mol. Design, 7 (1993) 23.
14 Rotstein, S.H. and Murcko, M.A., J. Med. Chem., 36 (1993) 1700.
15 Gillet, V.J., Johnson, A.P., Mata, P., Sike, S. and Williams, P., J. Comput.-Aided Mol. Design, 7 (1993) 127.
16 Gillet, V.J., Newell, W., Mata, P., Myatt, G., Sike, S., Zsoldos, Z. and Johnson, A.P., J. Chem. Inf. Comput. Sci., 34 (1994) 207.
17 Pearlman, D.A. and Murcko, M.A., J. Comput. Chem., 14 (1993) 1184.
18 Tschinke, V. and Cohen, N.C., J. Med. Chem., 36 (1993) 3863.
19 Ho, C.W.M. and Marshall, G.R., J. Comput.-Aided Mol. Design, 7 (1993) 623.
20 Leach, A.R. and Lewis, R.A., J. Comput. Chem., 15 (1994) 233.
21 Leach, A.R. and Kilvington, S.R., J. Comput.-Aided Mol. Design, 8 (1994) 283.
22 Eisen, M.B., Wiley, D.C., Karplus, M. and Hubbard, R.E., Protein Struct. Funct. Genet., 19 (1994) 199.
23 Clark, D.E., Frenkel, D., Levy, S.A., Li, J., Murray, C.W., Robson, B., Waszkowycz, B. and Westhead, D.R., J. Comput.-Aided Mol. Design, 9 (1995) 13.
24 Waszkowycz, B., Clark, D.E., Frenkel, D., Li, J., Murray, C.W., Robson, B. and Westhead, D.R., J. Med. Chem., 37 (1994) 3994.
25 Klebe, G., J. Mol. Biol., 237 (1994) 212.
26 Holland, J.H., Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975.
27 Holland, J.H., Sci. Am., 267 (1992) 44.
28 Goldberg, D.E., Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.
29 Davis, L. (Ed.) Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, NY, 1991.
30 Lucasius, C.B. and Kateman, G., Chemometrics Intelligent Lab. Syst., 19 (1993) 1.
31 South, M.C., Wetherill, G.B. and Tham, M.T., J. Appl. Stats., 20 (1993) 153.
32 Judson, R.S., J. Phys. Chem., 96 (1992) 10102.
33 Dandekar, T. and Argos, P., Protein Eng., 5 (1992) 637.
34 Dandekar, T. and Argos, P., J. Mol. Biol., 236 (1994) 844.
35 Sun, S., Protein Sci., 2 (1993) 762.
36 Unger, R. and Moult, J., J. Mol. Biol., 231 (1993) 75.
37 Jones, D.T., Protein Sci., 3 (1994) 567.
38 Judson, R.S., Jaeger, E.P. and Treasurywala, A.M., J. Mol. Struct. (THEOCHEM), 308 (1994) 191.
39 McGarrah, D.B. and Judson, R.S., J. Comput. Chem., 14 (1993) 1385.
40 Judson, R.S., Jaeger, E.P., Treasurywala, A.M. and Peterson, M.L., J. Comput. Chem., 14 (1993) 1407.
41 Clark, D.E., Jones, G., Willett, P., Kenny, P.W. and Glen, R.C., J. Chem. Inf. Comput. Sci., 34 (1994) 197.
42 Blommers, M.J.J., Lucasius, C.B., Kateman, G. and Kaptein, R., Biopolymers, 32 (1992) 45.
43 Judson, R.S., Colvin, M.E., Meza, J.C., Huffer, A. and Gutierrez, D., Int. J. Quantum Chem., 44 (1992) 277.
44 Tufféry, P., Etchebest, C., Hazout, S. and Lavery, R., J. Comput. Chem., 14 (1993) 790.
45 Sanderson, P.N., Glen, R.C., Payne, A.W.R., Hudson, B.D., Heide, C., Tranter, G.E., Doyle, P.D. and Harris, C.J., Int. J. Pept. Protein Res., 43 (1994) 588.
46 Le Grand, S.M. and Merz Jr., K.M., J. Global Opt., 3 (1993) 49.
47 Brodmeier, T. and Pretsch, E., J. Comput. Chem., 15 (1994) 588.
48 May, A.C.W. and Johnson, M.S., Protein Eng., 7 (1994) 475.
49 Payne, A.W.R. and Glen, R.C., J. Mol. Graph., 11 (1993) 74.
50 Rogers, D. and Hopfinger, A.J., J. Chem. Inf. Comput. Sci., 34 (1994) 854.
51 Leardi, R., J. Chemometrics, 8 (1994) 65.
52 Walters, D.E. and Hinds, R.M., J. Med. Chem., 37 (1994) 2527.
53 Blaney, J.M., Dixon, J.S. and Weininger, D., Paper presented at the Molecular Graphics Society Meeting on Binding Sites, York, U.K., March 1993.
54 Glen, R.C., Paper presented at the Molecular Graphics Society Meeting on Binding Sites, York, U.K., March 1993.
55 Cramer, R.D., CDA News, 8 (1993) 32.
56 Radcliffe, N.J., In Männer, R. and Manderick, B. (Eds.) Parallel Problem Solving from Nature, Vol. 2, Elsevier, Amsterdam, 1992, pp. 259–268.
57 Herndon, W.C., In King, R.B. (Ed.) Chemical Applications of Topology and Graph Theory, Elsevier, Amsterdam, 1983, pp. 231–242.
58 Morgan, H.L., J. Chem. Doc., 5 (1965) 107.
59 Moreau, G., Nouv. J. Chim., 4 (1980) 17.
60 Kuyper, L.F., In Perun, T.J. and Propst, C.L. (Eds.) Computer-Aided Drug Design, Marcel Dekker, New York, NY, 1989, pp. 327–369.