Bioengineered 4:5, 266–278; September/October 2013; © 2013 Landes Bioscience

Naturally selecting solutions

The use of genetic algorithms in bioinformatics

Timmy Manning,[1] Roy D Sleator[2,*] and Paul Walsh[1]

[1]Department of Computer Science; Cork Institute of Technology; Cork, Ireland; [2]Department of Biological Sciences; Cork Institute of Technology; Cork, Ireland

For decades, computer scientists have looked to nature for biologically inspired solutions to computational problems; ranging from robotic control to scheduling optimization. Paradoxically, as we move deeper into the post-genomics era, the reverse is occurring, as biologists and bioinformaticians look to computational techniques, to solve a variety of biological problems. One of the most common biologically inspired techniques are genetic algorithms (GAs), which take the Darwinian concept of natural selection as the driving force behind systems for solving real world problems, including those in the bioinformatics domain. Herein, we provide an overview of genetic algorithms and survey some of the most recent applications of this approach to bioinformatics based problems.

Keywords: genetic algorithm, optimization, multiple sequence alignment, protein structure prediction

Abbreviations: GA, genetic algorithm; TSP, travelling salesman problem; MSA, multiple sequence alignment; PSP, protein structure prediction; SOP, sum of pairs; PDT, pareto domination tournament; HP, hydrophobic-polar; SOGA, self-organizing genetic algorithm

Submitted: 10/16/12

Revised: 11/26/12

Accepted: 11/28/12

http://dx.doi.org/10.4161/bioe.23041

*Correspondence to: Roy D Sleator;
Email: roy.sleator@cit.ie

Introduction

First introduced by JH Holland in 1975,[1] GAs represent a biologically inspired field of evolutionary computation and, as such, characterizes a form of machine learning that has been applied to an array of bioinformatics based problems. As outlined in **Figure 1**, which plots the distribution of 150 papers referencing genetic algorithms since 1995, as indexed in the PubMed database, interest in GAs has grown significantly over the past 15 years.[2]

An analysis of the subject matter of this set of 150 peer-reviewed publications shows that the majority (88%) are concerned with the application of GAs to nucleic acid and protein based sequence analysis (**Fig. 1B**). Against this backdrop, we provide a case study of the recent application of GAs to the field of sequence analysis. To further demonstrate the flexibility of GAs, we also consider their application to the complex field of protein structure prediction (PSP).

Genetic Algorithms

A typical application of genetic algorithms is to efficiently search a large "space" of possible solutions to a problem for an optimal solution, e.g., identifying an optimal order for a number of variables or finding an optimal set of weights and parameters for an experiment.[3-5] GAs achieve this through an evolutionary search on a population of randomly generated individuals over a number of generations (**Fig. 2**). Successive generations of the population are generated from the fittest members of the previous generation, mimicking the Darwinian concept of natural selection.[6]

GAs begin with an initial population of randomly generated candidate solutions for a problem.[7,8] In each generation, the fittest population members are identified, ranked, and used as "parents" to form the basis for the next population (or next "generation"), replacing the current population.[9] Repeating this process propagates elements of successful solutions and should produce increasingly capable solution populations.[10] The genotypes of a new generation are created through the genetic operations of crossover (recombination) and mutation.[11] The actual implementation of crossover and mutation is dependent on the domain of application and the selected solution representation to produce valid solutions, as will be demonstrated in our examples.

The main components of a genetic algorithm are the genotype, phenotype, fitness function, selection algorithm, crossover operator and mutation operator.[12] Each of these topics will be discussed over the following sections, and an example provided of how the concepts can be combined to solve the traveling salesman problem (TSP)[13] using a genetic algorithm which follows the flowchart of **Figure 2**.

The goal of the TSP is to identify the most efficient path for a traveling salesman (perhaps selling sequencing machines) which visits a list of specified cities each exactly once and finishes at the starting point. The availability of a direct route between each pair of cities is assumed. A solution (good or bad) in the TSP is therefore a permutation of the set of cities in the order they must be visited.[14,15]

For a TSP with n cities, $(n-1)!/2$ possible solutions exist, meaning the complexity of the problem and the number of possible solutions grows rapidly as the number of cities increases ($n!$ possible solutions exist if equivalent solutions are considered unique where the starting cities and the direction in which the cities are traversed are different). For example, to plan an itinerary visiting every US state capital using brute force would require the selection of the best route among 3.04×10^{62} alternatives. If a computer running since The Big Bang was able to evaluate a path every nanosecond, 13.7 billion years later it would still have evaluated less than 1% of all possible routes.

The most efficient known approach to identifying an exact solution for the general TSP is the dynamic programming algorithm,[16,17] which can identify the optimal solution in $O(n^2 2^n)$ time.[18] However, heuristic approaches, such as GAs, can be employed to efficiently search the space of potential solution routes for good, if not optimal, solutions. A number of alternate exact and heuristic approaches are outlined and discussed in the paper "TSP–infrastructure for the traveling salesperson problem."[19]

The Genotype and Phenotype

Individual solutions are represented by a "genotype," a blueprint or set of instructions which is processed in software to produce an evaluable "phenotype."[13] The genotype is the solutions manipulable representation, while the phenotype is an evaluable solution to the problem.[20] A genotype chromosome is represented by a software data structure such as a string of bits, characters, integers or real numbers.[21] Each element in the structure represents a gene of the genotype. Depending on the problem domain and implementation technique, genotypes can be static or dynamic in terms of structure size.[22]

A possible genotype representation for the TSP is given as an example in **Figure 3A**. Here, each genotype is a permutation of the cities, whereas the phenotype, **Figure 3B**, is represented by a potential itinerary of visiting the associated cities. The phenotype conversion can include specific domain knowledge not encoded in the genotype, such as knowing that an additional connection is required between the last and first city.

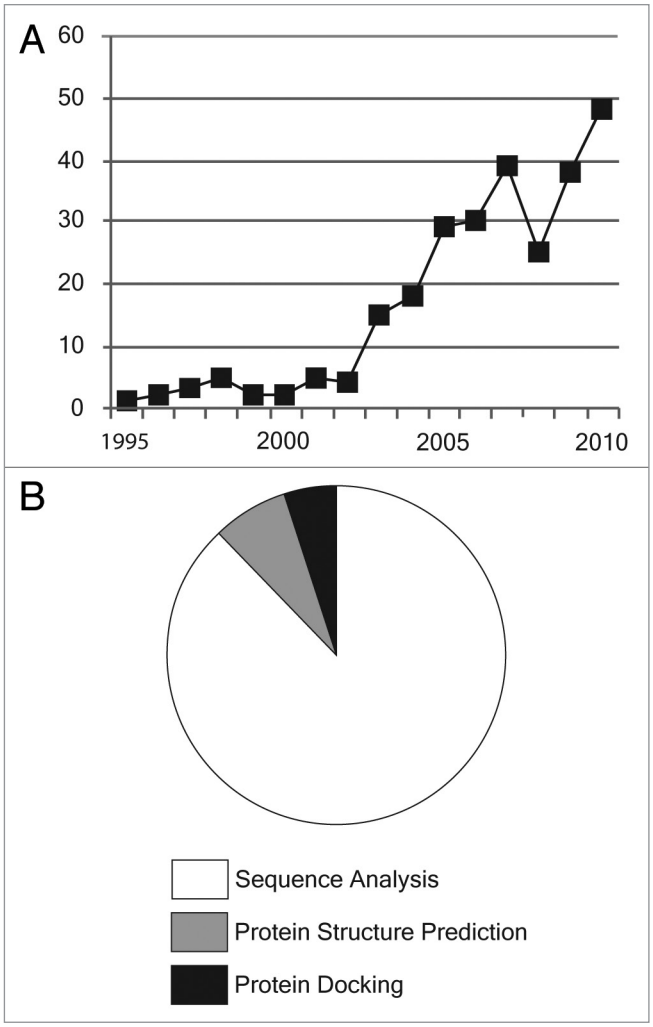

Figure 1. An analysis of 150 bioinformatics papers referencing GAs. (**A**) A plot of the distribution of the sampled papers (*y*-axis) plotted against the year of publication (*x*-axis), and (**B**) A breakdown of the subject matter addressed by the papers.

The Fitness Function

A "fitness function" is used to evaluate phenotypes to identify the fittest population members.[23] The fitness evaluation is a key aspect of the search heuristic and is commonly based on an objective measure within the domain of interest. It is the performance of the phenotypes on this fitness function that is optimized by the genetic algorithm.[24]

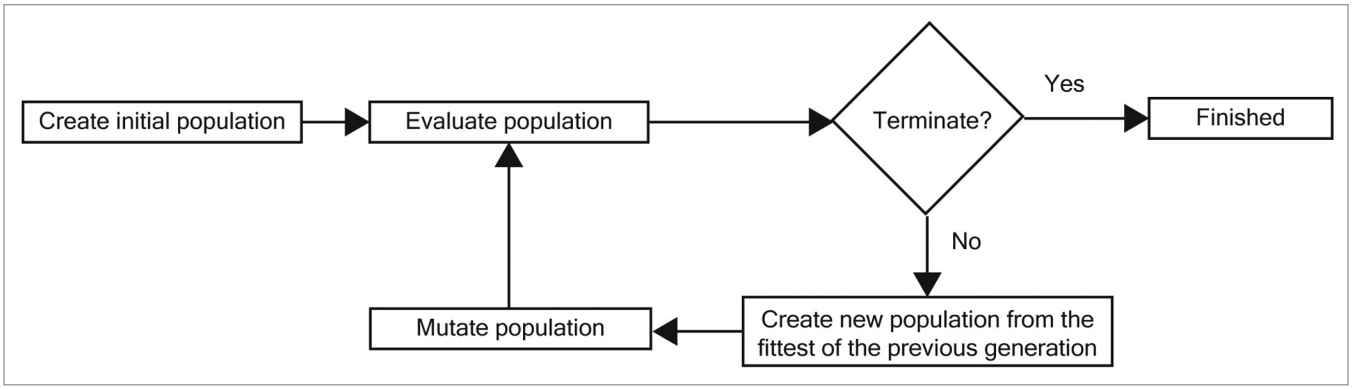

Figure 2. Flowchart for the execution of a genetic algorithm.

Selection

Once a population of solutions has been evaluated (using the fitness function), various algorithms exist for selecting which solutions will form the parents of the next generation of solutions. The selection algorithms should favor the selection of solutions with higher fitness and allow solutions to be the basis for several offspring (selection with replacement).[25] A common approach is the roulette algorithm, where each solution is apportioned an area of a (metaphorical) roulette wheel directly proportional to its fitness in the population. Selection is implemented by randomly selecting a point on the roulette wheel and accepting the associated solution.[26] With the roulette selection method, typically even the least fit solutions have a chance (albeit small) of forming the basis of a new solution in the proceeding population. This random selection ensures that some level of genetic diversity is maintained in the system, and allows potentially promising avenues of evolution to be further examined even if their potential is not immediately reflected in their fitness evaluation.[27] Some GA approaches implement a **survival rate**, which specify a percentage of the fittest solutions which are used exclusively to generate the next generation.[28]

An alternative approach is tournament selection.[23] A predefined value *s*, referred to as the tournament size, is selected which is less than the population size. To select a solution from the population, *s* solutions are first selected randomly from the current population. The simplest approach is to select the member of the tournament with the highest fitness. A tournament with $s = 1$ is equivalent to random selection. As the size of the tournament increases, the chance that the low ranked solutions will be selected decreases. Variations of this approach rank the members of the tournament and select solutions with a probability biased toward the fittest solutions.

The stochastic nature of the selection process (and the mutation operator, which will be discussed later) results in the potential for good solutions to be lost. "Elitism" allows the best performing solution (or solutions) identified thus far to be copied unmodified into each new generation.[29] This can prevent the loss of useful solutions found and speed up the progress of the algorithm. An alternate form of elitism is "steady-state" selection.[30] Instead of creating an entirely new population each generation, a segment of the unfit solutions are purged and replaced by new offspring spawned from the fit members.

Crossover

In crossover, segments of two parent genotypes are combined to form new genotypes. A well-designed crossover operator can increase the efficiency of the search.[31] **Figure 4** provides an example of a "one-point crossover" operation on two binary genotypes, creating two novel offspring.[32] In this example, an offset is selected between the fourth and fifth gene in both parent genotypes, dividing the parent genotypes each into two sections. The first child is created by combining the first four genes of parent 1 (P1[a]) with the last three genes of parent 2 (P2[b]). The second child comprises the first four genes of parent 2 (P2[a]) with the last three genes of parent 1 (P1[b]).

In "two-point crossover," two gene indexes are randomly selected in the parent genotypes.[33] The genes between these two points in both parents are switched in the parent genotypes to produce two new offspring, as shown in **Figure 5**. In this example, a contiguous string of genes are selected in each parent between the third and fifth genes inclusively. The selected sections are then switched between the parents to produce two new solutions. This approach can be extended to *k*-point (multi-point) crossover.

Two common forms of crossover specific to binary genotypes are uniform crossover and three-parent crossover. Uniform crossover uses a binary mask to decide how the parent genes will be combined. The mask is overlaid on both parents.[34] If the mask has a *1* value for gene *n*, gene *n* from the first parent is copied to the first offspring, and gene *n* from the second parent is copied to the second offspring. If the mask has a value of 0 for gene *n*, gene *n* from the first parent is copied to the second offspring, and gene *n* from the second parent is copied to the first offspring. In this way, two complete and complimentary offspring should be created by combining the genes of the parents. Three-parent crossover uses a third parent genotype to act as an arbiter, and produces a single offspring. Each gene which matches between the first two parent genotypes is copied into the offspring. If a gene value is different in the first two parents, the value of the gene from the third parent is used (forming a majority decision).[35]

Mutation

Once a new population has been created through selection and crossover, it is modified through mutation. Mutation refers to the modification of a genotype by some random process.[36] Not all mutations will result in increased fitness. For a binary genotype, the mutation operation can be as simple as inverting the value of one of the genes (flipping) or switching two adjacent values.[37] For genes represented by real numbers, the mutation could involve perturbing the value by a random amount under a Gaussian distribution.[38] New mutations which have a positive impact will increase the phenotype fitness and therefore increase its likelihood to be chosen for reproduction, meaning this positive mutation is more likely to be propagated across several solutions in future generations.[39] A common approach is to allow different levels of mutation; coarse-grained mutation to produce significantly new population members, providing increased coverage of the problem space, and fine-grained mutation to slowly edge good performing solutions toward optimality.[40] A number of the examples given in this paper also integrate local search optimizations with the mutation operator to increase the likelihood that superior offspring will be produced through mutation to increase the efficiency of the evolution.

Limitations of GAs

There are however **caveats** with the use of GAs. GAs are an approach to efficiently searching a space of possible solutions, but the final solutions produced may not be the optimal configuration as GAs can become trapped in "local optima" of the search space.[41] These locally optimal solutions may be significantly different from the optimal solution in terms of genotype, with a number of intermediate crossover and/or mutation operations required to convert any member of the current population to the optimal configuration.[42] As these locally optimal solutions are somewhat optimized, it is possible that a single mutation or crossover operator which makes the solution more similar to the globally optimal solution can actually decrease its fitness. As such, the genetic algorithm can become "trapped" on these local optima, and unlikely to improve. The results achieved can also be inconsistent, even when rerunning a GA with the same parameters, due to the stochastic nature of the process.[43]

A similar phenomenon is observed in the biological sciences in the form of the protein folding problem—more specifically, the folding funnel hypothesis for protein folding, representing a specific version of the energy landscape theory of protein folding, which assumes that a protein's native state corresponds to its free energy minimum under the solution conditions usually encountered in cells. Although the folding funnel hypothesis assumes that the native state is a deep free energy minimum with steep walls, corresponding to a single well-defined tertiary structure, energy landscapes are usually "rough," with many non-native local minima in which partially folded proteins can become trapped.[44]

The use of a well-designed coarse mutation operator (perhaps at a low usage rate) can help the algorithm to escape these local minima.[45] Running the GA several times with different initial solution configurations and increasing the population size are simple approaches to increasing the coverage of the search space, thus reducing the impact of this problem.[46]

In addition to the population size, other parameters of the genetic algorithm such as the crossover rate and the mutation rate may require tailoring to extract the best performance.[47] The crossover rate represents the percentage of offspring which will be produced using the crossover operator. The remaining offspring are copies of solutions from the previous generation.[48] With the exception of offspring produced through elitism, all offspring produced through crossover or otherwise are subjected to mutation. Typically a value in the range 60–90% is used for the crossover rate.[49-51]

Mutation rate refers to the probability with which each gene in a genotype should be mutated.[52] This should be low (typically about 1–5%), to allow the solutions being produced to examine their local areas for improvements and slowly move toward optimizing the solutions.[53] A mutation rate of 100% reduces the genetic algorithm to the equivalent of a random search. A mutation rate that is too low means the search of the potential solutions will be slow. The crossover and mutation rates are problem dependent and require trial and error for the identification of optimal values.[54,55]

Genetic algorithms are categorized as a "weak" method, as they carry out a blind search of a space of solutions without using prior domain knowledge.[56] As a result, although genetic algorithms are more efficient than a brute force search, they tend to be less efficient than "direct" methods which incorporate domain

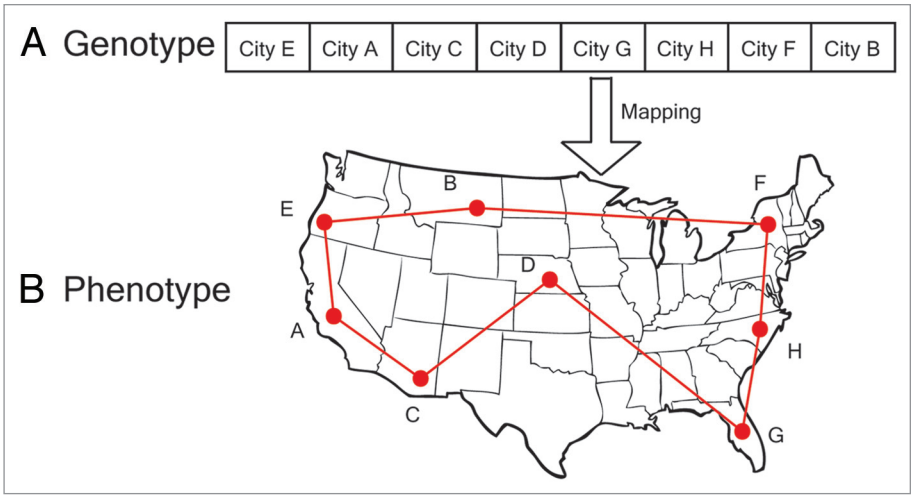

Figure 3. (**A**) A possible genotype representation for the TSP, describing a permutation of a list of cities. (**B**) A phenotype (the itinerary) generated from the genotype.

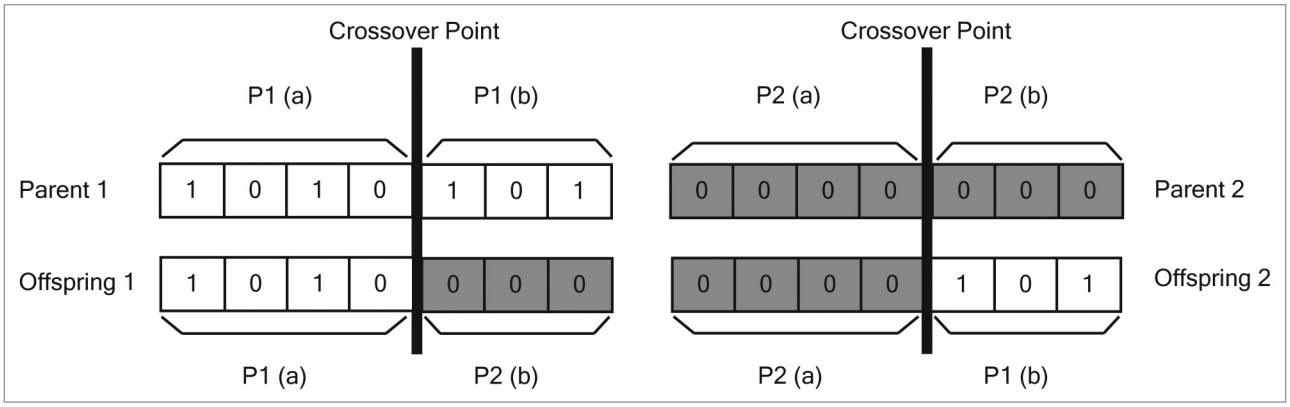

Figure 4. One-point crossover on two binary parent genotypes, creating two new offspring.

knowledge, where available. The incorporation of domain knowledge can result in a more "intelligent" search of the solution space. GAs should therefore be applied where the problem space is sufficiently large to make a brute force search impractical or intractable, and where no method exists to infer an optimal solution using domain knowledge. Further to this, Melanie Mitchell has described problems appropriate for the use of genetic algorithms as those where "the space to be searched is large, is known not to be perfectly smooth and unimodal, or is not well understood, or if the fitness function is noisy, and if the task does not require a global optimum to be found—i.e., if quickly finding a sufficiently good solution is enough."[57]

Linking the Concepts Together: A GA Strategy for Solving the TSP Problem

There are many possible valid approaches and variations on how a GA can be applied to solve the TSP.[58] In this section we provide an example of a typical approach which can be taken, discussed in terms of the previously defined concepts. To solve the TSP using a GA, one must first decide on a population size, survival rate, genotype representation and how the fitness function, crossover operator and mutation operator will be implemented. In this example, the genotype and phenotype are implemented as described in **Figure 3**, and the population size is set to 40. A roulette algorithm for selecting parents is used, with a survival rate of 40% (disregarding the worst performing 60% of each generation). The Euclidean distance which must be traveled when visiting the cities in a specified order (the length of the red line in **Figure 3**) is used as the fitness function. Here, lower values (shorter distances) are considered to be the more fit solutions.

A crossover algorithm for the TSP must ensure that the children produced are valid, in that they include each city exactly once. Crossover, in this example, is implemented by taking the cities (genes) up to a randomly selected offset in the first parent and combining them with the remaining cities taken in order from a second parent, as described in **Figure 6**.

The TSP requires a custom mutation algorithm to produce valid genotypes which fit the constraints of the problem domain (i.e., all cities listed exactly once). Three mutation operators are implemented for our example. The first is a simple fine grained mutation which involves switching the order of two adjacent genes, as demonstrated in **Figure 7**. The second mutation operator moves a randomly selected gene to a new offset in the genotype. The final mutation operator is a coarse grained approach which switches the order of 2 randomly selected genes. The coarse operator will typically have a large impact on the phenotypes produced, but facilitates an increased coverage of the search space.[59]

Once these parameters are chosen, the GA follows the flowchart shown in **Figure 2**. A typical TSP problem comprising 20 distributed cities is presented in **Figure 8A**. **Figure 8B** gives a potential solution path generated using a genetic algorithm after 100 generations. The stopping criterion can be a maximum number of generations (100 in this case), or reaching a fitness threshold. **Figure 9** shows a typical GA learning curve plotting the fitness of the best performing solution identified in each generation against the generation number. Note that the graph may be bumpy due to the stochastic nature of the algorithm where a potentially worse solution could be generated due to the mutation operator. If an elitist strategy is used, the fitness would be unable to decrease and consecutive generations without improvement would appear as horizontal lines in the fitness plot.[60]

Sequence Alignment Using a Genetic Algorithm

As for the genetic algorithms described, mutations can occur in natural evolution to the DNA, RNA or protein of a species in the form of insertions, deletions and substitutions. This results in variation between equivalent sequences with underlying similarities (conserved sequences) from different species with a divergent origin.[61] Sequence alignment is a key technique in bioinformatics that processes sequences of DNA, RNA or protein to identify regions of similarity that may be evidence of evolutionary relationships between the sequences. For a novel sequence, if a well understood homologous sequence can be identified, the features, function, structure or evolution of

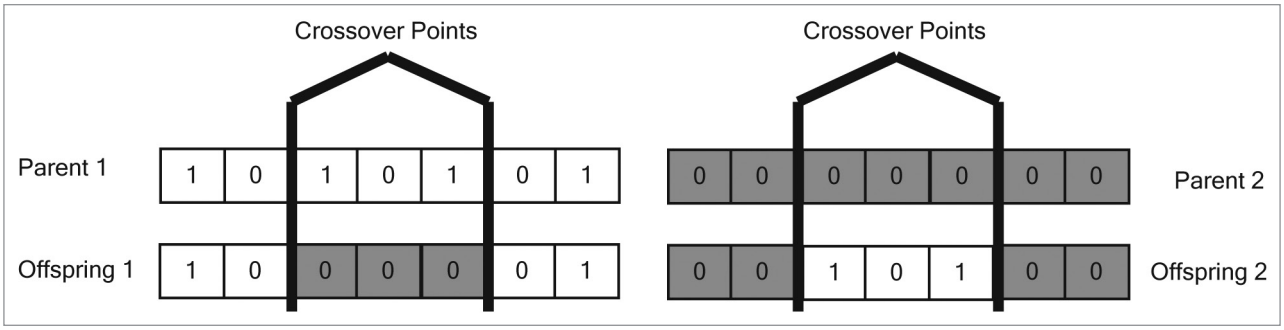

Figure 5. Two-point crossover. The parent genotypes are each divided into three segments, and the offspring are produced by combining alternate segments of the parents.

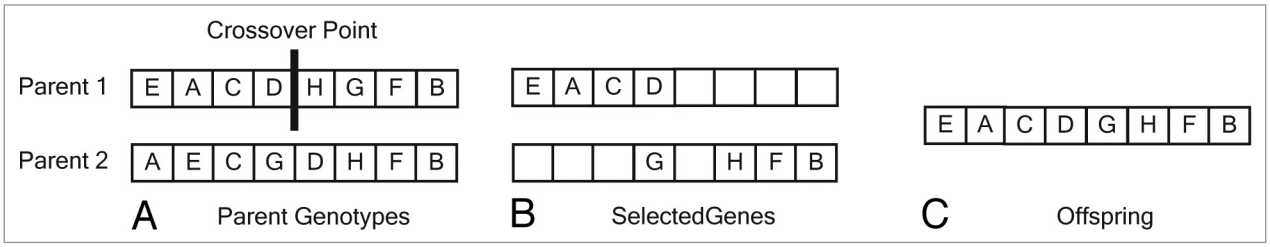

Figure 6. A possible crossover implementation for the TSP problem. (**A**) Two parents are chosen using a selection algorithm. A crossover point is randomly selected in the first parent. (**B and C**) The genes from one side of the crossover point are selected and are combined with the alternate genes in the order which they appear in the second parent to form a new child genotype.

the nucleic acid or encoded protein may be inferred.[62,63]

In pairwise alignment, one sequence is placed above the other. Gaps are inserted between the residues in either sequence such that the maximum numbers of identical or similar characters are aligned in successive columns.[64] Multiple-sequence alignment works in a similar way to pairwise alignment, but considers three or more sequences simultaneously.[65] Aligned sequences of nucleotide or amino acid residues are represented as rows within a matrix, as demonstrated in **Figure 10**. The use of multiple sequences limits the impact of coincidental alignments between two sequences and increases the impact of alignments observed across multiple sequences.[66] Multiple sequence alignment is however a complex and computationally expensive problem.

Example: Parallel Niche Pareto AlineaGA (PNPAlineaGA)

PNPAlineaGA by da Silva, Sánchez-Pérez, Gómez-Pulido and Vega-Rodríguez, is an example of an efficient genetic algorithm based approach to multiple sequence alignment for proteins.[67] PNPAlineaGA will be discussed in terms of its genotype representation, fitness function, selection, mutation operator, crossover operator, implementation and results achieved.

Genotype and phenotype. Every gene in this approach has 21 possible values, comprising 20 unique single character codes, each representing a different common amino acid, and the dash punctuation mark corresponding to gaps in the alignment from the loss or gain of amino acids in the sequences. The genotypes comprise a string of characters and gaps for each sequence in the alignment. The phenotype is represented in a two-dimensional array format with a row for each individual sequence in the alignment and a column for each gene of the sequence, as shown in **Figure 10**.

Fitness and selection. The fitness function employed here combines the sum-of-pairs (SOP) or identity (ID) scores to gauge the quality of alignments. In the SOP cost function, each pair of sequences is aligned and the "**cost**" of the current alignment is generated for each sequence pair. For example, with three sequences being aligned, alignment scores for the sequence pairs (1,2), (1,3) and (2,3) must be individually generated and combined. A simple cost function might place a one point penalty on each mismatched amino acid and a two point penalty for each amino acid aligned with a gap. The sum of pairs approach used in PNPAlineaGA utilizes the PAM350[68] scoring matrix to weight the cost of each mismatch based on the likelihood of the sequence mutation naturally occurring, with a penalty for alignments of amino acids with gaps. The ID score is a simple count of the number of columns in a multiple sequence alignment which contain the same value.

Previous work by da Silva, Sánchez-Pérez, Gómez-Pulido and Vega-Rodríguez used an SOP based fitness function.[69,70] The alignments produced indeed optimized the SOP score, but produced solutions showed low ID scores. Low ID scores limit the plausibility of the proposed alignments. PNPAlineaGA employs "Pareto optimality", an approach to multi-objective optimization, to optimize both the ID score and the SOP score simultaneously.

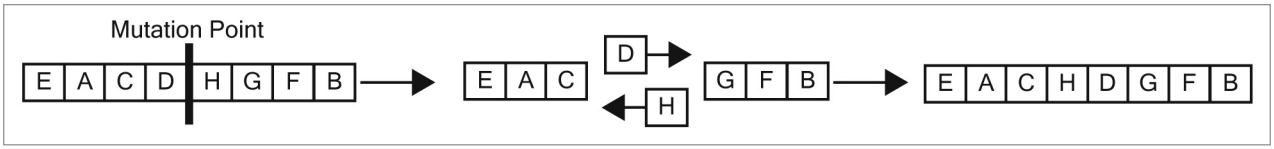

Figure 7. A possible mutation operator for the TSP. An offset in the genotype is selected. The genes either side of the offset are then switched to produce the new valid genotype.

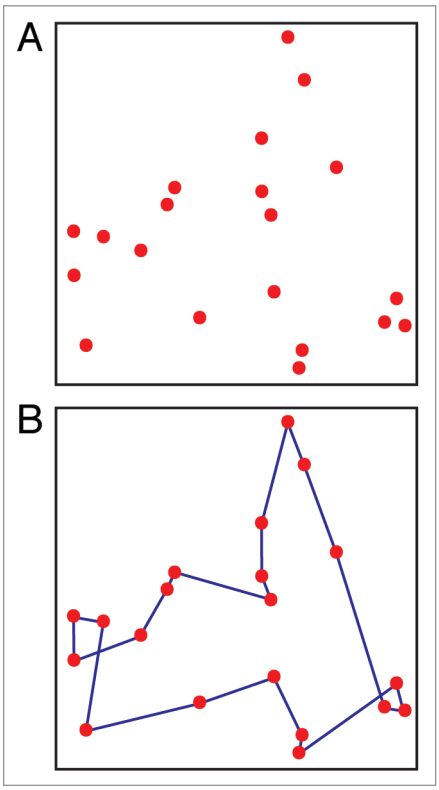

Figure 8. (**A**) An example distribution of 20 cities and (**B**) a path linking the cities found using a genetic algorithm.

Pareto optimality looks for the "Pareto front" of optimal solutions, which represent optimal solutions with varying trade-offs between the values being optimized, in this case the SOP and ID scores. A theoretical Pareto front is represented by the line in **Figure 11**. Solutions which lie under the front are said to be "dominated" (suboptimal) to those on the front. For any solution to be under the Pareto front means that its performance on at least one of the objectives can be increased without reducing the performance on the other objective.

PNPAlineaGA uses a "Pareto domination tournament" (PDT) for its selection mechanism. Two candidate solutions are selected and compared against a sampling (10% suggested) of the remaining solutions. If one of the solutions dominates the sampling it wins the tournament. If there is no winner for the tournament (both dominate the sampling, or members of the sampling dominate both candidates) an approach that favors solutions deemed to be less represented is used as a tie-breaker. The niching based selection mechanism is employed to keep a diverse range of solutions which avoids convergence of solutions to a single point on the front. The PDT approach is explained in more detail in "a niched Pareto genetic algorithm for multiobjective optimization."[71]

Mutation. The mutation operators must maintain the order of the amino acids in each sequence and all sequences must be kept the same length. The six mutation operators implemented for PNPAlineaGA therefore manipulate only the gap positions in a very controlled manner. The first three mutation operators are stochastic in nature:

• Gap insertion: insert a gap character in a random location in a sequence. Insert a gap at the beginning or end of every other sequence in the alignment;

• Gap shifting: move a gap character to a new point in the same sequence;

• Merge space: merges two or three gap characters (possibly with intermediate charecters), and shifts them to a new location in the sequence.

The remaining three mutation operators are "greedy" versions of those already described, which undo mutations which do not improve fitness, and retry with a new mutation. If the mutation is undone, the smart operators allow a new mutation to be attempted up to a maximum of 3 times. The smart operators also maintain a "direction probability," which uses its experience in each generation to decide which end of the alignments new gap insertions and shifts should be applied. The smart operators are discussed in more detail in the paper "optimizing multiple sequence alignment by improving mutation operators of a genetic algorithm."[72]

Crossover. Three crossover operators are used in PNPAlineaGA. The first two operators work in a manner similar to the one-point crossover operator already discussed. The first operator vertically splits the first parent at a specific column. The sequences of the second parent are then split such that the each subsequence contains the same amino acids to the equivalent sequence in the first parent. This divides the parents into complimentary parts which if combined could produce two new offspring, each containing all the sequences and amino acids in the correct order. An example of how two parents may be equivalently cut is given in **Figure 12**.

The second operator horizontally splits both parent genotypes into 2 sections along a specific row, for example as shown in **Figure 13**.

The third crossover operator, called RecombineMatchedCol, selects a column in the first parent in which all the amino acids match, but which are not aligned in the second parent. Gaps are inserted into the second parent as required to reproduce the column alignment of the first parent. In all 3 crossover operations gaps may be inserted at the ends of the child sequences as required to ensure they are kept the same length.

Implementation. PNPAlineaGA is implemented using an "island model" to increase the amount of genetic variation in the population. The population is divided into four or eight distinct subpopulations with a single centrally located master population. The use of more islands increases the coverage of the search space, but increases the execution time. Each sub-population evolves independently in parallel. After a prescribed number of generations (the migration rate), each slave population will send copies of its fittest

members to replace the weakest members of the master population. The master population sends copies of its fittest solutions to each slave. There is no direct connection between the slave populations. A sample representation of an island topology is given in **Figure 14**.

Results. While initial results reported for PNPAlineaGA indicate that GA based approaches to multiple sequence alignment may be able to produce results that can be compared with T-Coffee[73] (one of the leading multiple sequence alignment algorithms available), GA techniques would benefit from further research and evaluation in this area.

Example: Cyclic Genetic Algorithm for Multiple Sequence Alignment (CGA-MSA)

Proposed in a recent paper by Nizam, Ravi and Subbaraya, CGA-MAS takes a different approach to MSA in terms of both representation and implementation.[74] The genotype used here is fixed length, and specifies only the locations of the gaps in each sequence. The genotype comprises a binary string for each sequence. For each sequence in the alignment, the number of gaps is specified a priori such that all sequences are of the same length once the gaps are included. The phenotype is the sequences aligned when gaps are inserted at offsets according to the genotype (as in **Fig. 10**). **Figure 15** presents an example alignment with the gap offsets highlighted, and the corresponding genotype.

CGA-MSA employs a "self-organizing" genetic algorithm (SOGA) to reduce premature convergence to local minima and improve performance. A self-organizing genetic algorithm dynamically adjusts parameter values during execution without the need for operator interaction, based on how well the application is performing. The actual implementation begins with low parameter values for the crossover and mutation rate. The parameter values increase as changes in the fitness values produced each generation stagnate.

A restricted form of single point crossover is employed which is equivalent to the horizontal crossover of the PNPAlineaGA approach. Valid crossover points are between the genes corresponding to different sequences. The crossover rate is used to select the crossover point, such that the points become progressively offset from the start of the genotype as the crossover rate increases.

For the mutation operator, a specific offset in the genotype is selected relative to the size of the mutation rate. The binary digits up to the offset are grouped by sequence. Each group is then independently considered for mutation at the specified mutation rate. The mutation is a "binary shuffling," which randomizes the order of the bits but maintains the number of 0s and 1's in each group, thus ensuring the correct number of gaps per sequence. **Figure 16** gives an example of two binary shuffle operations used to create a new child genotype. In this example, the mutation operator creates three groups. The first group receives a binary shuffle, while the second is carried forward unchanged. The third group corresponds to only a segment of the third sequence and receives a binary shuffle. The remainder of the genotype beyond the mutation point remains unchanged.

In a small scale evaluation presented by the authors, the CGA-MSA approach was demonstrated to be able to produce better ID scores in considerably less execution time relative to a standard genetic algorithm approach.

Figure 9. A typical learning curve for a genetic algorithm.

Protein Structure Prediction Using a Genetic Algorithm

The sequence of the amino acids comprising a protein is referred to as its primary structure. The properties of the amino acids in the chain cause the chain to twist and turn, settling on the conformation with the lowest free energy.[75] It is this arrangement, referred to as the tertiary structure, which gives the protein its function.[76]

Su, Lin and Ting describe a GA based approach to determining the tertiary structure of a protein given its primary sequence.[77] The solutions produced in the GA are possible structures for the peptide represented as self-avoiding walks through a 2D triangular lattice structure. One such walk is given in **Figure 17A**. Edges representing the backbone of the protein join the amino acids to form a walk following the primary sequence of the protein. For a walk to be valid: (1) All nodes must be linked in the order of the primary sequence; (2) Edges must be connected to two amino acids; (3) Edges must not cross (the walk is a self-avoiding); (4) Only two edges can be connected to an amino acid; and (5) Edges can only connect adjacent intersect points on the lattice.

This approach uses the hydrophobic-polar (HP) model[78] for evaluating the fitness of each potential solution. Modeling

-	T	I	S	C	T	G	N	I	G	A	G	-	N	H	V	K	W	Y	Q	Q	L	P	G
-	R	L	S	C	S	S	I	F	S	S	-	-	Y	A	M	Y	W	V	R	Q	A	P	G
L	-	L	T	C	T	V	S	F	D	D	-	-	Y	Y	S	T	W	V	R	Q	P	P	G
P	E	V	T	C	V	V	S	H	E	D	P	Q	V	K	F	N	W	Y	V	Q	-	P	G

Figure 10. An example 4-sequence alignment. Individual sequences are represented on separate lines. Columns represent aligned amino acids. Adapted from reference 67.

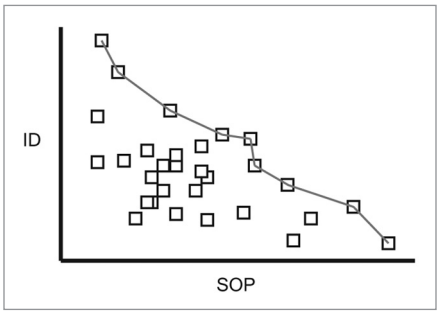

Figure 11. An example distribution of a set of solutions in a Pareto Optimality. Each box represents the fitness of a solution in terms of ID score plotted against its fitness in terms of the SOP score. The gray line represents the Pareto front. In this example the Pareto front shows optimal trade-offs between the ID and SOP scores.

the folding of a protein is a difficult problem affected by a number of attributes of the peptide chain. The HP model simplifies the problem by considering only the principle mechanism, hydrophobicity, in controlling the folding of a protein. The identities of the amino acids in the chain are reduced to either hydrophobic (H) or polar (P). "Good" solutions are identified as those which have the highest number of H-H contacts, which should represent the lowest free energy conformation of the protein under the simplified model. The plausibility of a conformation of a protein on a lattice can be evaluated using the HP model, where a contact is defined as two topological neighboring amino acids not connected by an edge.

For a protein *n* amino acids in length, the genotype of a solution consists of a list of *n*-1 directions from the list "**L, R, LU, RU, LD, RD**" representing "left," "right," "left up," "right up," "left down" and "right down" respectively. For example, the genotype for the walk given in **Figure 17** (**A**) would be **"RU, RD, R, RD, L, L, RD, L, LD, LU, RU, L, RU, L, L, RU, R, RU, RD,"** if starting at the circled hydrophobic amino acid. The phenotype is the actual walk through the lattice produced by the genotype and the fitness of each solution is defined using the number of H-H contacts.

Half of the population is generated using a tournament selection on the previous generation, with two point-crossover implemented at a rate of 0.8. Three mutation steps are performed on each offspring: (1) random gene mutations are performed and accepted only if they improve the fitness of the solution; (2) a standard uniform gene mutation at a rate of 0.4; and

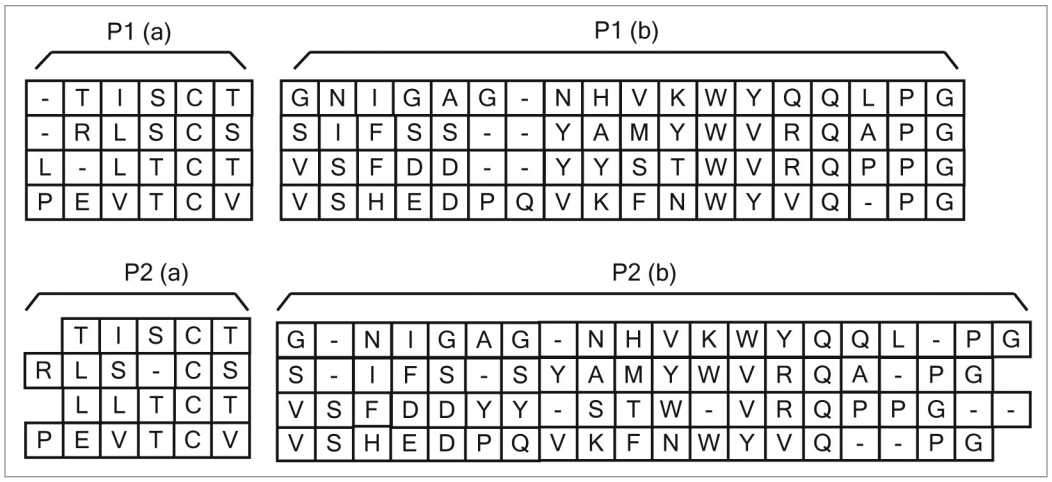

Figure 12. Example dissection of two parents for a vertical one-point crossover operation.

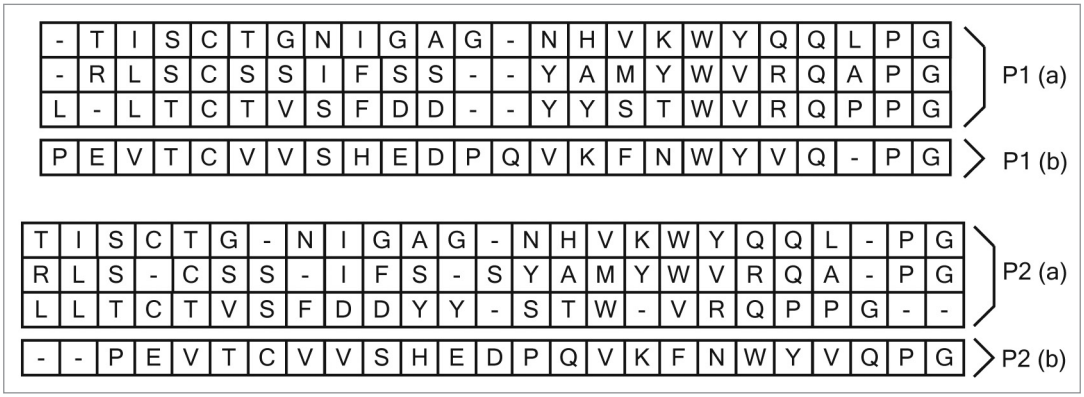

Figure 13. Example dissection of two parents for a horizontal one-point crossover operation.

(3) a segment of the genotype is rotated through all 6 angles allowed by the lattice (0°, 60°, 120°, 180°, 240° and 300°) and the best performing selected.

The first and third steps act as greedy local search optimizations to speed the convergence toward an optimal solution. Walks generated through crossover and mutation which violate the "no crossing" constraint of the lattice model are simply discarded and new instances generated. The second half of the new population is generated using an elite based reproduction where half the population is made up of a direct copy of the fittest solutions from the previous generation.

In evaluations, the use of a hybrid hill climbing (the greedy local search optimizations) and genetic algorithm combined with the elite based reproduction is shown to increase performance relative to a standard genetic algorithm approach. In evaluations, this approach achieved similar levels of performance to the "Tabu Search" approach[79] on a number of small proteins.

Lin and Su apply a hybrid of a GA and particle swarm optimization (HGA-PSO) for fitting proteins to a 3D cubic lattice, which shares many similarities with the 2D lattice model.[80] The 2D triangular lattice is used for modeling short proteins as it does not allow for overlapping structures. A 3D cubic lattice, such as that shown in **Figure 17B**, allows edges to overlap if they are at different elevations in the lattice. This allows the walks to form representations of secondary structures and complex 3D protein conformations resulting in more biologically plausible forms for large proteins than a 2D model can produce. The disadvantage of the cubic lattice relative to the triangular lattice is that amino acids in the primary sequence can only be topological neighbors if they are an odd number of amino acids apart.

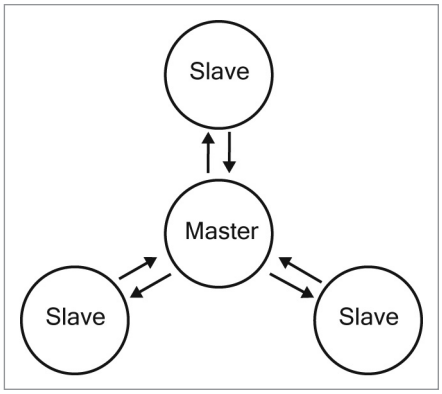

Figure 14. Graphical depiction of the flow of solution exchanges in 4-island model comprising three slave populations and a centrally located master population.

The genes in this implementation are single character codes representing the six possible steps which can be taken while traversing the 3D lattice; up, down, left, right, backward, and forward, represented

A

Sequence Alignment										Gap Offsets	Binary Representation
T	A	-	G	T	-	A	C	G	T	2,5	0010010000
-	A	C	G	-	-	T	A	C	G	0, 4, 5	1000110000
-	A	-	C	G	T	A	-	C	-	0, 2, 7, 9	1010000101
T	A	G	-	T	-	A	-	C	G	3, 5, 7	0001010100

B

0010010000	1000110000	1010000101	0001010100

Figure 15. (**A**) A sequence alignment for four sequence, and two equivalent encodings for the alignment. (**B**) The CGA-MSA genotype representing an alignment solution.

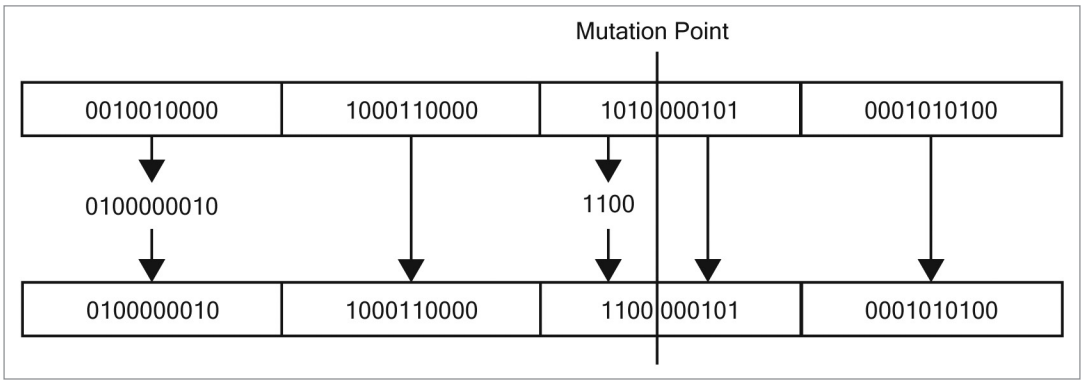

Figure 16. The CGA-MSA mutation operator. The binary digits up to the mutation point are grouped by sequence and considered for mutation individually.

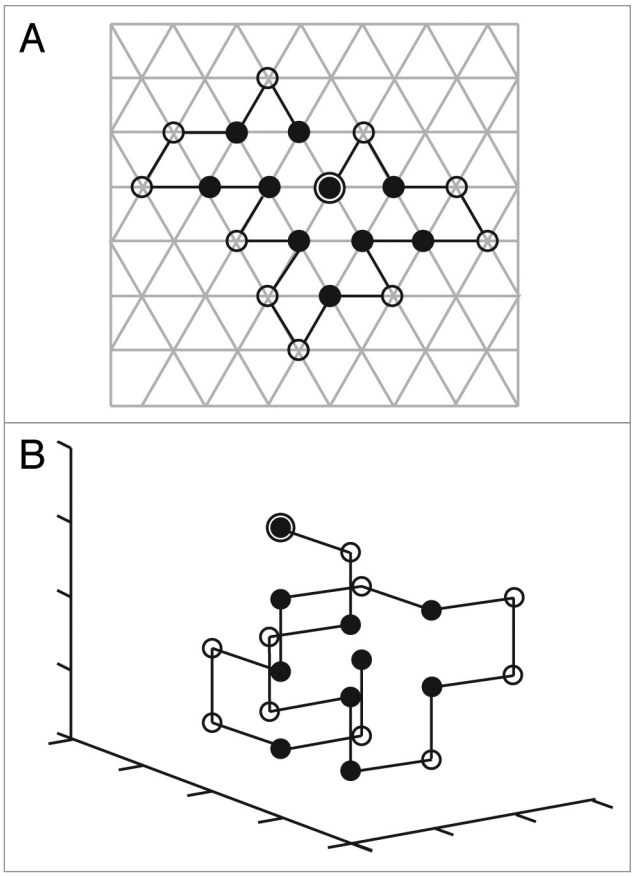

Figure 17. A protein with the format HPHPPHHPHPPHPHHPPHPH fitted to (**A**) a 2D triangular lattice, and (**B**) a 3D cubic lattice. The hydrophobic amino acids are represented as black dots and the polar amino acids as white circles. Both conformations are optimal under the HP model. Adapted from references 77 and 80, respectively.

Table 1. Performance of HGA-PSO, backtracking-EA, aging-AIS and ClonalgI on a set of 7 proteins

Sequence	Optimal Free Energy	HGA-PSO	Backtracking EA	Aging-AIS	Clonal gI
1	-11	-11	-11	-11	-11
2	-13	-13	-13	-13	-13
3	-9	-9	-9	-9	-9
4	-18	-18	-18	-18	-18
5	-29	-29	-25	-29	-29
6	-26	-26	-23	-23	-26
7	-49	-49	-39	-41	-48

by **"U, D, L, R, B, F"** respectively. For a protein with primary sequence of length *n*, the genotype is an *n*-1 sequence of genes. The phenotype is the 3D walk produced when the directions specified by a genotype are fit to the lattice. For example, the genotype for the walk in **Figure 17B** is **"F, D, L, D, R, D, R, U, R, U, L, B, L, D, B, D, F, R, U"**. As for the 2D lattice, the number of H-H contacts in the phenotype is used as the fitness of the corresponding genotype.

The HGA-PSO algorithm was evaluated against Backtracking-EA,[81] Aging-AIS[82] and ClonalgI[83] on seven standard benchmark sequences taken from "protein folding simulations of the hydrophobic–hydrophilic model by combining tabu search with genetic algorithms."[84] The best found result of each approach for every protein after 50 runs is presented in **Table 1**. In each experiment, HGA-PSO is able to identify an optimal conformation for the protein which maximizes the number of H-H contacts, achieving consistently equal or superior performance in comparison with the other approaches evaluated. Lower standard deviations in performance of the HGA-PSO approach compared with ClonalgI, the next best performing approach evaluated, are also reported.

Conclusion

This paper has presented a review of the application of GAs to computation problems in biology. While research efforts in this domain are encouraging, there are many remaining challenges if GAs are to fulfill the potential of harnessing evolutionary principles in silico. When deciding if a GA is suitable for producing a solution to a task at hand, it is important to bear in mind that any approach which does not evaluate all potential solutions cannot be guaranteed to identify the optimal solution. GAs are applicable if evaluating all potential solutions is infeasible in a reasonable amount of time, and "quickly finding a sufficiently good solution is enough."[57]

The examples given is this paper demonstrate two current efforts to address the inherent weaknesses of GAs; (1) optimizations of the genetic algorithm itself and (2) exploiting the strengths of GAs in combination with other disciplines. Optimizations of the genetic algorithm were demonstrated by the use of greedy mutation operators[67,77,80] and self-organizing parameters.[74] Similarly, it has already been mentioned that domain knowledge can be incorporated into the search operation to form an intelligent search, but this approach is domain specific. In this paper, the hybridization of GAs is demonstrated with PSO[80] and a Pareto Front,[67] but other recent interesting GA research includes topics such as the use of an Artificial Bee Colony (ABC) for protein structure prediction[85] and an Ant Colony Optimization algorithm for multiple sequence alignment,[86] for example.

In addition to the increasingly smart algorithms leveraging search coverage and execution time, the inherently parallel

nature of genetic algorithms means they have benefitted greatly from the recent surge of interest in distributed processing. In recent years, the availability of low cost multi-processor computers and cloud computing platforms have made the use of GAs more appealing as the population based approach of GAs can easily be adapted to take advantage of parallel environments.

There is also growing interest in other innovative techniques such as harnessing human interaction. The Foldit project is a successful example of such an approach, which presents the difficult problem of protein folding as a competitive computer game.[87] Foldit employs computer game psychology to encourage players to voluntarily download and replay a game which solves a real world problem. This approach harnesses the ingenuity, spatial reasoning and long-term vision of human players as well as their local processing power, with the competitive, collaborative and social aspects of Foldit acting as motivators. Foldit does not employ genetic algorithms, but it can be seen as harnessing human directed computing to perform a type of coarse search, suggesting the approach should work as part of a well-designed genetic algorithm. Given Foldit's success (on a number of problems, solutions produced in Foldit surpassed the state of the art Rosetta structure prediction program), popularity and generality of applicability to problems beyond protein folding, this approach is seen as a promising possible future area of development for interactive GAs.

Disclosure of Potential Conflicts of Interest

No potential conflicts of interest were disclosed.

Acknowledgments

This work was funded by a FP7-PEOPLE-2012-IAPP grant ClouDx-i to RDS and PW.

References

1. Holland JH. Adaptation in natural and artificial systems. Ann Arbor: University of Michigan Press 1975; 2.
2. Jiang W, Baker ML, Ludtke SJ, Chiu W. Bridging the information gap: computational tools for intermediate resolution structure interpretation. J Mol Biol 2001; 308:1033-44; PMID:11352589; http://dx.doi.org/10.1006/jmbi.2001.4633.
3. Kumar SN, Panneerselvam R. A Survey on the Vehicle Routing Problem and Its Variants. Intelligent Information Management 2012; 4:66-74; http://dx.doi.org/10.4236/iim.2012.43010.
4. Gemperline P, Niazi A, Leardi R. Genetic algorithms in chemometrics. J Chemometr 2012; 26:345-51; http://dx.doi.org/10.1002/cem.2426.
5. Sharma RN, Pancholi SS. Optimization techniques in pharmaceutical industry: A Review. Journal of Current Pharmaceutical Research 2011; 7:21-8.
6. McDowell JJ, Popa A. Toward a mechanics of adaptive behavior: evolutionary dynamics and matching theory statics. J Exp Anal Behav 2010; 94:241-60; PMID:21451751; http://dx.doi.org/10.1901/jeab.2010.94-241.
7. Álvarez D, Hornero R, Marcos JV, del Campo F. Feature selection from nocturnal oximetry using genetic algorithms to assist in obstructive sleep apnoea diagnosis. Med Eng Phys 2012; 34:1049-57; PMID:22154238.
8. Tian H, Liu C, Gao XD, Yao WB. Optimization of auto-induction medium for G-CSF production by Escherichia coli using artificial neural networks coupled with genetic algorithm. World J Microbiol Biotechnol 2012;1-9; PMID:23132252.
9. Naznin F, Sarker R, Essam D. Vertical decomposition with Genetic Algorithm for Multiple Sequence Alignment. BMC Bioinformatics 2011; 12:353; PMID:21867510; http://dx.doi.org/10.1186/1471-2105-12-353.
10. Wongsarnpigoon A, Grill WM. Energy-efficient waveform shapes for neural stimulation revealed with a genetic algorithm. J Neural Eng 2010; 7:046009; PMID:20571186; http://dx.doi.org/10.1088/1741-2560/7/4/046009.
11. Zaki N, Bouktif S, Lazarova-Molnar S. A combination of compositional index and genetic algorithm for predicting transmembrane helical segments. PLoS One 2011; 6:e21821; PMID:21814556; http://dx.doi.org/10.1371/journal.pone.0021821.
12. Agrawal D, Jaiswal HL, Singh I, Chandrasekaran K. An Evolutionary Approach to Optimizing Cloud Services. Computer Engineering and Intelligent Systems 2012; 3:47-54.
13. Larranaga P, Kuijpers CMH, Murga RH, Inza I, Dizdarevic S. Genetic algorithms for the travelling salesman problem: A review of representations and operators. Artif Intell Rev 1999; 13:129-70; http://dx.doi.org/10.1023/A:1006529012972.
14. Acuña DE, Parada V. People efficiently explore the solution space of the computationally intractable traveling salesman problem to find near-optimal tours. PLoS One 2010; 5:e11685; PMID:20686597; http://dx.doi.org/10.1371/journal.pone.0011685.
15. Urquhart N, Scott C, Hart E. Using an evolutionary algorithm to discover low CO 2 tours within a travelling salesman problem. Applications of Evolutionary Computation 2010:421-30; http://dx.doi.org/10.1007/978-3-642-12242-2_43.
16. Bellman R. Dynamic programming treatment of the travelling salesman problem. [JACM]. J ACM 1962; 9:61-3; http://dx.doi.org/10.1145/321105.321111.
17. Held M, Karp RM. A dynamic programming approach to sequencing problems. J Soc Ind Appl Math 1962; 10:196-210; http://dx.doi.org/10.1137/0110015.
18. Deineko VG, Woeginger GJ. A study of exponential neighborhoods for the travelling salesman problem and for the quadratic assignment problem. Math Program 2000; 87:519-42; http://dx.doi.org/10.1007/s101070050010.
19. Hahsler M, Hornik K. TSP–Infrastructure for the traveling salesperson problem. Journal of Statistical Software 2007; 23.
20. Hu T, Payne JL, Banzhaf W, Moore JH. Evolutionary dynamics on multiple scales: a quantitative analysis of the interplay between genotype, phenotype, and fitness in linear genetic programming. Genet Program Evolvable Mach 2012:1-33.
21. Raffe WL, Zambetta F, Li X. A survey of procedural terrain generation techniques using evolutionary algorithms. Evolutionary Computation (CEC), 2012 IEEE Congress on: IEEE, 2012:1-8.
22. Qodmanan HR, Nasiri M, Minaei-Bidgoli B. Multi objective association rule mining with genetic algorithm without specifying minimum support and minimum confidence. Expert Syst Appl 2011; 38:288-98; http://dx.doi.org/10.1016/j.eswa.2010.06.060.
23. Bakırlı G, Birant D, Kut A. An incremental genetic algorithm for classification and sensitivity analysis of its parameters. Expert Syst Appl 2011; 38:2609-20; http://dx.doi.org/10.1016/j.eswa.2010.08.051.
24. Lehman J, Stanley KO. Abandoning objectives: evolution through the search for novelty alone. Evol Comput 2011; 19:189-223; PMID:20868264; http://dx.doi.org/10.1162/EVCO_a_00025.
25. Xie H, Zhang M. Parent Selection Pressure Auto-tuning for Tournament Selection in Genetic Programming. IEEE Transaction on Evolutionary Computation 2013; 17:1-19; http://dx.doi.org/10.1109/TEVC.2011.2182652
26. Fernandez M, Caballero J, Fernandez L, Sarai A. Genetic algorithm optimization in drug design QSAR: Bayesian-regularized genetic neural networks (BRGNN) and genetic algorithm-optimized support vectors machines (GA-SVM). Mol Divers 2011; 15:269-89; PMID:20306130; http://dx.doi.org/10.1007/s11030-010-9234-9.
27. Kwak NS, Lee J. An implementation of new selection strategies in a genetic algorithm–population recombination and elitist refinement. Eng Optim 2011; 43:1367-84; http://dx.doi.org/10.1080/0305215X.2011.558577.
28. Lucas AW, Michelle DM. Cross-pollinating parallel genetic algorithms for multi-objective search and optimization. Int J Found Comput Sci 2005; 16:261-80; http://dx.doi.org/10.1142/S012905410500298X.
29. Jerin Leno I, Saravana Sankar S, Victor Raj M, Ponnambalam S. An elitist strategy genetic algorithm for integrated layout design. Int J Adv Manuf Technol 2012:1-17.
30. Whitley D. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. Proceedings of the third international conference on Genetic algorithms, 1989:116-21.
31. Sudholt D. Crossover speeds up building-block assembly. Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference: ACM, 2012:689-702.
32. Wong M, Pao WKS. A genetic algorithm for optimizing gravity die casting's heat transfer coefficients. Expert Syst Appl 2011; 38:7076-80; http://dx.doi.org/10.1016/j.eswa.2010.12.063.
33. Chen SH, Chang PC, Cheng T, Zhang Q. A Self-guided Genetic Algorithm for permutation flowshop scheduling problems. Comput Oper Res 2012; 39:1450-7; http://dx.doi.org/10.1016/j.cor.2011.08.016.
34. Yang CH, Cheng YH, Chuang LY, Changi HW. Confronting two-pair primer design for enzyme-free SNP genotyping based on a genetic algorithm. BMC Bioinformatics 2010; 11:509.
35. Eiben A, Raué P, Ruttkay Z. Genetic algorithms with multi-parent recombination. Parallel Problem Solving from Nature—PPSN III 1994:78-87.
36. Ramezanian R, Rahmani D, Barzinpour F. An aggregate production planning model for two phase production systems: Solving with genetic algorithm and tabu search. Expert Syst Appl 2012; 39:1256-63; http://dx.doi.org/10.1016/j.eswa.2011.07.134.

37. Aiello G, La Scalia G, Enea M. A multi objective genetic algorithm for the facility layout problem based upon slicing structure encoding. Expert Syst Appl 2012; http://dx.doi.org/10.1016/j.eswa.2012.01.125.
38. Abedini M, Nasseri M, Burn D. The use of a genetic algorithm-based search strategy in geostatistics: application to a set of anisotropic piezometric head data. Comput Geosci 2012; 41:136-46; http://dx.doi.org/10.1016/j.cageo.2011.08.024.
39. Taylor CM, Agah A. Data Mining and Hypothesis Refinement using a Multi-Tiered Genetic Algorithm. Journal of Intelligent Systems 2010; 19:191-226; http://dx.doi.org/10.1515/JISYS.2010.19.3.191.
40. Nair SSK, Subba Reddy NV, Hareesha KS. Exploiting heterogeneous features to improve in silico prediction of peptide status - amyloidogenic or non-amyloidogenic. BMC Bioinformatics 2011; 12(Suppl 13):S21; PMID:22373069; http://dx.doi.org/10.1186/1471-2105-12-S13-S21.
41. Kallel L, Naudts B, Rogers A. Theoretical aspects of evolutionary computing. Springer, 2001.
42. Rowe W, Platt M, Wedge DC, Day PJ, Kell DB, Knowles J. Analysis of a complete DNA-protein affinity landscape. J R Soc Interface 2010; 7:397-408; PMID:19625306; http://dx.doi.org/10.1098/rsif.2009.0193.
43. Morris MK, Saez-Rodriguez J, Clarke DC, Sorger PK, Lauffenburger DA. Training signaling pathway maps to biochemical data with constrained fuzzy logic: quantitative analysis of liver cell responses to inflammatory stimuli. PLoS Comput Biol 2011; 7:e1001099; PMID:21408212; http://dx.doi.org/10.1371/journal.pcbi.1001099.
44. Lei H, Chen C, Xiao Y, Duan Y. The protein folding network indicates that the ultrafast folding mutant of villin headpiece subdomain has a deeper folding funnel. J Chem Phys 2011; 134:205104; PMID:21639484; http://dx.doi.org/10.1063/1.3596272.
45. Bäck T. Optimal Mutation Rates in Genetic Search. Proceedings of the 5th International Conference on Genetic Algorithms: Morgan Kaufmann Publishers Inc., 1993:2-8.
46. Maaranen H, Miettinen K, Penttinen A. On initial populations of a genetic algorithm for continuous optimization problems. J Glob Optim 2007; 37:405-36; http://dx.doi.org/10.1007/s10898-006-9056-6.
47. Huang M, Narayana VK, Bakhouya M, Gaber J, El-Ghazawi T. Efficient Mapping of Task Graphs onto Reconfigurable Hardware Using Architectural Variants. Computers. IEEE Transactions on 2012; 61:1354-60.
48. Acampora G, Loia V, Salerno S, Vitiello A. A hybrid evolutionary approach for solving the ontology alignment problem. Int J Intell Syst 2012; 27:189-216; http://dx.doi.org/10.1002/int.20517.
49. Siregar MU. A New Approach to CPU Scheduling Algorithm: Genetic Round Robin. Int J Comput Appl 2012; 47:18-25.
50. González-García J, Cordero-Dávila A, Leal-Cabrera I, Robledo-Sánchez CI, Santiago-Alvarado A. Calculating petal tools by using genetic algorithms. Appl Opt 2006; 45:6126-36; PMID:16892113.
51. Eiben AE, Smith JE. Introduction to evolutionary computing. Springer, 2008.
52. Brain ZE, Addicoat MA. Optimization of a genetic algorithm for searching molecular conformer space. J Chem Phys 2011; 135:174106-10; PMID:22070291; http://dx.doi.org/10.1063/1.3656323.
53. Bauerly M, Liu Y. Evaluation and Improvement of Interface Aesthetics with an Interactive Genetic Algorithm. Int J Hum Comput Interact 2009; 25:155-66; http://dx.doi.org/10.1080/10447310802629801.
54. Nakamichi R, Ukai Y, Kishino H. Detection of closely linked multiple quantitative trait loci using a genetic algorithm. Genetics 2001; 158:463-75; PMID:11333253.
55. Gibbs MS, Maier HR, Dandy GC, Nixon JB. Minimum number of generations required for convergence of genetic algorithms. Evolutionary Computation, 2006 CEC 2006 IEEE Congress on: IEEE, 2006:565-72.
56. Michalewicz Z. Genetic algorithms+ data structures= evolution programs. springer, 1998.
57. Melanie M. An introduction to genetic algorithms. Cambridge, Massachusetts London, England, Fifth printing 1999.
58. Liu C, Kroll A. On designing genetic algorithms for solving small-and medium-scale traveling salesman problems. Swarm and Evolutionary Computation 2012:283-91.
59. Li Y, Hu CJ. Aesthetic learning in an interactive evolutionary art system. Applications of Evolutionary Computation 2010:301-10.
60. Raychaudhuri A, Khandelwal S, Chhalani S, Kakarania N. Image Binarization of Grey Level Images using Elitist Genetic Algorithm. Int J Comput Appl 2012:2012.
61. Nixon KC, Carpenter JM. On homology. Cladistics 2012; 28:160-9; http://dx.doi.org/10.1111/j.1096-0031.2011.00371.x.
62. Kumar S, Filipski A. Multiple sequence alignment: in pursuit of homologous DNA positions. Genome Res 2007; 17:127-35; PMID:17272647; http://dx.doi.org/10.1101/gr.5232407.
63. Thompson JD, Higgins DG, Gibson TJ. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic Acids Res 1994; 22:4673-80; PMID:7984417; http://dx.doi.org/10.1093/nar/22.22.4673.
64. Blazewicz J, Frohmberg W, Kierzynka M, Pesch E, Wojciechowski P. Protein alignment algorithms with an efficient backtracking routine on multiple GPUs. BMC Bioinformatics 2011; 12:181; PMID:21599912; http://dx.doi.org/10.1186/1471-2105-12-181.
65. Zufall RA. Beyond Simple Homology Searches: Multiple Sequence Alignments and Phylogenetic Trees. Current Protocols Essential Laboratory Techniques 2009:11.3. 1-.3. 7.
66. Marti-Renom MA, Madhusudhan MS, Sali A. Alignment of protein sequences by their profiles. Protein Sci 2004; 13:1071-87; PMID:15044736; http://dx.doi.org/10.1110/ps.03379804.
67. da Silva FJM, Pérez JMS, Pulido JA, Rodríguez MA. Parallel Niche Pareto AlineaGA--an evolutionary multiobjective approach on multiple sequence alignment. J Integr Bioinform 2011; 8:174; PMID:21926437.
68. Dayhoff M, Schwartz R, Orcutt B. A Model of Evolutionary Change in Proteins. Atlas of protein sequence and structure 1972; 5:345-52.
69. da Silva FJM, Sánchez-Pérez JM, Gómez-Pulido J, Vega-Rodríguez M. An evolutionary approach for performing multiple sequence alignment. IEEE, 2010:1-7.
70. da Silva FJM, Pérez JMS, Pulido JAG, Rodríguez M. Parallel AlineaGA: An island parallel evolutionary algorithm for multiple sequence alignment. IEEE, 2010:279-84.
71. Horn J, Nafpliotis N, Goldberg DE. A niched Pareto genetic algorithm for multiobjective optimization. Ieee, 1994:82-7 vol. 1.
72. da Silva FJM, Sanchez Perez J, Gomez Pulido J, Rodríguez MAV. Optimizing multiple sequence alignment by improving mutation operators of a genetic algorithm. IEEE, 2009:1257-62.
73. Notredame C, Higgins DG, Heringa J. T-Coffee: A novel method for fast and accurate multiple sequence alignment. J Mol Biol 2000; 302:205-17; PMID:10964570; http://dx.doi.org/10.1006/jmbi.2000.4042.
74. Nizam A. Cyclic Genetic Algorithm for Multiple Sequence Alignment. International Journal of Research and Reviews in Electrical and Computer Engineering (IJRRECE) 2011; 1.
75. Wang RYR, Han Y, Krassovsky K, Sheffler W, Tyka M, Baker D. Modeling disordered regions in proteins using Rosetta. PLoS One 2011; 6:e22060; PMID:21829444; http://dx.doi.org/10.1371/journal.pone.0022060.
76. Rodrigues JP, Levitt M, Chopra G. KoBaMIN: a knowledge-based minimization web server for protein structure refinement. Nucleic Acids Res 2012; 40(Web Server issue):W323-8; PMID:22564897; http://dx.doi.org/10.1093/nar/gks376.
77. Shih-Chieh S, Cheng-Jian L, Chuan-Kang T. An effective hybrid of hill climbing and genetic algorithm for 2D triangular protein structure prediction. Proteome Science; 9.
78. Dill KA. Theory for the folding and stability of globular proteins. Biochemistry 1985; 24:1501-9; PMID:3986190; http://dx.doi.org/10.1021/bi00327a032.
79. Böckenhauer HJ, Dayem Ullah A, Kapsokalivas L, Steinhöfel K. A local move set for protein folding in triangular lattice models. Algorithms in Bioinformatics 2008:369-81.
80. Lin CJ, Su SC. Protein 3D HP model folding simulation using a hybrid of genetic algorithm and particle swarm optimization. International Journal of Fuzzy Systems 2011; 13:140-7.
81. Cotta C. Protein structure prediction using evolutionary algorithms hybridized with backtracking. Artificial Neural Nets Problem Solving Methods 2003:1044-.
82. Cutello V, Morelli G, Nicosia G, Pavone M. Immune algorithms with aging operators for the string folding problem and the protein folding problem. Evolutionary Computation in Combinatorial Optimization 2005:80-90.
83. de Almeida C, Gonçalves R, Delgado M. A hybrid immune-based system for the protein folding problem. Evolutionary Computation in Combinatorial Optimization 2007:13-24.
84. Jiang T, Cui Q, Shi G, Ma S. Protein folding simulations of the hydrophobic–hydrophilic model by combining tabu search with genetic algorithms. J Chem Phys 2003; 119:4592; http://dx.doi.org/10.1063/1.1592796.
85. Benitez CMV, Parpinelli RS, Lopes HS. Parallelism, hybridism and coevolution in a multi-level ABC-GA approach for the protein structure prediction problem. Concurr Comput 2012; http://dx.doi.org/10.1002/cpe.1857.
86. Qu B, Wu Z. An efficient way of multiple sequence alignment. IEEE, 2011:442-5.
87. Cooper S, Khatib F, Treuille A, Barbero J, Lee J, Beenen M, et al. Predicting protein structures with a multiplayer online game. Nature 2010; 466:756-60; PMID:20686574; http://dx.doi.org/10.1038/nature09304.