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.

Introduction

First introduced by JH Holland in 1975, 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. 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. 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.

GAs begin with an initial population of randomly generated candidate solutions for a problem. 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. Repeating this process propagates elements of successful solutions and should produce increasingly capable solution populations. 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.

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
The main components of a genetic algorithm are the genotype, phenotype, fitness function, selection algorithm, crossover operator and mutation operator. 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) 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.

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. 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.”

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.” The genotype is the solutions manipulable representation, while the phenotype is an evaluable solution to the problem. A genotype chromosome is represented by a software data structure such as a string of bits, characters, integers or real numbers. 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.

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.
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). 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. 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. Some GA approaches implement a survival rate, which specify a percentage of the fittest solutions which are used exclusively to generate the next generation.

An alternative approach is tournament selection. 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 evolution of the population. Selection is implemented by 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. 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. Some GA approaches implement a survival rate, which specify a percentage of the fittest solutions which are used exclusively to generate the next generation.

An alternative approach is tournament selection. 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 evolution of the population. Selection is implemented by 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. 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. Some GA approaches implement a survival rate, which specify a percentage of the fittest solutions which are used exclusively to generate the next generation.

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. 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. 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. 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 to produce two new offspring. If the mask has a value of 0 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 1 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).
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. 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. For genes represented by real numbers, the mutation could involve perturbing the value by a random amount under a Gaussian distribution. 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 generations. 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. 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. 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. 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.

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.

The use of a well-designed coarse mutation operator (perhaps at a low usage rate) can help the algorithm to escape these local minima. 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. 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. 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. 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.

Mutation rate refers to the probability with which each gene in a genotype should be mutated. 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. 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.

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. 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...
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.”

**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. 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 from a second parent and combining them with the remaining cities taken in order from the first 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.

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.

**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. 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
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. Multiple-sequence alignment works in a similar way to pairwise alignment, but considers three or more sequences simultaneously. Aligned sequences of nucleotide or amino acid residues are represented as rows within a matrix, as demonstrated in Figure 11. The use of multiple sequences limits the impact of coincidental alignments between two sequences and increases the impact of alignments observed across multiple sequences. 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. 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 utilized in PNPAlineaGA employs the PAM350 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. 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.
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."72

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 characters), 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 an 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."73

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 mem-
bers of the master population. The genetic
type comprises a binary string for each
sequence. For each sequence in the align-
ment, 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
without the need for operator interaction,
and can be compared with T-Coffee73 (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-MSA takes a
different approach to MSA in terms of
both representation and implementa-
tion.74 The genotype used here is fixed
length, and specifies only the locations
of the gaps in each sequence. The geno-
type is a “binary shuffling,” which random-
izes 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
groups of three. The first group receives a
binary shuffle, while the second is car-
ried forward unchanged. The third
group corresponds to only a segment of
the sequence; the crossover point, such
that the points become progressively offset
from the start of the genotype as the cros-
sover rate increases.

For the mutation operator, a specific
offset in the genotype is selected rela-
tive to the size of the mutation rate. The
binary digits up to the offset are grouped
by sequence. Each group is then inde-
pendently considered for mutation at the
specified mutation rate. The mutation is
a “binary shuffling,” which random-
izes 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
groups of three. The first group receives a
binary shuffle, while the second is car-
rried forward unchanged. The third
group corresponds to only a segment of
the sequence; the crossover point, such
that the points become progressively offset
from the start of the genotype as the cros-
sover rate increases.

In a small scale evaluation presented by
the authors, the CGA-MSA approach was
demonstrated to be able to produce bet-
ter ID scores in considerably less execution
time relative to a standard genetic algo-
rithm approach.75

Protein Structure Prediction Using a Genetic Algorithm

The sequence of the amino acids com-
prising a protein is referred to as its pri-
mary structure. The properties of the
amino acids in the chain cause the chain
to twist and turn, settling on the confor-
mation with the lowest free energy.76 It is
this arrangement, referred to as the ter-
tiary structure, which gives the protein its
function.76

Su, Lin and Ting describe a GA based
approach to determining the tertiary struc-
ture 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 fol-
lowing the primary sequence of the pro-
tein. For a walk to be valid: (1) All nodes
must be connected 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
dges 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) model78 for evaluating the fit-
ness of each potential solution. Modeling

**Figure 10.** An example 4-sequence alignment. Individual sequences are represented on separate lines. Columns represent aligned amino acids. Adapted from reference 67.
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, RU, LD, RU, L, LU, RU, L, LD, LU, 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...
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 as single character codes:

- 0°, 60°, 120°, 180°, 240°, and 300°

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 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 triangular lattice 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.

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 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 triangular lattice 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.
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.”

The examples given in 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 and self-organizing parameters. 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 and a Pareto Front, but other recent interesting GA research includes topics such as the use of an Artificial Bee Colony (ABC) for protein structure prediction and an Ant Colony Optimization algorithm for multiple sequence alignment, for example.

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

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, U, R, D, R, U, R, U, L, B, L, D, B, D, F, 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, Aging-AIS and ClonalgI on seven standard benchmark sequences taken from “protein folding simulations of the hydrophobic–hydrophilic model by combining tabu search with genetic algorithms.” 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.

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 | ClonalgI |
|----------|---------------------|---------|-----------------|-----------|----------|
| 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     | -29             | -29       | -29      |
| 6        | -26                 | -26     | -23             | -23       | -26      |
| 7        | -40                 | -40     | -39             | -41       | -48      |

Figure 17. A protein with the format HPHPPHPPHPPHPPHPP 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.
nature of genetic algorithms means they have benefited 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 GA’s 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.8 Foldit employs computer game psychology to encourage players to voluntarily upload 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 future failure 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 FFP-PEOPLE-2012-IAPP grant CloudEx in RDS and P.W.

References

1. Holland JD. Adaptation in natural and artificial systems. Ann Arbor: University of Michigan Press 1975.

2. Jiang W, Baker ML, Leafak SJ, Chou CW. Bridging the information gap: computational tools for modern evolution structure interpretation. J Mol Biol. 2001; 306:183-44. PMID:11352099; http://dx.doi.org/10.1006/jmbi.2001.4835.

3. Kumar SN, Panoutsos B. A Survey on the Vehicle Routing Problem and its Variants. Intelligent Information Management. 2012; 4:64-74; http://dx.doi.org/10.5252/iim.2012.4-5010.

4. Geppert P, Niau A, Leard B. Genetic algorithms 4.0: a new and improved version of an artificial intelligence tool. http://dx.doi.org/10.1007/s10218-004-9121-1.

5. Smaragdinos NP, Panchali S. Optimization techniques in pharmaceutical industry. A Review. Journal of Pharmaceutical Research. 2011; 7:3-8.

6. McMinn JJ, Popa A. Tintin as a mechanism of adaptive behavior: evolutionary dynamics and matching theory status. Int J Anal Bioanal. 2010; 54:241-60. PMID:20417675; http://dx.doi.org/10.1007/s10218-004-9121-1.

7. Alavie D, Hermon R, Marcus IV, Del Campe F. Protein selection from sequential outcome using genetic algorithms to assist in obstructive sleep apnea diagnosis. Med Eng Phys. 2012; 34:1489-97. PMID:22134238.

8. Tian H, Lu C, Qin XD, Yao W. Optimization of irradiation medium for G-CSP production by Escherichia coli using artificial neural networks coupled with genetic algorithm. World J Microbiol Biotechnol. 2012; 28:195-202. PMID:22132723.

9. Naziri F, Sahker R, Emor D. Vertical descomosition with Gaines Algorithm for Multiple Sequence Alignment. BMC Bioinformatics. 2011; 12:395. PMID:21687560; http://dx.doi.org/10.1186/1471-2105-12-395.

10. Weisangangiyev A, Gillam WH. Energy-efficient waveform shape for neural stimulation revealed with a genetic algorithm. J Neural Eng. 2010; 7:46008. PMID:20875160; http://dx.doi.org/10.1088/1741-2552/7/4/46008.

11. Zakhi N, Rokhi S, Laszlo-Mohan S. A combination of compositional index and genetic algorithm for predicting transmembrane helical segments. PLoS One. 2011; 6:e23825. PMID:21844590; http://dx.doi.org/10.1371/journal.pone.0012385.

12. Agard D, Jewel RL, Singh J, Chandrasekaran K. An Evolutionary Approach to Optimizing Cloud Services. Computer Engineering and Intelligent Systems 2012; 3:47-54.

13. Lusinchi R, Kaplanay CMIH, Mezio RH, Sinau I, Difede JV. Genetic algorithm for the travelling salesman problem: A review of representations and operators. Artificial Intell Rev. 1995; 13:229-70; http://dx.doi.org/10.1007/BF01203172.

14. Acha DE, Pando V. People efficiently explore the solution space of the computationally intractable travelling salesman problem to find near-optimal tours. PLoS One. 2010; 5:e1468. PMID:21106879; http://dx.doi.org/10.1371/journal.pone.0014685.

15. Uzunýan N, Scott C, Hart E. Using an evolutionary algorithm to discover low CO2 tours within a travelling salesman problem. Applications of Evolutionary Computation 2010:421-40; http://dx.doi.org/10.1007/978-3-642-11244-8_3.42.

16. Hoffst D. Random dynamics near the travelling salesman problem. [JACM]. J ACM 1962; 9:67-91. http://dx.doi.org/10.1145/321019.321021.

17. Held M, Karp RM. A dynamic programing approach to sequencing problems. J Soc Ind Appl Math. 1962; 10:196-209; http://dx.doi.org/10.1137/0110015.

18. Drezner W, Wesonger GJ. A study of expe - rimental neighborhoods for the travelling salesman problem and for the quadratic assignment problem. Math Program 2000; 87:519-42; http://dx.doi.org/10.1007/s10107-000-0010-8.

19. Harker M, Heinrich K. TSP-Infrastructures for the travelling salesperson problem. Journal of Statistical Softw 2007; 12.

20. Hu T, Poon HO, Ren SC, Chai J. The travelling salesman problem: An intelligent algorithm without specifying minimum support and minimum confidence. IEEE Trans Syst Man Cybern Part B Cybern. 2011; 41:256-258. http://dx.doi.org/10.1109/TSMCB.2010.208304.

21. Raffel WL, Zacharias F, Li X. On the importance of selecting good genetic operators for combinatorial optimization. Comp Oper Res. 2007; 34:1835-1846. http://dx.doi.org/10.1016/j.cor.2006.08.005.

22. Lehrman J, Stanley K. Abandoning objectives: Evolution through the search for novelty alone. Evolutionary Computation 2011; 19:689-92; PMID:21086264; http://dx.doi.org/10.1162/evco_a_00025.

23. Lusinchi R, Kaplanay CMIH, Mezio RH, Sinau I, Difede JV. Genetic algorithm for the travelling salesman problem: A review of representations and operators. Artificial Intell Rev. 1995; 13:229-70; http://dx.doi.org/10.1007/BF01203172.

24. Acha DE, Pando V. People efficiently explore the solution space of the computationally intractable travelling salesman problem to find near-optimal tours. PLoS One. 2010; 5:e1468. PMID:21106879; http://dx.doi.org/10.1371/journal.pone.0014685.

25. Acha DE, Pando V. People efficiently explore the solution space of the computationally intractable travelling salesman problem to find near-optimal tours. PLoS One. 2010; 5:e1468. PMID:21106879; http://dx.doi.org/10.1371/journal.pone.0014685.

26. Held M, Karp RM. A dynamic programing approach to sequencing problems. J Soc Ind Appl Math. 1962; 10:196-209; http://dx.doi.org/10.1137/0110015.

27. Held M, Karp RM. A dynamic programing approach to sequencing problems. J Soc Ind Appl Math. 1962; 10:196-209; http://dx.doi.org/10.1137/0110015.

28. Drezner W, Wesonger GJ. A study of expe - rimental neighborhoods for the travelling salesman problem and for the quadratic assignment problem. Math Program 2000; 87:519-42; http://dx.doi.org/10.1007/s10107-000-0010-8.

29. Harker M, Heinrich K. TSP-Infrastructures for the travelling salesperson problem. Journal of Statistical Softw 2007; 12.

30. Hu T, Poon HO, Ren SC, Chai J. The travelling salesman problem: An intelligent algorithm without specifying minimum support and minimum confidence. IEEE Trans Syst Man Cybern Part B Cybern. 2011; 41:256-258. http://dx.doi.org/10.1109/TSMCB.2010.208304.

31. Raffel WL, Zacharias F, Li X. On the importance of selecting good genetic operators for combinatorial optimization. Comp Oper Res. 2007; 34:1835-1846. http://dx.doi.org/10.1016/j.cor.2006.08.005.

32. Lehrman J, Stanley K. Abandoning objectives: Evolution through the search for novelty alone. Evolutionary Computation 2011; 19:689-92; PMID:21086264; http://dx.doi.org/10.1162/evco_a_00025.
Algorithm. Int J Hum Comput Interact 2009; 25:155-60.

Genetic Algorithms: Morgan Kaufmann Publishers

Proceedings of the 5th International Conference on Evolutionary Computation. Springer, 2001.

Multiple Sequence Alignments and Phylogenetic Trees. Camel Optimization Laboratory. Technical reports 2005:3-7.

Optimal Mutation Rates in Genetic Search. Morgan Kaufmann Publishers 1995.

A New Approach to CPU Scheduling Algorithm. IEEE Transactions on Parallel and Distributed Systems 2012; 23:1049-1055.

A hybrid evolutionary approach for solving the strategic alignment problem. Int J Intel Syst 2012; 27:189-206; PMID:22720537.

A New Approach to CPU Scheduling Algorithm. IEE Comput Intell 2012; 47:18-25.

Genetic algorithms and the Hydrophobic-Internal Model. Bioinformatics 2004; 20:165-166; PMID:1478213.

Multiobjective Optimization and Evolutionary Algorithms. In: Held M, Kallrath J, eds. Handbook of Optimization. Berlin, Heidelberg: Springer, 2011:1-28.

A novel method for fast and accurate multiple sequence alignment. J Mol Biol 2000; 302:205-221; PMID:10849132.

A new algorithm for 2D triangular protein structure prediction. Algorithms in Molecular Biology 2011; 6:5-16; doi:10.1186/1748-7188-6-5.

Multiple Sequence Alignments and Phylogenetic Trees. Camel Optimization Laboratory. Technical reports 2005:3-7.

A Model of Learning and Generalization using Genetic Algorithms. Machine Learning 1994; 13:199-220; PMID:7536831.

Multiple Sequence Alignments and Phylogenetic Trees. Camel Optimization Laboratory. Technical reports 2005:3-7.

Optimal Mutation Rates in Genetic Search. Morgan Kaufmann Publishers 1995.

An algorithm for 2D triangular protein structure prediction. Algorithms in Molecular Biology 2011; 6:5-16; doi:10.1186/1748-7188-6-5.

A Model of Learning and Generalization using Genetic Algorithms. Machine Learning 1994; 13:199-220; PMID:7536831.

A Hybrid Immune-Based System for the Protein Folding Problem. Evolutionary Computation in Combinatorial Optimization. Berlin, Heidelberg: Springer, 2011:1-5.

A Hybrid Immune-Based System for the Protein Folding Problem. Evolutionary Computation in Combinatorial Optimization. Berlin, Heidelberg: Springer, 2011:1-5.

A Novel Hybrid Algorithm for 2D Triangular Protein Structure Prediction. Algorithms in Molecular Biology 2011; 6:5-16; doi:10.1186/1748-7188-6-5.

A New Approach to CPU Scheduling Algorithm. IEE Comput Intell 2012; 47:18-25.

A Hybrid Immune-Based System for the Protein Folding Problem. Evolutionary Computation in Combinatorial Optimization. Berlin, Heidelberg: Springer, 2011:1-5.

A Novel Hybrid Algorithm for 2D Triangular Protein Structure Prediction. Algorithms in Molecular Biology 2011; 6:5-16; doi:10.1186/1748-7188-6-5.