Application of Evolutionary Algorithms for Multiple Sequence Alignment

Rahul Choudhury
BIOC 218
Stanford University
SUNetID: rahul99

Abstract

Multiple Sequence Alignment is a crucial task in Bioinformatics. Most of the commonly used multiple alignment methods are based on a dynamic programming approach. This approach however requires time proportional to the product of the sequence lengths and also doesn't provide an extensible platform for evaluating different objective functions. Tree-based algorithms, which combine results from pairwise alignments, have also been proposed. However, these algorithms depend on the existence of a tree that describes the relations between the sequences, and this tree cannot always be obtained. Recently Evolutionary Algorithms have been used successfully in a wide variety of applications to find solutions for hard optimization problems. They offer the advantage of operating on several solutions simultaneously, combining exploratory search through the solution space with exploitation of current results. In this paper, the focus is on usage of Evolutionary Algorithms for multiple Sequence alignment. Aspects of Evolutionary Algorithm, which are of immediate concern to multiple sequence alignments, are discussed. Also a critical analysis of two evolutionary algorithms for multiple sequence alignment, namely MSA-EA (an Evolutionary Algorithm to improve Clustal-V) and SAGA (a genetic algorithm dedicated for multiple sequence alignment) has been done. Results from these two evolutionary algorithms have also been compared with traditional approaches.

Table of Contents

1. Introduction

Multiple Sequence Alignment is an optimization problem that appears in diverse scientific fields. During the last decade, there has been an increasing interest in the biosciences for methods that can efficiently solve this problem for sequences such as biological macromolecules, DNA and proteins. Multiple Sequence Alignment has been used to address many critical problems in bioinformatics. Multiple sequence alignment of a set of sequences can provide information as to the most alike regions in the set. This in turn helps help demonstrate homology between new sequences and existing families. Multiple sequence alignment has been used to help find diagnostic patterns for families (Bairoch et al., 1997), and to predict secondary or tertiary structure of new sequences (Rost and Sander, 1993). Another use for consensus of information retrieved from a multiple sequence alignment is for the prediction of specific probes for other members of the same group or family of similar sequences in the same or other organisms. Once a consensus pattern has been found, database-searching programs may be used to find other sequences with a similar pattern. In the laboratory, a reasonable consensus of such patterns may be used to design polymerase chain reaction (PCR) primers of amplification of related sequences. Once the alignment is found, the number or types of changes in the aligned sequence residues may be used for a phylogenic analysis (Felenstein, 1998). The resluting alignments can be used to generate profiles (Gribskov et al., 1987) or Hidden Markov Models (HMM) (Haussler et al., 1993) that can be used to search databases for distantly related members of the family.

In general, two basic classes of multiple alignment programs have been developed. When making a global alignment, the algorithm attempts to align sequences chosen by the user over their entire length. Local alignment algorithms automatically discard portions of sequences that do not share any homology with the rest of the set.

1.1 Problems with automatic generation of Multiple Sequence Alignment

A multiple alignment is used to discover relationships within a set of sequences that may have been diverging for millions of years. The relationships that researchers are interested in are typically evolutionary, structural, and functional. To discover meaningful relationships, one would require an in-depth knowledge of the evolutionary history and structural properties of these sequences. However in real life this information is rarely available. Researchers instead use generic empirical models of protein evolution [1, 2], based on sequence similarity. Unfortunately, these can prove difficult to apply when the sequences are less than 30% identical. Also, accurate optimization methods that use these models are typically very expensive (in terms of computation resources) for more than a handful of sequences. This is why most multiple alignment methods rely on approximate heuristic algorithms. These heuristics are usually a complex combination of ad hoc procedures mixed with some elements of dynamic programming. Overall, two key properties characterize them: the optimization algorithm and the criteria (objective function) an algorithm attempts to optimize.

1.2 Classification of Optimization Algorithms

To get an idea of where Genetic Algorithms fall in the currently available array of multiple sequence alignment applications, its important to look at the major categories. Optimization algorithms roughly fall in three categories: the exact, the progressive, and the iterative algorithms.

- **Exact:** These types of algorithms try to perform an optimal or a sub-optimal alignment within some well defined bounds [3], [5]. Unfortunately, typically they are limited by the number of sequences they can handle and the type of objective function they can optimize.
- **Progressive**: Progressive alignment [4, 6] methods use dynamic programming to build a multiple alignment starting with the most related sequences and then progressively adding less related sequences to the initial alignment. This approach has the advantage of speed and simplicity. However the major problem with the progressive alignment methods is that errors in the initial alignments of the most closely related sequences are propagated to the multiple alignments.
- **Iterative**: Iterative alignment methods depend on algorithms able to produce an alignment and to refine it through a series of cycles (iterations) until no more improvement can be made. Iterative methods can be deterministic or stochastic, depending on the strategy used to improve the alignment. The simplest iterative strategies are deterministic. They involve extracting sequences one by one from a multiple alignment and realigning them to the remaining sequences [10] [16]. The procedure is terminated when no more improvement can be made (convergence). Stochastic iterative methods include HMM training, simulated annealing (SA) [17, 20] and evolutionary computation such as genetic algorithms (GAs) [21, 22, 23, 24] and evolutionary programming [25, 26]. Their main advantage is to allow for a good separation between the optimization process and evaluation criteria (objective function). It is the objective function that defines the aim of any optimization procedure.

In this paper, the focus is on the evolutionary algorithms.

1.3 The Objective Function

In an evolutionary algorithm, the objective function is the criteria used to evaluate the quality (fitness) of a solution (individual). To be of any use, the value that this function associates to an alignment must reflect its biological relevance and indicate the structural or the evolutionary relation that exists among the aligned sequences. In theory, a multiple alignment is correct if in each column the aligned residues have the same evolutionary history or play similar roles in the three-dimensional fold of RNA or proteins. Since evolutionary or structural information is rarely at hand, it is common practice to replace them with a measure of sequence similarity. The rationale behind this is that similar sequences can be assumed to share the same fold and the same evolutionary origin as long as they are more than 30% identical over 100 residues or more.

Accurate measures of similarity are obtained using substitution matrices. A substitution matrix is a pre-computed table of numbers where each possible substitution/conservation receives a weight indicative of its likeliness as estimated from data analysis. In these matrices, substitutions (conservations) observed more often than one would expect by chance receive positive values while under-represented mutations are associated with negative values. Given such a matrix the correct alignment is defined as the one that maximizes the sum of the substitution (conservations) score. An extra factor is also applied to penalize insertions and deletions (Gap penalty). The most commonly used model for that purpose is named ‘affine gap penalties’. It penalizes an insertion/deletion once for its opening (gap opening penalty, abbreviated GOP) and then with a factor proportional to its length (gap extension penalty, abbreviated GEP). This measure can be extended for the alignment of multiple sequences in many ways. For instance, it is common practice to set the score of the multiple alignment to be the sum of the score of every pairwise alignment it contains (sums of pairs)[27]. While that scoring scheme is the most widely

used, its main drawback stems from the lack of an underlying evolutionary scenario. It assumes that every sequence is independent and this results in an overestimation of the number of substitutions..

An objective function always defines a mathematical optimum, that is to say an alignment in which the sequences are arranged in such a manner that they yield a score that cannot be improved. Evolutionary algorithms can be very useful to answer questions like whether an objective function can be optimized and whether it is biologically relevant or not. They make it possible to design new scoring schemes without having to worry, at least in the first stage, about optimization issues.

In the next section, one of these evolutionary techniques known as genetic algorithms (GA) is discussed. Genetic Algorithms are described along with another closely related stochastic optimization algorithm: simulated annealing.

2. Evolutionary Algorithms

Evolutionary algorithms are stochastic search methods that mimic the metaphor of natural biological evolution. Evolutionary algorithms operate on a population of potential solutions applying the principle of survival of the fittest to produce better and better approximations to a solution. At each generation, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain an d breeding them together using operators borrowed from natural genetics. This process leads to the evolution of populations of individuals that are better suited to their environment than the individuals that they were created from, just as in natural adaptation. Evolutionary algorithms model natural processes, such as selection, recombination, mutation, migration, locality and neighborhood.

Two stochastic strategies have been widely used for sequence analysis: **simulated annealing (SA) and genetic algorithms (GA)**.

2.1 Simulated Annealing

Simulated annealing (SA) [28] was the first stochastic algorithm used to attempt solving the multiple sequence alignment problem. Simulated Annealing does not actually belong to the field of evolutionary computation. But in practice it has influenced the genetic algorithms used in sequence analysis to a great extent. SA relies on an analogy with physics. The idea is to compare the solving of an optimization problem to some crystallization process (cooling of a metal). In practice, given a set of sequences, a first alignment is randomly generated. A perturbation is then applied (shifting of an existing gap or introduction of a new one) and the resulting alignment is evaluated with the objective function. If that new alignment is better than the previous one, the previous one is replaced. Otherwise it replaces it with a probability that depends on the difference of score and on the current temperature. The higher the temperature the more likely an important score difference will be accepted. Every cycle the temperature decreases slightly until it reaches 0. From the perspective of an evolutionary algorithm, SA can be regarded as a population with one individual only. Perturbations are similar to the mutations used in evolutionary algorithms. Apart from the population size of one, the main difference between SA and any true evolutionary algorithm is the extrinsic annealing schedule. However Simulated Annealing based algorithms are typically extremely slow. This serious limitation makes it much harder to use it as the black box one needs to evaluate the design new objective functions.

2.2 Genetic Algorithms

It is in an attempt to overcome the limits of SA that evolutionary algorithm were adapted to the multiple sequence alignment problem. Genetic algorithm is an optimization technique that was formulated during the early years of the 1970's by John Holland [29]. This technique is useful for finding the optimal or near optimal solutions for combinatorial optimization problems that traditional methods fail to solve efficiently.
The genetic algorithms approach is based on the assumption that simulating an evolutionary process in a population of potential solutions can eventually "evolve" good solutions. Biological terms are conveniently used to describe this process: chromosomes are the potential solutions. Every chromosome is composed of several genes, the solution parameters. Many chromosomes form a population. Successive populations are referred to as generations. Crossover is the exchange of genes (solutions parameters) between two chromosomes (solutions). Mutation is the random change of one or more genes in a chromosome. Offsprings are the new chromosomes created by two parent chromosomes by crossover. The genetic algorithms process starts with an initial population composed of random chromosomes, which form the first generation. Crossover is used to combine genes from the existing chromosomes and create new ones. Then, the best chromosomes are selected to form the next generation. This selection is based on a fitness function, which assigns a fitness value to every chromosome. The ones with the best fitness value "survive" to give offsprings for the new generation, and the process is repeated until satisfactory solutions evolve.

The main advantage of genetic algorithms over other optimization methods is that there is no need to provide a particular algorithm to solve a given problem. It only needs a fitness function to evaluate the quality of different solutions. Also since it is an implicitly parallel technique, it can be implemented very effectively on powerful parallel computers to solve exceptionally demanding large-scale problems.

2.2.1 Comparison with Simulated Annealing

Evolutionary algorithms are parallel stochastic search tools. Unlike SA, which maintains a single line of descent from parent to offspring, evolutionary algorithms maintain a population of trials for a given objective function. Evolutionary algorithms are among the most interesting stochastic optimization tools available today. However implementation of an evolutionary algorithm dedicated to multiple alignment is much less straightforward than with simulated annealing. In other areas of computational biology, evolutionary algorithms have already been established as powerful tools. These include RNA [30, 31] and protein structure analysis [32, 33]. Among all the existing evolutionary algorithms (genetic algorithms, genetic programming, evolution strategies, and evolutionary programming) genetic algorithms have been by far the most popular in the field of computational biology.

2.2.2 Key Ingredients of Genetic Algorithm

Any Genetic Algorithm strategy deals with two important ingredients: **the selection method and the operators**. Selection is established in order to lead the search toward improvement. It means that the best individuals (as evaluated using the objective function) must be the most likely to survive. To serve the GA purpose, this selection strategy cannot be too strict. In stead it should allow some variety to be maintained all along the search in order to prevent the GA population from converging toward the first local minimum it encounters. Evolution toward the optimal solution also requires the use of operators that modify existing solutions and create diversity (mutations) or optimize the use of the existing diversity (crossovers) by combining existing motifs into an optimal solution. The main difficulty to

overcome when adapting a GA to a problem like multiple sequence alignment is the design of a well-suited series of operators. The reason for this is in genetic algorithms the operators (and the problem representation) largely control the manner in which a solution landscape is analyzed.

2.2.3 Existing Genetic Algorithm Based solutions for Multiple Sequence Alignment

Usage of evolutionary algorithms in multiple sequence alignment has been in practice for quite some time.

- Attempts to apply evolutionary algorithms to the multiple sequence alignment problem started in 1993 when Ishikawa et al. published a hybrid Genetic Algorithm [34] that does not try to directly optimize the alignment but rather the order in which the sequences should be aligned using dynamic programming. Of course, this limits the algorithm to objective functions that can be used with dynamic programming. Even so, the results obtained that way were convincing enough to prompt the development of the use of GAs in sequence analysis.
- The first Genetic Algorithm able to deal with sequences in a more general manner was described a few years later by Notredame and Higgins [24], shortly before a similar work by Zhang [23]. In these two GAs, the population is made of complete multiple sequence alignments and the operators have direct access to the aligned sequences: they insert and shift gaps in a random or semi-random manner.
- Over the following years, at least three new multiple sequence alignment strategies based on evolutionary algorithms have been introduced [22], [26] and [25]. Each of these relies on a principle similar to SAGA: a population of multiple alignments evolves by selection, combination and mutation. The population is made of alignments and the mutations are string-processing programs that shuffle the gaps using complex models. The main difference between SAGA and these recent algorithms has been the design of better mutation operators that improve the efficiency and the accuracy of the algorithms.

These new results have strengthened the idea that the essence of the adaptation of GAs to multiple sequence alignments is the design of proper operators, reflecting as well as possible the true mechanisms of molecular evolution.

In the next section, two examples of Genetic Algorithms are discussed to analyze the many ingredients used in the context of multiple sequence alignment. For both the cases, the process for initialization, the usage of appropriate operators and the way a given objective function is evaluated is analyzed.

3. MSA-EA: an Improvement of Clustal V by Evolutionary Algorithm

Thomsen et. el extended previous work with evolutionary algorithms (EA) by using MSA solutions obtained from the well-known Clustal V algorithm [35]. They took the Clustal V as a candidate solution seed of the initial EA population. They also made an effort to provide fair performance comparison between EA and the Clutsal V.

3.1 Initialization

The population of initial parent alignment matrices was generated via random initialization of rows in the following way: First, for each sequence s(i) with length l(i) a random permutation from the set 1,2,….,w (representing all columns in the matrix). Second, the first l(i) numbers of the permutation were sorted in increasing order. Third, the permutation numbers were used as indices in the matrix row indicating where the according amino acid symbols from the sequence s(i) were placed. Finally, all positions in the matrix row not associated with an amino acid symbol were filled with a gap symbol. This procedure was repeated for all n sequences.

For each MSA experiment below, the solution from Clustal V was used as one “seed" in the EA population in the hopes of starting the evolution in a more useful region of the search space than with a purely random seed. This of course bears the risk of driving the search towards the local optimum of the Clustal V seed solution. However, introducing just one single Clustal V solution among a population of random solutions, which are recombined and mutated during the run, yielded solutions that quickly arrived at more appropriate final solutions

3.2 Operators

During the evolutionary process the individuals were exposed to different variation operators in order to alter the candidate alignments. Only the important operators briefly mentioned here:

The **LocalShuffle** operator picks a random amino acid from a randomly chosen row (sequence) in the alignment and checks whether one of its neighbors is a gap. If this is the case, the algorithm swaps (exchanges) the selected amino acid with a gap neighbor. If both neighbors are gaps then one of them is picked randomly.

The **BlockShuffle** operator is very similar to LocalShuffle. First, a random block of consecutive amino acids is picked from a randomly chosen row (sequence). The block is then moved to the left or right by one position (depending on which side contains gaps) if there is a neighbor position with a gap. If the block has gaps on both sides, it picks the direction of movement randomly.

The **GrowMatchedColumns** operator randomly selects a fully matched column in the alignment (containing no gaps). If possible, the operator tries to swap amino acids with gaps in all sequences, such that a new fully matched adjacent column is generated either to the left or right of the column.

The **RecombineMatchedColumns** operator takes two alignment genomes from the population and randomly selects a fully matched (non-gap) column in each of them if present, which must not involve the same amino acids in the sequence order. The operator then tries to create an alignment by swapping gaps with amino acids, such that the o spring contains feasible sequences that contain both matched columns

The **CleanUpGapColumns** operator moves matched gap columns to the end (right-hand side) of the sequence, since they are only disrupting the current alignment. The operator is always applied after a modification of an individual by one of the four operators mentioned above.

3.3 Evaluation

Prior to the candidate solution evaluation, all columns containing gaps only are moved to the right-hand side of the alignment matrix using the CleanUpGapColumns operator and are ignored in the evaluation process. The optimization task is then defined as a fitness maximization problem, such that matching amino acids across sequences are rewarded and gaps (excluding those in matched gap columns on the very right hand side) are penalized

$$Fitness = SymbolScore - GapPenaltyScore$$

Where SymbolScore is the sum of all pairwise symbol matches, which is specified by a similarity matrix (such as PAM) and blocks substitution matrix (BLOSUM). These similarity matrices contain score for all possible matches and mismatches of amino acids symbols based on the frequency of occurrence of these changes in known protein sequence databases. Amino acids that are identical between two sequences at a particular location receive the highest score, whereas an unlikely amino acid mismatch at a given position will have lowest score.

$$\text{SymbolScore} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} PAM(li, lj)$$

4. SAGA: Genetic Algorithm for Sequence Alignment

SAGA is a genetic algorithm dedicated to multiple sequence alignment. It follows the general principles of the simple genetic algorithms described by Goldberg [36]. In SAGA, each individual is a multiple alignment. The data structure chosen for the internal representation of an individual is a straightforward two-dimensional array where each line represents an aligned sequence and each cell is either a residue or a gap. The population has a constant size and does not contain any duplicate (i.e. identical individuals). The pseudo-code of the algorithm is reproduced on Figure 1.

4.1 Initialization

The challenge of the initialization (also known as seeding) is to generate a population as diverse as possible in terms of 'genotype' and as uniform as possible in terms of scores. In SAGA, generation 0 consists of a 100 multiple alignments randomly generated that only contain terminal gaps. These initial alignments are less than twice the length of the longest sequence of the set (longer alignments can be generated later). To create one of these individuals, a random offset is chosen for each sequence (between 0 and the length of the longest sequence); each sequence is shifted to the right, according to the offset and empty spaces are padded with null signs in order to give the same length L to all the sequences. Seeding can also be carried out by generating sub-optimal alignments using an implementation of dynamic programming that incorporates some randomness.

4.2 Operators

As mentioned earlier, the design of an adequate set of operators has been the main point of focus in the work that lead to SAGA. According to the traditional nomenclature of genetic algorithms, two types of operators coexist in SAGA: crossover and mutation. An operator is designed as an independent program that inputs one or two alignments (the parents) and outputs one alignment (the child). Each operator requires one or more parameters that specify how the operation is to be carried out. For instance, an operator that inserts a new gap requires three parameters: the position of the insertion, the index of sequence to modify and the length of the insertion.

These parameters may be chosen completely at random (in some pre-defined range). In that case, the operator is used in a stochastic manner. Alternatively, all but one of the parameters may be chosen randomly, leaving the value of the remaining parameter to be fixed by exhaustive examination of all possible values. The value that yields the best fitness is kept.

Crossover Operators: Crossovers are meant to generate a new alignment by combining two existing ones. Two types of crossover coexist in SAGA: the one point crossover that combines two parents through a single point of exchange and the uniform crossover that promotes multiple exchanges between two parents by swapping blocks between consistent bits. The uniform crossover is much less disruptive than its one-point counterpart, but it can only be applied if the two parents share some consistency, a condition rarely met in the early stages of the search. Of the two children produced by a crossover, only the fittest is kept and inserted into the new population (if it is not a duplicate). Crossovers are essential for promoting the exchange of high quality blocks within the population. They make it possible to efficiently use existing diversity. However, the blocks present in the original population only represent a tiny proportion of all the possibilities. They may not be sufficient to reconstruct an optimal alignment, and since crossovers cannot create new blocks, another class of operators is needed: mutation.

Mutation Operators: SAGA has multiple mutation operators described in [24]. However the gap insertion operator deserves special attention. With this operator an attempt is made to reconstitute backward some of the events of insertion/deletions through which a set of sequences might have evolved. The aligned sequences are split into two groups. Within each group, every sequence receives a gap insertion at the same position. Groups are chosen by randomly splitting an estimated phylogenetic tree (as given by ClustalW [37]). In the stochastic version, the length of the inserted gaps and the two insertion positions are randomly chosen while in the semi-hill climbing mode the second insertion position is chosen by exhaustively trying all the possible positions and comparing the scores of the resulting alignments.

4.3 Evaluation

Fitness in SAGA is measured by scoring each alignment according to the chosen objective function. The better the alignment, the better its score and the higher its fitness. To minimize sampling errors, raw scores are turned into a normalized value known as the expected offspring (EO). The EO indicates how many children an alignment is likely to have. In SAGA, EOs are stochastically derived using a predefined recipe: 'the remainder stochastic sampling without replacement'. This gives values that are typically between 0 and 2. Only the weakest half of the population is replaced with the new offspring while the other half is carried over unchanged to the next generation. This practice is known as overlapping generations.

It is during the breeding that new individuals (children) are generated. The EO is used as a probability for each individual to be chosen as a parent. This selection is carried out by weighted wheel selection without replacement and an individual's EO is decreased by one unit each time it is chosen to be a parent. An operator is also chosen and applied onto the parent(s) to create the newborn child. Twenty-two operators are available in SAGA. They all have their own usage probability and can be divided in two categories: mutations that only require one parent and crossovers that require two parents. Since no duplicate is allowed in the population, a newborn child is only accepted if it differs from all the other members of the generation already created. When a duplicate arises, the whole series of operations that lead to its creation is canceled. Breeding is over when the new generation is complete, and SAGA proceeds toward producing the next generation unless the finishing criterion is met.

Conditions that could guarantee optimality are not met in SAGA and there is no valid proof that it may reach a global optimum, even in an infinite amount of time (as opposed to SA). For that reason an empirical criterion is used for termination: the algorithm terminates when the search has been unable to improve for more than 100 generations. That type of stabilization is one of the most commonly used condition to stop a GA when working on a population with no duplicate (i.e. a population where all the individuals are different from one another).

5. Discussion: Comparison with Traditional Algorithms

The primary advantage behind an evolutionary algorithm design is that it enables a robust platform, which any objective function could be tested in a seamless manner. Such a system allows evaluation of functions that are biologically relevant and those that are not.

For instance, let us consider the popular weighted sums of pairs. It owes its popularity to the fact that algorithmic methods exist that allow its approximate optimization. MSA [3] is an algorithm that makes it possible to deliver an optimal (or a very close suboptimal) multiple sequence alignment using the sums of pairs measure. This sophisticated heuristic performs multi-dimensional dynamic programming in a bounded hyper-space. It is possible to assess the level of optimization reached by SAGA and MSA-EA V by comparing it to MSA while using exactly the same objective function.

The sums-of-pairs principle is to associate a cost to each pair of aligned residues in each column of an alignment (substitution cost), and another similar cost to the gaps (gap cost). The sum of these costs yields the global cost of the alignment. Major variations involve: i) using different sets of costs for the substitutions; ii) different schemes for the scoring of gaps; iii) different sets of weights associated with each pair of sequence. Formally, one can define the cost of a multiple alignment (A) as:

$$\textbf{ALIGNMENT COST (A)} = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} Wi,j \;\; \textbf{\textit{COST (Ai, j)}}$$

Where N is the number of sequences, Ai the aligned sequence i, COST is the alignment score between two aligned sequences (Ai and Aj) and Wi,j is the weight associated with that pair of sequences. The COST includes the sum of the substitution costs as given by a substitution matrix and the cost of the insertions/deletions using a model with affine gap penalties (a gap opening penalty and a gap extension penalty). Two schemes exist for scoring gaps: natural affine gap penalties and quasi-natural affine gap penalties [1].

Below a summary of comparison between MSA & SAGA and MSA & MSA-EA is given with respect to the weighted sum of pairs objective function.

5.1 MSA & SAGA

Using SAGA, it was possible to design new objective functions that make use of more complex gap penalties, take into account non-local dependencies or use position specific scoring schemes and to ask if this increased sophistication results in an improvement of the alignments biological quality.

It is common practice to validate a new method by comparing the alignments it produces with references assembled by experts. In the case of multiple alignments, one often uses structure based sequence alignments that are regarded as the best standard of truth available. For SAGA, validation was carried out using 3Dali [38]. Biological validation should not be confused with the mathematical validation also required for an optimization method. In the case of SAGA, both validations were simultaneously carried out.

Firstly, SAGA was used to optimize the sums of pairs with quasi-natural gap penalties, using MSA derived alignments as a reference. In two thirds of the cases, SAGA reached the same level of optimization as MSA. In the remaining test sets, SAGA outperformed MSA, and in every case that improvement correlated with an improvement of the alignment biological quality, as judged by comparison with a reference alignment. This suggests that SAGA is an adequate optimization tool that competes well with the most sophisticated heuristics.

In a second aspect of that validation, SAGA was used to align test cases too large to be handled by MSA, and using as an objective function the weighted sums of pairs with natural gap penalties. ClustalW was the nonstochastic heuristic used as a reference. The use of natural penalties lead to some improvement over the optimization reached by ClustalW, and that mathematical improvement was also correlated with a biological improvement. Altogether, these results are indicative of the versatility of SAGA as an optimizer and of its ability to optimize functions that are beyond the scope of standard dynamic programming based algorithmic methods.

5.2 MSA & MSA-EA

MSA-EA and Clustal V were compared with respect to the three data sets mentioned below.

Data Set	**N**	**LSEQ avg (min, max)**
Histone H4	71	101.1 (71, 107)
Globin	12	146 (136, 153)
Cytochrome C	6	108.0 (82, 135)

N = Number of Sequences

LSEQ = Length of Sequences

Three sets of MSA data were used. First 71 protein sequences were used from histone H4, which is an essential protein for the folding of DNA into chromatin and is considered to be one of the most highly conserved proteins known. For this reason, alignment of histone H4 should be an easy task for both Clustal V and the EA. The second data set contained 12 sequences of proteins from the globin family, which were also used in Notredame et. al. Globins are also conserved across storage and transport. Third, 6 sequences of the protein cytochrome C were used. The proteins are involved in electron transport and are also quite well conserved. All the protein sequences were collected from Protein Data Bank (PDB)

In all experiments, the characteristic performance differences between the MSA EA and Clustal V was observed. A typical observation was that the MSA EA made rapid improvements during the first 20000 fitness evaluations. Further improvements happened throughout the entire evolutionary optimization process although the EA slowly seemed to settle on a good alignment and only improved the alignment once in a while during the end of the run. However their fitness curve shows that the evolutionary process was far from stagnation by the end of the run and additional evaluations could have improved the results.

6. Conclusion

As discussed in the previous sections, Genetic Algorithms (such as MSA-EA and SAGA) are able to solve very complex optimization problems with a reasonable level of accuracy. This clearly indicates the importance and the interest of these methods in the field of sequence analysis. However currently genetic algorithms in this context suffer from two major drawbacks: unreliability and lack of speed.

6.1 Unreliable

Given a set of sequences, a general Genetic Algorithm may not deliver twice the same answer, owing to the stochastic nature of the optimization process and to the difficulty of the optimization. This may be a great cause of concern to the average biologist who expects to use his multiple alignment as a prediction tool and possibly as a decision aide for the design of expensive wet lab experiments. For example, if we consider the protein test cases analyzed here, SAGA reaches its best score in half of the runs on average. Same problem can be seen with MSA-EA case as well. If one is only interested in validating a new objective function, this is not a major source of concern since even in the worse cases the sub-optimal solutions are within a few percent of the best found solution.

6.2 Slow

More than instability, genetic algorithms currently suffer from another drawback: computation efficiency. A common problem with almost all Genetic Algorithm approaches for sequence alignment has been the long computational time required for useful results. When starting with a random initialization, evolutionary approaches can take hours to search for useful alignments, whereas Clustal V and Clustal W only require a matter of seconds to achieve similar results on sequences of reasonable length and percent identity. In a real-world scenario the user does not want to wait several hours for a good alignment, he/she wants a reasonable alignment right away and will usually be satisfied with the ones provided by the Clustal series. In the case of MSA-EA, the solution to this problem was to use the Clustal solution as a seed in the initial population of candidate alignments. Although this bears the risk of misguiding the optimization process toward local optima, it ensures that the EA solution will be at least as good as the one provided by Clustal. In case of SAGA, using Clustal as the seed solution was not feasible as SAGA is a more general Evolutionary Algorithm based approach. However both MSA-EA and SAGA are designed for parallel implementation, which can improve the speed significantly.

In spite of the drawbacks of Genetic Algorithms in multiple sequence alignment, there are at least two important fields of application exist for which they uniquely suited. The first one is the analysis of rare and very complex problems for which no other alternative is available, such as the folding of very long RNAs. Secondly, Genetic Algorithms provide us with a unique way of probing very complex problems with little concern; even with a very simple GA one can quickly ask very important questions and decide weather a thread of investigation is worth being pursued or should simply be abandoned.

7. Figure 1: Pseudo-Code of MSA-EA Algorithm

This is the pseudo-code of MSA-EA [35]

```
Begin
    Initialize with Clustal's seed and random individuals
    Cleanup gap columns
    Evaluate
    While (not termination-condition) do
        Begin
            Mutate individuals
            Recombine individuals
            Clean up gap columns
            Evaluate
            Selection
        End
    End
```

8. Figure 2: Pseudo-Code of SAGA Algorithm

This is a layout of the SAGA Algorithm. This pseudo-code indicates the main steps that take place during the optimization [24]

Initialization:	1. Create G0, an initial random population
Selection:	2. Evaluate the population of n(Gn)
	3. If the population is stabilized than END
	4. Select the individuals to replace
	5. Evaluate the expected offspring
Variation:	6. Select the parent(s) from Gn
	7. Select an operator
	8. Generate the offspring
	9. Keep or discard new offspring in Gn+1
	10. Go to step 6 until all Gn+1 is complete
	11. n = n +1
	12. Go to step 2
End:	13. End

9. References

[1] S Henikoff and J.G Henikoff, Amino Acid substitution matrices for protein blocks, Proc. Natl. Acad. Sci., 89 (1992), pp. 10915-10919.

[2] S. A. Benner, M. A. Cohen and G. H. Gonnet, Response to Barton's letter, Science, 257 (1992), pp. 1609-1610.

[3] D. J. Lipman, S. F. Altschul and J. D. Kececioglu, A Tool for multiple sequence alignment, Proc. Natl. Acad. Sci. USA, 86 (1989), pp. 4412-4415.

[4] P. Hogeweg and B. Hesper, CLUSTAL: A package for performing multiple sequence alignment on a microcomputer, Gene, 73 (1988), pp 237-244, J. Mol. Evol., 20 (1984), pp. 175-186.

[5] J. Stoye, V. Moulton and A. W. Dress, DCA: an efficient implementation of divide-and-conquer approaches to simultaneous multiple sequence alignment, Comput ApplBiosci, 13 (1997), pp. 625-6.

[6] W. R. Taylor, A flexible method to align large numbers of biological sequences, Journal of Molecular Evolution, 28 (1988), pp. 161-169.

[7] Lewis, P. O. (1998). A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data. Mol Biol Evol, 15(3), 277-283.

[8] Burke, D. S., De Jong, K. A., Grefenstette, J. J., Ramsey, C. L., & Wu, A. S. (1998). Putting more genetics into genetic algorithms. Evol Comput, 6(4), 387-410.

[9] Cui, Y., Chen, R. S., & Wong, W. H. (1998). Protein folding simulation with genetic algorithm and supersecondary structure constraints. Proteins, 31(3), 247-257.

[10] O. Gotoh, Significant Improvement in Accuracy of Multiple Protein Sequence Alignments by Iterative Refinements as Assessed by Reference to Structured Alignments, J. Mol. Biol., 264 (1996), pp. 823-838.

[11] Weber, L. (1998). Applications of genetic algorithms in molecular diversity. Curr Opin Chem Biol, 2(3), 381-385.

[12] Pedersen, J. T., & Moult, J. (1996). Genetic algorithms for protein structure prediction. Curr Opin Struct Biol, 6(2), 227-231.

[13] Lehtonen, J. V., Denessiouk, K., May, A. C., & Johnson, M. S. (1999). Finding local structural similarities among families of unrelated protein structures: a generic non-linear alignment algorithm. Proteins, 34(3), 341-355.

[14] Holland J.H., "Genetic Algorithms", Scientific American, pp. 66-72, July 1992.

[15] A. A. Rabow and H. A. Scheraga, Improved genetic algorithm for the protein folding problem by use of a cartesian combination operator, Protein Science, 5 (1996), pp. 1800-1815.

[16] J. Heringa, Two Strategies for sequence comparison: profile-processed and secondary structure-induced multiple alignments, Computers and Chemistry, 23 (1999), pp. 341-364.

[17] J. Kim, S. Pramanik and M. J. Chung, Multiple Sequence Alignment using Simulated Annealing, Comp. Applic. Biosci., 10 (1994), pp. 419-426.

[20] J. Kim, J. R. Cole and S. Pramanik, Alignment of possible secondary structures in multiple RNA sequences using simulated annealing, Comp. Applic. Biosci., 12 (1996), pp. 259-267.

[21] L. A. Anabarasu, Multiple Sequence Alignment using parallel genetic algorithm, The Second Asia-Pacific Conference on Simulated Annealing, Canberra, Australia 1998

[22] R. R. Gonzalez, Multiple Protein Sequence comparison by genetic algorithms, SPIE-98, 1999

[23] C. Zhang and A. K. Wong, A genetic algorithm for multiple molecular sequence alignment, Comput Appl Biosci, 13 (1997), pp. 565-81.

[24] C. Notredame and D. G. Higgins, SAGA: sequence alignment by genetic algorithm Nucleic Acids Res., 24 (1996), pp. 1515-1524.

[25] L. Cai, D. Juedes and E. Liakhovitch, Evolutionary Computation Techniques for multiple sequence alignment, Congress on evolutionary computation 2000, 2000, pp. 829-835.

[26] K. Chellapilla and G. B. Fogel, Multiple Sequence Alignment using evolutionary programming, Congress on Evolutionary Computation 1999, 1999, pp. 445-452.

[27] S. F. Altschul, Gap Costs for Multiple Sequence Alignment, J. Theor. Biol., 138 (1989), pp. 297-309.

[28] S. Kirkpatrick, C. D. J. Gelatt and M. P. Vecchi, Optimization by Simulated Annealing, Science, 220 (1983), pp. 671-680.

[29] Holland J.H., “Adaptation in Natural and Artificial Systems”, Ann Arbor: The University of Michigan Press, 1975.

[30] A. P. Gultayaev, F. D. H. van Batenburg and C. W. A. Pleij, Computer Simulation of RNA Folding Pathways Using Genetic Algorithm, J. Mol. Biol., 250 (1995), pp.37-51.

[31] B. A. Shapiro and J. C. Wu, Predicting RNA HType pseudonots with the massively parallel genetic algorithm, Comp. Applic. in Biosci., 13 (1997), pp. 459-471.

[32] R. Unger and J. Moult, Genetic Algorithms for Protein Folidng Simulation, J. Mol. Biol., 231 (1993), pp. 75-81.

[33] A. C. May and M. S. Johnson, Improvised genetic algorithm-based protein structure comparisons: pairwise and multiple superpositions, Protein Eng, 8 (1995), pp. 873-82.

[34] M. Ishikawa, T. Toya and Y. Tokoti, Parallel Iterative Aligner with Genetic Algorithm, Artificial Intelligence and Genome Workshop, 13th International Conference on Artificial Intelligence, Chambery, France, 1993, pp. 13-22.

[35] Thomsen, R., Fogel, G., B. and Krink, T. (2002). A Clustal Alignment Improver using Evolutionary Algorithms. In: Proceedings of the Fourth Congress on Evolutionary Computation (CEC-2002), vol. 1, p. 121-126

[36] D. E. Goldberg, genetic algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, New York, 1989.

[37] J. Thompson, D. Higgins and T. Gibson, CLUSATL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice, Nucleic Acids Res., 22 (1994), pp.4673-4690.

[38] S. Pascarella and P. Argos, $A data bank merging related protein structures and sequences,Protein Eng., 5 (1992), pp. 121-137.