Using Genetic Algorithms for Pairwise and Multiple Sequence Alignments

Cédric Notredame

Information Génétique et Structurale
CNRS-UMR 1889
31 Chemin Joseph Aiguier
13 006 Marseille
Email: cedric.notredame@igs.cnrs-mrs.fr

1 Introduction

1.1 Importance of Multiple Sequence Alignment

The simultaneous alignment of many nucleic acid or amino acid sequences is one of the most commonly used techniques in sequence analysis. Given a set of homologous sequences, multiple alignments are used to help predict the secondary or tertiary structure of new sequences [51]; to help demonstrate homology between new sequences and existing families; to help find diagnostic patterns for families [6]; to suggest primers for the polymerase chain reaction (PCR) and as an essential prelude to phylogenetic reconstruction [19]. These alignments may be turned into profiles [25] or Hidden Markov Models (HMMs) [27, 9] that can be used to scour databases for distantly related members of the family.

Multiple alignment techniques can be divided into two categories: global and local techniques. 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. They constitute a greater challenge since they increase the amount of decision made by the algorithm. Most multiple alignment methods are global, leaving it to the user to decide on the portion of sequences to be incorporated in the multiple alignment. To aid that decision, one often uses local pairwise alignment programs such as Blast [3] or Smith and Waterman [56]. In the context of this chapter, we will focus on global alignment methods with a special emphasis on the alignment of protein and RNA sequences.

Despite its importance, the automatic generation of an accurate multiple sequence alignment remains one of the most challenging problems in bioinformatics. The reason behind that complexity can easily be explained. A multiple alignment is meant to reconstitute relationships (evolutionary, structural, and functional) within a set of sequences that may have been diverging for millions and sometimes billions of years. To be accurate, this reconstitution would require an in-depth knowledge of the evolutionary history and structural properties of these sequences. For obvious reasons, this information is rarely available and generic empirical models of protein evolution [18, 28, 8], based on sequence similarity must be used instead. Unfortunately, these can prove difficult to apply when the sequences are less than 30% identical and lay within the so-called “twilight zone” [52]. Further, accurate optimization methods that use these models can be extremely demanding in computer resources for more than a handful of sequences [12, 62]. 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) this algorithm attempts to optimize.

1.2 Standard Optimization Algorithms

Optimization algorithms roughly fall in three categories: the exact, the progressive, and the iterative algorithms. *Exact algorithms* attempt to deliver an optimal or a sub-optimal alignment within some well defined bounds [40], [57]. Unfortunately, these algorithms have very serious limitations with regard to the number of sequences they can handle and the type of objective function they can optimize. *Progressive alignments* are by far the most widely used [30, 14, 45]. They depend on a progressive assembly of the multiple alignment [31, 20, 58] where sequences or alignments are added one by one so that never more than two sequences (or multiple alignments) are simultaneously aligned using dynamic programming [43]. This approach has the great advantage of speed and simplicity combined with reasonable sensitivity even if it is by nature a greedy heuristic that does not guarantee any level of optimization. *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 [7] [24] [29]. The procedure is terminated when no more improvement can be made (convergence). Stochastic iterative methods include HMM training [39], simulated annealing (SA) [37, 36, 33] and evolutionary computation such as genetic algorithms (GAs) [44, 47, 34, 64, 5, 23] and evolutionary programming [11, 13]. 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 and in our case, it is also the objective function that contains the biological knowledge one tries to project in the alignment.

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 [52] as long as their level of identity is above the so-called "twilight zone" (more than 30% identity over more than 100 residues).

Accurate measures of similarity are obtained using substitution matrices [18, 28]. A substitution matrix is a pre-computed table of numbers (for proteins, this matrix is 20*20, representing all possible transition states for the 20 naturally occurring amino acids) 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). Since any gap can be explained with one mutational event only, the aim of that scheme is to make sure that the best scoring evolutionary scenario involves only a small number of insertions or deletions (indels) in the alignment. This will result in an alignment with few long gaps rather than many short ones. The resulting score can be viewed as a measure of similarity between two sequences (pairwise). 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)[1]. 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. It is to counterbalance that effect that probability based schemes were introduced in the context of HMMs. Their purpose is to associate each column of an alignment with a generation probability [39]. Estimations are carried out in a Bayesian context where the model (alignment) probability is evaluated simultaneously with the probability of the data (the aligned sequences). In the end, the score of the complete alignment is set to be the probability of the aligned sequences to be generated by the trained HMM. The major drawbacks of this model are its high dependency on the number of sequences being aligned (i.e. many sequences are needed to generate an accurate model) and the difficulty of the training. More recently, new methods based on consistency were described for the evaluation of multiple sequence alignments. Under these schemes, the score of a multiple alignment is the measure of its consistency with a list of pre-defined constraints [42, 46, 10, 45]. It is common practice for these pre-defined constraints to be sets of pairwise, multiple or local alignments. Quite naturally, the main limitation of consistency-based schemes is that they make the quality of the alignment greatly dependent on the quality of the constraints it is evaluated against.

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. The mathematically optimal alignment should never be confused with the correct alignment, the biological optimum. While the biological optimum is by definition correct, a mathematically optimal alignment is biologically only as good as it is similar to the biological optimum. This depends entirely on the quality of the objective function that was used to generate it. In order to achieve this result, there is no limit to the complexity of the objective functions one may design, even if in practice the lack of appropriate optimization engines constitutes a major limitation.

What is the use of an objective function if one cannot optimize it and how is it possible to tell if an objective function is biologically relevant or not? Evolutionary algorithms come in very handy to answer these questions. 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, we introduce one of these evolutionary techniques known as genetic algorithms (GA). GAs are described along with another closely related stochastic optimization algorithm: simulated annealing. Three examples are reviewed in details, in which GAs were successfully applied to sequence alignment problems.

2 Evolutionary Algorithms and Simulated Annealing.

An evolutionary algorithm is a way of finding a solution to a problem by means of forcing sub-optimal solutions to evolve through some perturbations (mutations and recombination). Most evolutionary algorithms are stochastic in the sense that the solution space is explored in a random rather than ordered manner. In this context, randomness provides a non-null probability to sample any potential solution, regardless of the solution space size, providing that the mutations allow such an exploration. The drawback of randomness is that all potential solutions may not be visited during the search (including the global optimum). In order to correct for this problem, a large number of heuristics have been designed that attempt to bias the way in which the solution space is sampled. They aim at improving the chances of sampling an optimal solution. For that reason, most stochastic strategies (including evolutionary computation) can be regarded as a tradeoff between greediness and randomness. Two stochastic strategies have been widely used for sequence analysis: simulated annealing and genetic algorithms. Strictly speaking, SA does not belong to the field of evolutionary computation, yet, in practice, it has been one of the major source of inspiration for the elaboration of genetic algorithms used in sequence analysis.

2.1 Simulated Annealing

Simulated annealing (SA) [38] was the first stochastic algorithm used to attempt solving the multiple sequence alignment problem [33, 37]. 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, it replaces it, 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.

While the principle is very sound and the adequacy to multiple alignment optimization and objective function evaluation is obvious, SA suffers from a very serious drawback: it is extremely slow. Most of the studies conducted on simulated annealing and multiple alignments concluded that although it does reasonably well, SA is too slow to be used for *ab-initio* multiple alignments and must be restricted to being used as an alignment improver (i.e. improvement of a seed alignment). 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. 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. One of the reason why these algorithms have received so little attention in the context of multiple sequence alignment is probably due to the fact that the 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 [26, 55, 48] and protein structure analysis [53, 60, 41]. 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.

Although one could argue on who exactly invented GAs, the algorithms we use today were formally introduced by Holland in 1975 [32] and later refined by Goldberg to give the Simple Genetic Algorithm[22]. GAs are based on a loose analogy with the phenomenon of natural selection. Their principle is relatively straightforward. Given a problem, potential solutions (individuals within a population) compete with one another (selection) for survival. These solutions can also evolve: they can be modified (mutations), or combined with one another (crossovers). The idea is that acting together, variation and selection will lead to an overall improvement of the population via evolution. Most of the concepts developed here about GAs are taken from [22, 16].

Two ingredients are essential to the GA strategy: the selection method and the operators. Selection is established in order to lead the search toward improvement. It means that the best individuals (as judged using the objective function) must be the most likely to survive. To serve the GA purpose, this selection strategy cannot be too strict. It must 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.

Given such a crude layout, the potential for variation is infinite and the study of new GA models is a very active field in its own right. This being said, the main difficulty to overcome when adapting a GA to a problem like multiple sequence alignment is not the choice of a proper model, but rather the conception of a well suited series of operators. This is a well known problem that has also received some attention in the field of structure prediction both for proteins [50] and RNA [54]. A simple justification is that the operators (and the problem representation) largely control the manner in which a solution landscape is being analyzed. For instance, the neighborhood of a solution is mostly defined by the exploration capabilities of the operators. Well chosen they can smoothen very rugged landscapes. Yet, on the other hand, if they are too ’smart’ and too greedy, they may prevent a proper exploration to be carried out. Finding the right trade off can prove rather a complex task. When applying GAs to sequence alignments, previous work on SA proved instrumental. It provided researcher with a set of operators well tested and perfectly suitable for integration within most evolutionary algorithms.

Attempts to apply evolutionary algorithms to the multiple sequence alignment problem started in 1993 when Ishikawa et al. published a hybrid GA [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 GA able to deal with sequences in a more general manner was described a few years later by Notredame and Higgins[44], shortly before a similar work by Zhang [64]. 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. In 1997, SAGA was applied to RNA analysis [47] and parallelized for that purpose using an island model. This work was later duplicated by Anabrasu et al. [5] who carried out an extensive evaluation of this model, using ClustalW as a reference. Over the following years, at least three new multiple sequence alignment strategies based on evolutionary algorithms have been introduced [23], [13] and [11]. 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 order to expose each of the many ingredients that constitute a GA specialized in sequence alignments, the example of SAGA will now be reviewed in details.

3 SAGA: a GA Dedicated to Sequence Alignment.

3.1 The Algorithm.

SAGA is a genetic algorithm dedicated to multiple sequence alignment [44]. It follows the general principles of the simple genetic algorithms (sGA) described by Goldberg [22] and later refined by Davis [17]. 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 given on figure 1. Each of these steps is developed over the next sections.

3.1.1 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. This is the case in RAGA [47], an implementation of SAGA specialized in RNA alignment.

3.1.2 Evaluation

Fitness 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 (scores are inverted if the OF is meant to be minimized). 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 pre-defined recipe: 'the remainder stochastic sampling without replacement' [22]. 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 [16].

3.1.3 Breeding

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 [22] 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.

3.1.4 Termination

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) [17].

3.2 Designing the 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 [22], 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 said to be used in a *stochastic* manner [44]. 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. An operator applied this way is said to be used in a *semi-hill climbing* mode. Most operators may be used either way (stochastic or semi-hill climbing). For the robustness of the GA, it is also important to make sure that the operators are completely independent from any characteristic of the objective function, unless one is interested in creating a very specific operator for the sake of efficiency.

.

3.2.1 The Crossovers

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 (Figure 2a) and the uniform crossover that promotes multiple exchanges between two parents by swapping blocks between consistent bits (Figure 2b). 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.

3.2.2 Mutations: Example of the Gap insertion Operator.

SAGA’s mutation operators are extensively described in [44]. We will only review here the gap insertion operator, a crude attempt to reconstitute backward some of the events of insertion/deletions through which a set of sequences might have evolved. When that operator is applied, alignments are modified following the mechanism shown on Figure 3. 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 [59]). The stochastic and the semi-hill climbing versions of this operator are implemented. 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.

3.2.3 Dynamic Scheduling of the Operators

When creating a child, the choice of the operator is just as important as the choice of the parents. Therefore, it makes sense to allow operators to compete for usage, just as the parents do for survival, in order to make sure that useful operators are more likely to be used. Since one cannot tell in advance the good operators from the bad ones, they initially all receive the same usage probability. Later during the run, these probabilities are dynamically reassessed to reflect each operator individual performances. The recipe used in SAGA is the dynamic scheduling method described by Davis [16]. It easily allows the adding and removal of new operators without any need for retuning. Under this model, an operator has a probability of being used that is a function of its recent efficiency (i.e. improvement generated over the 10 last generations). The credit an operator gets when performing an improvement is also shared with the operators that came before and may have played a role in this improvement. Thus, each time a new individual is generated, if it yields some improvement over its parents, the operator that was directly responsible for its creation gets the largest part of the credit (e.g. 50%); then the operator(s) responsible for the creation of the parents also get their share on the remaining credit (50% of the remaining credit); and so on. This credit report goes on for some specified number of generation (e.g. 4). Every 10 generations, results are summarized for each operator and the usage probabilities are reassessed based on the accumulated credit. To avoid early loss of some operators, each of them has a minimum usage probability higher than 0. It is common practice to set these minimal usage probabilities so that they sum to 0.5. To that effect one can use for each operator a minimum probability of 1/(2*number of operators)).

A very interesting property of that scheme is that it helps using operators only when they are needed. For instance, uniform crossovers are generally more efficient than their one point counterpart. Unfortunately, they cannot be properly used in the optimization early stages because at that point there is not enough consistency within the population. The dynamic scheduling adapts very well to that situation by initially giving a high usage probability to the one point crossover, and by shifting that credit to the uniform crossover when the population has become tidy enough to support its usage. It is interesting to notice that these two operators are competing with one another although the GA does not explicitly know they both belong to the crossover category.

3.3 Parallelisation of SAGA

Long running times were SAGA's main limitation. This became especially acute when aligning very long sequences such as ribosomal RNAs (>1000 nucleotides). It is common practice to use parallelisation in order to alleviate such problems. The technique applied on SAGA is specific of GAs and known as an island parallelisation model [21]. Instead of having a single GA running, several identically configured GAs run in parallel on separate processors. Every 5 generations they exchange some of their individuals. The GAs are arranged on the leaves and the nodes of a *k*-branched tree and the population exchange is unidirectional from the leaves to the root of the tree (Figure 4). By default, the individuals migrating from one GA to another are those having the best score. The GA node where they come from keeps a copy of them but they replace low scoring individuals in the accepting GA [44]. Initially implemented in RAGA, the RNA version of SAGA, this model was also extended to SAGA, using a 3-branched trees with a depth of 3 that requires 13 GAs. These processes are synchronous and wait for each other to reach the same generation number before exchanging populations.

This distributed models benefits from the explicit parallelisation and is about 10 times faster than a non-parallel version (i.e. about 80% of the maximum speedup expected when distributing the computation over 13 processors). It also benefits from the new constraints imposed by the tree topology on the structure of the population. It seems to be the lack of feedback that makes it possible to retain within the population a much higher degree of diversity than a single unified population could afford. These are the terminal leaves that behave as a diversity reservoir and yield a much higher accuracy to the parallel GA than a non-parallel version with the same overall population. Nonetheless, these preliminary observations remain to be firmly established, using some extra thorough benchmarking.

4 Applications: Choosing an Objective Function

The main motivation behind SAGA's design was the creation of a robust platform or a black box on which any OF one could think of could be tested in a seamless manner. Such a black box allows discriminating between the functions that are biologically relevant and those that are not. For instance, let us consider the weighted sums of pairs. This function is one of the most widely used. It owes its popularity to the fact that algorithmic methods exist that allow its approximate optimization [43, 40]. Yet we know this function is not very meaningful from a biological point of view [4]. The three main limitations that have caught biologists' attention are the crude modeling of the insertions/deletions (gap), the assumed independence of each position and the fact that the evaluation cannot be made position dependant.

Thanks to 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. The following section reviews three classes of objective functions that were successfully optimized using SAGA [44, 47, 46].

4.1 The Weighted Sums of Pairs

MSA [40] is an algorithm that makes it possible to deliver an optimal (or a very close sub-optimal) 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 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 (PAM matrices [18] or BLOSUM tables [28]); ii) different schemes for the scoring of gaps [1]; iii) different sets of weights associated with each pair of sequence [2]. Formally, one can define the cost of a multiple alignment (*A*) as:

$$ALIGNMENT\ COST\ (A) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} W_{i,\,j} COST\ (A_i,\ A_j)$$

Where *N* is the number of sequences, A_i the aligned sequence *i*, *COST* is the alignment score between two aligned sequences (A_i and A_j) and $W_{i,j}$ 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]. Quasi-natural gap penalties are the only scheme that the MSA program can efficiently optimize. This is unfortunate since these penalties are known to be biologically less accurate than their natural counterparts [1] because of a tendency to over-estimate the number of gaps. Under both scheme, terminal gaps are penalized for extension but not for opening.

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 [24]. For SAGA, validation was carried out using 3Dali [49]. 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, and a summary of the results obtained when optimizing the sums of pairs is shown of Table 1.

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. Although they fall short of a demonstration, these figures suggest 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 non-stochastic heuristic used as a reference. As expected, 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.

4.2 Consistency Based Objective Functions: The COFFEE Score

Ultimately, a multiple alignment aims at combining within a single unifying model every piece of information known about the sequences it contains. However, it may be the case that a part of this information is not as reliable as one may expect. It may also be the case that some elements of information are not compatible with one another. The model will reveal these inconsistencies and require decisions to be made in a way that takes into account the overall quality of the alignment.

A new objective function can be defined that measures the fit between a multiple alignments and the list of weighted elements of information. Of course, the relevance of that objective function will depend greatly on the quality of the pre-defined list. This list can take whatever forms one wishes. For instance, a convenient source is the generation of a list of pair wise alignments [46, 45] that given a set of N sequences will contain all the N^2 possible pair wise alignments. In the context of COFFEE (Consistency Based Objective Function For alignmEnt Evaluation), that list of alignments is named a library, and the COFFEE function measures the level of consistency between a multiple alignments and its corresponding library. Evaluation is made by comparing each pair of aligned residues observed in the multiple alignments to the list of residue pairs that constitute the library. During the comparison, residues are only identified by their index within the sequences. The consistency score is equal to the number of pairs of residues that are simultaneously found in the multiple alignment and in the library, divided by the total number of pairs observed in the multiple sequence alignment. The maximum is 1 but the real optimum depends on the level of consistency found within the library. To increase the biological relevance of this function, each pair of residues is associated with a weight indicative of the quality of the pair-wise alignment it comes from (a measure of the percentage of identity between the two sequences).

The COFFEE function can be formalized as follows. Given N aligned sequences $S_1...S_N$ in a multiple alignment. $A_{i,j}$ is the pair wise projection (obtained from the multiple alignment) of the sequences S_i and S_j. $LEN\ (A_{i,j})$ is the number of ungapped columns in this alignment. $SCORE\ (A_{i,j})$ is the overall consistency between $A_{i,j}$ and the corresponding pair-wise alignment in the library and $W_{i,j}$ is the weight associated with this pair-wise alignment.

$$\text{COFFEE score}=\Big[\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} W_{i,j} * \text{ SCORE } (A_{i,j})\Big] / \Big[\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} W_{i,j} * \text{LEN}(A_{i,j})\Big]$$

If we compare this function to the weighted sums of pairs developed earlier, we will find that the main difference is the library that replaces the substitution matrix and provides a position dependant mean of evaluation. It is also interesting to note that under this formulation an alignment having an optimal COFFEE score will be equivalent to a Maximum Weight Trace alignment using a 'pair-wise alignment graph' [35].

Table 2 shows some of the results obtained using SAGA/COFFEE on 3Dali. For that experiment, the library of pair wise alignments had been generated using ClustalW alignments, and the resulting alignments proved to be of a higher biological quality than those obtained with alternative methods available at the time. Eventually, these results were convincing enough to prompt the development of a fast non-GA based method for the optimization of the COFFEE function. That new algorithm, named T-Coffee, was recently made available to the public [45].

4.3 Taking Non-Local Interactions Into Account: RAGA.

So far, we have reviewed the use of SAGA for sequence analysis problems that consider every position as independent from the others. While that approximation is acceptable when the sequence signal is strong enough to drive the alignment, this is not always the case when dealing with sequences that have a lower information content than proteins but carry explicit structural information, such as RNA or DNA. To illustrate one more usage of GAs it will now be interesting to examine a case where SAGA was used to optimize an RNA structure super-imposition in which the OF takes into account local and non-local interactions altogether. RNA was chosen because its fold, largely based on Watson and Crick base pairings [63], generates characteristic structures (stems-loops) that are easy to predict and analyze [65]. Since the pairing potential of two RNA bases can be predicted with reasonable accuracy, the evaluation of an alignment can easily take into account structure (*Se*) and sequence (*Pr*) similarities altogether. The version of SAGA in which that new function is implemented is named RAGA (RNA Alignment by Genetic Algorithm) [47]. In RAGA, the OF evaluates the alignment of two RNA sequences, one with a known secondary structure (master) and one that is homologous to the master but whose exact secondary structure is unknown (slave). It can be formalized as follow:

$$\textit{Alignment score} = Pr + (\lambda * Se) - \textit{Gap Penalty} \qquad (2)$$

λ is a constant (1-3) and *Gap penalty* is the sum of the affine gap penalties within the alignment. *Pr* is simply the number of identities. *Se* uses the secondary structure of the master sequence and evaluates the stability of the folding it induces onto the slave sequence. If two bases form a base pair (part of a stem) in the master, then the two 'slave' bases they are aligned to should be able to form a Watson and Crick base pair as well. *Se* is the sum of the score of these induced pairs. The energetic model used in RAGA is very simplified and assigns 3 to GC pairs and 2 to UA and UG.

Assessing the accuracy and the efficiency of RAGA is a problem very similar to the one encountered when analyzing SAGA. In this case, the reference alignments were chosen from mitochondrial ribosomal small subunit RNA sequence alignments established by experts [61]. The human sequence was used as a master and realigned by RAGA to seven other homologous mitochondrial sequences used as slaves. Evaluation was made by comparing the optimized pairwise alignments to those contained in the reference alignment. The results on Table 3 indicate very clearly, that a proper optimization took place and that the secondary structure information was efficiently used to enhance the alignment quality. This is especially sensible for very divergent sequences that do not contain enough information at the primary level for an accurate alignment to be determined on these bases alone. It is also interesting to point out that RAGA could also take into account some elements of the tertiary structure known as pseudoknots, that were successfully added to the objective function. These elements, that are beyond the scope of most dynamic programming based methods, lead to even more accurate alignment optimization [47].

5 Conclusion: GA versus Heuristic Methods.

Section 4 of this chapter illustrates three situations in which GAs proved able to solve very complex optimization problems with a reasonable level of accuracy. On its own, this clearly indicates the importance and the interest of these methods in the field of sequence analysis. Yet, when applied to that type of problems, GAs suffer from two major drawbacks: they are very slow and unreliable. By unreliable, we mean that given a set of sequences, a GA 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. How severe is this problem? If we consider the protein test cases analyzed here, SAGA reaches its best score in half of the runs on average. For RAGA, maybe because the solution space is more complex, this proportion goes down to 20%. 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. However, this instability is not unique to GAs and is not as severe as the second major drawback: the efficiency. Although much more practical than SA, GAs slowness means that they cannot really be expected to become part of any of the very large projects that require millions of alignments to be routinely made over a few days [15]. More robust, if less accurate, techniques are required for that purpose.

Is the situation hopeless then? The answer is definitely no since two important fields of application exist for which GAs are 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. The second aspect is more general. GAs provide us with a unique way of probing very complex problems with little concern, at least in the first stages, for the algorithmic issues involved. It is quite remarkable that 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.

The COFFEE project is a good example of such a cycle of analysis. It followed this three steps process: (i) an objective function was first designed without any concern for the complexity of its optimization and the algorithmic issues. (ii) SAGA was used to evaluate the biological relevance of that function. (iii) This validation was convincing enough to prompt the conception of a new dynamic programming algorithm, much faster and appropriate for this function[45]. This non-GA based algorithm was named T-Coffee (Tree based COFFEE). The mere evocation of these two projects respective developing time makes a good case for the use of SAGA: the COFFEE project took four months to be carried out, while completion of the T-Coffee project required more than a year and a half for algorithm development and software engineering.

6 Availability

SAGA, RAGA, COFFEE and T-Coffee are all available free of charge from the author either via Email (cedric.notredame@igs.cnrs-mrs.fr) or via the WWW (http://igs-server.cnrs-mrs.fr/~cnotred).

7 Acknowledgements

The author wishes to thank Dr Hiroyuki Ogata and Dr Gary Fogel for very helpful comments and for an in-depth review of the manuscript.

8 References

[1] S. F. Altschul, *Gap costs for multiple sequence alignment*, J. Theor. Biol., 138 (1989), pp. 297-309.

[2] S. F. Altschul, R. J. Carroll and D. J. Lipman, *Weights for data related by a tree*, Journal of Molecular Biology, 207 (1989), pp. 647-653.

[3] S. F. Altschul, W. Gish, W. Miller, E. W. Myers and D. J. Lipman, *Basic local alignment search tool*, Journal of Molecular Biology, 215 (1990), pp. 403-410.

[4] S. F. Altschul and D. J. Lipman, *Trees, stars, and multiple biological sequence alignment*, SIAM J. Appl. Math., 49 (1989), pp. 197-209.

[5] L. A. Anabarasu, *Multiple sequence alignment using parallel genetic algorithms, , The Second Asia-Pacific Conference on Simulated Evolution (SEAL-98)*, Canberra,australia, 1998.

[6] A. Bairoch, P. Bucher and K. Hofmann, *The PROSITE database, its status in 1997*, Nucleic Acids Research, 25 (1997), pp. 217-221.

[7] G. J. Barton and M. J. E. Sternberg, *A strategy for the rapid multiple alignment of protein sequences: confidence levels from tertiary structure comparisons*, Journal of Molecular Biology, 198 (1987), pp. 327-337.

[8] S. A. Benner, M. A. Cohen and G. H. Gonnet, *Response to Barton's letter: Computer speed and sequence comparison*, Science, 257 (1992), pp. 1609-1610.

[9] P. Bucher, K. Karplus, N. Moeri and K. Hofmann, *A flexible motif search technique based on generalized profiles*, Comput Chem, 20 (1996), pp. 3-23.

[10] K. Bucka-Lassen, O. Caprani and J. Hein, *Combining many multiple alignments in one improved alignment*, Bioinformatics, 15 (1999), pp. 122-30.

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

[12] H. Carrillo and D. J. Lipman, *The multiple sequence alignment problem in biology*, SIAM J. Appl. Math., 48 (1988), pp. 1073-1082.

[13] K. Chellapilla and G. B. Fogel, *Multiple sequence alignment using evolutionary programming, , Congress on Evolutionary Computation 1999*, 1999, pp. 445-452.

[14] F. Corpet, *Multiple sequence alignment with hierarchical clustering*, Nucleic Acids Res., 16 (1988), pp. 10881-10890.

[15] F. Corpet, F. Servant, J. Gouzy and D. Kahn, *ProDom and ProDom-CG: tools for protein domain analysis and whole genome comparisons*, nucleic acids res, 28 (2000), pp. 267-9.

[16] L. Davis, *The handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

[17] P. J. Davis and R. Hersh, *The mathematical experience*, Birkauser, Boston, 1980.

[18] M. O. Dayhoff, *Atlas of Protein Sequence and Structure*, National Biomedical Research Foundation, Washington, D. C., U. S. A., 1978.

[19] J. Felsenstein, *PHYLIP: phylogeny inference package*, Cladistics, 5 (1988), pp. 355-356.

[20] D.-F. Feng and R. F. Doolittle, *Progressive sequence alignment as a prerequisite to correct phylogenetic trees*, Journal of Molecular Evolution, 25 (1987), pp. 351-360.

[21] A. L. Goldberg and R. E. Wittes, *Genetic code: aspects of organization*, Science, 153 (1966), pp. 420-424.

[22] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, New York, 1989.

[23] R. R. Gonzalez, *Multiple protein sequence comparison by genetic algorithms, , SPIE-98*, 1999.

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

[25] M. Gribskov, M. McLachlan and D. Eisenberg, *Profile analysis: Detection of distantly related proteins*, Proceedings of the National Academy of Sciences, 84 (1987), pp. 4355-5358.

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

[27] D. Haussler, A. Krogh, I. S. Mian and K. Sjölander, *Protein Modeling using Hidden Markov Models: Analysis of Globins*, in L. Hunter, ed., *Proceedings for the 26th Hawaii International Conference on Systems Sciences*, Los Alamitos, CA: IEEE Computer Society Press, Wailea, HI, U.S.A., 1993, pp. 792-802.

[28] S. Henikoff and J. G. Henikoff, *Amino acid substitution matrices from protein blocks*, Proc. Natl. Acad. Sci., 89 (1992), pp. 10915-10919.

[29] J. Heringa, *Two strategies for sequence comparison: profile-prprocessed and secondary structure-induced multiple alignment*, Computers and Chemistry, 23 (1999), pp. 341-364.

[30] D. G. Higgins and P. M. Sharp, *CLUSTAL: a package for performing multiple sequence alignment on a microcomputer*, Gene, 73 (1988), pp. 237-244.

[31] P. Hogeweg and B. Hesper, *The alignment of sets of sequences and the construction of phylogenetic trees. An integrated method*, J. Mol. Evol., 20 (1984), pp. 175-186.

[32] J. H. Holland, *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbour, MI, 1975.

[33] M. Ishikawa, T. Toya, M. Hoshida, K. Nitta, A. Ogiwara and M. Kanehisa, *Multiple sequence alignment by parallel simulated annealing*, Comp. Applic. Biosci., 9 (1993), pp. 267-273.

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

[35] J. D. Kececioglu, *The maximum weight trace problem in multiple sequence alignment*, Lecture Notes in Computer Science, 684 (1983), pp. 106-119.

[36] 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.

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

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

[39] A. Krogh, M. Brown, I. S. Mian, K. Sjölander and D. Haussler, *Hidden Markov Models in Computational Biology: Applications to Protein Modeling*, J. Mol. Biol., 235 (1994), pp. 1501-1531.

[40] 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.

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

[42] B. Morgenstern, A. Dress and T. Wener, *Multiple DNA and Protein sequence based on segment-to-segment comparison*, Proc. Natl. Acad. Sci. USA, 93 (1996), pp. 12098-12103.

[43] S. B. Needleman and C. D. Wunsch, *A general method applicable to the search for similarities in the amino acid sequence of two proteins*, J. Mol. Biol., 48 (1970), pp. 443-53.

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

[45] C. Notredame, D. G. Higgins and J. Heringa, *T-Coffee: A novel algorithm for multiple sequence alignment*, JMB, 302 (2000), pp. 205-217.

[46] C. Notredame, L. Holm and D. G. Higgins, *COFFEE: an objective function for multiple sequence alignments*, Bioinformatics, 14 (1998), pp. 407-22.

[47] C. Notredame, E. A. O'Brien and D. G. Higgins, *RAGA: RNA Sequence Alignment by Genetic Algorithm,* Nucleic Acids Res., 25 (1997), pp. 4570-4580.

[48] H. Ogata, A. Yutaka and K. Minoru, *A genetic algorithm based molecular modeling technique for RNA stem-loop structures*, Nucleic Acids res., 23 (1995), pp. 419-426.

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

[50] 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.

[51] B. Rost and C. Sander, *Prediction of protein secondary structure at better than 70% accuracy*, Journal of Molecular Biology, 232 (1993), pp. 584-599.

[52] C. Sander and R. Schneider, *Database of homology-derived structures and the structurally meaning of sequence alignment*, Proteins: Structure, Function, and Genetics, 9 (1991), pp. 56-68.

[53] S. Schulze-Kremer, *Genetic algorithms for protein tertiary structure prediction*, in R. Männer and B. Manderick, eds., *Proceedings of the 2nd Conference on Parallel Problem Solving from Nature,* Elsevier Science Publishers, Amsterdam, 1992, pp. 391-400.

[54] B. A. Shapiro and J. C. Wu, *An annealing mutation operator in the genetic algorithm for RNA folding,* Comp. Applic. Biosci., 12 (1996), pp. 171-180.

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

[56] T. F. Smith and M. S. Waterman, *Comparison of biosequences*, Adv. Appl. Math., 2 (1981), pp. 482-489.

[57] J. Stoye, V. Moulton and A. W. Dress, *DCA: an efficient implementation of the divide-and-conquer approach to simultaneous multiple sequence alignment*, Comput Appl Biosci, 13 (1997), pp. 625-6.

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

[59] J. Thompson, D. Higgins and T. Gibson, *CLUSTAL 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.

[60] R. Unger and J. Moult, *Genetic Algorithms for Protein Folding Simulations*, J. Mol. Biol., 231 (1993), pp. 75-81.

[61] Y. Van de Peer, J. Jansen, P. De Rijk and R. De Watcher, *Database on the structure of small ribosomal RNA*, Nucleic Acids res., 25 (1997), pp. 111-116.

[62] L. Wang and T. Jiang, *On the complexity of multiple sequence alignment*, Journal of computational biology, 1 (1994), pp. 337-348.

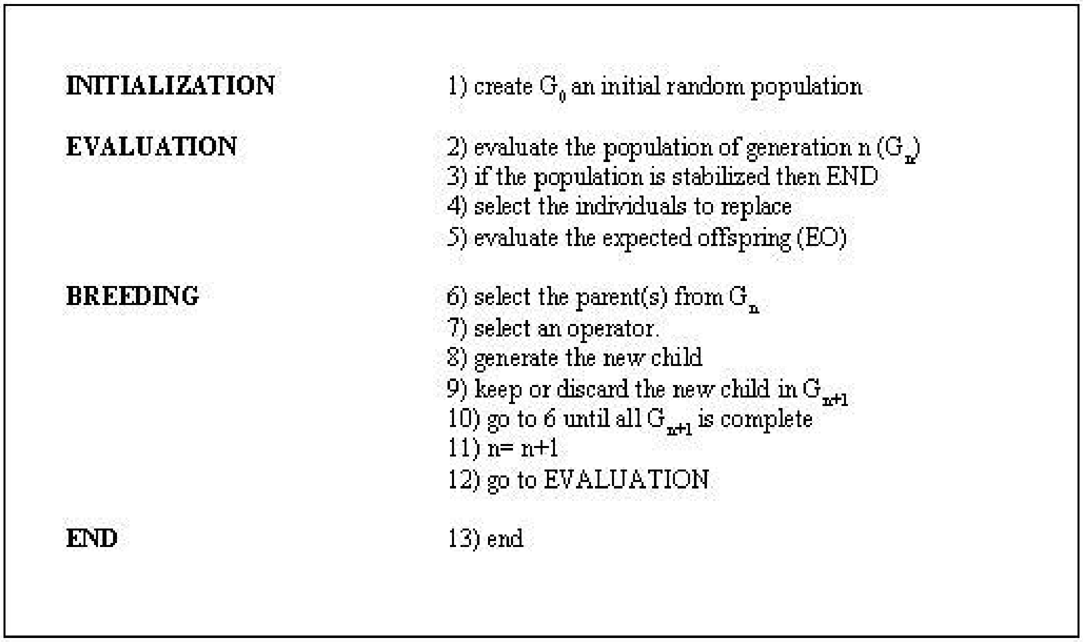
[63] J. D. Watson and F. H. C. Crick, *Molecular structure of nucleic acids. A structure for deoxyribose nucleic acid*, Nature, 171 (1953), pp. 737-738.

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

[65] M. Zuker and P. Stiegler, *Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information*, Nucleic Acids Res., 9 (1981), pp. 133-148.

Figure legends

Figure 1



```
INITIALIZATION      1) create $G_0$ an initial random population

EVALUATION          2) evaluate the population of generation n ($G_n$)
                    3) if the population is stabilized then END
                    4) select the individuals to replace
                    5) evaluate the expected offspring (EO)

BREEDING            6) select the parent(s) from $G_n$
                    7) select an operator.
                    8) generate the new child
                    9) keep or discard the new child in $G_{n+1}$
                    10) go to 6 until all $G_{n+1}$ is complete
                    11) n= n+1
                    12) go to EVALUATION

END                 13) end
```

Figure 1
Layout of the SAGA algorithm.
This pseudo-code indicates the main steps that take place during the optimization carried out by SAGA. See the text for full details.

Figure 2a)One point Crossover

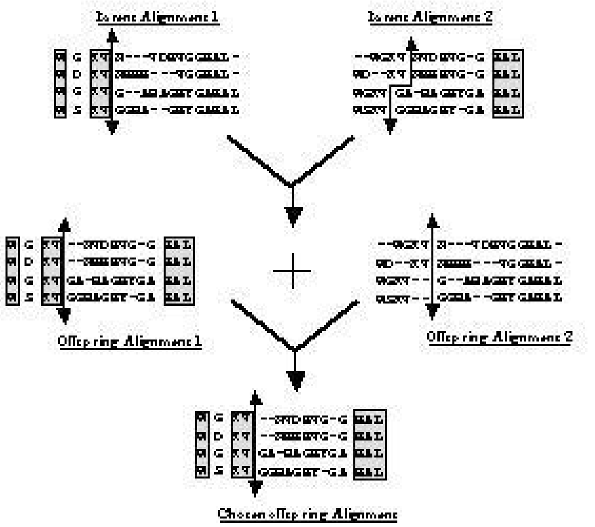
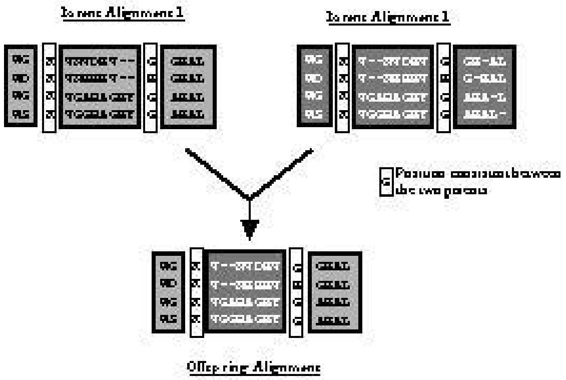


Figure 2
Crossovers used in SAGA
a) *One point crossover* between two parent alignments to produce two children. The arrows indicate the way the two parents are cut having randomly chosen a position in the left hand alignment. Child 1 is produced by combining the left side of parent 1 and the right side of

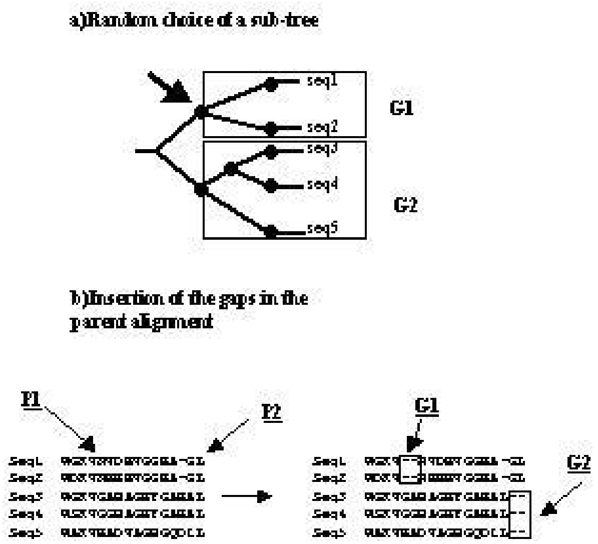
parent 2. Child 2 is produced by combining the right side of parent 1 and the left side of parent 2. Only one of these two children alignments is kept (whichever scores better). The boxed sections show some patterns from the parent alignments that are combined in the child.

Figure 2b) Uniform Crossover



b)*Uniform crossover*. All of the positions in the two parents that are consistent between the two alignments are marked (stars). Children are produced by swapping blocks between the two parents where each block is randomly chosen between two consistent positions.

Figure 3) Gap Insertion Operator


a)Random choice of a sub-tree

b)Insertion of the gaps in the parent alignment

Figure 3

Gap insertion Operator
a) The estimated phylogenetic tree connecting the five sequences is randomly divided into two sub trees. This gives two groups of sequences (G1 and G2).
b) Two positions P1 and P2 are randomly chosen in the alignment. A gap of random length (here 2 nulls) is inserted at position P1 in the sequences of subgroup G1, and the same number of nulls are inserted at position P2 in subgroup G2.

Figure 4) Island Parallelisation

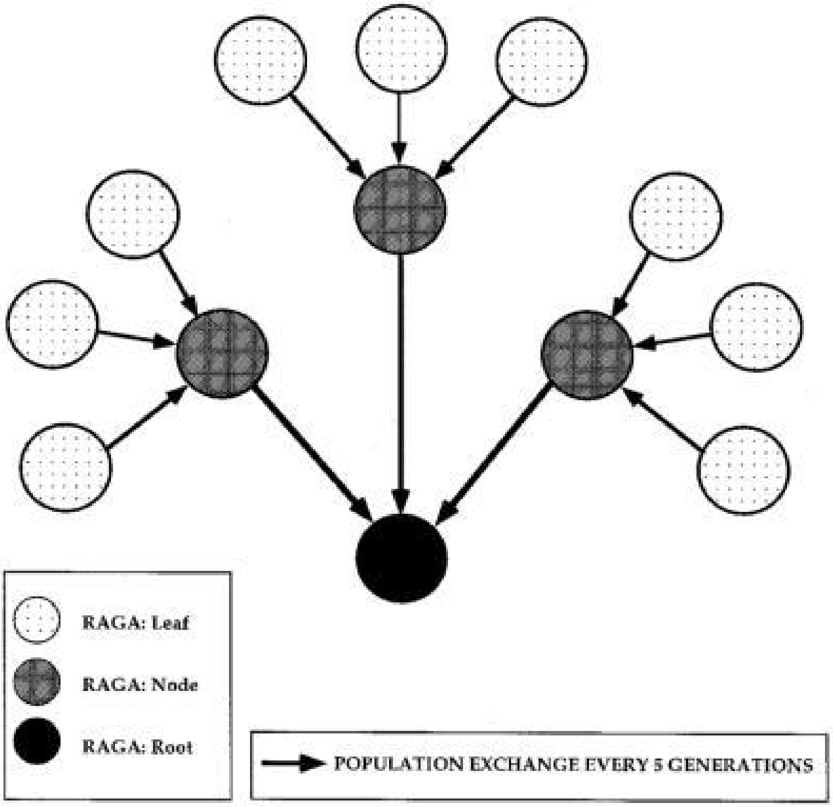


Figure 4.

Layout of the Parallel version of RAGA.

Each circle represents a RAGA process. The best individuals migrate from top to bottom. The best solution is to be found in the root (bottom).

Tables

Table 1. Mathematical validation of SAGA against MSA using 3Dali.

			MSA			SAGA-MSA		
Test case	Nseq	Length	Score	Q	CPU	Score	Q	CPU
Cyt c	6	129	1051257	74.2	7	1051257	74.2	960
Gcr	8	60	371875	75.0	3	**371650**	82.0	75
Ac protease	5	183	379997	80.1	13	379997	80.1	331
S protease	6	280	574884	91.0	184	574884	91.0	3500
Chtp	6	247	111924	*	4525	**111579**	*	3542
Dfr secstr	4	189	171979	82.0	5	**171975**	82.5	411
Sbt	4	296	271747	80.1	7	271747	80.1	210
Globin	7	167	659036	94.4	7	659036	94.4	330
Plasto	5	132	236343	54.0	22	**236195**	54.0	510

Nseq is the number of sequences; Length is the length of the final SAGA alignment; Score is the score of the alignment returned by MSA using the weighted sums-of-pairs with quasi-natural affine gap penalties (the function is minimized and the best scores are the lowest). The columns marked ‘Q' give the percentage of an MSA alignment that matches the structural alignment. CPU time is given in seconds. SAGA-MSA indicates similar results with alignments obtained by SAGA. In the Score column, alignments for which SAGA outperforms MSA are indicated in bold. The PDB structure identifiers for each test case can be found in 3Dali. The PDB structure identifiers for each test case are as follows. *Cyt c*: 451c, 1ccr, 1cyc, 5cyt, 3c2c, 155c. *Gcr*: 2gcr, 2gcr-2, 2gcr-3, 2gcr-4, 1gcr, 1gcr-2, 1gcr-3, 1gcr-4. *Ac protease*: 1cms, 4ape, 3app, 2apr, 4pep. *S protease*: 1ton, 2pka, 2ptn, 4cha, 3est, 3rp2. *Dfr secstr*: 1dhf, 3dfr, 4dfr, 8dfr. *Chtp*: 3rp2, M13143 (EMBL accession number), 1gmh, 2tga, 1est, 1sgt. *Sbt*: 1cse, 1sbt, 1tec, 2prk. *Globin*: 4hhb-2, 2mhb-2, 4hhb, 2mhb, 1mbd, 2lhb, 2lh1. *Plasto*: 7pcy, 2paz, 1pcy, 1azu, 2aza.

Table2. Biological validation of the COFFEE function using 3Dali.

Test case	Nseq	Length	SAGA-MSA	SAGA-COFFEE	CLUSTAL
ac_prot	21	14	**51.2**	50.2	39.2
binding	31	7	64.2	**64.5**	50.0
cytc	42	6	67.3	**90.7**	89.1
fnIII	17	9	45.2	**47.0**	42.0
gcr	36	8	80.8	**83.1**	80.8
globin	24	17	78.0	85.2	**86.4**
igb	24	37	70.1	**78.1**	74.8
lzm	39	6	**72.3**	**72.3**	72.2
Phenyldiox	22	8	55.6	**64.7**	58.5
Sbt	61	7	96.0	**96.9**	96.7
s-prot	27	15	**68.5**	66.6	62.5

Nseq is the number of sequences; Length is the length of the final SAGA alignment; SAGA-MSA is the percentage of the alignment that matches the structural alignment when SAGA is run to optimize the weighted sums-of-pairs with natural affine gap penalties. SAGA-COFFEE is similar but using the COFFEE function. CLUSTALW is similar with a comparison made on the default output of ClustalW. The method giving the best result is in bold.

Table 3. Biological validation of an RNA-specific objective function.

Master	Slave	Dist.	Pairs (%)	Len.	Q(%)	
					DP	PRAGA
Homo sapiens	*Oxytrichia nova*	0.41	82.5	1914	83.9	86.6
Homo sapiens	*Giarda ardeae*	0.57	82.1	1895	72.2	76.1
Homo sapiens Mit.	*Latim. chalum. Mit.*	0.31	81.2	998	85.9	92.5
Homo sapiens Mit.	*Xenopus laevis Mit.*	0.43	84.9	985	83.9	92.5
Homo sapiens Mit.	*Dros. virilis Mit.*	0.76	82.6	973	66.8	76.6
Homo sapiens Mit.	*Apis mellifera Mit.*	1.23	72.1	977	45.2	56.0
Homo sapiens Mit.	*Penicil. chryso. Mit.*	1.26	81.3	1478	37.7	63.8
Homo sapiens Mit.	*Chla. reinha. Mit.*	1.30	66.6	1271	34.1	53.2
Homo sapiens Mit.	*Sacc. cerevis. Mit.*	1.33	80.3	1699	31.6	60.2

Master: Sequence with a known structure. Slave: sequence with an unknown structure. Dist.: estimated mean number of substitutions per site between the master and the slave measured on the reference alignment. Pairs: percentage of residues involved in the master secondary structure. Len.: length of the reference alignment. Q: measure m1 (overall level of identity with the reference alignment) made on a Dynamic Programming with local gap penalties alignment (DP) or on a RAGA alignment. The sequences EMBL accession numbers are as follow: *Homo sapiens* (X03205), *Homo sapiens* mitochondria (V00702), *Oxytrichia nova* (X03948), *Giarda ardeae* (Z177210), *Latimeria chalumnae* mitochondria (Z21921), *Xenopus laevis* mitochondria (M27605), *Drosophila virilis* mitochondria (X05914), *Apis mellifera* mitochondria (S51650), *Penicillium chrysogenum* mitochondria (L01493), *Chlamydomonas reinhardtii* mitochondria (M25119), *Saccharomyces cerevisiae* mitochondria (V00702).