4570–4580 *Nucleic Acids Research, 1997, Vol. 25, No. 22* *© 1997 Oxford University Press*

RAGA: RNA sequence alignment by genetic algorithm

Cédric Notredame[1,*], Emmet A. O'Brien[1,2] and Desmond G. Higgins[1,2]

[1]EMBL Outstation–The European Bioinformatics Institute, Welcome Trust Genome Campus, Hinxton, Cambridge CB10 1SD, UK and [2]Department of Biochemistry, University College, Cork, Ireland

Received July 23, 1997; Revised and Accepted October 1, 1997

ABSTRACT

We describe a new approach for accurately aligning two homologous RNA sequences when the secondary structure of one of them is known. To do so we developed two software packages, called RAGA and PRAGA, which use a genetic algorithm approach to optimize the alignments. RAGA is mainly an extension of SAGA, an earlier package for multiple protein sequence alignment. In PRAGA several genetic algorithms run in parallel and exchange individual solutions. This method allows us to optimize an objective function that describes the quality of a RNA pairwise alignment, taking into account both primary and secondary structure, including pseudoknots. We report results obtained using PRAGA on nine test cases of pairs of eukaryotic small subunit rRNA sequence (nuclear and mitochondrial).

INTRODUCTION

Most methods of alignment are based on the primary structure of the sequences to be analysed (1). Alignment may be straightforward when the primary structure is conserved but becomes less and less accurate as the evolutionary distance increases. In the case of RNA it may be possible to use secondary structure information to supplement the weak primary structure information. Such alignments, using primary and secondary constraints, have been built for rRNAs (2,3). Their construction is at least partially manual and is usually based on identification of sets of correlated mutations which suggest secondary structure interactions.

One justification for such methods is the fact that accurate alignment is still the main non-experimental way to establish a reliable secondary structure for a long RNA molecule. The only other alternative is *ab initio* prediction. Several techniques of this type have been developed over time (4,5), but they recently received renewed attention through the use of stochastic heuristic-based approaches, like simulated annealing (6,7) and genetic algorithms (8,9). Nevertheless, they remain limited by the fact that our understanding of the *in vivo* folding process is still incomplete. In contrast, homology analysis based on alignments does not have these limitations. Multiple alignments reveal the positions of the sequences on which some constraints exist, regardless of the actual cause of these constraints. Several algorithms have been developed for aligning RNA sequences taking into account primary and secondary information. Some methods attempt to simultaneously align and fold sequences (10–12). Their main drawback is that they remain limited to sets of short sequences (<200 nt long).

To reduce the complexity of this problem it is also possible to align a sequence (or a set of sequences) of unknown structure to some pre-established reference master structure. Such alignments include non-local interactions and their solution has been shown to be NP hard (13). Nevertheless, for small sequences sensible results can be obtained. Several methods of this type, based on the use of stochastic context free grammar (SCFG) for the description of RNA non-pseudoknotted secondary structure, have been described (11,14,15). Pseudoknots, however, are important motifs in RNA folding (16) and recently some new results have been obtained on this aspect of RNA analysis. This includes the work of Tabaska and Stormo (17) for aligning an RNA sequence to a pseudoknotted structure in polynomial time. Unfortunately, all these methods have the limitation of being computationally very expensive and therefore remain restricted to small sequences (<200 nt long).

This problem can be partially overcome by heuristic methods. Corpet and Michot have described an approach of this type (18). In this case a heuristic allows identification of the portions of an alignment that can be made without using secondary structure information. The remaining portions, if they are small enough, can then be aligned using non-local interactions. This is done with a specialized dynamic programming algorithm. Although this algorithm is less efficient than that described for SCFG-based alignments, the heuristic filtering makes it possible, in some cases, to align long RNA molecules (e.g. >1500 nt). At the moment this is largely beyond the scope of any SCFG-based algorithm. Unfortunately, the algorithm cannot deal with very divergent sequences and does not support the computation of pseudoknots. As opposed to the SCFG-based scoring scheme, that used by Corpet and Michot has no real theoretical justification. Nevertheless, it has the merit of being conceptually simple as well as leading to computation of sensible alignments (as judged by comparison with established reference alignments) (18).

For this reason we took the overall approach and scoring scheme of Corpet and Michot but used a genetic algorithm (GA) to carry out the optimization. This has two significant advantages. Firstly, in the GA context there is no difference between the handling of pseudoknots and any other secondary structure. Secondly, it is possible to attempt to find alignments between much longer sequences, such as complete small subunit rRNAs,

*To whom correspondence should be addressed. Tel: +44 1223 494449; Fax: +44 1223 494468; Email: cedric.notredame@ebi.ac.uk

which can be >2000 nt in length. The genetic algorithms (GA) (19,20), like simulated annealing (21) or Gibbs sampling (22), is a stochastic optimization technique. It involves an attempt at optimizing some cost function (objective function, OF) by modifying and combining a population of solutions (individuals). GAs do not guarantee an optimal solution, but are known to perform well with combinatorial or enumeration problems.

We based our approach directly on a previous package, SAGA (Sequence Alignment by Genetic Algorithm) (23). This algorithm was improved and parallelized. A suitable cost function that describes the quality of a RNA alignment was introduced, mostly based on the function described by Corpet and Michot (18). The package was name PRAGA for Parallel RNA Alignment by Genetic Algorithm. We compared this algorithm with traditional techniques of sequence alignment and, to some extent, with the program RNAlign (18).

METHODS

The aim is to align two related sequences of RNA, knowing the secondary structure (master structure) of one of them (master sequence), in order to predict the position of these structural elements in the second sequence (slave sequence). In the correct alignment elements of the two sequences sharing the same structure and/or sequence should be aligned (Fig. 1a–c). A measure (OF) can be designed that allows evaluation of the quality of such an alignment. This measure takes into account the quality of the sequence alignment and the stability of the folding induced by the master sequence onto the slave sequence. To produce the best scoring alignment according to this measure we used a GA. We also describe a parallel GA that we have designed to gain some speed over a serial one. The results obtained with several sets of sequences were compared with established reference alignments of the same sequences, using three comparison methods.

Objective function

The function we use was described by Corpet and Michot (18). We implemented it in RAGA without any modification. It combines three different scores: Pr, the primary score; Se, the secondary score; a gap penalty score. The overall score is a combination of these three values. The higher this score, the better the alignment between the two sequences.

Pr is a function of the aligned pairs of residues in the alignment. It depends on a matrix where each possible pair of residues is given a score. In the case of RNA a simple identity matrix is used with a mismatch score of 0 and a match score of 1. All positions containing a gap are ignored at this point. Pr is therefore equal to the number of matches in the pairwise alignment.

Se is based on the secondary structure. It evaluates the stability of the folding induced by the master onto the slave sequence. If two nucleotides form a base pair (part of a stem) in the master, then the two nucleotides in the slave sequence aligned with them should be able to form a Watson–Crick base pair as well if the secondary structure is conserved. Since pairings are relatively well defined in RNA, it is possible to assign a score to the pairing potential of the sequence of unknown structure. This can be formalized as follows. Given two sequences A (master) and B (slave) with (A_i,A_j) being two nucleotides of A and (B_k,B_n) the equivalent aligned nucleotides of B, if A_i and A_j are known to be

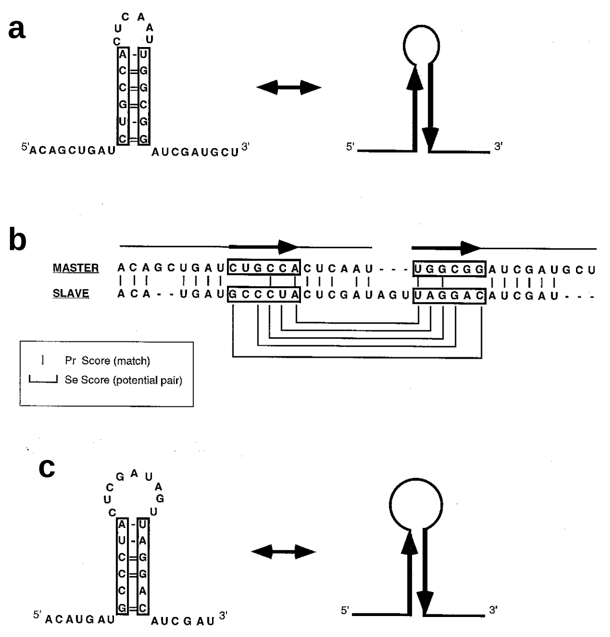

a
C A
U A
C U
A U
C G
C G
G C
U G
C G
5'ACAGCUGAU
AUCGAUGCU3'
5'
3'
b
MASTER
A C A G C U G A U C U G C C A C U C A A U - - - U G G C G G A U C G A U G C U
SLAVE
A C A - - U G A U G C C C U A C U C G A U A G U U A G G A C A U C G A U - - -
Pr Score (match)
Se Score (potential pair)
c
G A U
C A
U G
C U
A U
U A
C G
C G
C A
G C
5' ACAUGAU
AUCGAU 3'
5'
3'

Figure 1. RNA alignment. (**a**) Master sequence with a known secondary structure. (**b**) Alignment of the master sequence with a slave sequence of unknown structure using primary (Pr score) and secondary structure (Se score) information. (**c**) Prediction of the structural elements of the slave sequence shared with the master sequence.

paired, A_i is aligned with B_k and A_j with B_n then the score Se is equal to the pairing score of B_k with B_n. In practice, a very simplified model was used to assign pairing scores, giving 2 for GC pairs and 1 for UA and UG pairs (UG is not a conventional Watson–Crick base pair but stems in RNA frequently contain UG pairs) or any other interaction involving one of the wildcards X or N. The other pairs are given a score of 0. Se is the sum of the scores associated with each pair in the structure of B induced by the structure of A.

It is usually necessary to insert gaps into one or both sequences in order to perform the alignment. These gaps represent insertion or deletion events that have occurred over time in both sequences. They may not occur completely at random and may happen more frequently in loops or in non-structured domains. In order to reflect this in the OF, two position-specific gap penalties are used following the model of Corpet and Michot (18): GOS, a penalty for opening a gap between two stacked pairs; GO, a penalty for opening a gap in non-structured regions.

A third penalty (GEP) is used to penalize gap length. It is calculated as GEP × length of gap. Terminal gaps are not penalized. We use the values proposed by Corpet and Michot with GO = 5, GOS = 8, GEP = 0.3. The total gap penalty of an alignment is equal to:

$$\text{gap penalty} = (a \times \text{GOS}) + (b \times \text{GO}) + (c \times \text{GEP}) \qquad \mathbf{1}$$

Where *a* is the number of gaps between stacked pairs in stems, *b* is the number of other non-terminal gaps and *c* is the total length of all the non-terminal gaps. The complete alignment score is calculated using a new parameter λ, which is always a positive value.

$$\text{alignment score} = \text{Pr} + (\lambda \times \text{Se}) - \text{gap penalty} \qquad \mathbf{2}$$

The parameter λ has the effect of balancing the contribution from primary structure information and from secondary structure.

Optimization of the objective function

The function in equation **2** was shown to be a good indicator of the quality of an alignment (18). The main drawback is that its optimization is difficult when λ is given a value other than 0. In the case of $\lambda = 0$ the contribution of secondary structure to the alignment is ignored. Such a function can be optimized by regular dynamic programming with local gap penalties (1,24). However, when λ is not zero optimization becomes much harder. A variation of dynamic programming has been described. It requires $O(M^2N^2)$ space and $O(M^2N^3)$ time (18), M and N being the lengths of the sequences or profiles to align. Such a high complexity makes it hard to apply this algorithm to anything other than small sequences or fragments of alignments. Another limitation of this approach is that it cannot easily deal with pseudoknots. In order to overcome these two problems we used a GA.

RAGA is derived from the Simple Genetic Algorithm described by Goldberg (20). It involves using a population of solutions which evolve by means of natural selection. The population we consider is made of pairwise alignments. Initially a generation zero (G_0) is created (initialization). In this population each individual consists of one possible alignment for the sequences to be aligned. The size of this population is kept constant. To go from one generation to the next, children are derived from parents that are chosen by some kind of 'natural selection', based on their fitness as measured by OF (i.e. the better the parent, the more children it will have). To create a child an operator is selected that can be a crossover (mixing the contents of the two parents) or a mutation (modifying a single parent). There are several types of mutations, modifying the alignments in different ways. Each of these has a probability of being chosen that is dynamically optimized during the run (dynamic scheduling).

These steps are repeated iteratively, generation after generation (evaluation/breeding). During these cycles new pieces of alignment appear because of the mutations and are combined by the crossovers. This selection makes sure that good pieces survive and dynamic setting of the operators helps the population to improve by creating the children it needs. Following this simple process the average fitness of the population increases until no more improvement can be made. The best alignment obtained in this way is taken as a result.

Initialization. The first step of the algorithm is initialization, during which a population of solutions is created (seeding). The two desirable properties of an initial population are to have as much diversity as possible and to contain as many good scoring blocks as possible (i.e. individuals as good as possible). Seeding in a random manner allows one to have high diversity, but very few good scoring blocks. Such a population will usually improve slowly. On the other hand, seeding with greedy methods [e.g. ClustalW (25) and other alignment software] gives a population with a better initial score at the cost of lower diversity. Such a population usually improves very quickly, but tends to get stuck into a local minimum close to the starting point.

In RAGA we tried to find a good trade-off between these two extremes. A variation on dynamic programming described by Gerstein (26) was used to produce our initial alignments. This method (Dynamic Programming with Added Noise, DPAN in this paper) allows addition of a random amount of noise to the regular dynamic programming method (27,28), therefore producing sub-optimal alignments centred around the mathematical optimum obtained by dynamic programming without noise. In practice, when two sequences are aligned several times with DPAN long stretches of conserved residues tend to be kept intact, while diversity accumulates in less stable regions of the alignment. It is possible to control the overall amount of noise added. In RAGA this noise was tuned so that the average score of the alignments used for seeding would be the same as the average score of a population of random alignments. Even with such a bad initial average score, a population generated in this way improves about three times faster than a completely random population.

Evaluation/breeding/end. The rest of the GA procedures involved in RAGA are taken directly from SAGA. Individuals are first evaluated through the OF described earlier and then given an expected offspring score that reflects their quality in comparison with the rest of the individuals of the same generation. At each generation half of the population (lowest scoring alignments) is replaced by newly generated individuals (children). To produce a child an operator is selected (mutation or crossover). In the case of a mutation one parent is chosen to which the mutation is applied to produce a modified alignment that is put back into the population. In the case of a crossover the procedure is the same, but two parents are used. Children are put back into the population only when they are different from all the other children (population without duplicates). The selection of a parent is made by weighted wheel selection, a standard practice in GA: a virtual wheel is spun where each individual has a number of slots proportional to its expected offspring. Therefore the fittest are more likely to be chosen as parents, while the weakest still have a chance to survive.

Each operator has a probability of being used which varies along the run, depending on how well it performed. This automatic process is known as dynamic scheduling of the operators and has been described in greater detail (19,23). When no improvement has been made for a specified number of generations (typically 100 generations on a run of 400 generations), the GA is stopped.

The operators. According to the traditional nomenclature of genetic algorithms (20) two types of operators are used in RAGA: crossovers and mutations. In RAGA we do not make any distinction between these two types with regard to how we apply them. They are designed as independent programs that input one or two alignments (the parents) and output one alignment (the child). The difference in score between the input and output is used for future evaluation of the usefulness of the operator. Each operator requires one or more parameters which specify where and how the operation is to be carried out. For instance, an operator inserting a new gap must be told where (at which position in the alignment) and in which sequences the gap is to be inserted and how long this gap will be. The operators display several levels of greediness. Some are completely stochastic (i.e. the values of the parameter are determined randomly in some reasonable range), while others aim at some local optimization and rely on enumeration or DPAN.

The crossovers. These operators allow good pieces to be recombined and therefore play a central role in improving the population. In RAGA only one type of crosover is implemented:

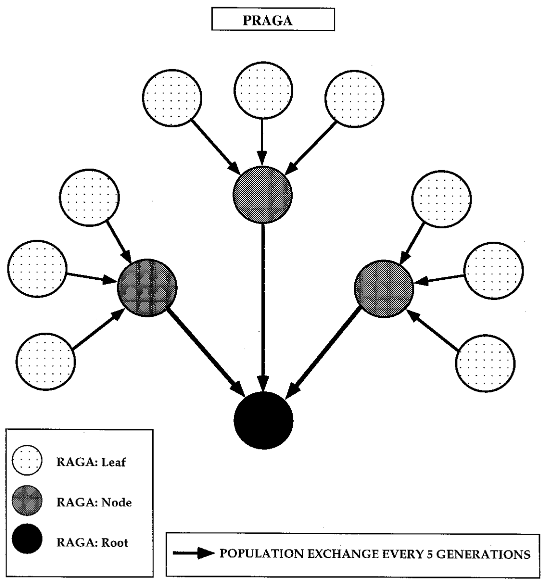


Figure 2. Layout of the parallel genetic algorithm PRAGA. Each circle represents a RAGA process. The best individuals migrate from top to bottom. The best solution is to be found in the root (bottom).

the uniform crossover (UCO) previously used in SAGA. The principle is to map the areas of alignment which are consistent (identical) between the two parents. The child will contain all these identical blocks as well as non-identical blocks taken from either of the parents. The choice of a non-consistent block can be random or deterministic. Both versions (random and deterministic) are implemented in RAGA.

Gap shifting. In order to keep improving the population it is necessary to introduce new alignment configurations. This can be done by shifting gaps. To do so a gap is randomly chosen in an alignment and moved to another position. The choice of the new position can be random or greedy, in which case the gap is slowly shifted in one direction as long as the score of the alignment keeps improving.

Gap insertion. This operation is made by DPAN as described earlier. It is only performed on a portion of random size that is extracted from the alignment, re-aligned and re-introduced.

Island parallelization. In order to decrease run times we implemented an island parallelization model (20). Instead of having a single copy of RAGA, several identically configured GAs were used, running independently in parallel and exchanging individuals every *N* generations, where *N* is typically 5. The algorithms are arranged on a k-branched tree and population exchange only takes place in one direction, from the leaves to the bottom of the tree (Fig. 2). By default the individuals migrating from one RAGA to another are those having the best score. They replace individuals with lower scores in the node to which they move. The node from which they come keeps a copy, so that in each RAGA process the population remains constant.

We found that this model gives results comparable with what would be obtained with a single copy of RAGA having a population equivalent to the overall population of all the GAs. By default, after trial and error optimization, we used a three-branched tree with a depth of three, as shown in Figure 2. This model requires 13 GAs. The processes are synchronous and wait for each other to reach the same generation number before exchanging their populations. In terms of CPU time this implementation can be ~10 times faster than a single GA with the same overall population. This means that we get ~80% of the maximum speed-up. A typical population size for each GA is 30 and population migration occurs every five generations. A GA that receives new individuals (node) replaces half of its population that way (15 individuals). These 15 individuals are made up of three groups of five individuals coming from the previous nodes/leaves. This parallel GA was named PRAGA for Parallel RNA Alignment Genetic Algorithm. All of our results were obtained using PRAGA.

Test cases

To assess the efficiency and accuracy of PRAGA several test cases were designed. We used aligned rRNA sequences, obtained from a manual expert alignment of small subunit (SSU) sequences (29). This database contains large alignments of rRNA made by hand. These alignments come with predicted secondary structure. To build a test case two sequences were extracted from a multiple alignment. This initial alignment was kept as a reference. From the same alignment the structure (master structure) of one of the two sequences (master sequence) was then extracted. In this structure we kept only the elements documented as belonging to the conserved core which is found in most SSU rRNAs (30,31). These elements were chosen because they were likely to exist in the second sequence (slave sequence).

We designed two large test cases (test cases 1 and 2 in Table 1) using full-length eukaryotic nuclear sequences. These two sets use the human SSU rRNA as a master sequence and *Oxytrichia nova* and *Giardia ardeae* as slaves. These sequences are 65 and 75% identical respectively to the master human sequence. The purpose of these two test cases is to show the ability of the GA to optimize long alignments consisting of sequences of ~2 kb in length. In order to obtain test cases with a wider range of identity between the master and the slave sequence we turned to the mitochondrial SSU rRNA sequences. These sequences diverge faster than their nuclear counterparts (32), have a wider spectrum of identity and are also generally smaller (~1–1.5 kb in length). This allowed an extensive study of some of the properties of our algorithm. This set of seven test cases (Table 1, test cases 3–9) was created using the procedure already described. Sequences and structures were extracted from the database alignment (29). We used the human mitochondrial SSU rRNA as a master sequence and seven other mitochondrial sequences as slaves. Their identity with the human sequence ranges from 70 to 43%. Some of the slave sequences do not contain all the structural elements described in the core structure used for the alignment (Table 1, pairs column). This gave us a chance to analyse the effect of this type of noise on our optimization procedure.

The distances between the two sequences of a given test case were measured using the program Dnadist in the package Phylip (33). We used this program to assess the 'Kimura with 2 parameters' distance (34), with a default 'transition/transvertion' ratio set to 2 and one category of substitution rates.

Table 1. Test cases and general results

TC	Master	Slave	Distance	Pairs (%)	Length	λ	m1 (%)		m2 (residues)		m3 (%)	
							DP	PRAGA	DP	PRAGA	DP	PRAGA
1	*Homo sapiens*	*Oxytrichia nova*	0.41	82.5	1914	1.00	83.9	86.6	0.15	0.06	85.3	94.7
2	*Homo sapiens*	*Giarda ardeae*	0.57	82.1	1895	3.00	72.2	76.1	0.53	0.45	65.2	81.3
3	*Homo sapiens* mitochondrion	*Latimeria chalumnae* mitochondrion	0.31	81.2	998	1.00	85.9	92.5	0.64	0.10	82.6	96.1
4	*Homo sapiens* mitochondrion	*Xenopus laevis* mitochondrion	0.43	84.9	985	1.00	83.9	92.5	0.41	0.20	77.4	96.7
5	*Homo sapiens* mitochondrion	*Drosophila virilis* mitochondrion	0.76	82.6	973	3.00	66.8	76.6	2.08	1.59	48.6	68.5
6	*Homo sapiens* mitochondrion	*Apis mellifera* mitochondrion	1.23	72.1	977	4.00	45.2	56.0	3.83	2.91	24.1	55.1
7	*Homo sapiens* mitochondrion	*Penicillium chrysogenum* mitochondrion	1.26	81.3	1478	4.00	37.7	63.8	4.96	3.21	15.7	77.0
8	*Homo sapiens* mitochondrion	*Chlamydomonas reinhardtii* mitochondrion	1.30	66.6	1271	4.00	34.1	53.2	13.4	8.26	8.90	50.0
9	*Homo sapiens* mitochondrion	*Saccharomyces cerevisiae* mitochondrion	1.33	80.3	1699	6.00	31.6	60.2	14.7	3.70	21.6	70.0

TC, test case number (as used in the text); Master, sequence with a known structure; Slave, sequence with an unknown structure, Distance, estimated mean number of substitutions per site between the master and the slave measured on the reference alignment; Pairs, pairs defined in the core structure of the master and present in the slave sequence, as judged from the reference alignment; Length: length of the reference alignment; λ, optimal value of λ, measured from graphs similar to those shown in Figure 4 [in cases where the three graphs (m1, m2, m3) did not indicate the same optimum we chose a value that was at least consistent with two of the graphs]; m1, measure m1 (overall level of identity with the reference alignment) obtained by dynamic programming with local gap penalties alignment (DP) or by PRAGA alignment obtained with the optimal λ (PRAGA); m2, average offset measured on the structure (m2 should be as small as possible); m3, percent of pairs found correctly aligned (the reference is the number of pairs in the master core structure conserved in the slave sequence). The sequence EMBL accession nos are as follows: *Homo sapiens*, X03205; *Homo sapiens* mitochondrion, V00702; *Oxytrichia nova*, X03948; *Giarda ardeae*, Z177210; *Latimeria chalumnae* mitochondrion, Z21921; *Xenopus laevis* mitochondrion, M27605; *Drosophila virilis* mitochondrion, X05914; *Apis mellifera* mitochondrion, S51650; *Penicillium chrysogenum* mitochondrion, L01493; *Chlamydomonas reinhardtii* mitochondrion, M25119; *Saccharomyces cerevisiae* mitochondrion, V00702.

Evaluation

PRAGA was evaluated by comparing the results on the test cases with results obtained using traditional dynamic programming, RNAlign and by comparison with the reference alignments. Dynamic programming was implemented using Gotoh's algorithm (35) with local gap penalties, so as to make it comparable with optimizing the OF with $\lambda = 0$. Due to the length of the sequences and the memory requirement, it was only possible to run RNAlign (18) on two of the test cases (3 and 5).

Comparison of an alignment with the reference taken from the databases can be done in several ways. We use three different measures: m1, m2 and m3. m1 is the percentage of the aligned columns of nucleotides in the reference alignment that are reproduced in the test alignment (columns with gaps are ignored). m2 is based on the alignment of stems. It is the average offset of stems between the reference and the test alignment. If a position A_k of the master sequence is aligned with B_i in the reference alignment and with B_j in the new alignment the offset will be $(i - j)$. m2 is equal to the average of each offset absolute value. The better the alignment, the smaller m2. The main advantage of m2 is that it takes into account some close sub-optima that would otherwise be completely disregarded by m1. Giving some credit to these types of alignments makes sense, especially when aligning similar structures with very divergent sequences. m3 is a measure very similar to m1. In m3 we only consider residues that form a pair in the secondary structure (stems). To be considered correctly aligned both residues of a pair must be aligned in a similar way to the reference. m3 is the percentage of such residues over the total number of pairs in the common core structure.

Implementation

RAGA and PRAGA are written in ANSI C and run under UNIX. PRAGA can be run on a variety of different UNIX platforms as long as they can each run RAGA. For RAGA the memory requirement is ~20 MB for an average alignment length close to 2000 regardless of the population size. A beta release for PRAGA and RAGA is available free of charge from the corresponding author by Email request, including 'RAGA or PRAGA' in the title.

RESULTS

Dynamic programming reference

For each test case a pairwise alignment was produced using dynamic programming with local gap penalties. Another was made without local penalties using ClustalW. We compared these alignments to their reference using m1, m2, m3 and found that alignments made with local penalties were ~10% (as measured

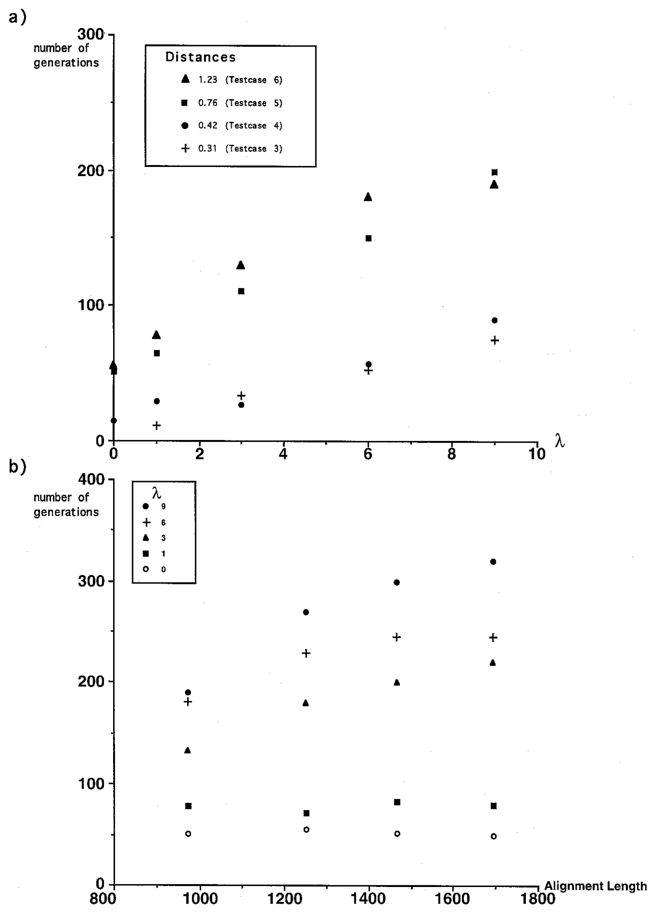


Figure 3. Complexity. (**a**) Time (in generations) required to find a solution as a function of λ. The distances are as in Table 1 and measure the distance between the slave and the master sequence. Four test cases producing alignments of similar length were used (6, 5, 4 and 3, which have lengths comprised between 900 and 1000 nucleotides). (**b**) Time (in generations) needed to find an optimum as a function of alignment length. For each of the four test cases measures were made varying λ. The four test cases used (6, 8, 7 and 9) have a comparable distance between the master and the slave sequence (1.23–1.33). In our system (see Methods) the time required for one generation was ~54 s.

with m1 and m3) more similar to the reference than alignments made without local gap penalties. We then measured the distance between the sequences in each reference alignment, as described in the previous section. Similarity to the reference alignment, using measures m1, m2 and m3, was plotted against these distances. As expected, we find that there is a clear correlation between the level of identity of the sequences aligned and the similarity of their pairwise alignment to the reference. In order to improve on these results we introduced secondary structure information into the alignment procedure and used PRAGA to do so.

Efficiency and accuracy of PRAGA

Since the optimization procedure is central to our work, we analysed PRAGA for its ability to perform this task. We looked at two criteria: the accuracy of optimization and the consistency of the results. Our algorithm, being a stochastic heuristic, can be expected to give different results when run several times with the same set of parameters. In order to have a program that is as

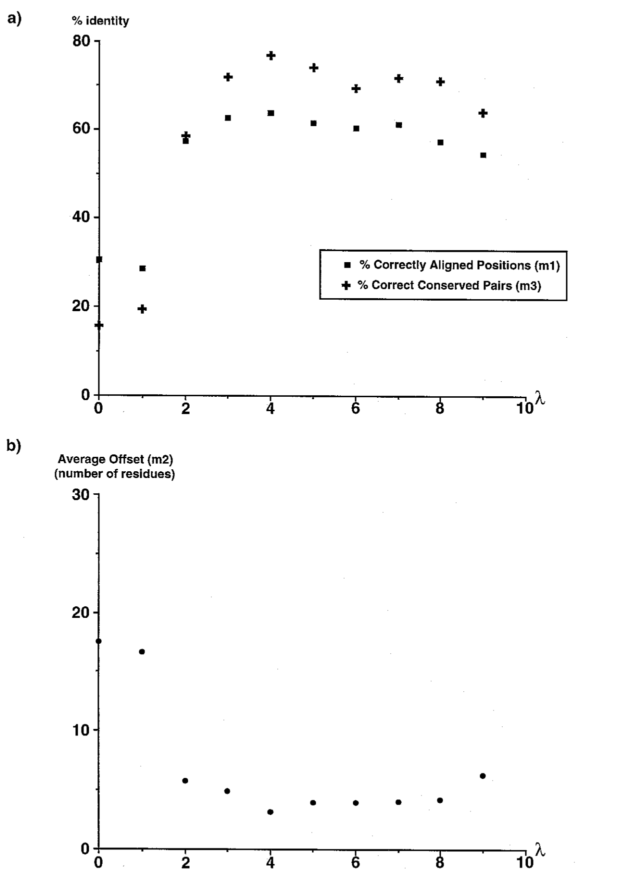


Figure 4. Evaluation of optimum λ on test case 7. (**a**) m1 and m3 (see Methods) were measured on the alignment produced by PRAGA with different values of λ. (**b**) As (a) but with m2.

reliable as possible one would like to minimize the level of variation from one run to another.

First, we checked, through the use of crossovers and mutations, that our program was able to reproduce the patterns of gaps and matches present in any of the reference alignments. We did so by using as an OF the measure of overall identity (m1) between a PRAGA alignment and the reference alignment. For all the test cases the GA was able to produce alignments 100% identical to the reference. Several runs were made for each test case that showed a total consistency in the scores. This is a good sign that PRAGA has the potential to explore the whole solution space when aligning two sequences.

Since dynamic programming with local gap penalties is equivalent to the OF described in the method with λ = 0, we checked that when using such an OF PRAGA was able to reproduce the dynamic programming alignments. In all cases it managed to produce alignments having exactly the same score as the dynamic programming reference. Here again we found a very good consistency from run to run (<0.1% deviation). When looking at the similarity between these alignments and the reference we found that the deviation was significantly higher (2.2% on m1, 2.1% on m3, 0.1 residues on m2). The highest variations were found for alignments where the two sequences aligned shared a low level of identity. This fact is not surprising, since it is well known that several alternative alignments of the

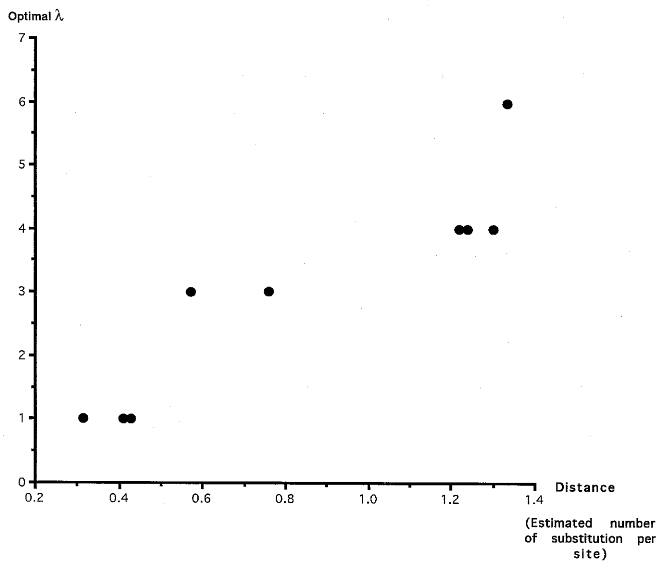


Figure 5. Optimal values of λ as a function of the slave/master distance. For each test case the values of λ leading to the best alignment were measured on plots similar to that shown in Figure 4 and plotted against the slave/master distance.

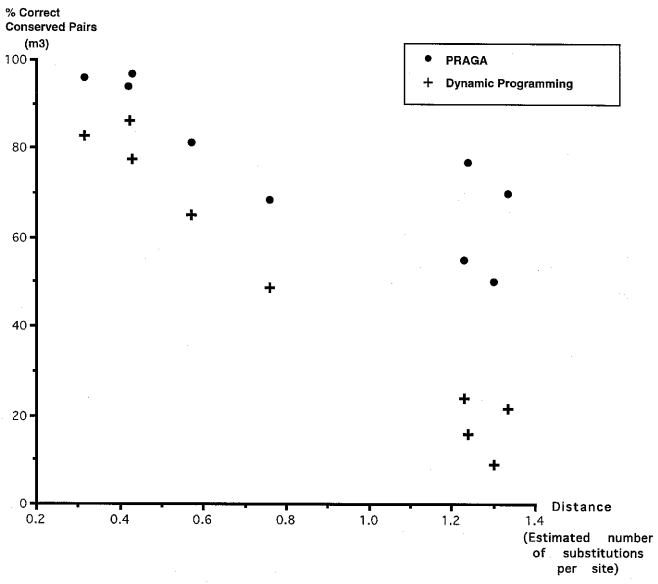


Figure 6. Comparison of PRAGA and dynamic programming with local gap penalties using the m3 measure. Each point corresponds to one of the test cases in Table 1. PRAGA alignments were obtained using an optimal value for λ.

same sequences can share the same score (36). This is simply a consequence of the OF properties.

In a second stage PRAGA was tested with values of λ set between 1 and 9. Each test case was analysed, four runs being made with each value of λ. Since no mathematical optimal solution was available to serve as a benchmark for these alignments, we focused our analysis on the consistency of the program. We found that overall the deviation of the score of equivalent alignments was <0.5%. This deviation tended to remain constant with different values of λ. The deviation of the score of the comparison with the reference alignment was higher (3.2% on m1, 2.1% on m2 and 0.4 residues on m3) and tended to increase slightly with higher values of λ. In order to verify that the use of dynamic programming (DPAN) was not adding some uncontrolled bias, most of these experiments were repeated while switching off the DPAN seeding and DPAN mutation described in Methods. Results obtained in that way were consistent with the rest of our experiment. This also allowed us to confirm that the use of DPAN gives an ~3-fold speed-up to the optimization procedure and does not create any premature convergence problem.

Finally, an attempt was made to establish the complexity of the algorithm as a function of the different parameters (Fig. 3a and b). Due to the properties of the OF the time needed to compute one generation increases linearly with the average length of the alignments in the population. This average length is roughly similar to the length of a regular dynamic programming alignment. For a typical test case (7 in Table 1) the average time needed for one generation is of the order of 54 s CPU time. The number of generations needed to reach an optimal solution, however, is a function of several factors, including the value of λ, the length of the alignment and the similarity of the sequences. Our experiments show that the level of similarity has a significant effect on the time requirement. This is in agreement with previous observations made on protein sequences using a similar model of alignment (23). The complexity of the gap pattern (as seen from the point of view of the operators) tends to increase with the distance between the two sequences, making it harder for the GA to find the right configuration.

We also found (Fig. 3a) that for a fixed length and a fixed level of identity the number of generations needed to reach the optimum increases with λ. However, more remarkable is that for λ = 0 the number of generations required tends to be independent of the alignment length. This means that under these conditions the time requirement increases almost linearly with the sequence length (this observation holds when seeding is done in a random way). In theory this is a clear improvement over dynamic programming, which requires at least a quadratic amount of time. In practice, however, the overhead is so large that the sequences to be aligned would need to be extremely long (>10 000 nt) for this speed-up to become really noticeable and we still need to check that this linearity holds for such long sequences.

Tuning of λ

Corpet and Michot described their OF as giving the best results with λ = 3. Since we aligned sequences with a wide range of identity, it was important to know whether λ should be set to a value that is a function of the distance between the slave and the master sequence. For each of our seven test cases the accuracy, as measured by m1, m2 and m3, was plotted against λ. Most of the graphs show reasonable continuity, as shown in Figure 4a and b. We deduced an optimal value for λ from each of these graphs and found the results to be mostly consistent with each similarity measure used (m1, m2 or m3). Figure 5 is a plot of ‘optimal λ’ against the slave/master distance. It shows that the value of λ should roughly reflect the level of identity between the two sequences analysed. It should be higher for sequences of low identity and low for very similar sequences.

Our results also indicate that λ is quite a robust parameter and that a variation of one or two around the optimal value has little effect on the actual quality of the alignment. Such a robustness means that one can perform a dynamic programming alignment beforehand, measure the distance (using Kimura or any other scheme) and deduce from this a reasonable λ. For instance, λ should be set to 1 for closely related sequences (distance <0.5 estimated substitutions/site), 3 for more distantly related pairs (distance <1) and 5 for more remote homologues (distance >1).

Comparison with dynamic programming

The new alignments generated with PRAGA were compared with those obtained by dynamic programming. In all cases (Fig. 6, Table 1) we found that using our method leads to a significant improvement over the dynamic programming approach regardless of the comparison measure. Although both methods follow the same trend and decrease in accuracy when the slave/master distance increases, PRAGA is clearly less affected than dynamic programming. The accuracy of the alignments produced by PRAGA is also clearly a function of the slave/master structural similarity. For instance, let us consider test cases 6–9. These alignments have comparable distances (1.23–1.33 estimated substitutions/site), therefore it seems that the factor responsible for the lower accuracy observed for 6 and 8 is mostly due to the fact that in these alignments the level of conservation of secondary structure is lower than for 7 and 9 (see column ‘pairs’ in Table 1).

Comparison with RNAlign

An attempt was made to align each of the nine test cases using the program RNAlign (18). This was done on a Pentium PC with 64 MB memory. Only two of the test cases (3 and 5) could be aligned successfully. All the others caused the machine to run out of memory or the program to issue a warning message. For 3 and 5 we tuned λ as we did for PRAGA. The optimal values found were the same as those reported with the GA. We found RNAlign alignments to be roughly similar to PRAGA with the three measures (for instance measure m3 gave 89.3% for test case 3 and 68.6 for 5). These results are quite consistent with those obtained with PRAGA (Table 1) and constitute one more piece of evidence that the optimization procedure is accurately performed by our program.

The reason why the other test cases could not be aligned has to do with the way RNAlign works. It first produces a dynamic programming alignment with local gap penalties. In the second stage it identifies some ‘anchor points’ in this alignment. These are regions of the alignment that can be considered as correctly aligned, using some conservative evaluation scheme. During the last stage, considering the area in between the anchor points, RNAlign performs a complex dynamic programming that takes into account both primary and secondary structure constraints. This dynamic programming is very intensive in terms of time [O(M^2N^3)] and memory [O(M^2N^2)]. This means that when trying to align sequences with low levels of identity the setting of anchor points is difficult and leads the program to re-align stretches of the alignment much too long to be handled in that way. In practice, the longer the sequences, the more similar they need to be for RNAlign to align them. To overcome these limitations the authors use a multiple alignment (Bank) instead of a single master sequence. From this multiple alignment they remove areas of very low identity by a semi-automatic method and then use this reduced profile in RNAlign.

Computation of pseudoknots

Pseudoknots are structures that involve interaction of a loop with a domain on the 3′- or 5′-side of its stem (16,37). They can be considered as RNA tertiary motifs. Computationally, prediction of pseudoknots is very difficult using traditional approaches (38). In their method Corpet and Michot (18) had to exclude pseudoknots. It is interesting to notice that in the case of PRAGA there is no real distinction between a pseudoknot and any other type of Watson–Crick interaction. This means that the algorithm should have no more difficulty in aligning pseudoknots than normal stems. In the previous experiments, in order to remain consistent with RNAlign, we excluded these interactions from the master structure. In order to demonstrate the ability of PRAGA to deal with such structures we re-introduced some of them. We only considered pseudoknots involving more than two residues and associated through Watson–Crick interactions. By doing so it is possible to add 6 bp to the previously used structure. These base pairs are boxed in green in Figure 7a and b.

The GA was then used with this new master structure, setting λ to the optimal value previously reported. The experiment was performed on four of the test cases (4, 6, 7 and 9) and the results are given in Table 2. They show unambiguously that our program can efficiently use pseudoknot information in order to improve the alignment. It should be noted that the computation of pseudoknots has no noticeable effect on the algorithmic complexity previously discussed. The fact that even without having pseudoknots present in the master structure (Table 2, PRAGA–PN) PRAGA improves the alignment is due to the constraint imposed by other structures in the vicinity of these pseudoknots (see Fig. 7).

Table 2. Incorporating pseudoknot information

TC	Distance	m2 (residues)			m3 (%)		
		DP	PRAGA (Struc –PN)	PRAGA (Struc +PN)	DP	PRAGA (Struc –PN)	PRAGA (Struc +PN)
4	0.43	0.00	0.00	0.00	100	100	100
6	1.23	2.80	0.61	0.00	16.6	50.0	100
7	1.26	4.90	0.25	0.00	0.00	50.0	100
9	1.33	13.5	0.20	0.00	0.00	66.6	100

Alignments were made incorporating into the master structure some of the positions known to form a pseudoknot (green boxes in Fig. 7). These positions make a total of six new pairs of nucleotides. The alignments were compared to the reference for their accuracy on the newly added positions. m2 and m3 were calculated on the new pairs. The results (Struc +PN) were compared with those obtained by dynamic programming with local gap penalties (DP) and by PRAGA without the pseudoknot information (Struc –PN).

a

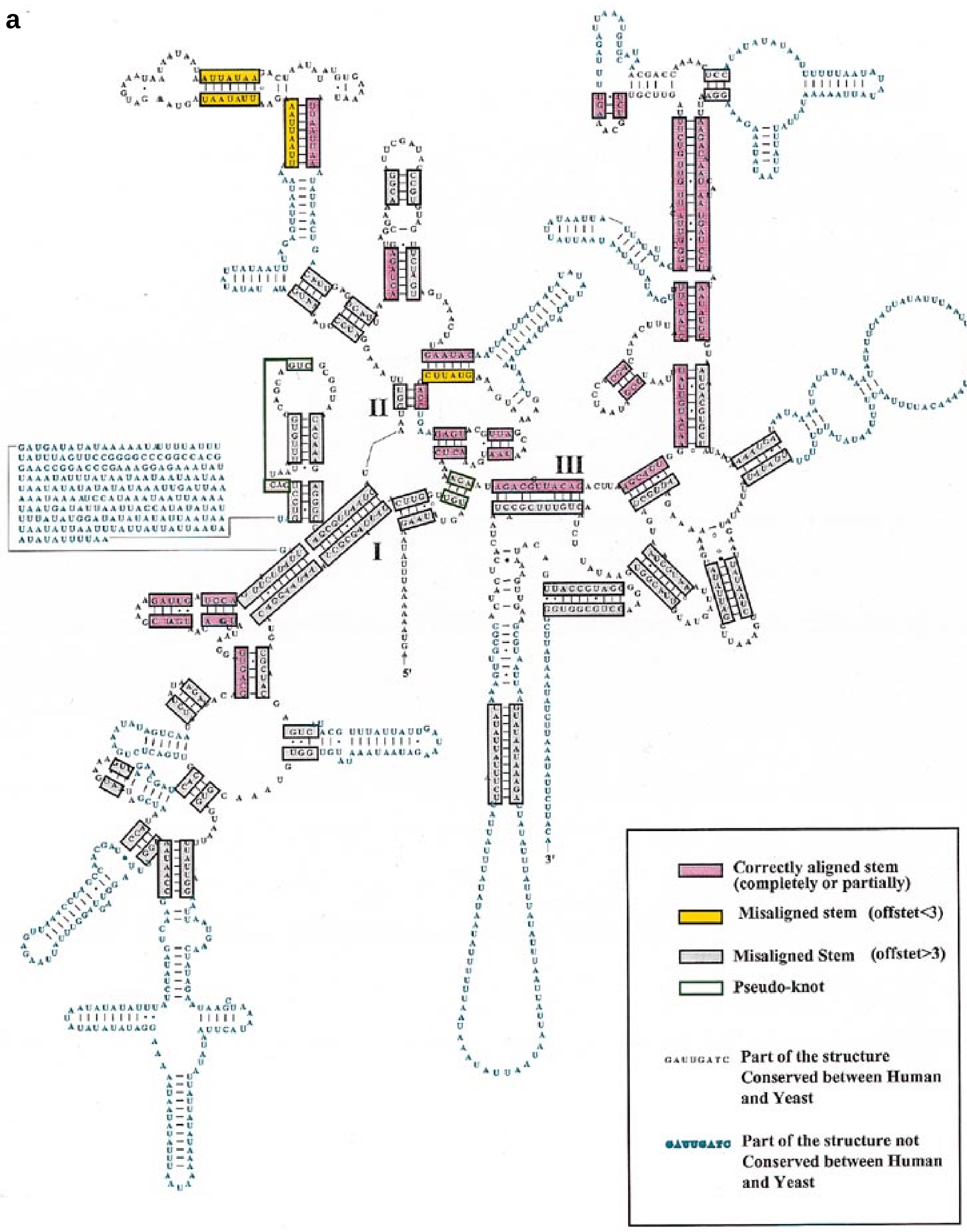
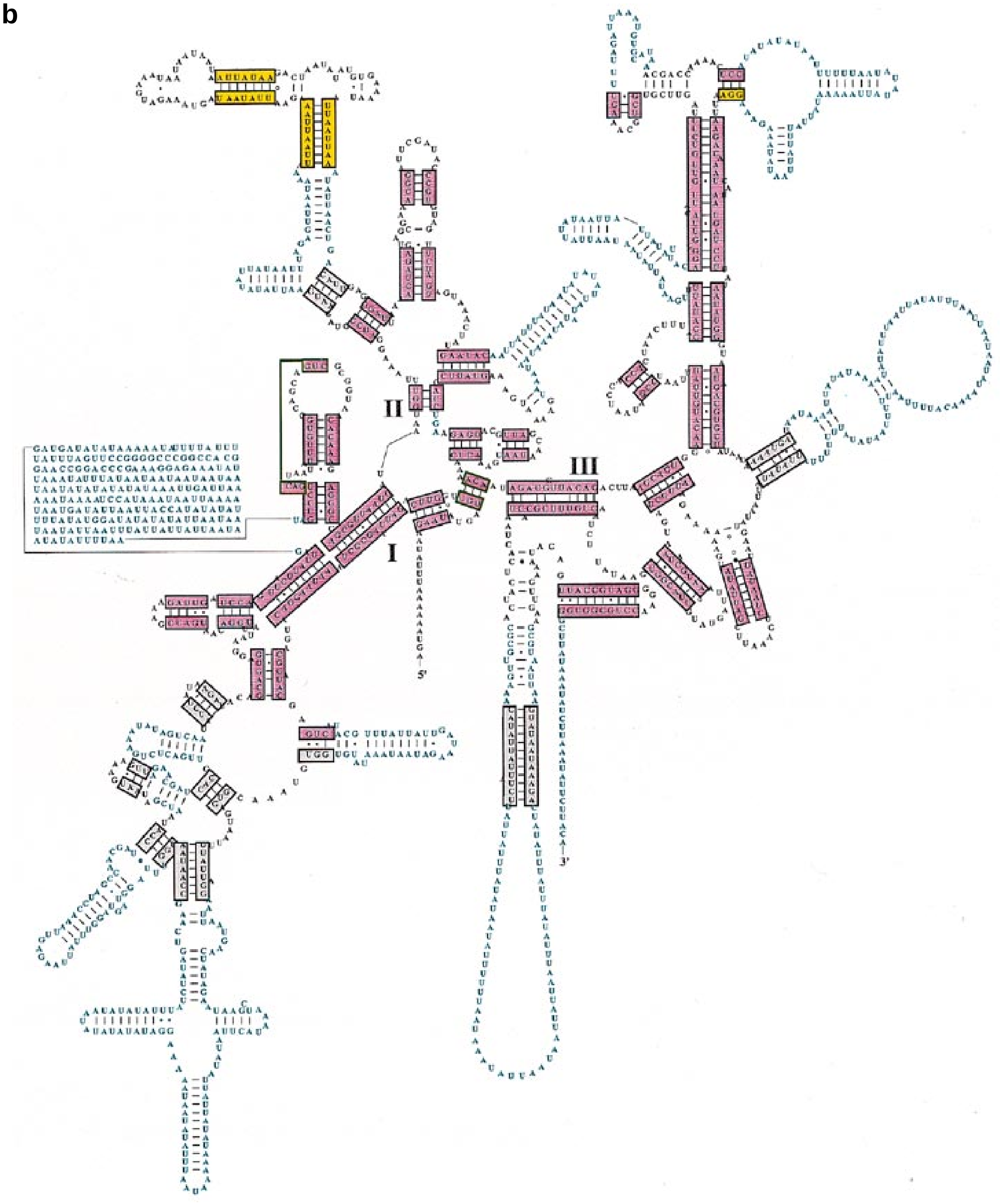

GAUGAUAUAUAAAAAAUAUUUUAUUU
UAUUUAGUUCCGGGGCCCGGCCACG
GAACCGGACCCGAAAGGAGAAAUAU
UAAAUAUUUAUAAUAAUAAUAAUAA
UAAUAUAUAUAUAUAUAAAUUGAUUAA
AAAUAAAAUCCAUAAAUAAUUAAAA
UAAUGAUAUUAAUUACCAUAUAUAU
UUUAUAUGGAUAUAUAUAUUAAUAA
UAAUAUUAAUUUAUUAUUAUUAAUA
AUAUAUUUUAA
I
II
III
5'
3'
Correctly aligned stem (completely or partially)
Misaligned stem (offstet<3)
Misaligned Stem (offstet>3)
Pseudo-knot
GAUUGATC Part of the structure Conserved between Human and Yeast
GAUUGAUC Part of the structure not Conserved between Human and Yeast

Figure 7. Comparison of dynamic programming (**a**) and PRAGA (**b**) on test case 9. The boxed stems indicate stems that were included in the master structure (*Homo sapiens* mitochondrion). Blue portions are the elements not shared by the two structures (and therefore not part of the master structure). A stem is considered correctly aligned if at least one position of its alignment is strictly identical to the reference alignment. This scheme does not take pairing into account. This explains why it is possible to have only the left or the right strand of a stem correctly aligned. The pseudoknots boxed in green are those that were used in the alignment (see text).

b



DISCUSSION

PRAGA is a powerful tool for RNA alignment. Using existing structures allowed us to predict quite accurately the core structure of several ribosomal sequences, even those remotely related to the sequence for which the master structure was known. We also show that this type of analysis can capture some of the tertiary properties of the folding, such as pseudoknot interactions.

The next step with PRAGA will be implementation of an OF that allows the use of a whole alignment instead of a single master sequence. We are currently investigating ways of maximizing the information that can be extracted from such alignments (sequence weighting, local penalties, local substitution schemes, use of secondary structure, etc.). Using a GA gives us a lot of freedom in the design of the OF. In practice, almost any type of constraint or information can be built into an OF and used for optimization purposes. This could include, for instance, SCFG-based functions, which have sounder theoretical justifications than the function we have been using here (39). Another possible extension of PRAGA would be to use the alignment to predict non-conserved stems between correctly aligned core stems with traditional folding prediction methods (4). We are currently investigating ways of defining the local reliability of a given alignment in order to produce a complete structure.

The RAGA algorithm itself is mostly adapted from SAGA (23). This means that the operators used by RAGA were initially designed for aligning protein sequences (linear alignments). It is surprising that these operators do so well in a context where long range non-linear pairings are involved. We believe that this can be explained by the balance between potentially disruptive operators, such as crossovers, and the ability of the other operators (mutations) to fix these disruptions. This balance is automatically maintained by dynamic scheduling of the operators usage probability. It should be noted that the uniform crossover is much less disruptive than would be other forms of exchange, such as the one point crossover described for SAGA. In many cases the uniform crossover tends to respect long range interactions, simply because they often belong to stable parts of the alignment and are likely to be widespread in the population. This approach could probably be taken further and new operators could be designed with respect to the RNA tree structures (40) or using stochastic context-free grammar.

A very important aspect of PRAGA is its ability to deal with noise. By noise we mean unconserved secondary elements that are present in the master structure and absent in the slave structure. We have shown that although such elements significantly decrease the performance of our algorithm, overall, even in these cases, PRAGA remains significantly better than any other alternative we know of. In the future we will focus our attention on defining a better OF that would be able to discriminate this type of event, therefore minimizing their negative effect.

Although our algorithm has been specifically designed for aligning rRNA, it can in theory be applied to any structured RNA. An aspect of PRAGA that still needs to be studied would be its ability to discriminate between RNA structural homologues. The program is too slow to be used for scanning databases as it is now, but since it is a GA, it can also produce sub-optimal solutions in a shorter time. In our experience the quality of the solution found by PRAGA in the first 10% of a run is usually a good indicator of the overall score that may be reached after further optimization. This property could be used to design a 'short PRAGA' as some sort of filter, later focusing the long runs on candidates having a potentially good structural match with some query sequence.

ACKNOWLEDGEMENTS

This work was partly supported by a grant from the EC Biotechnology Program (BIO4-CT95 0130). We wish to thank Burkhart Rost and Miguel Andrade for useful comments. We also wish to thank the three anonymous referees for their useful remarks and interesting suggestions.

REFERENCES

1 Needleman,S.B. and Wunsch,C.D. (1970) *J. Mol. Biol*., **48**, 443–453.
2 Michot,B., Qu,L.H. and Bachellerie,J.P. (1990) *Eur. J. Biochem*., **188**, 219–229.
3 Gutell,R.R., Weiser,B., Woese,C.R. and Noller,H.F. (1985) *Prog. Nucleic Acid Res. Mol. Biol*., **32**, 155–216.
4 Zuker,M. (1989) *Science*, **244**, 48–52.
5 Gouy,M. (1987) In Bishop,M.J. and Rawlings,C.J. (eds), *Nucleic Acid and Protein Sequence Analysis: A Practical Approach*. IRL Press, Oxford, UK, pp. 259–284.
6 Schmitz,M. and Steger,G. (1996) *J. Mol. Biol*., **255**, 254–266.
7 Shapiro,B.A. and Wu,J.C. (1996) *Comput. Applicat. Biosci*., **12**, 171–180.
8 Gultayaev,A.P., van Batenburg,F.D.H. and Pleij,C.W.A. (1995) *J. Mol. Biol*., **250**, 37–51.
9 Ogata,H., Yutaka,A. and Minoru,K. (1995) *Nucleic Acids Res*., **23**, 419–426.
10 Sankoff,D. (1985) *SIAM J. Applicat. Math*., **45**, 810–825.
11 Eddy,S.R. and Durbin,R. (1994) *Nucleic Acids Res*., **22**, 2079–2088.
12 Kim,J., Cole,J.R. and Pramanik,S. (1996) *Comput. Applicat. Biosci*., **12**, 259–267.
13 Lathrop,R.H. (1994) *Protein Engng*, **7**, 1059–1068.
14 Sakakibara,Y., Brown,M., Underwood,R.C., Mian,I.S. and Haussler,D. (1994) In *27th Hawaii International Conference on System Sciences*. IEEE Computer Society Press, Los Alamitos, CA, pp. 284–293.
15 Lefebvre,F. (1995) In *ISMB-95*. AAAI Press, CA, pp. 222–230.
16 Pleij,C. (1990) *Trends Biochem. Sci*., **15**, 143–147.
17 Tabaska,E.J. and Stormo,G.S. (1997) In *ISMB-97*. AAAI Press, Menlo Park, CA, pp. 311–318.
18 Corpet,F. and Michot,B. (1994) *Comput. Applicat. Biosci*., **10**, 389–99.
19 Davis,L. (1991) *The Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, NY.
20 Goldberg,D.E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, New York, NY.
21 Kirkpatrick,S., Gelatt,C.D.J. and Vecchi,M.P. (1983) *Science*, **220**, 671–680.
22 Lawrence,C.E., Altschul,S.F., Boguski,M.S., Liu,J.S., Neuwald,A.F. and Wootton,J.C. (1993) *Science*, **262**, 208–214.
23 Notredame,C. and Higgins,D.G. (1996) *Nucleic Acids Res*., **24**, 1515–1524.
24 Thompson,J.D. (1995) *Comput. Applicat. Biosci*., **11**, 19–29.
25 Thompson,J., Higgins,D. and Gibson,T. (1994) *Nucleic Acids Res*., **22**, 4673–4690.
26 Gerstein,M. and Levitt,M. (1996) In *Fourth International Conference on Intelligent Systems for Molecular Biology*. AAAI Press, Menlo Park, CA, pp. 59–67.
27 Gotoh,O. (1993) *Comput. Applicat. Biosci*., **9**, 361–370.
28 Myers,E.W. and Miller,W. (1988) *Comput. Applicat. Biosci*., **4**, 11–17.
29 Van de Peer,Y., Jansen,J., De Rijk,P. and De Watcher,R. (1997) *Nucleic Acids Res*., **25**, 111–116.
30 Neefs,J.M. and De Wachter,R. (1990) *Nucleic Acids Res*., **18**, 5695–5704.
31 Neefs,J.M., Van de Peer,Y., De Rijk,P., Chapelle,S. and De Wachter,R. (1993) *Nucleic Acids Res*., **21**, 3025–3049.
32 Subramanian,A.R. (1985) *Essays Biochem*., **21**, 45–85.
33 Felsenstein,J. (1988) *Cladistics*, **5**, 355–356.
34 Kimura,M. (1980) *J. Mol. Evol*., **16**, 111–120.
35 Gotoh,O. (1982) *J. Mol. Biol*., **162**, 705–708.
36 Gotoh,O. (1990) *Bull. Math. Biol*., **52**, 509–525.
37 Westhof,E. and Jaeger,L. (1992) *Curr. Opin. Struct. Biol*., **2**, 327–333.
38 Abrahams,J.P., van der Berg,M., van Batenburg,E. and Pleij,C. (1990) *Nucleic Acids Res*., **18**, 3035–3044.
39 Grate,L. (1995) In *ISMB-95*. AAAI Press, Menlo Park, CA, pp. 136–144.
40 Shapiro,B.A., Maizel,J., Lipkin,L.E., Currey,K. and Whitney,C. (1984) *Nucleic Acids Res*., **12**, 75–88.