BIOINFORMATICS

Vol. 18 no. 11 2002
Pages 1494–1499

A simulated annealing algorithm for finding consensus sequences

Jonathan M. Keith [1,*], Peter Adams [1], Darryn Bryant [1], Dirk P. Kroese [1], Keith R. Mitchelson [2,3], Duncan A.E. Cochran [1,2] and Gita H. Lala [1,2]

[1]Department of Mathematics, The University of Queensland, Qld 4072, Australia, [2]Australian Genome Research Facility, The University of Queensland, Qld 4072, Australia and [3]Institute for Molecular Biosciences, The University of Queensland, Qld 4072, Australia

Received on February 26, 2002; revised on April 22, 2002; accepted on April 25, 2002

ABSTRACT

Motivation: A consensus sequence for a family of related sequences is, as the name suggests, a sequence that captures the features common to most members of the family. Consensus sequences are important in various DNA sequencing applications and are a convenient way to characterize a family of molecules.

Results: This paper describes a new algorithm for finding a consensus sequence, using the popular optimization method known as simulated annealing. Unlike the conventional approach of finding a consensus sequence by first forming a multiple sequence alignment, this algorithm searches for a sequence that minimises the sum of pairwise distances to each of the input sequences. The resulting consensus sequence can then be used to induce a multiple sequence alignment. The time required by the algorithm scales linearly with the number of input sequences and quadratically with the length of the consensus sequence. We present results demonstrating the high quality of the consensus sequences and alignments produced by the new algorithm. For comparison, we also present similar results obtained using ClustalW. The new algorithm outperforms ClustalW in many cases.

Availability: The software is made available upon request.

Contact: jonathan@maths.uq.edu.au

INTRODUCTION

The need to determine a consensus sequence for a family of related sequences arises in various applications within computational biology. For example, in DNA sequencing the same region of a genome is sequenced multiple times and a consensus sequence is determined. A consensus sequence also provides a convenient summary of the common elements in a family of sequences. Somewhat related to the problem of finding a consensus sequence is that of forming a multiple sequence alignment. Multiple sequence alignment is a ubiquitous theme in computational biology and is used to detect structural, functional and evolutionary relationships in families of DNAs, RNAs and proteins. Three-dimensional structures can also be characterized as sequences, and alignments involving such structures provide insights into the structure and function of biological molecules. A recent survey on multiple sequence alignment is given in Jiang and Wang (2002).

Although our main focus in this present paper is on an algorithm for constructing a consensus sequence, we also include significant digressions on multiple sequence alignment. A consensus sequence induces a multiple sequence alignment with very little additional work and our results indicate that alignments obtained in this manner compare very favourably with alignments produced by the well-known Clustal software (Thompson *et al.*, 1997). This finding adds substantially to the value and general interest of this work.

There is very little literature devoted exclusively to the problem of finding a consensus sequence. Day and McMorris (1993) showed that the problem is NP-complete in general. Gusfield (1997) provides a brief but valuable discussion of the problem. We mention two existing algorithms. The first is well-known (and not attributable to any single author or group) and consists of forming a multiple sequence alignment for the family of sequences in question, then taking a (possibly weighted) consensus of the characters in each column of the alignment. The second (which is due to Gusfield (1997)) closely resembles that of Wang *et al.* (1996) for the phylogenetic alignment problem. However, this second algorithm merely selects one of the members of the family of sequences as a candidate consensus sequence. Mention should also be made

*To whom correspondence should be addressed.

of Li *et al.* (2000), which describes several polynomial time approximation schemes for determining consensus sequences under various assumptions. Finding a consensus sequence may be regarded as a special case of phylogenetic alignment (Altschul and Lipman, 1989; Wang and Gusfield, 1998; Wang *et al.*, 2000, and references) and many of the algorithms designed for that problem could potentially be modified to find consensus sequences.

We require the following notation and definitions. Let $\mathcal{A}$ be a finite set of characters, which we refer to as the *alphabet*, and let $|\mathcal{A}|$ be the number of distinct characters in the alphabet. Let Ω be the set consisting of all sequences formed from characters in the alphabet. Let $S = \{s_1, \ldots, s_q\} \subset \Omega$ be a finite set of $q \geq 2$ sequences, that is, the family of sequences for which we aim to determine a consensus sequence. (In fact, the sequences are not necessarily distinct, so S could be a multiset.) Let n_i be the length of s_i for each $i = 1, \ldots, q$. The essential idea of the algorithm presented in Section 3 is to search the space Ω for an object known as a *Steiner consensus string*. A *Steiner consensus string* for a set S of sequences is, loosely speaking, a sequence which is minimally distant from each of the sequences in the set. To formalise this concept, we require the following definitions, which resemble those given by Gusfield (1997).

DEFINITION 1. *Let Ω be the set consisting of all sequences formed from characters in an alphabet $\mathcal{A}$. Given a family $S = \{s_1, s_2, \ldots, s_q\}$ of sequences in Ω, an arbitrary sequence s (not necessarily in S) and a pairwise scoring function $D : \Omega \times \Omega \to \mathbb{R}$, the* consensus error of s relative to S *is $E(s) = \sum_{i=1}^{q} D(s_i, s)$. A sequence s^* is a* Steiner consensus string *for S if $E(s^*) \leq E(s)$ for all $s \in \Omega$.*

The term 'Steiner consensus string' appears to have been coined by Gusfield (1997) and is appropriate because such a sequence is a *Steiner point* for the set S when (Ω, D) is a metric space. Gusfield also abbreviates the term to 'Steiner string'. Where it would not be confusing to do so, we use the word 'sequence' in preference to 'string'.

In this paper, the pairwise scoring function $D : \Omega \times \Omega \to \mathbb{R}$ is the *edit distance*, that is, the minimum number of single-base insertions, deletions and substitutions needed to transform one sequence into the other. However, we emphasise that the algorithms can be readily adapted to accommodate a wide range of scoring functions. The scoring function does not even have to be symmetric or satisfy the triangle inequality. The method can therefore be adapted for a range of applications in DNA sequencing and sequence alignment.

METHODS

Our algorithm to search for a Steiner string uses the optimization technique known as *simulated annealing*. This technique brings insights from statistical mechanics to bear on optimization, and emulates the way in which a thermodynamic system settles into a low-energy state as the system cools. Since its introduction by Kirkpatrick *et al.* (1983), simulated annealing has been widely used and has been found to perform well on a variety of large combinatorial optimization problems. Simulated annealing uses Markov chain sampling (described below) to search for a global optimum of a real-valued function E on a set $\mathcal{X}$. We may here assume that $\mathcal{X}$ is finite. The essence of the technique is to induce a family of probability mass functions $\{\pi_\gamma : \gamma \in (0, \infty)\}$ on $\mathcal{X}$ such that the probability mass $\pi_\gamma(x)$ associated with a point $x \in \mathcal{X}$ is proportional to $\exp[-E(x)/\gamma]$ for each $\gamma > 0$. A distribution of this form is known as a *Boltzmann distribution* and the term γ is called the *temperature*. Markov chain sampling is then used to sample from a succession of Boltzmann distributions, each with a lower temperature than the previous one. The sequence of temperatures is termed the *annealing schedule*. As the temperature approaches zero, the sequence of samples settles into a local minimum of the function E. With a carefully chosen annealing schedule, the technique will in many cases find a global minimum of the function.

Markov chain sampling (more commonly called Markov chain Monte Carlo simulation) is a technique for sampling elements from a space $\mathcal{X}$ according to a specified distribution π. The idea is to start with an arbitrary element $X_0 = x \in \mathcal{X}$ and generate a Markov chain $\{X_0, X_1, \ldots\}$ in such a way that the limiting distribution of the chain is equal to π. There are two common forms of Markov Chain sampling, namely the Metropolis–Hastings algorithm (Metropolis *et al.*, 1953; Hastings, 1970) and Gibbs sampling (Geman and Geman, 1984; Gelfand and Smith, 1990). In the Metropolis–Hastings algorithm, potential new elements of the chain are generated in accordance with a transition matrix $Q(x, y)$ (where $Q(x, y)$ gives the probability of the transition from x to a new element y). The sampler consists of the following steps performed iteratively:

(1) Given $X_n = x$, draw $Y = y \in \mathcal{X}$ in accordance with the distribution $Q(x, \cdot)$.

(2) Draw a Uniform(0,1) random variable U.

(3) If $U \leq \min\{1, \pi(y)Q(y, x)/\pi(x)Q(x, y)\}$ then set $X_{n+1} = Y$, otherwise set $X_{n+1} = X_n = x$.

Under fairly mild conditions on Q, the limiting distribution of the chain is π. Gibbs sampling was originally developed to sample from Gibbs distributions, but is applicable whenever the state variable is a random vector. Let the dimension of the state space $\mathcal{X}$ be d. Suppose X is a random vector with distribution π. Let $\pi(\cdot \mid x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_d)$ represent the distribution of

the ith co-ordinate of X conditional on the other components (being $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_d$)). Then the sampler consists of the following steps performed iteratively:

(1) Given $X_n = (x_{n.1}, \ldots, x_{n.d})$, generate $Y = (Y_1, \ldots, Y_d)$ consecutively as follows:
Given the values $Y_1 = y_1, \ldots, Y_{i-1} = y_{i-1}$ draw Y_i in accordance with the conditional distribution $\pi(\cdot|y_1, \ldots, y_{i-1}, x_{n.i+1}, \ldots, x_{n.d})$.

(2) Let $X_{n+1} = Y$.

Under certain conditions, specifically irreducibility and aperiodicity of the Markov process, the resulting chain has limiting distribution π. Gibbs sampling is advantageous if it is easier to sample from the conditional distributions than from the full distribution. Gibbs sampling can be shown to be a special case of the single component Metropolis–Hastings algorithm (Gilks *et al.*, 1996).

The Markov chain sampling procedure used by simulated annealing algorithms is usually the Metropolis–Hastings algorithm. However, there is no reason why simulated annealing algorithms cannot use other Markov chain sampling procedures such as Gibbs sampling. In the algorithm presented in the next section we use a new Markov sampler, which generalizes the Gibbs sampler. The new sampler is described in detail by Keith *et al.* (submitted). Here we merely observe that the new sampler allows Gibbs-like sampling from a space in which the dimension of individual elements varies.

ALGORITHM

The idea of the algorithm is to generate a series of sequences in Ω, beginning with the null sequence " " and, ideally, ending with a Steiner string for S. Each sequence is derived from the previous one by a *point mutation*, that is, a substitution, an insertion or a deletion of a single character. The sequences are generated using our generalized Gibbs sampler applied to a succession of Boltzmann distributions.

The algorithm is divided into iterations, and the iterations are sub-divided into phases. For the purposes of this paper, the number of iterations is fixed at $J > 0$. The number of phases at Iteration j ($1 \leq j \leq J$) is K_j. Each phase generates a new sequence in Ω. The current element (that is, the most recently generated sequence) at the beginning of iteration j is given the symbol x_{j0} and the sequence generated by phase k of iteration j is given the symbol x_{jk}. Note that $x_{jK_j} = x_{(j+1)0}$. Thus the order in which sequences are generated is $\{x_{10} =$ " ", $x_{11}, \ldots, x_{1K_1} = x_{20}, x_{21}, \ldots, x_{(J-1)K_{J-1}} = x_{J0}, \ldots, x_{JK_J}\}$. Let $L(x_{jk})$ denote the length of x_{jk} for all $1 \leq j \leq J, 0 \leq k \leq K_j$. We will see that K_j is not determined until the end of iteration j.

The temperature of the Boltzmann distribution is kept constant over all phases of a given iteration. Let the temperature for iteration j be γ_j. At the end of each iteration, the temperature is reduced by a factor r in the range $0 < r < 1$. This simple annealing scheme is rather primitive, particularly because it does not ensure that the chain reaches equilibrium before changing the temperature. Nevertheless, this annealing scheme was found to be adequate for the numerical experiments presented in Section 4. The overall algorithm can now be described as follows:

ALGORITHM 1. **Overall algorithm.**

1. Initialize γ_1 and r.
2. Set $x_{10} =$" ".
3. For $j = 1, \ldots, J$,
   (a) Generate $x_{j1}, \ldots, x_{jK_j}$ using Algorithm 2.
   (b) Set $\gamma_{j+1} = r\gamma_j$.
4. Output x_{JK_J}.

In the numerical experiments in Section 4, we used $\gamma_1 = 1$ and $r = 0.95$.

At Iteration j, Step 3(a) of Algorithm 1 generates $x_{j1}, \ldots, x_{jK_j}$ by calling the following algorithm. For ease of description, we append a *termination character* to each sequence. This allows us to describe an insertion at the end of x_{jk} as an insertion immediately in front of character $L(x_{jk}) + 1$.

ALGORITHM 2. **Iteration j.**

1. Set $k = 0$ and $m = 1$.
2. While $m \leq L(x_{jk}) + 1$
   (a) If $L(x_{jk}) < MAXLEN$, call Algorithm 3 to generate $x_{j(k+1)}$ by deciding whether to insert a character and which character to insert immediately before character m of x_{jk}. Otherwise, set $x_{j(k+1)} = x_{jk}$.
   (b) Set $k = k + 1$.
   (c) If a character was inserted at step 2(a),
      i. Set $m = m + 1$.
      ii. Go back to step 2(a).
   (d) If $m \leq L(x_{jk})$,
      i. Call Algorithm 4 to generate $x_{j(k+1)}$ by deciding whether to delete or substitute character m of x_{jk}, and in the latter case, which character to substitute.
      ii. Set $k = k + 1$.
      iii. If $m \leq L_{jk}$ and a character was deleted at step 2(d)i, go back to step 2(d)i.
   (e) Set $m = m + 1$.

The value of $MAXLEN$ in Step 2(a) should be specified in advance. A very conservative value is $\sum_{i=1}^{q} n_i$. One may set $MAXLEN = \infty$, but this allows the possibility of infinite looping in Step 2. The insertion algorithm referred to at Step 2(a) of Algorithm 2 and the deletion/substitution algorithm referred to at step 2(d)i are described below. Note that both algorithms refer to the consensus error $E(x)$, defined in terms of a pairwise scoring function $D : \Omega \times \Omega \to \mathbb{R}$. For the numerical experiments described in Section 4, D is the edit distance.

ALGORITHM 3. **Try an Insertion.**

1. For each character $a \in \mathcal{A}$,
   (a) Form a sequence $s^+(a)$ by inserting character a into x_{jk} immediately before the mth character.
   (b) Calculate $E(s^+(a)) = \sum_{i=1}^{q} D(s_i, s^+(a))$.
2. Select one of $\{x_{jk}\} \cup \{s^+(a) \mid a \in \mathcal{A}\}$ where x_{jk} is weighted by $e^{-E(x_{jk})/\gamma_j}/(2L(x_{jk})+1)$ and $s^+(a)$ is weighted by $e^{-E(s^+(a))/\gamma_j}/(2L(x_{jk})+3)$ for each $a \in \mathcal{A}$.
3. Return the selected sequence as $x_{j(k+1)}$.

ALGORITHM 4. **Try a Deletion.**

1. Form a sequence s^- by deleting character m of x_{jk}.
2. Calculate $E(s^-) = \sum_{i=1}^{q} D(s_i, s^-)$.
3. For each character $a \in \mathcal{A}$
   (a) Form a sequence $s(a)$ by substituting character a for the mth character of x_{jk} (note x_{jk} is one of the $s(a)$).
   (b) Calculate $E(s(a)) = \sum_{i=1}^{q} D_W(s_i, s(a))$.
4. Select one of $\{s^-\} \cup \{s(a) \mid a \in \mathcal{A}\}$ where s^- is weighted by $e^{-E(s^-)/\gamma_j}/(2L(x_{jk})-1)$ and $s(a)$ is weighted by $e^{-E(s(a))/\gamma_j}/(2L(x_{jk})+1)$ for each $a \in \mathcal{A}$.
5. Return the selected sequence as $x_{j(k+1)}$.

Note that the probability of selecting any particular sequence in Algorithms 3 and 4 is proportional to a Boltzmann distribution multiplied by an additional term involving the length of the new sequence. This additional term is a feature of the generalized Gibbs sampler and is explained in detail in Keith *et al.* (submitted).

To predict the complexity of the overall algorithm, note firstly that the consensus error of $|\mathcal{A}|+1$ sequences is computed in each phase of each iteration. Each computation of a consensus error requires an edit distance to be computed for each $s_i \in S$. This can be done in time proportional to the product of the lengths of the two sequences by dynamic programming (see Gusfield, 1997). Thus the time required by phase k of Iteration j is $O(|\mathcal{A}|L_{jk}\sum_{i=1}^{q} n_i)$. However, computing the edit distance between s_i and each of the candidates for $x_{j(k+1)}$ is not very different to computing the edit distance between s_i and x_{jk}. It is possible to take advantage of the similarities in these calculations to reduce the time required by each phase to $O(|\mathcal{A}|\sum_{i=1}^{q} n_i)$, but we will not present the details of the speed-up here. Now, one might expect that the number of phases in a given iteration is $O(L)$, where L is the length of the Steiner consensus string, and indeed this is what we find experimentally. Since the number of iterations is fixed at $J > 0$, the overall efficiency of the algorithm is $O(J|\mathcal{A}|L\sum_{i=1}^{q} n_i)$. If we further assume that n_i is approximately L for $i = 1, \ldots, q$ then this expression simplifies to $O(J|\mathcal{A}|qL^2)$. That is, the algorithm is linear in the number of sequences in S, and quadratic in the length of the sequences. At this stage, it is unclear whether J can be fixed irrespective of the values of L and q, but our preliminary investigations indicate that the number of iterations required for convergence increases only very slowly with sequence length, if at all.

Once a Steiner consensus string s has been found, a multiple sequence alignment of the sequences in S can be constructed using an algorithm described by Feng and Doolittle (1987) (see also Gusfield, 1997) applied to a star tree with the consensus string at the centre. The idea of that algorithm is to first determine the optimal pairwise alignment of each sequence in S to the consensus string and then to form a multiple sequence alignment consistent with these pairwise alignments. For the readers' convenience, we provide the following specialised version of the alignment algorithm:

ALGORITHM 5. **Star alignment.**

1. For each $i = 1, \ldots, q$, find an optimal pairwise alignment M_i of s_i and s with respect to some scoring function $V(M_i)$ (this is always possible because there are only finitely many alignments of s_i and s).
2. For each pairwise alignment M_i $(i = 1, \ldots, q)$, let $\sigma_i(k)$ be the number of spaces inserted into the row of M_i corresponding to s between characters k and $k+1$ of s for $0 < k < L$. Let $\sigma_i(0)$ be the number of spaces inserted in front of s and let $\sigma_i(L)$ be the number of spaces inserted at the end of s. For each k, $0 \le k \le L$, define $\sigma(k) = \max_{i=1,\ldots,q} \sigma_i(k)$.

3. For each pairwise alignment M_i $(i = 1, \ldots, q)$, form a new matrix M_i' as follows.
   (a) Insert $\sigma(0) - \sigma_i(0)$ columns of spaces in front of M_i.
   (b) Insert $\sigma(k) - \sigma_i(k)$ columns of spaces immediately after character k of s, for $0 < k \leq L$.
4. We claim that in each M_i', the rows corresponding to s are identical and that consequently the matrices M_i' all have the same length L'. It is therefore possible to form an alignment M of S by:
   (a) Removing the row corresponding to s from each alignment M_i'.
   (b) Placing the remaining rows into a matrix of q rows and L' columns.
   (c) Removing any columns that contain only spaces.

NUMERICAL EXPERIMENTS

To test the algorithm, extensive simulations were performed. Fragments of DNA sequence were randomly selected from a database containing sequences of total length approximately 7 Mbp. For each fragment, simulated point mutations were performed at random to generate a number of variant sequences. The algorithm described above was then used to reconstruct the original sequence, given the variant sequences as input. The reconstruction was compared to the original and the number of differences between them was calculated. The results of these simulations are shown in Table 1 and Table 2. Table 1 contains results for simulations involving five variant sequences ($q = 5$) and Table 2 contains results for simulations involving 20 variant sequences ($q = 20$). Each row of the tables corresponds to approximately 1000 simulations. The first column of each table gives the fragment length. The second column gives the probability that a new character will be inserted between any two characters of the original sequence, or at the beginning and end of the sequence. Multiple insertions were allowed at each possible location, with the probability of a second insertion being equal to the probability of the first insertion, and so on. The third and fourth columns respectively give the probabilities that each character will be deleted or substituted with a different character. If the decision was made to substitute a given character, a new character was selected from the three alternatives, with each alternative being given equal probability. The fifth column gives the average number of differences (that is, the average edit distance) between the original sequences and the consensus sequences produced by our algorithm.

For comparison, the variant sequences from each simulation were aligned using the well-known ClustalW software for multiple sequence alignment (Thompson *et al.*, 1994), with the default parameters. A consensus character was determined for each character of the alignment, thus producing a consensus sequence. This sequence was then compared to the original sequence, and the average number of differences is shown in the sixth column of each table. Note that our algorithm produced a sizable improvement over ClustalW in all rows.

Table 1. Results for $q = 5$ sequences

length	P_i	P_d	P_s	$D(O,S)$	$D(O,C)$	d_s	d_c
400	0.01	0.01	0.10	3.33	7.82	88.1	95.0
2000	0.01	0.01	0.10	17.05	39.10	440.9	474.8
400	0.01	0.01	0.20	21.14	33.47	149.3	158.9
2000	0.01	0.01	0.20	106.44	167.65	747.2	794.0
400	0.05	0.05	0.20	46.07	103.50	185.5	221.1
2000	0.05	0.05	0.20	234.57	516.19	928.8	1101.0
400	0.10	0.10	0.10	40.51	113.08	178.1	226.2
2000	0.10	0.10	0.10	204.35	568.83	887.8	1126.7

Table 2. Results for $q = 20$ sequences

length	P_i	P_d	P_s	$D(O,S)$	$D(O,C)$	d_s	d_c
400	0.01	0.01	0.10	0.06	1.76	88.2	107.1
2000	0.01	0.01	0.10	0.34	8.30	440.9	535.6
400	0.01	0.01	0.20	0.65	5.10	149.9	173.9
2000	0.01	0.01	0.20	3.32	24.93	749.6	870.4
400	0.05	0.05	0.20	5.90	58.03	190.2	258.4
2000	0.05	0.05	0.20	29.50	284.56	949.9	1285.7
400	0.10	0.10	0.10	5.82	83.49	183.7	272.1
2000	0.10	0.10	0.10	28.98	415.37	917.7	1355.5

The multiple sequence alignments produced by our algorithm were also scored by summing the pairwise edit distances between all pairs of sequences in the alignment. This sum was then divided by the number of pairs to give the average number of differences between any two rows of the alignment. The averages of these scores over 1000 simulations are shown in the seventh column of each table. For comparison, the alignments produced by ClustalW were scored in a similar manner and the results are shown in the eighth column of each table. Again, our algorithm produced better results in all cases. Note that as the amount of mutation increases, the difference between the results produced by the two methods becomes increasingly apparent. Also note that the difference is more pronounced when 20 variant sequences are used instead of five.

These numerical experiments were performed on a collection of Pentium III PCs, running the Linux Operating system. The code was written in C and compiled using the Gnu C compiler. Running time varied significantly with

sequence length, initial temperature and cooling parameter, but the algorithm was generally slower than ClustalW.

DISCUSSION

We mentioned above that the annealing scheme we use is somewhat primitive and does not ensure the chain reaches equilibrium before lowering the temperature. We also mentioned that the distance used here is only edit distance; some improvement may result from using a biologically motivated (and application-specific) score (for example, a score based on PAM matrices). With these caveats, the authors are satisfied that this procedure is an adequate method for generating consensus sequences and their associated multiple sequence alignments.

The experiments described in the previous section indicate that the new algorithm performs substantially better than the widely used Clustal software both as a method for determining a consensus sequence and as a method for forming a multiple sequence alignment. These are very encouraging results, but we stress that the tests performed here do not conclusively demonstrate the superiority of the new approach. In particular, each test involved a collection of *independently generated* variants of a common ancestral sequence. In many realistic applications, the input sequences are not independently generated, and Clustal may produce more meaningful results on such data. The results remain impressive, nonetheless, and further investigation of the new approach seems warranted.

It should be pointed out that the Metropolis–Hastings algorithm could have been used as the basis for the simulated annealing search instead of our generalized Gibbs sampler. However, we chose to use a Gibbs-like sampler because it seemed more efficient to work through the available moves systematically rather than the randomized method of selection used in the Metropolis–Hastings algorithm. With Metropolis–Hastings, some point mutations might not be considered for many iterations. Another possible disadvantage of the Metropolis–Hastings algorithm is that it considers only two potential actions at each step of the chain, one of which is to repeat the most recent element. Our sampler considers more options at each step, and this should lead to faster convergence.

The algorithm presented in this paper involves a combination of stochastic and deterministic search techniques. The space of all possible sequences is searched stochastically, but the optimal pairwise alignment scores are found using a deterministic algorithm. This combination of techniques is probably not essential to the technique, and gains in efficiency may result from using heuristic or stochastic techniques to form the pairwise alignments.

ACKNOWLEDGEMENT

The support of the Australian Research Council is gratefully acknowledged.

REFERENCES

Altschul,S.F. and Lipman,D. (1989) Trees, stars, and multiple sequence alignment. *SIAM J. Appl. Math.*, **49**, 197–209.

Day,W.H. and McMorris,F.R. (1993) The computation of consensus patterns in DNA sequence. *Math. Comput. Model.*, **17**, 49–52.

Feng,D. and Doolittle,R.F. (1987) Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J. Mol. Evol.*, **25**, 351–360.

Gelfand,A.F. and Smith,A.F.M. (1990) Sampling-based approaches to calculating marginal densities. *J. Am. Stat. Assoc.*, **85**, 398–409.

Geman,S. and Geman,D. (1984) Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE T. Pattern Anal.*, **6**, 721–741.

Gilks,W.R., Richardson,S. and Spiegelhalter,D.J. (1996) *Markov Chain Monte Carlo in Practice*. Chapman and Hall.

Gusfield (1997) *Algorithms on strings, trees and sequences*. Cambridge University Press.

Hastings,W.K. (1970) Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, **57**, 97–109.

Jiang,T. and Wang,L. (2002) Algorithmic methods for multiple sequence alignment. In Jiang,T., Xu,Y. and Zhang,M.Q. (eds), *Current Topics in Computational Biology*. MIT Press.

Keith,J.M., Kroese,D.P. and Bryant,D. (submitted) A generalised Markov chain sampler.

Kirkpatrick,S., Gelatt,C.D. and Vecchi,M.P. (1983) Optimization by simulated annealing. *Science*, **220**, 671–680.

Li,M., Ma,B. and Wang,L. (2000) Near optimal multiple alignment within a band in polynomial time. In *32nd ACM Symposium on Theory of Computing (STOC2000)*. pp. 425–434.

Metropolis,N., Rosenbluth,A.W., Rosenbluth,M.N. and Teller,A.H. (1953) Equations of state calculations by fast computing machines. *J. Chem. Phys.*, **21**, 1087–1092.

Thompson,J.D., Gibson,T.J., Plewniak,F., Jeanmougin,F. and Higgins,D.G. (1997) The ClustalX windows interface: flexible strategies for multiple sequence alignment aided by quality analysis tools. *Nucleic Acids Res.*, **25**, 4876–4882.

Thompson,J.D., Higgins,D.G. and Gibson,T.J. (1994) CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, **22**, 4673–4680.

Wang,L., Jiang,T. and Lawler,E.L. (1996) Approximation algorithms for tree alignments with a given phylogeny. *Algorithmica*, **16**, 302–315.

Wang,L. and Gusfield,D. (1998) Improved approximation algorithms for tree alignment. *J. Algorithms*, **25**, 255–273.

Wang,L., Jiang,T. and Gusfield,D. (2000) A more efficient approximation scheme for tree alignment. *SIAM J. Comput.*, **30**, 283–299.