Rethinking the DNA Fragment Assembly Problem

João Meidanis

Relatório Técnico DCC–23/93

Setembro de 1993

Rethinking the DNA Fragment Assembly Problem

João Meidanis*

Abstract

DNA fragment assembly (FA) is an important problem in molecular biology. It appears in large-scale DNA sequencing tasks. Research related to the FA problem has mainly focused into two approaches: (1) development of software tools useful in practice but using heuristic methods difficult to analyze formally, and (2) formal modeling through theoretical problems, which captures some but not all of the real issues in FA. Our goal is to hybridize these two approaches by building a software tool that is useful to the biologists *and* has well-understood formal properties.

1 Introduction

The acronym DNA (which stands for deoxyribonucleic acid) denotes a class of molecules that carry genetic information in living organisms. Chemically, a DNA molecule is composed of two parallel *strands*, running in opposite directions (Figure 1.) Each strand is a sequence of units called *nucleotides*, identified by their *bases*. There are four bases, indicated by A, C, G, and T (A = adenine, C = cytosine, G = guanine, T = thymine) [28, 17].

The two strands have the same length and the sequence in one strand determines the sequence in the other because A always pairs with T and

*Departamento de Ciência da Computação, Universidade Estadual de Campinas, Caixa Postal 6065, 13081-970 Campinas, SP. E-mail: meidanis@dcc.unicamp.br

C with G in the opposite strand. Thus, we obtain the opposite strand sequence by exchanging A $\leftrightarrow$ T and C $\leftrightarrow$ G and then reversing the resulting string. This is called the *reverse complement* of the original string. The *sequence* of a DNA molecule is just the sequence of one of its strands. Hence, a DNA molecule has two possible sequences, one the reverse complement of the other.

The sequence of a DNA molecule completely characterizes its genetic and biochemical properties, and this results in a great interest in determining it, or *sequencing* DNA. However, current laboratory technology permits the direct sequencing of contiguous stretches of up to about 500 bases. For longer sequences, a divide-and-conquer approach has to be used. One of the most popular such methods is illustrated in Figure 2. In this method, several copies of a single DNA molecule have their strands separated and subjected to random cuts distributed more or less uniformly along each strand. The resulting fragments are then sequenced.

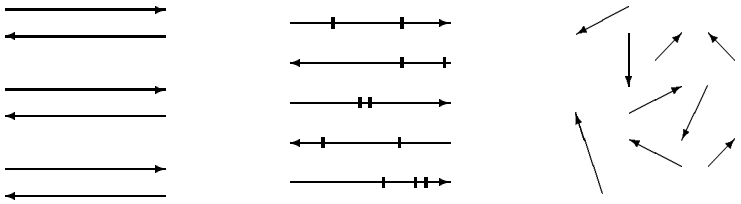
We remark that the sequencing process gives fragment sequences in their correct orientation, but cannot tell which strand they come from. Also, errors usually occur during the process, so that the sequences of two fragments coming from the same region may be slightly different. Finally, notice that regardless of the size of a fragment, at most 500 bases or so will be determined by direct sequencing.

The fragment assembly (FA) problem is then to reconstruct the original molecule's sequence based on the fragment's sequences.

Research related to the FA problem has mainly focused into two approaches. Many papers describe software tools aimed at helping sequencing projects at various levels [25, 26, 27, 4, 31, 22, 3, 14, 7, 12].

Figure 1: Representation of a DNA molecule.

Figure 2: Random sequencing.



On the other hand, several researchers studied the problem from a theoretical standpoint, usually modeling it as finding the shortest common supersequence [8, 30, 2, 15]. In general, papers fall in only one of these categories. (A notable exception is the work by Peltola et al. [22] in which the authors describe a practical tool based on a mathematical model.)

In this paper we try to mix these two approaches. Our goal is to provide an easy-to-use, efficient tool which produces high-quality solutions to the assembly problem. We intend to achieve this by a careful choice of already successful ideas combined with new ones to improve efficiency. In addition, the system will be geared toward large-scale problems, which present unique issues not found in smaller efforts. In contrast to many existing implementations, it will be distributed to interested groups and individuals free of charge.

To be able to establish the formal properties of the program, a formal definition of the assembly problem is needed. The FA problem has been modeled as finding the Shortest Common Supersequence (SCS) of the collection of fragments, but this approach has several drawbacks. A better definition, given by Peltola et al., will be the basis of our own new proposal.

Several of the ideas presented here were conceived when the author was a student at the University of Wisconsin-Madison. Some of them

are described in a recent report on a computerized assembly system [11]. While this system is designed to be a comprehensive tool, here we concentrate in one particular piece that is sometimes called "initial assembly", in which a collection of fragments is received and a preliminary layout is produced. The comprehensive system will include support for editing, inclusion/removal of fragments, statistical analysis, etc.

The rest of the paper is organized as follows. Section 2 is a brief survey of earlier work done in the area. Section 3 discusses the SCS approach and our new definition of the problem. An overview of the system tool we are building appears in Section 4. Finally, we conclude with some remarks and plans for future work in Section 5.

2 Previous Work

In this section we review the literature on software tools for DNA sequencing.

Rodger Staden seems to be the most prolific researcher in this subject [25, 26, 27, 4]. He maintains a software package since the late 70's. His system was initially written in FORTRAN for the VAX[1] and later ported to Unix[2] platforms. Recently, a graphic user interface was incorporated into the system [4]. This interface is written in ANSI C and runs on an X Windows environment.

The basic strategy is as follows. Fragments are processed one at a time. Each new fragment is compared to the previous data, and groups of related fragments spanning a contiguous region are kept in structures called *contigs*. As more fragments are processed, new contigs are created (in case a new fragment doesn't match anything so far), old contigs grow, or contigs are joined (if a connecting fragment appears). After each contig modification, a "consensus" sequence is recomputed, representing the most probable base composition of the region spanned. The output of the system is the contigs remaining after all fragments have been

[1] VAX is a trademark of Digital Equipment Corporation.
[2] Unix is a trademark of Unix Systems Laboratories.

processed.

As Staden himself points out, not much has been published in this area, in spite of its importance and the activity it generates. One probable reason for that is that most of the effort goes into developing commercial products instead of reporting results. Many available computational biology packages, e.g., GCG [5], contain programs that aid DNA sequencing in one way or another. Gunnar von Heijne's book [9] contains a list of available software products for computational biology, including DNA sequencing.

To give some idea of the existing programs, we comment on a few typical implementations in more detail below.

Johnston, Mackenzie, and Dougherty describe a typical system designed to be an inexpensive, readily available tool [12]. It was written in BASIC and runs on IBM-compatible microcomputers, handling up to about 100 fragments. It operates by searching for blocks of contiguous characters that appear in more than one fragment and then linking the fragments that share a number of blocks. Block size is 10 to 12 bases. Although the authors don't give explicit running times, they comment that speed it not an issue because the lab work is usually the limiting factor in a sequencing task. They say their program can process a full day's work by running overnight. While this might be acceptable for small-scale projects, large-scale ones may generate much higher volumes of data, and prompt processing becomes more critical.

Among the practical papers, the one by Peltola et al. is almost unique in its use of deep mathematical concepts [22]. The authors provide a sound formal definition for FA that will be the basis of our own definition presented in Section 3. They take into account both the orientation of the fragments and approximate similarities, but still keep the requirement for a shortest answer, which, as they themselves point out, lacks biological motivation.

Their approach is interesting in several ways. They begin by pairwise comparing all fragments to detect overlap, and constructing a graph where nodes are the fragments and edges link overlapping fragments. Then they seek a subgraph that is an *interval graph*, i.e., its nodes corre-

spond to intervals in the line and adjacent nodes are those that overlap. They don't comment on how they solve orientation incompatibilities.

Peltola et al. report also on the use of their system in practice. Results for input instances with up to 100 fragments are shown, with target molecule sizes of the order of 1.5 kbp (kilo base pairs). Total processing time is between 150 and 200 seconds, 98% of which is spent in the pairwise comparison step. This step uses a dynamic programming scheme improved to take only δmn, where m and n are the sizes of the sequences being compared and δ is the error rate (0.1 or less). Our approach replaces the dynamic programming comparison by a faster fingerprint technique, reducing this time considerably and producing a more balanced system. Time saved in this phase can be used later to improve the alignment.

Finally, they mention that repetitive sequences can confuse any automatic assembly tool, and theirs is no exception. Dear and Staden [4] also comment on this problem and on how they handle it: a fragment will always be put where it matches best. In cases when the fragment matches two places in the same contig, or more than two places in one or more contigs (indicating a possible case of repetitive sequences), the name of the fragment is written to an error log file.

Our approach is similar. We place each fragment where it matches best and issue a warning message if the same fragment is selected for insertion in an incompatible position. As we will see later, we process *pairs* of overlapping fragments instead of single fragments.

Another interesting point in Staden's method is the assessment of the quality of a solution. It depends on the number of times the region has been sequenced on each strand, on whether it is well determined on each strand, and on whether the two strands agree or not.

Other papers describing software aids for DNA sequencing are listed in the reference section.

3 Modeling Fragment Assembly

Let Σ be a fixed alphabet. A *string* is just a word over Σ. A *substring* u of a string w is a part of w involving consecutive characters; w is then a *superstring* of u. For instance, `AG` is a substring of `AGA`, while `AA` is not. The length of a string u will be denoted by $|u|$. For us, a *DNA alphabet* is either the set $\{A, C, G, T\}$ or an extended alphabet including error characters commonly used in fragment assembly [21]. In either case, a complementation operation is defined on its characters so that we can talk about reverse complements. If Σ is a DNA alphabet, we denote by u^{RC} the reverse complement of u, as defined in the introduction.

The *Shortest Common Superstring* problem is to find, for a given collection of strings, a string with minimum length that is simultaneously a superstring of all of them. This problem is known to be NP-hard [6]. Many researchers have studied it and some present DNA fragment assembly as an application [2, 8, 15, 30].

It is certainly the case that exact or approximate solutions to SCS can valuably help solving FA algorithmically. However, as we point out in the sequel, there are a few extra complications in a real FA problem that are not captured by SCS.

First, there is the question of exact versus approximate matching. The SCS formulation assumes that each fragment is an exact copy of a substring of the solution, whereas in practice some errors usually occur. It would be better to allow for, say, k errors, each error being an insertion, deletion, or substitution. More generally, each fragment should be similar to a substring of the solution, with similarity formalized through one of the various string distance measures proposed in the literature [23].

Second, there is the orientation issue. Because the sequencing process cannot identify the strand from which each fragment originated, we have to consider both the fragment and its reverse complement. If one of them fits well with the solution, that is enough. However, the SCS formulation forces us to consider all fragments in the given orientation. This can lead to poorer solutions. For instance, in the input consists of just two sequences, `AATCG` and `CGATTG`, then the SCS is `AATCGATTG`, while the

solution `CAATCG` is shorter and matches exactly the first sequence and the reverse complement of the second sequence.

Third, the requirement that the solution be the shortest possible seems artificial. Of course, without it the problem becomes trivial — just concatenate all input strings. We replace this by the requirement that each fragment should be strongly linked to at least one other fragment, in the sense that a prefix of one of them is strongly similar to a suffix of the other (or one is strongly similar to a substring of the other.)

A related issue is what to do if a fragment fails to align well at all, or, more generally, if the input collection is partitioned into several groups, with members of each group strongly linked together but showing no similarity to members of other groups. The SCS approach is to link the groups by the best existing match (even if it involves just one or two bases), or, in the absence of matches, to concatenate the groups in an arbitrary order. We prefer to leave the groups separate, and admit that sometimes a solution may consist of several unrelated pieces of strongly linked fragments. This agrees with the approach followed by various software packages for FA, written by biologists, in which these pieces are called "contigs" [4, 9].

It is important to permit solutions with many contigs. In practice, it is often the case that an assembly is done with only a subset of the fragments that will be used. Further planning depends on the results of this first assembly, which can tell which regions need more fragments.

A better model for FA is given by Peltola et al. [22]. Their formal definition is as follows. Given a set of fragments $f_1, \ldots, f_k$ and a positive constant δ, find a shortest string F such that for each f_i there is a substring of F whose *edit distance* from f_i or from f_i^{RC} is at most $\delta|f_i|$. For a definition of string edit distances, refer to the book of Sankoff and Kruskal [23]. The particular distance used by Peltola et al. permits character insertions, deletions, substitutions, and transpositions, each with an associated cost.

Notice that the above statement takes into account both the fact that matches can be approximate and the orientation issue. However, they keep the requirement that F must be the shortest possible, even

though they concede that this is not biologically motivated. With this formulation, the problem is still NP-hard [22].

We propose instead the following new definition, which incorporates the possibility of more than one contig and replaces the requirement for a shortest string by conditions ensuring that contigs are well formed and not mergeable.

We need some additional concepts before proceeding. The proposed definition depends on two parameters, δ and k, which control the minimum amount of acceptable approximate similarity and the minimum acceptable overlap length, respectively.

We assume the reader is familiar with the concept of an *alignment* of two sequences and with alignment scores. Given score values for each pair of characters in the alphabet, alignment scores are computed by summing up individual scores for each column, based on this table containing the scores for each pair of bases plus a gap penalty. More details can be found in some of the references [24].

For an alignment α between two sequences, let $s(\alpha)$ be the score of α charging for gaps in the extremities of both sequences, and $s'(\alpha)$ the score without charging for gaps in the extremities of either sequence. We assume that these score measures have been normalized so that the score of a match is 1. Let also $\mathrm{ovlp}(\alpha)$ be the length of the actual overlap between the two sequences in α.

We say that sequences u, v *exhibit prefix-suffix similarity* if there is an alignment α between u and v such that

$$\begin{aligned} \mathrm{ovlp}(\alpha) &\geq k, \text{and} \\ s'(\alpha) &\geq (1-\delta)\mathrm{ovlp}(\alpha). \end{aligned}$$

We are now ready for our new formulation. Given a set of fragments (sequences over a DNA alphabet Σ) $f_1, \ldots, f_k$, find strings $c_1, \ldots, c_m$ over the same alphabet such that

1. For each f_i there is a substring g_i of some c_j and an alignment α between f_i and g_i or between f_i^{RC} and g_i such that

$$s(\alpha) \geq (1-\delta)|f_i|.$$

These g_i's are *fixed* and their *location* within c_j is part of the solution.

2. Each base in each c_j belongs to at least one such g_i.

3. Each contig c_j is well formed in the following sense. Construct a graph with nodes being the g_i's belonging to this c_j and edges between two nodes if the intersection of the corresponding g_i's has at least k bases. The contig c_j is well formed if this graph is connected.

4. For each pair of distinct contigs c_j and c_l we have that c_j does not exhibit prefix-suffix similarity with either c_l or c_l^{RC}.

The above conditions are just minimal requirements to make a solution sensible in biological terms. Further study of the *quality* of a solution is necessary. Dean and Staden implicitly suggest a way of doing this [4]. They define a scale from -2 to $+2$ for the quality of each position in a contig: 0 means that the sequence is well determined in both strands and they agree; 1 and -1 mean that the sequence is well determined in one strand only; 2 and -2 mean that the sequence is well determined in both strands but they disagree.

Reasonable values for δ and k are 0.2 and 10, respectively. Setting $\delta = 0.2$ excludes from consideration similarities of less that 80%. The value $k = 10$ disregards overlaps of less than 10 bases.

The question of evaluating an FA algorithm is another point where we depart from the traditional approach. Although the algorithm has to produce *some* output for any input, no matter how contrived, its behavior on "nice" inputs is more important. "Nice" inputs are those for which a reasonable answer can be expected. In other words, we prefer an algorithm that performs well on good inputs and is lousy on bad inputs to one that does average in all inputs. Contrast this with the SCS literature, where an algorithm that achieves $2\times$ optimum length for all inputs is regarded as good.

This way of evaluating FA algorithms creates a new problem: the problem of evaluating input instances. While we don't know how to solve

this problem in general, we present a simple method of generating good inputs, in which the algorithms can be run for testing. Start with a long sequence generated at random. Choose at random several substrings of this sequence. Take the reverse complement of some of them, leaving the others as they are. Then induce random errors along each such fragment, and give them as input to the algorithm. Finally, compare the output given by the algorithm to the long original sequence.

We see here three parameters that affect the input quality. First, there is the error rate. The lower the error rate, the higher the input quality. Then there is the number of contigs, each contig being in this case a connected component of the interval graph formed by the substrings. Less contigs mean better input. Finally, for each contig containing more than one substring a quality measure was recently developed [18]. This measure, called the *weakest link*, is the minimum overlap that has to be detected in order to link all fragments in this component. If the weakest link is too low, the algorithm is in some sense "forgiven" for not assembling all the fragments of this contig in one single piece.

4 New ideas

Following general practice for the FA problem, our method consists of a first phase of pairwise comparison followed by contig construction by processing pairs of related fragments. This represents a mixture of methods from Peltola et al. [22], Staden [26] and algorithms that have been proposed for the SCS [2]. Some notable differences exist, however, and they are likely to improve substantially both the performance and the accuracy of the system. In addition, new algorithms were developed to interact with the data structures needed.

In this section we concentrate on three new ideas that we are implementing in the new system. These are (1) a fast method for pairwise comparison, (2) processing pairs of fragments instead of single fragments, and (3) a new algorithm for converting an alignment graph into an alignment matrix.

Our pairwise comparison phase, where each fragment is compared to

all others in both orientations, does not use dynamic programming (DP) but uses instead a fast fingerprint technique explained below. In typical input instances, it has been observed that very few pairs share some similarity. For instance, if we have 300 fragments spanning a 10 kbp region, only about 3% of the pairs will be related, assuming fragments uniformly distributed along this region [18]. To run a full DP algorithm on all these pairs just to find out they are not related seems a waste of time. The method we use is about 20 times faster than DP and is able to quickly separate stringly related pairs from unrelated ones. The price to pay for this speed is that the test is less reliable. Some overlapping pairs can be missed, especially the ones with shorter overlap, while some unrelated pairs can be flagged as similar. Nevertheless, such mistakes do not necessarily jeopardize the final assembly. All putatively related pairs found here will be tested through DP later on, and missed pairs with short overlap can be caught by transitivity.

The method works as follows. For each fragment, we slide a window of fixed length (usually 7) across the sequence. For each window, a hash value is computed as in the Karp-Rabin string matching algorithm [13], and the corresponding bit in a hash table is set to one. We thus obtain one such hash table per fragment, and this works as a fingerprint of the fragment. When we need to compare two fragments, we count the number of common bits in both tables, and from that observed value we estimate the actual overlap by elementary statistical techniques, computing what we call the *score* of the pair. Pairs with score above a certain threshold are declared similar. More details can be found in [18]. Other fingerprints can be used, but bit manipulation and counting have proven to be very fast in practice.

Apart from the Karp-Rabin scheme, the idea of using a fast all-pairs comparison is not new, but it was used in a different way in the past, which we will describe in the sequel. Some systems process the fragments one at a time as follows. A list of current contigs is kept. When a new fragment is processed, it is compared (DP) to all current contigs and placed where it matches best. This strategy can be very expensive if the current contig list is long, so an effort is made to keep this list short.

In these systems the order in which fragments are processed is often critical, both to avoid the early appearance of many contigs and to promote strong, stable consensus sequences at the outset. To determine a good processing order, a pairwise comparison step has been employed to select the best sequences and process these first. However, once the processing order is determined, information from the pairwise comparison is no longer used.

This led us to our second improvement, which is the processing by *pairs* of fragments. The idea is to use the pairwise comparison information to avoid searching for the right contig for a fragment. Instead of starting with no contigs and adding fragments one at a time, we start with one contig per fragment. Pairs of fragments declared as similar are processed in decreasing order by score. Each time we process a pair of related fragments we join the corresponding contigs. Candidate pairs are run through a DP comparison first to confirm their similarity. An *union-find* structure [29] is used to quickly locate the contig a fragment belongs to. Because the fragments themselves are compared, this method saves also on consensus computation, since there is no need to have a consensus sequence for the contigs. Only in the final step of the whole process consensus sequences are computed for the contigs.

The implementation of this part is extremely similar to the minimum spanning tree algorithm devised by Kruskal [29].

Orientations are easily handled in such a scheme. In the union-find structure, an extra field records the relative orientation between a node and its parent. The relative orientation between a node and the root of its tree can be determined with the same complexity of a *find* operation, that is, almost constant in the average. Two fragments in the same tree can be tested for orientation inconsistency using the root as intermediate.

Contig stored information must also include the sequences of the fragments involved and a multiple alignment of these. Some systems use a large two-dimensional matrix to store aligned fragments, with each fragment occupying one row and aligned bases occupying the same column. Although this is the preferred contig representation in human-readable output, there is no need to keep the same structure internally. In fact,

it has many drawbacks. For instance, inserting a new fragment that causes a gap in the existing alignment is expensive (a large portion of the array may have to be moved one position to the right). Space for this big matrix is usually allocated statically in the beginning, which is also undesirable.

Instead, we use a dynamic structure. Fragments are represented by linked lists of bases, with aligned fragments sharing bases where they agree. Similar structures have been used in many applications, including multiple alignment [10], approximate pattern matching [19, 20], and large vocabularies [16, 1].

These structures are in fact directed acyclic graphs, and in some cases they are seen as finite automata. This is not appropriate for the FA context, however, because sequences usually do not start in the same node (there is no single initial state) and do not end in the same node. On the other hand, the kind of multiple alignment that arises in a contig is peculiar, because not all fragments are directly related, in general. The structure is usually long and thin, while multiple alignments involving cliques of related sequences are in general shorter and thicker.

Our contribution here is a new algorithm to convert such a structure into the familiar matrix format. Formally, we solve the following problem. Given a directed acyclic graph G, find a function c that maps nodes to integers such that, for every pair of nodes u, v with v reachable from u we have

$$c(v) - c(u) \geq \mathrm{ld}(u, v), \tag{1}$$

where $\mathrm{ld}(u, v)$ is the longest distance between u and v measured in number of edges. We require further that equality holds in (1) if either u is a source or v is a sink. Intuitively, $c(u)$ is the column in which u must be placed. Once column numbers are given, we know where fragments overlap, and lines can be allocated without conflicts. Alternatively, a separate line can be used for each fragment, although this would be wasteful, since a line can be reused after a fragment allocated to it ends.

The algorithm we propose to solve this problem starts at an arbitrary source u making $c(u) = 0$. After that, a series of alternate forward- and backward-propagations follow, assigning column numbers to nodes in

a way that resembles Dijkstra's algorithm for shortest paths [29]. To ensure that nodes are visited in topological order, priority queues are maintained with the nodes yet to be visited during a given propagation. The algorithm runs in $O(m + n \log n)$ steps in graphs with n nodes and m edges. More details can be found in the author's thesis [18].

5 Conclusions

We believe that the ideas contained here represent a step forward in understanding the real issues in fragment assembly and developing effective tools and theories for its study. Nevertheless, several important points still need further work, for instance, dealing with repeated patterns in the target DNA.

As in many problems in computational biology, it is hard to imagine a fully automatic solution, because of the many exceptional circumstances that can arise. At best, we can hope for a software tool that produces a good starting point for a solution, which can then be examined and edited by the biosciences professional until a satisfactory solution is reached.

Acknowledgements

I thank Fred Blattner for sharing with us all his experience in DNA sequencing and computerized fragment assembly, Deborah Joseph for suggesting the use of the Karp-Rabin method, and Augusto Almeida for many clarifying discussions.

References

[1] A. W. Appel and G. J. Jacobson. The world's fastest Scrabble program. *Commun. ACM*, 31(5):572–578,585, 1988.

[2] A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis. Linear approximation of shortest superstrings. In *Proc. 23rd ACM Symposium on Theory of Computing*, pages 328–336, 1991.

[3] Anne Dankaert, Colombe Chappey, and Serge Hazout. 'Size Leap' algorithm: An efficient extraction of the longest common motifs from a molecular sequence set. Application to the DNA sequence reconstruction. *Computer Applications in the Biosciences (CABIOS)*, 7(4):509–513, 1991.

[4] S. Dean and R. Staden. A sequence assembly and editing program for efficient management of large projects. *Nucleic Acids Research*, 19(14):3907–3911, 1991.

[5] J. Devereux, P. Haeberli, and D. Smithies. A comprehensive set of sequence analysis programs for the VAX. *Nucleic Acids Research*, 12:387–395, 1984.

[6] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[7] T. Gingerias, J. Milazzo, D. Sciaky, and R. Roberts. Computer programs for assembly of DNA sequences. *Nucleic Acids Research*, 7:529–545, 1979.

[8] Dan Gusfield, Gad M. Landau, and Baruch Schieber. An efficient algorithm for the all pairs suffix-prefix problem. *Information Processing Letters*, 41:181–185, 1992.

[9] Gunnar von Heijne. *Sequences Analysis in Molecular Biology: Treasure Trove or Trivial Pursuit.* Academic Press, San Diego, CA, 1987.

[10] Jotun Hein. Unified approach to alignment and phylogenies. In Russell F. Doolittle, editor, *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, volume 183 of *Methods in Enzymology*, pages 626–645. Academic Press, 1990.

[11] W. Istvanick, A. Kryder, G. Lewandowski, J. Meidanis, A. Rang, S. Wyman, and D. Joseph. Dynamic methods for fragment assembly

in large scale genome sequencing projects. In *Proc. of the Twenty-Sixth Annual Hawaii Int'l Conf on System Sciences*, volume I, pages 534–543, January 1993.

[12] R.E. Johnston, J. M.Jr. Mackenzie, and W.G. Dougherty. Assembly of overlapping dna sequences by a program written in basic for 64k cp/m and MS-DOS IBM-compatible microcomputers. *Nucleic Acids Research*, 14(1):517–527, 1986.

[13] R. Karp and M. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. of Research and Development*, 31:249–260, 1987.

[14] J. Kececioglu and E. Myers. A robust and automatic fragment assembly system. Unpublished, 1991.

[15] Ming Li. Towards a DNA sequencing theory (learning a string). In *Proc. 31st Symposium on Foundations of Computer Science*, pages 125–134, 1990.

[16] C. L. Lucchesi and T. Kowaltowski. Applications of finite automata representing large vocabularies. *Software — Practice and Experience*, 23(1):15–30, 1993.

[17] C. K. Mathews and K. E. van Holde. *Biochemistry.* Benjamin/Cummings, 1990.

[18] Joao Meidanis. *Algorithms for Problems in Computational Genetics.* PhD thesis, University of Wisconsin-Madison, 1992.

[19] E. W. Myers and W. Miller. Approximate matching of regular expressions. *Bull. Math. Biol.*, 51:5–37, 1989.

[20] Gene Myers. Approximate matching of network expressions with spacers. In *Proc. First Latin American Theoretical Informatics*, volume 583 of *Lecture Notes in Computer Science*, pages 372–386, 1992.

[21] William R. Pearson. Rapid and sensitive sequence comparison with FASTP and FASTA. In Russell F. Doolittle, editor, *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, volume 183 of *Methods in Enzymology*, pages 63–98. Academic Press, 1990.

[22] H. Peltola, H. Söderlund, and E. Ukkonen. SEQAIDS: A DNA sequence assembling program based on a mathematical model. *Nucleic Acids Research*, 12:307–321, 1984.

[23] D. Sankoff and J. B. Kruskal. *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison.* Addison-Wesley, Reading, MA, 1983.

[24] T. F. Smith, M. S. Waterman, and W. M. Fitch. Comparative biosequence metrics. *J. Molec. Evol.*, 18:38–46, 1981.

[25] R. Staden. A strategy of DNA sequencing employing computer programs. *Nucleic Acids Research*, 6:2601–2610, 1979.

[26] R. Staden. A new computer method for the storage and manipulation of DNA gel reading data. *Nucleic Acids Research*, 8(16):3673–3694, 1980.

[27] R. Staden. Automation of the computer handling of gel reading data produced by the shotgun method of DNA sequencing. *Nucleic Acids Research*, 10:4731–4751, 1982.

[28] R. Tamarin. *Principles of Genetics.* Wm. C. Brown Publishers, 3rd edition, 1991.

[29] Robert Endre Tarjan. *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics (SIAM), 1983.

[30] Jonathan S. Turner. Approximation algorithms for the shortest common superstring problem. *Information and Computation*, 83:1–20, 1989.

[31] R.B. Wallace. DNASTAR - a microcomputer-based DNA sequence management system. *Biotechnology Software*, 1:6, 1984.

Relatórios Técnicos – 1992

01/92 **Applications of Finite Automata Representing Large Vocabularies,** *C. L. Lucchesi, T. Kowaltowski*

02/92 **Point Set Pattern Matching in d-Dimensions,** *P. J. de Rezende, D. T. Lee*

03/92 **On the Irrelevance of Edge Orientations on the Acyclic Directed Two Disjoint Paths Problem,** *C. L. Lucchesi, M. C. M. T. Giglio*

04/92 **A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams,** *W. Jacometti*

05/92 **An (l, u)-Transversal Theorem for Bipartite Graphs,** *C. L. Lucchesi, D. H. Younger*

06/92 **Implementing Integrity Control in Active Databases,** *C. B. Medeiros, M. J. Andrade*

07/92 **New Experimental Results For Bipartite Matching,** *J. C. Setubal*

08/92 **Maintaining Integrity Constraints across Versions in a Database,** *C. B. Medeiros, G. Jomier, W. Cellary*

09/92 **On Clique-Complete Graphs,** *C. L. Lucchesi, C. P. Mello, J. L. Szwarcfiter*

10/92 **Examples of Informal but Rigorous Correctness Proofs for Tree Traversing Algorithms,** *T. Kowaltowski*

11/92 **Debugging Aids for Statechart-Based Systems,** *V. G. S. Elias, H. Liesenberg*

12/92 **Browsing and Querying in Object-Oriented Databases,** *J. L. de Oliveira, R. de O. Anido*

Relatórios Técnicos – 1993

01/93 **Transforming Statecharts into Reactive Systems,** *Antonio G. Figueiredo Filho, Hans K. E. Liesenberg*

02/93 **The Hierarchical Ring Protocol: An Efficient Scheme for Reading Replicated Data,** *Nabor das C. Mendonça, Ricardo de O. Anido*

03/93 **Matching Algorithms for Bipartite Graphs,** *Herbert A. Baier Saip, Cláudio L. Lucchesi*

04/93 **A lexBFS Algorithm for Proper Interval Graph Recognition,** *Celina M. H. de Figueiredo, João Meidanis, Célia P. de Mello*

05/93 **Sistema Gerenciador de Processamento Cooperativo,** *Ivonne. M. Carrazana, Nelson. C. Machado, Célio. C. Guimarães*

06/93 **Implementação de um Banco de Dados Relacional Dotado de uma Interface Cooperativa,** *Nascif A. Abousalh Neto, Ariadne M. B. R. Carvalho*

07/93 **Estadogramas no Desenvolvimento de Interfaces,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

08/93 **Introspection and Projection in Reasoning about Other Agents,** *Jacques Wainer*

09/93 **Codificação de Seqüências de Imagens com Quantização Vetorial,** *Carlos Antonio Reinaldo Costa, Paulo Lício de Geus*

10/93 **Minimização do Consumo de Energia em um Sistema para Aquisição de Dados Controlado por Microcomputador,** *Paulo Cesar Centoducatte, Nelson Castro Machado*

11/93 **An Implementation Structure for RM-OSI/ISO Transaction Processing Application Contexts,** *Flávio Morais de Assis Silva, Edmundo Roberto Mauro Madeira*

12/93 **Boole's conditions of possible experience and reasoning under uncertainty,** *Pierre Hansen, Brigitte Jaumard, Marcus Poggi de Aragão*

13/93 **Modelling Geographic Information Systems using an Object Oriented Framework,** *Fatima Pires, Claudia Bauzer Medeiros, Ardemiris Barros Silva*

14/93 **Managing Time in Object-Oriented Databases,** *Lincoln M. Oliveira, Claudia Bauzer Medeiros*

15/93 **Using Extended Hierarchical Quorum Consensus to Control Replicated Data: from Traditional Voting to Logical Structures,** *Nabor das Chagas Mendonça, Ricardo de Oliveira Anido*

16/93 **$\mathcal{LL}$ – An Object Oriented Library Language Reference Manual,** *Tomasz Kowaltowski, Evandro Bacarin*

17/93 **Metodologias para Conversão de Esquemas em Sistemas de Bancos de Dados Heterogêneos,** *Ronaldo Lopes de Oliveira, Geovane Cayres Magalhães*

18/93 **Rule Application in GIS – a Case Study,** *Claudia Bauzer Medeiros, Geovane Cayres Magalhães*

19/93 **Modelamento, Simulação e Síntese com VHDL,** *Carlos Geraldo Krüger e Mário Lúcio Côrtes*

20/93 **Reflections on Using Statecharts to Capture Human-Computer Interface Behaviour,** *Fábio Nogueira de Lucena e Hans Liesenberg*

21/93 **Applications of Finite Automata in Debugging Natural Language Vocabularies,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

22/93 **Minimization of Binary Automata,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

Departamento de Ciência da Computação — IMECC
Caixa Postal 6065
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

reltec@dcc.unicamp.br