ALGORITHMS FOR LARGE-SCALE DNA SEQUENCING

F. R. CERQUEIRA, J. MEIDANIS

Institute of Computing, University of Campinas, Av. Albert Einstein, 1251, P.O.Box 6176, 13083-970 Campinas, Sao Paulo, Brazil

Estamos em plena era genômica. As descobertas científicas nesta área estão tendo um enorme impacto na medicina, agricultura e outros setores fundamentais da atividade humana. Os países que não dominarem as tecnologias envolvidas correm o risco de de tornar perigosamente depenedentes dos que as dominam. Neste trabalho damos um passo importante para dominar uma técnica crucial: algoritmos para seqüenciamento em larga escala de DNA.

The world is experiencing a true genomic revolution. The new findings in this area have a tremendous impact on medicine, agriculture, and other fundamental sectors of human activity. The countries that do not dominate the technology involved risk becoming dangerously dependent on the ones that dominate them. This work is an important step towards dominating a crucial technique: algorithms for large-scale DNA sequencing.

1 Introduction

In the era of large-scale genome sequencing, algorithms for DNA fragment assembly are of utmost importance. There are many programs available to do DNA fragment asssembly (phrap, CAP3, Staden, etc.) but new ones are constantly being built to deal with the increasing complexity and size of the genomes being tackled. The private company Celera Genomics, Inc., had to build their own assembler for the very large and complex *Drosophila* and human genomes. It is likely that Celera considers their software strategic and will not release it to the general public, not even commercially.

In this work we present our recipe for assembling fragments using overlap graphs. We used this model in an implementation of an assembler named `concam`, that performed very well when compared to `phrap`[1], one of the most popular DNA assemblers today. Our implementation performs basically a *clean-up* step that trims fragments, a *matching* phase that looks for a matching in the overlap graph, and a *recombination* phase to deal with cycles in the graph. An important characteristic of this implementation is the use of *iteractions*: the solution to the problem is built by successive applications of the steps above, relaxing the conditions at each iteration.

This work is organized as following. In Section 2 we introduce the DNA fragment assembly problem with its peculiarities, and select one formalization for the problem. In Section 3 we present details of our implementation. Some experimental results come in Section 4, followed by our conclusions and proposal of extensions to the tools.

2 The DNA Fragment Assembly Problem

DNA is a molecule that lies in each cell in every living organism, and serves as a cookbook that a cell uses to build proteins. Proteins, in turn, ultimately make life possible. DNA is a double stranded molecule. Every strand is a linear arrangement of smaller molecules called nucleotides or bases. Nucleotides are four: adenine (`A`), cytosine (`C`), guanine (`G`) and thymine (`T`). The two strands of a DNA molecule are bound in such a way that an `A` in one strand is paired to a `T` in the other and a `C` in one strand is paired to a `G` in the other. The strands have opposite *orientations* and are said to be *complemented* with respect to each other.

Sequencing a DNA molecule is to obtain the string of letters that correspond to its nucleotides. This data is very useful for molecular biologists and geneticists. A DNA molecule size ranges from a few thousand nucleotides to more than a billion nucleotides. Current sequencing technology enables the direct sequencing of about 1000 nucleotides with the additional drawback that reliability of sequence is low in regions close to the ends. It also does not contain information about which strand originated the fragment. DNA sequencing strategies can be roughly seen as: obtaining enough copies of the DNA, splicing the DNA to obtain fragments that overlap and can be directly sequenced, obtaining enough copies of each fragment, sequencing the fragments, and assembling them together.

Issues that a DNA assembler has to deal with include: the *fragment orientation*, that is, which strand a fragment come from; *base call* errors, that is, the deletion, insertion or substitution of nucleotides; *contamination* of DNA during the copy process, adding DNA from organism (vector) that is used to make copies; *chimeric fragments*, that is, fragments from different regions of DNA that bind together during the splicing phase; *lack of coverage* of some portions of DNA by at least one fragment; *repeated regions*, that is, copies of a fraction of DNA that appear elsewhere in the same molecule.

The *shortest common substring* model does not account for most of these problems. The models *Reconstruction*[2,3], *Multicontig*[4] and *minimum k-contig*[5] deal with some of these situations, but all lead to NP-hard problems. The model we used is a variation of minimum k-contig.

2.1 The Minimum k-contig Formalization

Let us denote the prefix with length l of a sequence s by $\mathit{prefix}(s,l)$ and the suffix with length l of s by $\mathit{suffix}(s,l)$.

For a given set of fragments $\mathcal{F}$ we build the *overlapping graph* $G(\mathcal{F}) = (V,E)$ as follows. For every fragment $s \in \mathcal{F}$ we add a vertex in $G(\mathcal{F})$. Given two vertices in $s,t \in V$, we add an oriented edge (s,t) if and only if $s \neq t$, $s \neq \bar{t}$, and there is an integer l, with $1 \leq l \leq \min(|s|,|t|) - 1$ such that $\mathit{suffix}(s,l) = \mathit{prefix}(t,l)$. If $s,t \in V$ and s is a proper subsequence of t, we call s a *Steiner sequence*. We call a non-Steiner sequence a *terminal sequence*. Since sequences are also vertices of V, we use the term *Steiner vertex* to refer to a vertex corresponding to a Steiner sequence, and *terminal vertex* analogously.

Let be two sequences s e t such that $\mathit{suffix}(s,l) = \mathit{prefix}(s,l)$ and t's length is n. We represent by $s|_l t$, the concatenation of s with the last $n-l$ characters of t. To represent the successive concatenation of m sequences from the leftmost to the rightmost we use $s_1|_{l_1} s_2|_{l_2} \ldots s_{m-1}|_{l_{m-1}} s_m$. A $k-$*path* in $G(\mathcal{F})$ is a path $P = (s_1, s_2, ..., s_m)$ such that for every pair s,t of adjacent vertices in the path we have $\mathit{suffix}(s,l) = \mathit{prefix}(t,l)$ with $k \leq l$. The sequence $S(P) = s_1|_{l_1} s_2|_{l_2} \ldots s_{m-1}|_{l_{m-1}} s_m$ is called a k-*consensus* in $\mathcal{F}$. The multiple alignment between the sequences in P represents a k-*contig* in $\mathcal{F}$.

The *mininum k-contig problem* (MkCP) is, given a set $\mathcal{F}$ of sequences, to find a minimum set C of disjoint k-paths that cover every terminal sequence in $\mathcal{F}$. The MkCP does not take account of orientation problem. So, an extension of MkCP, denoted by MkCP^r, is needed. In this case, the vertex set of the overlap graph $G(\mathcal{F})$ will be twice as large as $\mathcal{F}$. It will contain the sequences of $\mathcal{F}$ in the two possible orientations. Our problem, now, is the same for MkCP, but the k-paths must now cover the terminal sequences in exactly one orientation, not in both.

Finding optimal covers in a directed graph is NP-hard [5]. However, finding optimal covers by paths *and* cycles is easy, as stated in the following theorem. We basically have to find maximum matching on a related graph defined as follows.

Let $G = (V, E)$ be a directed graph. The *double* of G, is an undirected, bipartite graph $G' = (V_1, V_2, A)$ such that V_1 and V_2 are copies of V and for each directed edge $(u, v) \in E$ we have an edge $(u_1, v_2) \in A$, with $u_1 \in V_1$ and $v_2 \in V_2$. No other edges exist in G'. The graphs G and G' have therefore the same number of edges.

Theorem 1 *Let G be a directed graph and G' its double. Every matching M in G' corresponds to a set C of disjoint paths and cycles that cover all the vertices in G with*

$$|M| + |C| = |V|. \tag{1}$$

Conversely, every such set C corresponds to a matching M in G' satisfying (1).

Two special cases are worth mentioning here. First, when G is acyclic the covering problem becomes polynomial, because it is equivalent to finding a maximum matching. Second, if G has a Hamiltonian path, which is the case when we have sufficient coverage to encompass the entire DNA molecule, a maximum matching in G' will correspond to a solution with at most one path. We can then try and "cross-over" the path with each cycle in turn to end up with a cycle-free solution, that is, a single contig.

3 Our Implementation

In our fragment assembler named `concam`, we have tried to build a really useful toolkit, taking care of all difficulties shown on Section 2 and producing good results within acceptable running time limits. The `concam` assembler is a set of seven `Perl` programs, three C++ programs, and makes use of three other well known programs from the `phrap` package: `phred`, `phd2fasta` and `cross_match` [6,7,8,1]. In the sequel, we give a summarized description of our work.

The first step is to process the chromatograms, which are the output of sequencing machines. From the chromatograms the sequence of bases is extracted together with a quality value for each letter. The quality value is an integer between 0 and 100 and is related to the probability p that the letter is wrong by $p = 10^{-q/10}$[6,7]. To extract the letters and their quality values, `concam` uses the programs `phred` and `phd2fasta`. After that, program `cross_match` is invoked to indentify the stretches on both ends of fragments that came from vector DNA. Each letter belonging to those stretches will be replaced by character `X`. At the end of these steps, there are two text files in `FASTA` format, one with sequences of fragments and other with quality values for sequences. These are the initial inputs for our set of programs.

3.1 Preprocessing

Fragment preprocessing is necessary before we start building the graph. First, we exclude the ends belonging to vectors, so it will be easier to compare fragment sequences later. Then, we identify on each fragment the interval of good quality. As we said before, the ends of fragments have in general very poor quality, so we want to mark the initial and final positions in the fragments that correspond to good quality regions. This will be useful in the graph building phase. To help find those regions we use two parameters: a *minimum average quality* q and a *window size* w. The sequences are scanned starting first on left side. We calculate the average quality of each window (substring) of size w. When this average is greater or equal than the minimum q we stop and start the same process on the right size.

During our search, in some cases there are fragments having poor quality in their whole extension. Those kinds of fragments will be discarded from the assembly since they are unreliable. Another case of exclusion is when the interval of good quality of a fragment is shorter than a *minimum length* l. The default values for q, w and l is 20, 10 and 100.

3.2 Overlap Graph and Matchings

To build the graph, the fragments are compared pairwise. `Cross_match` is used to perform the comparisons. It returns the alignments of the sequences. The alignments produced by `cross_match` are checked to see how the fragments relate to each other. In pratice, we rarely will have a fragment s covering all a fragment t or even a suffix of a fragment s matching exatly a prefix of a fragment t. In first case, we should establish a *minimum coverage* c instead of demanding 100% of coverage. Besides, when we encounter a situation that the *minimum coverage* is not satisfacted, we can check if end regions of sequence being covered, that are not in alignment, are in or out of the sequence interval of good quality. If it is out we can just ignore the size of those regions with respect to the whole size of the sequence and recalculate the percentage of coverage. It will not permite that the bad quality of the ends disturbs the comparisons.

We have the same situation concerning the matching between prefix and suffix. When a pair s and t of sequences being compared seams to have no suffix-prefix matching, we can check if the right end region of s and the left end region of t, that are not present in alignment, are out of respectives intervals of good quality of the sequences. If so, we ignore these regions and get a suffix-prefix alignment, adding an arc for our graph. If not, we can still check if the regions length is less or equal than a threshold *tol* called *tolerance* multiplied by the alignment size. The idea is to be unconcerned with end regions out of alignment, if their length are very small if compared to alignment size. Tipical values for c and *tol* would be 80% and 0.05, but in a latter section, when we talk about iteractivity, we will see that good initial values would be 99% and 0.01. In each following iteraction these requirements are relaxed.

The next steps are removing Steiner vertices from the graph, building its double and finding any maximal matching in the graph. We remove the Steiner vertices following the idea that if our solution covers the terminals vertices it will also cover the Steiner vertices. Besides, our graph will be much smaller, taking less computing effort. We compute macthings using an implementation by Cherkassky and colleagues [9].

3.3 Recombination

At this point the program has found some paths and cycles. The program uses a heuristic to combine paths and cycles in a greedy way. This means that when `concam` finds a favorable path to combine to a given cycle it will perform the recombination without looking for another possible path. The recombination is simply breaking the cycle and joining it to the path through arcs that were not in the original solution. If edges (a, b) and (c, d) belong to the path and the cycle, respectively, and edges (a, d) and (c, b) exist in the graph, we replace the first pair by the second. Figure 1 shows an example where a graph has two possible solutions, one with a cycle and the other without, and Figure 2 shows the result of recombination. If, after verifying all possible paths, we realize that recombination of a given cycle is not possible, then `concam` simply breaks the cycle in some random edge, because the final solution cannot have cycles. It is an extreme situation but in our tests we had rare cases in which

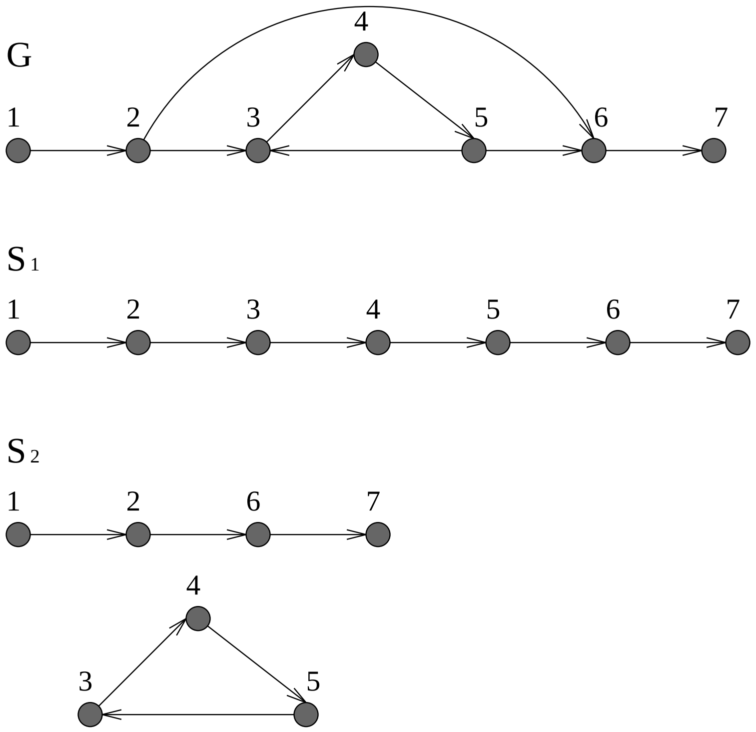

Figure 1: Graph G with with two possible solutions: S_1 and S_2.

recombination could not be done.

3.4 Alignment and Consensus

As described in Section 2, the overlap graph is formed by fragments and their reverse complement. So far we have found paths in the graph covering sequences in both orientations. But we want a solution that covers each sequence in one orientation only. Our assembler uses another heuristic to remove complementary vertices from the current solution. If a vertex s and its complement $\overline{s}$ are both in the solution, but in different paths, `concam` removes the one belonging to the smallest path and also removes the edges adjacent to it. If s and $\overline{s}$ are in the same path, the program removes the vertex closer to an end of the path. This heuristic tries to keep the paths as long as possible.

The next step performed by `concam` is to transform the final paths into multiple alignments corresponding to contigs. To do this, the program scans each path beginning with the starting vertex and joins the parwise alignments produced by `cross_match` in the order given by the path. To join the alignments `concam` makes use of base graphs [10], which represent an excelent data

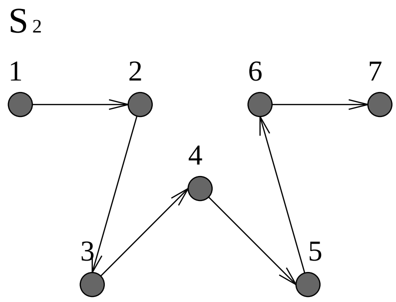

Figure 2: Recombination of a cycle and a path in solution S_2 from the previous figure.

structure for this purpose.

The last step is to derive the consensus for the multiple alignments obtained earlier. The qualities of each base are very important here. Our idea was inspired in `consed`[11,1]. Given a multiple alignment, for a certain column i, let a be a base having quality q_1 and belonging to a fragment A. For the same column i, let b be a base of the same kind of a, having quality q_2 and beloging to a fragment B in opposite orientation with respect to A, such that no other base of i of the same kind and beloging to another fragment in the same orientation as B, has quality greater than q_2. The result of $q_1 + q_2$ is what we call *modified quality* of a. Each column will be represented in the consensus by the base with highest *modified quality*.

When we have spaces in the alignment, their quality will be the average between qualities of their adjacent bases (rounded up to the nearest integer.) Figure 3 shows a simple example. The resulting consensus is written to an `ace` file[11,1], so that they can be visualized by the `consed` graphical editor.

3.5 Iteractivity

We came to realize in our experiments that one of the most difficult part of assembly is to build good overlap graphs, given the pitfalls mentioned in Section 2. No matter what technique is used to find paths, if our graph does not represent the real relationship between fragments, the solution will be not of interest.

In spite of the fact that the parameters *minimum average quality*, *window size*, *tolerance* and *minimum coverage* help us reach better graphs, they were not sufficient to bring us to only one contig in our solution. When we have low *tolerance* and high *minimum coverage* some arcs will be wrongly discarded and some vertices will be wrongly considered as terminals. If we relax those values, having high *tolerance* and low *minimum coverage*, bad arcs and too

```
               A  G  -  C  C  T  G  T  C  A  A  G  (u)
               6  8  11 15 18 25 23 25 22 17 12 9
      C  T  T  A  G  T  C  C  -  -  T  G  A  (u)
      9  8  10 15 16 19 25 22 14 14 7  8  5
                                 G  -  C  A  T  G  G  (c)
                                 12 13 15 23 18 12 15
                     T  C  C  T  G  T  C  C  T  G  (c)
                     8  5  10 15 15 19 24 18 14 8
                  G  T  T  T  T  G  T  G  C  T  G  G  A  T  (u)
                  7  5  10 7  12 18 26 23 28 20 15 10 11 6
G  G  C  T  T  A  A  -  T  T  T  A  T  G  C  (u)
10 8  9  15 17 20 18 22 27 23 19 11 15 10 7
--------------------------------------------------------------
G  G  C  T  T  A  A  T  C  C  T  G  T  C  C  T  G  G  A  T
10 8  9  15 17 20 18 27 30 32 40 38 45 46 46 38 27 25 11 6
```

Figure 3: Modified qualities. The (c) and (u) at the end of sequences mean "complemented" and "uncomplemented," respectively

many Steiner vertices will appear. Even after tuning these values we had too many contigs in our solution.

Those experiments resulted in our decision of introducing interactivity into the assembler. In the first iteraction, `concam` is very restrictive. *Minimum coverage* is set very close to 100% and *tolerance* very close to 0%. This will result in many contigs in this iteraction, but they are very strongly connected and reliable. Then we use these contigs to feed the next iteration, relaxing the parameters a little, that is, *minimum coverage* will decrease while *tolerance* will increase. It will allow some contigs to join and others (flagged as Steiner vertices in the current iteraction) to be eliminated. We proceed in this way until we reach a maximum allowed number of iteractions *max* or until we obtain only one contig.

To a given iteraction $i > 1$, we calculate *tolerance* tol_i and *minimum coverage* min_i, by $tol_i = tol_{i-1} inc$ and $min_i = 100 - (100 - min_{i-1} dec)$, where inc and dec are increasing and decreasing factors. The default values for dec, inc and max are 3, 3 and 10 respectively. In our implementation we let *tolerance* grow without restrictions and let *minimum coverage* decrease until it reaches 0. We believe the most difficult parameters to tune are tol_1, min_1, inc and dec. Our strategy performed very well, as we can see in results shown in the next section.

Table 1: Sets used for tests.

Name	# Chromat.	Length(bp)	Repeats
00I03	17	1750	no
07A01	632	40000	no
07C03	1067	40000	yes
11A02	1037	40000	yes

4 Results

In this section we present some results with four selected sets of fragments from regions of the genome of *Xylella fastidiosa*[12]. The sets are `00I03`, `07A01`, `07C03` and `11A02`. For tests the shown here the minimum accepted length for overlap was 30, the *window size* was 10, the *minimum average quality* was 20, the *minimum fragment length* was 100, *increasing* and decreasing factors were 3, *tolerance* started at 0.01 and *minimum coverage* started at 99%. The tests were performed under an `UltraSparc` running `SunOS` 5.6 with two $300MHz$ processors and $1GB$ of RAM. Table 1 shows fragment set information. Note that sets `07C03` and `11A02` have repeats.

The results are shown in Table 2. There are four blocks, one for each fragment set. Each line has information about each iteraction except the first and last line in each block, which contain the number of removed fragments in preprocessing and the total running time, respectively. Notice that many fragments are removed even before the first iteration.

5 Conclusions

In this work we provide a reliable high-level algorithm for large-scale DNA sequencing. The results we achieved were based on extensive experimentation with real data from the *Xylella fastidiosa* genome project.

We summarize here the main lessons learned with our experiments. An important component of this work is the use of qualities. In the preprocessing phase it helps eliminate many bad fragments from our set. The determination of Steiner vertices, arcs, and consensus bases all profited from the use of quality values.

Another issue was Steiner vertex elimination. In the beginning we were not certain it would be a good idea. But our tests indicate it clearly helps: the overlap graph size drops, decreasing the total running time, and we were still able to reach the right solution.

Table 2: Test results. Column headings are: **It.** - Iteraction; v - Number of vertices of overlap graph G; a - Number of arcs of G; t - Number of terminal vertices of G; s - Number of Steiner vertices of G; V - Number of vertices of G after Steiner vertex removal; A - Number of arcs of G after Steiner vertex removal; cy - Number of cycles in the graph without Steiner vertices; **Rec.** - Indicates whether cycles were recombined or not; C - Number of contigs obtained; $T(s)$ - Running time, in seconds.

Set 00I03: total number of fragments: 17
Number of fragments eliminated in preprocess: 0

It.	v	a	t	s	V	A	cy	Rec.	C	T(s)
1	34	124	14	20	14	36	0	-	1	3.74
Total running time										3.74

Set 07A01: total number of fragments: 632
Number of fragments eliminated in preprocess: 53

It.	v	a	t	s	V	A	cy	Rec.	C	T(s)
1	1158	7690	556	602	556	1980	0	-	57	179.99
2	114	128	24	90	24	22	0	-	5	35.30
3	10	2	4	6	4	0	0	-	2	15.87
4	4	0	2	2	2	0	0	-	1	15.12
Total running time:										246.28

Set 07C03: total number of fragments: 1067
Number of fragments eliminated in preprocess: 143

It.	v	a	t	s	V	A	cy	Rec.	C	T(s)
1	1848	19802	860	988	860	4118	0	-	125	318.40
2	250	382	54	196	54	42	0	-	15	45.06
3	30	0	8	22	8	0	0	-	4	21.88
4	8	4	8	0	8	4	0	-	2	25.47
5	4	0	2	2	2	0	0	-	1	19.20
Total running time:										430.01

Set 11A02: total number of fragments: 1037
Number of fragments eliminated in preprocess: 184

It.	v	a	t	s	V	A	cy	Rec.	C	T(s)
1	1706	18318	726	980	726	3672	1	yes	83	310.97
2	166	244	32	134	32	20	0	-	8	39.62
3	16	0	10	6	10	0	0	-	4	18.39
4	10	0	6	4	6	0	0	-	3	17.14
5	6	2	4	2	4	2	0	-	1	21.65
Total running time:										407.77

We must emphasize the usage of matchings for finding paths in overlap graph. The tests convinced us that the matching technique is quite efficient, providing solutions with a small number of cycles, probably due to particular features of overlap graphs for real problems.

Using matchings with our heuristics for elimination of complementary vertices and recombination of cycles into paths represents an improvement upon naive implementations based directly on the *minimum k-contig* formalization. Instead of avoiding cycles in the solution and restricting this solution to cover the sequences in one of their orientations, our approach relaxes the problem, eliminating these costly constraints, and compensates by using the heuristics. Every component is polynomial if the number of resulting cycles is polynomial. However, we tried this approach only on sets of fragments without repeats or with simple repeats (both `07C03` and `11A02` have one repeated region present in two copies). The next step is to perform tests with more complex data sets.

Acknowledgments

The authors acknowledge the financial support from Brazilian agencies FAPESP and CNPq.

References

1. The Phred/Phrap/Consed System Home Page. http://www.phrap.org/.
2. H. Peltola, H. Soderlund, J. Tarhio, and E. Ukkonen. Algorithms for some string matching problems arising in molecular genetics. *In Information Processing 83: Proceedings of the International Federation for Information Processing (IFIP) Ninth World Computer Congress*, pages 53–64, 1983.
3. J. D. Kececioglu and E. W. Myers. Combinatorial algorithms for DNA sequence assembly. *Algorithmica*, pages 13:7–51, 1995.
4. Joao Setubal and Joao Meidanis. *Introduction to Computational Molecular Biology.* PWS, 1997.
5. Carlos E. Ferreira, Cid C. de Souza, and Yoshiko Wakabayashi. Rearrangement of DNA fragments: a branch-and-cut algorithm. Technical report, IME-USP e IC-UNICAMP, 1997.
6. Peter Richterich. Estimation of errors in “Raw” DNA sequences: A validation study. *Genome Research*, pages 8:251–259, 1998.
7. B. Ewing and P. Green. Base-calling of automated sequencer traces using phred. II. error probabilities. *Genome Research*, pages 8:186–194, 1998.

8. B. Ewing and P. Green. Base-calling of automated sequencer traces using phred. I. accuracy assessment. *Genome Research*, pages 8:175–185, 1998.
9. B.V. Cherkassky, A.V. Goldberg, P. Martin, J. C. Setubal, and J. Stolfi. Augment or push? a computational study of bipartite matching and unit capacity flow algorithms. *Proceedings of the Workshop on Algorithm Engineering*, pages 1–10, 1997.
10. Augusto A. M. de Almeida, João Meidanis, and Alexandre Moriya. Um sistema para auxílio na montagem de fragmentos de DNA. *XXI Seminário Integrado de Software e Hardware*, pages 533–545, 1994.
11. David Gordon, Chris Abajian, and Phil Green. Consed: A graphical tool for sequence finishing. *Genome Research*, pages 8:195–202, 1998.
12. *Xylella fastidiosa* Genome Project. http://www.lbi.ic.unicamp.br/xf/.