Parallel DNA Sequence Assembly

Jacek Błażewicz[†,‡] Marek Figlerowicz[‡] Przemysław Jackowiak[†] Dariusz Janny[†]
Dariusz Jarczyński[†] Marta Kasprzak[†,‡] Maciej Nalewaj[†] Bartosz Nowierski[†]
Rafał Styszyński[†] Łukasz Szajkowski[†]
Paweł Widera[†]

[†] Institute of Computing Science, Poznań University of Technology
Piotrowo 3A, 60-965 Poznań, Poland
[‡] Institute of Bioorganic Chemistry, Polish Academy of Sciences
Noskowskiego 12, 61-704 Poznań, Poland
blazewic@put.poznan.pl

Abstract

In the paper, a heuristic algorithm for the DNA sequence assembly problem is presented. Its sequential implementation is described as well as the way of its parallelization. Computational experiment shows how the parallel algorithm speed depends on a number of processes. Tests on real data coming from experiments with SARS coronavirus are also discussed, where the outcome of our algorithm has appeared to be biologically correct.

Key words: *DNA sequence assembly, pairwise alignment, Hamiltonian path.*

1. Introduction

The process of reading genomic sequences can be divided into three stages depending on the length of the analyzed sequence: *sequencing* — determining a sequence of nucleotides in a DNA fragment of a length of a few hundred nucleotides, *assembly* — combining the sequenced fragments into longer contigs, often being whole genes, and *mapping* — placing the assembled contigs in proper chromosome regions (see e.g. [2]). This process is fundamental and takes an important position in molecular biology. Algorithms supporting it are applied in medicine (e.g. in recognition of genetic diseases or in development of gene therapy), biology (understanding gene functions, phylogenetic analysis), or agriculture (adjustment of species). Despite the fact that there are several assembly algorithms available, there is still a need for a development of the new ones with better performance characteristic, allowing for a better usage of parallel computing systems [4].

The problem of *DNA sequence assembly* can be defined formally in the following way. On the input we have a multiset of sequences over alphabet {A, C, G, T}. These letters stand for four nucleotides composing DNA chains. The input sequences generally have different lengths (from a few hundred to a few thousand nucleotides) and there may exist an inclusion relation between them. The solution is a sequence containing all the input sequences as substrings, where the criterion of an evaluation of the solution can be its length (minimized during computations) or its likelihood (maximized) [11, 9, 7, 6].

The input sequences are outcomes of the DNA sequencing process. The goal of the assembly is to compose them in one sequence in a proper order. Unfortunately, the sequencing outcomes usually contain misreadings (insertions, deletions, and substitutions of nucleotides) coming from biochemical steps as well as from a weakness of a sequencing program. Thus, inexact matches of sequences have to be allowed. Of course, to disable accidental overlaps, some limit of mismatch acceptance must be defined. The input data can come from one or from both strands of an assembled fragment of the DNA helix. In the latter case, a part of the input sequences have the opposite orientation than the others and they should be matched obeying the rule of complementarity. The complementarity means, that in one strand A stands against T in the other strand and C stands against G. For example, the reverse complement for AACATG is CATGTT. Example 1 shows an illustration of the sequence assembly with sequences on input containing errors and coming from both DNA strands.

Proceedings of the Fifth Mexican International Conference in Computer Science (ENC'04)
0-7695-2160-6/04 $20.00 © 2004 **IEEE**

Example 1 Let a set of the input sequences for the DNA assembly process be {AGCA, ATCAAGCAAC, GACTC, TAGAA, TTTGCC}. Because we assume that the sequences may contain misreadings, we must allow for inexact matchings. And because they may come from both DNA strands, appearance of some reverse complements on output is permitted. One of possible results is shown in Figure 1. In printing the resulting sequence (the one above the line) the majority rule has been used (the character appearing the greatest number of times in a column is the winner). □

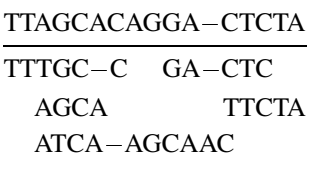

Figure 1. A possible assembly of sequences from Example 1.

The DNA sequence assembly problem is strongly NP-hard, even in the case of data without errors and derived from one DNA strand (compare with Shortest Common Superstring [3]).

The aim of this paper is to present a heuristic algorithm for the DNA sequence assembly. In the algorithm a graph is constructed on the base of the input data, and a weighted Hamiltonian path is looked for. Parallel and sequential implementations are described. Their performance is checked in a computational experiment on instances derived from the genome of *Drosophila melanogaster*. Tests on real data coming from biological experiments with SARS coronavirus are also discussed, where the outcome of our algorithm has appeared to be biologically correct.

2. Heuristic for DNA sequence assembly

The problem solved by the algorithm presented in this section can be formulated as follows.

Problem 1 *DNA sequence assembly with errors — search version*
Instance: Multiset S of sequences over alphabet {A, C, G, T} coming from both strands of a DNA fragment.
Answer: A sequence of a maximum likelihood value containing as substrings all elements of S, taken straight or as reverse complements, with some limit of mismatches allowed.

Usually the assembled sequences are composed of the letters {A, C, G, T}, but they can be written as well as strings of aminoacids. The algorithm accepts S over any alphabet. The *likelihood* is defined as the sum of weights of these arcs in a graph constructed on the basis of the input data, which compose a path representing the solution (cf. [5]). The algorithm can have two optional parameters: the minimum overlap, i.e. the minimal number of characters on which neighboring input sequences must overlap in a solution, and the error bound, being the limit on the percentage of mismatches allowed in overlaps of two neighboring sequences. The heuristic can return a solution in more than one part, and to get one output sequence the parts may be concatenated. In practical applications the concatenation based on expert knowledge is used.

Because S contains sequences coming from both DNA strands, some of them should be replaced by their complementary counterparts. Unfortunately, the assignment of sequences to strands is not known. In the heuristic we simply add to S reverse complements created for all initial sequences from S. During the computations, if a sequence is added to the solution, its reverse complement is also marked as "used".

The method starts with building a multigraph with vertices corresponding to sequences from S not contained in others. Arcs between a pair of vertices correspond to possible overlaps of the sequences observing the assumed error bound. In order to keep reasonable connections of sequences only, the error bound should be set to a relatively small value. The acceptable overlaps are computed by a simplified dynamic programming method based on the standard algorithm for the pairwise semiglobal sequence alignment from [11]. (The semiglobal alignment compares in a best way a suffix of the first sequence with a prefix of the second one.) However, the computations for all possible pairs of sequences would consume huge amount of time, so we choose a subset of these pairs by a heuristic based on a hash function.

Using the hash function we compute characteristic numbers for all short substrings of all elements of S. Two sequences having a great part of their characteristic numbers common, with a high probability are neighbors in the original sequence. Vertices corresponding to these sequences should be joined by an arc, and the semiglobal alignment for the sequences is determined. The simplified version of the alignment algorithm does not fill the whole dynamic programming table, but only the entries around some diagonal (or diagonals). The appropriate diagonals gather significiant number of entries, which correspond to substrings of the two aligned sequences with equal characteristic numbers. (Similar approach to selecting good diagonals was realized in [12].)

Initially all arcs have weights equal to 1. A great part of the arc set in the graph represents redundant information and can be removed. In the reduction stage there are

removed all the arcs (i, j) corresponding to a shift d_{ij}, for which such a vertex k can be found, that there exist two arcs (i, k) and (k, j) with shifts d_{ik} and d_{kj}, respectively, summing up to d_{ij}. Every removed arc (i, j) adds its weight to the weights of the two arcs (i, k) and (k, j), in order to increase their credibility. Of course, the arcs are removed in a descending way (an arc with the greatest shift first).

Next, the Hamiltonian path of the greatest sum of weights of its arcs is looked for in the graph, but because of several errors and incompleteness of the input, it may be found in more than one part. As the first element of the path the one is chosen, which has the worst connection as a successor with other elements. To find it, the arc of maximum weight from among the arcs entering a vertex is chosen. Then the arc of the smallest weight within the set of the best entering arcs in the whole graph is chosen. If more than one subfragment have such the worst connection, the one having the greatest error (in percents) of this connection is preferred.

Every next vertex of the constructed path must have the greatest value of a function f among all not visited yet candidates. This function is defined in the following way:

$$f = \frac{w}{lim1} + \frac{w}{lim2}$$

where w is the maximum among weights of the arcs from the last element of the current path to a candidate vertex; $lim1$ means the greatest weight for the set of arcs leaving the last vertex and $lim2$ means the greatest weight for the set of arcs entering the candidate vertex. Such value of w, which is twice normalized, gives a good trade-off for an analyzed connection, from the side of the last vertex of the current path as well as from the side of its potential successor. If more than one candidate have a maximal value of f, the one is chosen, which has the smallest error (in percents) of the connection with the last vertex of the current path (concerning the same arc for which the maximal value of f was calculated). If still both these criteria are satisfied for more than one vertex, the winner is chosen in a similar way as for the selection of the first element of the path.

If there is no arc from the last vertex of the current path to a not visited one, a next disjoint part of the solution is constructed and its first vertex is chosen in a similar way as at the beginning of the algorithm, but taking into account only the connections between elements not yet used in the current solution. When all subfragments are present in the solution, and it consists of more than one part, the procedure of reordering them is called. Of course, it is not possible to connect one part with the other constructed later, but the arc may exist which joins the last vertex of a part with the first vertex of another part constructed earlier. The procedure searches for such connections and chooses the ones minimizing the length of the solution.

The resulting sequence is printed on the output by a complex procedure. It joins a set of pairwise alignments of neighboring sequences into some multiple alignment using the majority rule: this character is chosen which appears the greatest number of times at the considered position of the alignment.

3. Parallel version

The sequential algorithm from Section 2 has been implemented as a parallel version using one master and a number of slave processes (cf. [1]). The master part is responsible, among others, for reading data and program parameters, for printing a solution, and obviously for distributing tasks among slave processes. The distribution takes place in several procedures of the algorithm:

- Filling the hash table. Every slave determines characteristic numbers for some subset of sequences.
- Computing acceptable overlaps. Every slave determines successors for some subset of sequences.
- Removing redundant information from the multigraph. Every slave determines arcs to delete that are outgoing from some subset of vertices.
- Choosing a next element of the Hamiltonian path. Every slave determines the best candidate among some subset of vertices. However, if the number of successors for the current vertex is less than the assumed bound, all these computations are done by the master process.

The size of tasks assigned to slaves was chosen in a series of preliminary tests. Data packs of various sizes were sent to slaves and the size resulting in the greatest efficiency was selected. The cardinality of the subsets of sequences/vertices assigned each time to a slave has been set to 5 in the computational experiment.

4. Computational experiment

The tests were done on the server SUN Fire 6800 with processors UltraSparc III 900 MHz and 20 GB RAM, placed in Poznań Supercomputing and Networking Center [8]. The algorithm was implemented in C with the use of the MPI library. The instances used in tests with results shown in Figure 2 and in Table 1 were derived from a string of nucleotides of chromosome arm2R from *Drosophila melanogaster*, published by Celera Inc. In the first part of the experiment (Figure 2) both program parameters, i.e. the minimum overlap and the error bound, were set to 10.

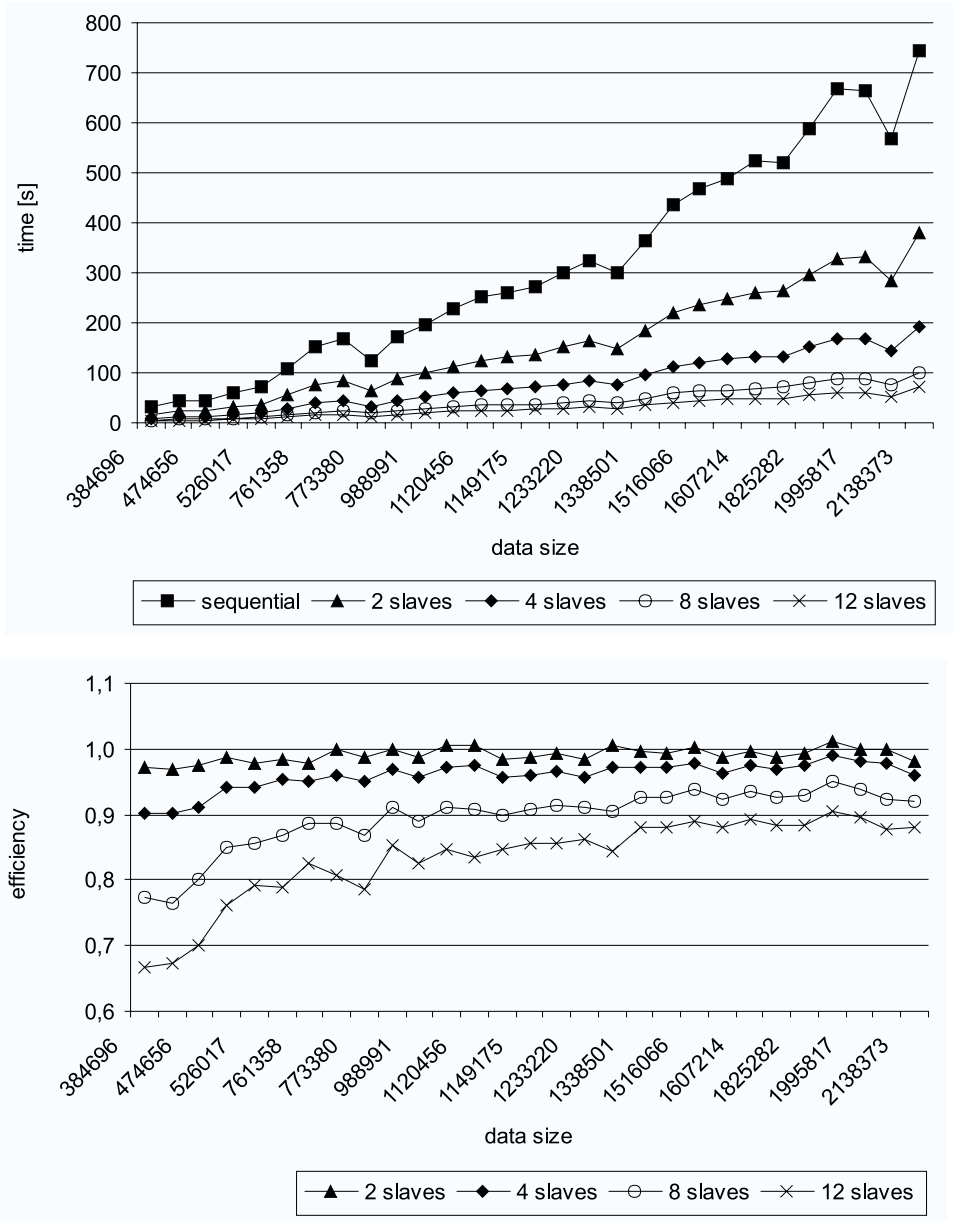

Figure 2. Computation time of sequential and parallel versions of the algorithm (the upper graph) and efficiency of the parallel computations, being the ratio of the computation time of the sequential version to the time of the parallel one, the latter multiplied by the number of slaves (the lower graph).

		A	B	A∪B
No. of instances		83	18	101
Coverage [%]	1^{st}	100.0	73.7	–
	2^{nd}	–	25.8	–
	$1^{st} + 2^{nd}$	100.0	97.0	99.5
Similarity [%]		97.9	97.5	97.8
Data size [b]		91126	182061	107332

Table 1. Results of the comparison of sequences generated by the algorithm and the original ones.

The results presented in Figure 2 show, that the parallel version is highly scalable. Addition of new processors allows for solving larger and larger instances of the assembly problem.

The second part of the computational experiment was devoted to check the quality of sequences generated by the algorithm. Here, the error bound was set to 8 and the instances were derived from 6, 8, or 10 copies of original sequences. Rows "A", "B", and "A∪B" in Table 1 concern, respectively, the instances for which a sequence as long as the original one was generated, the remaining instances, and all the instances used. The table contains average values calculated for these sets of instances, with the cardinalities specified in the first column. The coverage shows how bis is a part of the original sequence being covered by the generated components after aligning them. Only two longest generated components have been analyzed here in order to simplify the presentation, they are denoted as "1^{st}" and "2^{nd}". The similarity shows the quality of this alignment.

Finally, the algorithm was used to assembly the genome of SARS coronavirus. As the input real shotgun data were used, coming from TOR2 sequence and downloaded from Michael Smith Genome Science Centre (Canada) [10]. The algorithm found a few contigs covering 98% of the entire genome. After manual reordering based on expert knowledge we obtained a sequence almost the same (of the same quality) as the one published by the Centre.

References

[1] Błażewicz J. (ed.), Algorytmy asemblacji łańcuchów DNA, report RA-003/2002, Poznań Supercomputing and Networking Center, Poznań 2002.

[2] Błażewicz J., Formanowicz P., Kasprzak M., Selected combinatorial problems of computational biology, *European Journal of Operational Research*, to appear.

[3] Garey M.R., Johnson D.S., *Computers and Intractability. A Guide to the Theory of NP-Completeness*, San Francisco, W.H. Freeman and Company 1979.

[4] Human Genome Program, U.S. Department of Energy, Genomics and its Impact on Medicine and Society: A 2001 Primer.

[5] Kasprzak M., ASSEMBL — opis programu i wyniki obliczeń na procesorze Intel Pentium 4, report RA-002/2002, Poznań Supercomputing and Networking Center, Poznań 2002.

[6] Myers E.W., Weber J.L., Is whole human genome sequencing feasible?, **in:** *Computational Methods in Genome Research*, ed. S. Suhai, New York, Plenum Press 1996, 73–89.

[7] Pevzner P.A., *Computational Molecular Biology: an Algorithmic Approach*, Cambridge, MIT Press 2000.

[8] PROGRESS Grid project, http://progress.psnc.pl/.

[9] Setubal J., Meidanis J., *Introduction to Computational Molecular Biology*, Boston, PWS Publishing Company 1997.

[10] TOR2 sequence of SARS coronavirus, http://www.bcgsc.ca/bioinfo/SARS/.

[11] Waterman M.S., *Introduction to Computational Biology. Maps, Sequences and Genomes*, London, Chapman & Hall 1995.

[12] Wilbur W.J., Lipman D.J., Rapid similarity searches of nucleic acid and protein data banks, *Proceedings of the National Academy of Sciences of the USA* 80, 1983, 726–730.

0-7695-2160-6/04 $20.00 © 2004 **IEEE**