Acta Biochimica Polonica

Vol. 51 No. 4/2004

983–993

QUARTERLY

Assembling the SARS-CoV genome — new method based on graph theoretical approach

Jacek Błażewicz[1,2], Marek Figlerowicz[2], Piotr Formanowicz[1,2]✉, Marta Kasprzak[1,2], Bartosz Nowierski[1], Rafał Styszyński[1], Łukasz Szajkowski[1], Paweł Widera[1] and Mariusz Wiktorczyk[1]

[1]*Institute of Computing Science, Poznań University of Technology, Poznań, Poland;* [2]*Institute of Bioorganic Chemistry, Polish Academy of Sciences, Poznań, Poland*

Received: 26 April, 2004; revised: 14 September, 2004; accepted: 05 October, 2004

Key words: SARS-CoV, assembling, graphs, algorithm

Nowadays, scientists may learn a lot about the organisms studied just by analyzing their genetic material. This requires the development of methods of reading genomes with high accuracy. It has become clear that the knowledge of the changes occuring within a viral genome is indispensable for effective fighting of the pathogen. A good example is SARS-CoV, which was a cause of death of many people and frightened the entire world with its fast and hard to prevent propagation. Rapid development of sequencing methods, like shotgun sequencing or sequencing by hybridization (SBH), gives scientists a good tool for reading genomes. However, since sequencing methods can read fragments of up to 1000 bp only, methods for sequence assembling are required in order to read whole genomes. In this paper a new assembling method, based on graph theoretical approach, is presented. The method was tested on SARS-CoV and the results were compared to the outcome of other widely known methods.

Infectious diseases are one of the most serious world health problems. According to a recent WHO report, they are the most frequent cause of human death. Moreover, in 2003, 50% of the world population has suffered from bacterial, fungal or viral infection. As we could learn during the last decade epidemics caused by viruses, especially those whose genomes are composed of RNA molecules (RNA viruses and retroviruses), are particularly dangerous (e.g., human immunodeficiency virus — HIV, hepatitis C virus — HCV,

✉Correspondence to: Piotr Formanowicz, Institute of Computing Science, Poznań University of Technology, Piotrowo 3a, 60-965 Poznań, Poland; phone: (48 61) 852 8503 ext. 276; fax: (48 61) 877 1525; e-mail: piotr@cs.put.poznan.pl

Abbreviations: *eb*, error bound; HCV, hepatitis C virus; HIV, human immunodeficiency virus; *mo*, minimum energy; SARS-CoV, severe acute respiratory syndrom corona virus; *ws*, window size.

severe acute respiratory syndrome corona virus – SARS-CoV). There are several lines of evidence suggesting that the major source of our problems with RNA-based viruses is their enormous genetic variability (Figlerowicz *et al.*, 2003). Mechanisms generating the polymorphism of RNA genomes are involved in both the appearance of drug resistant mutants and the emergence of new, often dangerous, viral strains and species. Therefore, it seems that the elaboration of effective methods enabling fast and precise analysis of pathogen genomes can be of great use in fighting viral infections. One may note here that it is a more global trend. The development of new mathematical and computational tools for molecular biology has been recently highlighted by devoting an issue of the *Science* magazine to this topic (Chin *et al.*, 2004).

In fact, it is thought that the basic factor allowing a proper choice of medication during HIV therapy is the continual analysis of the changes occurring in the viral genome (Richman, 2001). Similar conclusions have been reached by scientists investigating the mechanisms responsible for the development of chronic hepatitis C. They found that one can predict the final outcome of anti-HCV therapy by analyzing the genetic changes within the virus population (Farci *et al.*, 2002). Moreover, the easiest way of identifying a new virus is by sequencing its genome.

Recently, during the SARS-CoV epidemic it became apparent how important the methods of sequencing and analysis of viral genomes are for modern medicine. Reading the SARS-CoV genome, discovering that it is a corona virus and describing its proteins took some time (Marra *et al.*, 2003; Rota *et al.*, 2003). In order to speed this process up one needs to develop better and faster DNA assembling procedures. In this paper, such a method, based on graph theoretical approach, is presented. The method has some advantages over the existing algorithms. When tested on the SARS-CoV genome data coming from the second fatal case from Toronto (http://www.bcgsc.ca/bioinfo/SARS/), it has produced SARS-CoV genome sequences slightly differing from the one reported in NCBI, labeled as AY274119.3.

In this paper, the *New DNA assembling method* subsection of Results describes the basic idea and the implementation of the method, while *Computational experiment with the SARS-CoV genome* and Discussion report the results of computational experiments based on the SARS-CoV genome data.

MATERIALS AND METHODS

The presented DNA assembling method is an algorithm which has to be implemented in a programming language to be useful. Our choice was the *ANSI C* language with the *gcc* compiler since it provides a clean and fast code. In order to further increase the speed of the assembling application, a distributed version was created (this version, however, is not described in this paper). Here, particularly useful were the MPI library with its *MPICH-G2* implementation responsible for communication between processes of the application, and *Globus* – a grid environment platform for executing distributed applications. The assembling application was developed and tested on SUN Fire 6800 machines, with UltraSparc III 900 MHz processors on board, localized at the Poznań Supercomputing and Networking Center. (It was developed as part of the *PROGRESS* project and is now publicly available as a computational service at its portal (http://progress.psnc.pl/); the application is also available on the Computational Biology Server at the Institute of Computing Science, Poznań University of Technology (http://bio.cs.put.poznan.pl/)).

The original sequence data on which the application was tested were obtained during the SARS-CoV genome sequencing (TOR2 – the second fatal case in Toronto). The shotgun data is publicly available at the Canada's Michael Smith Genome Science Centre website

(http://www.bcgsc.ca/bioinfo/SARS/). There is also the entire genome, assembled by them, published in NCBI, which was used as a reference to the results given by the presented algorithm.

RESULTS

Problem description

A standard approach to genome reading is divided into three hierarchical stages: sequencing, assembling and mapping (Gusfield, 1997; Pevzner, 2000). Genome mapping is used only for very large genomes, thus, sequencing and assembling are the most standard techniques used nowadays. Sequencing, either using gel electrophoresis (Setubal & Meidanis, 1997) or hybridization approach (Southern, 1988; Bains, 1991; Błażewicz *et al.*, 1999a; 1999b; 2000; 2004a), can read DNA sequences of up to 1000 base pairs. Next, these fragments are put together in the assembling stage. Although a big progress in the latter technique has been observed recently (Myers *et al.*, 2000) and some procedures are publicly available (e.g., http://www.phrap.org/, http://genome.cs.mtu.edu/cap/cap3.html), there is still a need to develop better and faster algorithms that can deal with the quickly changing viral genomes, e.g. HIV, HCV and SARS-CoV. Such a method, based on a graph theoretical approach, was developed here and tested on the SARS-CoV genome data.

The input data for the DNA assembling problem is a set of short sequences of up to 1000 bp obtained by shotgun sequencing. The problem of DNA assembling is solvable only when all regions of the original sequence are covered by a sufficient number of input sequences (coverage of 6–10 times is recommended) whose positions in the original sequence are mostly different. Thus, the input sequences overlap with each other – this information is crucial for the reconstruction of the original sequence. In an ideal case the input sequences are assumed to have no errors introduced by the sequencing phase, hence the overlaps are perfect. Such a problem is illustrated in Fig. 1.

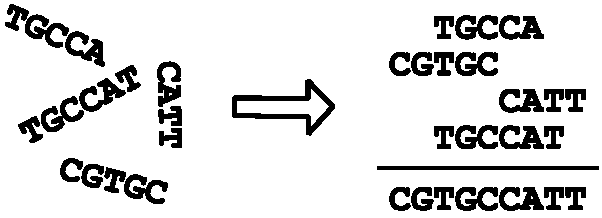

Figure 1. DNA assembling example – an ideal case.

In reality, however, sequencing is not perfect and the sequences obtained are erroneous. Here, we consider inaccuracy errors, which can be of three types:

- insertion – a nucleotide occurs in the obtained sequence, although it is not in fact present in the DNA being analyzed;
- deletion – a nucleotide was omitted while reading;
- substitution – a nucleotide was erroneously identified as another nucleotide.

Additionally, we have to face another problem. During sequencing the sequences are read from clones of the original sequence which may be of either orientation. It is usually impossible to determine which strand the input sequence comes from. Thus, for example, the sequences TGTAATC and GATTACA (reverse complement) are equivalent. It means that given a sequence on input, its reverse complement may be used instead.

New DNA assembling method

The algorithm presented in this paper is driven by two parameters. One of them is *mo* (Minimum Overlap). It denotes a minimal number of nucleotides by which two sequences must overlap for the overlap to be found significant (values not less than 10 are recommended; 15–30 should be enough in most cases). In this way, we limit accidental

overlapping of input sequences, which do not come from the same region of the original sequence. The other parameter is *eb* (Error Bound). This is a maximum acceptable error (ranging from 0 to 1) occurring between the overlapping fragments of sequences. Error is a function of matching and mismatching nucleotides in overlapping fragments, as well as the length of the overlap – it will be described in details later. Since the presented method is a heuristic and is well time-optimized, these parameters are considered only in a weak sense. To be more precise, overlaps shorter than *mo* and with error greater than *eb* are not accepted, however, not all overlaps which satisfy these limits take part in the solution construction. It barely affects the quality of results, but it allows seriously decreasing the time of computations.

The algorithm is composed of four stages. The first stage identifies overlaps between sequences (along with errors concerning these overlaps) and creates an overlap graph, being a directed multigraph. This is the most complicated and time-consuming stage. The second stage reduces the size of the constructed graph by deleting those arcs which are unnecessary and could disturb further computations. It also calculates a reliability score for the remaining arcs, basing on their relation with the deleted ones. The third stage is a heuristic trying to produce a special path in the graph, which represents the solution. The path should be as long as possible, with regard to some constraints, and possibly the most reliable and the least erroneous one. The last stage is the construction of the consensus sequence from the path found in the previous stage. This is only a general idea; the details of the procedure will be presented later in this section.

Before the algorithm is described in detail, a few words about the alignment problem should be said (Waterman, 1995; Setubal & Meidanis, 1997; Pevzner, 2000). The alignment is a way of sequence comparison. Informally speaking, two sequences (or their parts) are placed next to each other and "stuffed" with spaces so that they look as similar as possible. Such an arrangement is evaluated by rewarding matching nucleotides as well as penalizing mismatching ones and spaces. In the presented algorithm, the following values are used: +1 for a match, –1 for a mismatch and –2 for a space. The *global alignment* is considered in the context of the entire sequence comparison. The *semi-global alignment* is a modification of the *global alignment* where spaces at the beginning of the first sequence and at the end of the second sequence are not penalized. Thus, prefixes of the first sequence are compared with suffixes of the second sequence – this is the way the overlaps between sequences are identified. Having the alignment and its evaluation (denoted as *score*), we calculate the *error* (in order to compare it with *eb*) by normalizing the *score*, using the formula:

$$\text{error} = \left(1 - \frac{score + length}{2 \cdot length}\right),$$

where *length* denotes the number of overlapping nucleotides (note that the best possible alignment for any pair of sequences can have the score in the interval [–*length*, *length*]).

In the first stage of the algorithm (construction of the overlap graph), an alignment algorithm being a modification of the Smith-Waterman alignment algorithm (Smith & Waterman, 1981) with addition of appropriate pruning (further in this paper it is referred to as *diagonal-bounded Smith-Waterman alignment algorithm*) and the Lipman-Wilbur fast alignment algorithm (Wilbur & Lipman, 1983) are used. The idea of the proposed alignment algorithm benefits mainly from two observations.

Observation 1: if two sequences overlap with low error, then the path representing the best alignment in the Smith-Waterman matrix goes mainly through some diagonal (or diagonals) and its neighborhood.

Observation 2: overlaps not shorter than $\frac{1}{eb}$ must have a perfectly matching region of length at least $\frac{1}{eb} - 1$ (they are usually much bigger or there are many of them if an overlap is long enough).

Let a region of a sequence of some fixed size be called a window and let *ws* be its size (Window Size). The value of *ws* should be more or less equal to $\frac{1}{eb} - 1$. (We cannot always stick strictly to this formula, as too big values could cause bad results and too small values could cause algorithm to drastically slow down.) The idea is to compare all the windows of all the sequences of size *ws* with each other. If *ws* is large enough this can be done very efficiently using for example a hashing technique (Cormen *et al.,* 1990). Knowing which windows are equal to each other (i.e. match exactly), the step of finding overlaps can be drastically reduced. There is no need to align every pair of sequences (which is very time-costly), only pairs of sequences with a significant number of equal windows need to be aligned, because only those pairs have a chance to overlap. Observation 2 says that, by doing so, overlaps which satisfy *eb* are not missed (unless they are too short, but it turned out not to be a big problem in practice). Additionally, information about equal windows could be used to prune computations of Smith-Waterman alignment algorithm. When windows of two compared sequences match, this match is marked in the appropriate place in the Smith-Waterman matrix. If there are many common windows for the two aligned sequences and the sequences overlap well, then by Observation 1, many of such marks are along some specific diagonal (or diagonals). Now, it is enough to fill in the matrix in a fixed-width neighborhood of the diagonal(s) when running the Smith-Waterman algorithm. This is what we have called diagonal-bounded Smith-Waterman algorithm. It computes the alignment in time proportional to the length of the longer of the aligned sequences. It is much faster than the original Smith-Waterman algorithm, which computes the alignment in time proportional to the product of the lengths of the aligned sequences.

Before the first stage even begins, for every input sequence its reverse complement is created and added to the set of input sequences. It cannot be determined which of the two orientations is the correct one, so the new sequences are treated in the same manner as their originals and almost independently. Only stage three of the algorithm needs information about the correspondence between the sequences and their reverse complements.

Now, creation of an overlap graph may begin. In such a graph each input sequence is represented by a vertex. There is an arc from vertex *v* to vertex *u* when the sequences corresponding to these vertices overlap. Obviously, not every overlap is taken into account, but only the ones with the overlap length not less than *mo* and the error value not greater than *eb* (as mentioned earlier, this method is a heuristic, so some overlaps satisfying these constraints might not be taken into account). Both the overlap length and the error can be obtained after the alignment is done from the Smith-Waterman matrix. Additionally, with every arc, the error value and the shift between overlapping sequences (which can be calculated from the lengths of the sequences and the overlap) are associated. This information is important at later stages. Example 1 illustrates the first stage of the algorithm.

Example 1. Let us consider the following set of sequences received as a result of sequencing: (1) `ACTTAGTC`, (2) `AGTCCATG`, (3) `TTGTCCA` and (4) `CCAAGACT` (for the sake of clarity of the example, adding reverse complements is skipped). In Fig. 2, there are presented all feasible and some examples of infeasible overlaps for *mo* = 3 and *eb* = 0.25 (penalized positions are marked with small letters).

All the feasible overlaps form the graph presented in Fig. 3. This graph is the output from

the first stage and becomes the input for the second stage of the algorithm.

FEASIBLE OVERLAPS		
(1) (2)	ACTTAGTC---- ----AGTCCATG	*overlap* = 4 (*shift* = 4) *error* = 0
(1) (3)	ACTTaGTC-- --TT-GTCCA	*overlap* = 6 (*shift* = 2) *error* = 0.25
(1) (3)	ACTTaGTC-- ---TtGTCCA	*overlap* = 5 (*shift* = 3) *error* = 0.2
(2) (4)	AGTCCAtG--- ---CCAaGACT	*overlap* = 5 (*shift* = 3) *error* = 0.2
(3) (2)	TtGTCCA-- -aGTCCATG	*overlap* = 6 (*shift* = 1) *error* ≈ 0.17
(3) (4)	TTGTCCA----- ----CCAAGACT	*overlap* = 3 (*shift* = 4) *error* = 0
(4) (1)	CCAAGACT----- -----ACTTAGTC	*overlap* = 3 (*shift* = 5) *error* = 0
INFEASIBLE OVERLAPS		
(3) (1)	TTGTCCA------- ------ACTTAGTC	*overlap* = 1 < *mo* *error* = 0
(4) (2)	CCAGAcT----- ----AgTCCATG	*overlap* = 3 *error* ≈ 0.33 > *eb*

Figure 2. Feasible and infeasible overlaps.

In the second stage, arcs which duplicate some information are removed from the graph. An arc from vertex (sequence) v to vertex (sequence) w with shift s (denoted as $v \rightarrow_s w$) is deleted if there exist two other arcs $v \rightarrow_p u$ and $u \rightarrow_q w$ such that $s = p + q$ (note that this is a so called transitive arc). It means that the information about sequences v and w overlapping with shift s could be reconstructed from the information provided by the smaller arcs, thus, it is redundant. Additionally, it is easy to notice that the arc $v \rightarrow_s w$ confirms that arcs $v \rightarrow_p u$ and $u \rightarrow_q w$ are not accidental, thus makes them more reliable. Therefore, a score is assigned to each arc, which is a measure of its reliability. Initially, each arc has the score equal to 1. When the arc $v \rightarrow_s w$ is removed, its score is added to the scores of $v \rightarrow_p u$ and $u \rightarrow_q w$. To avoid disposal of arcs necessary to delete some other arcs (with greater shifts), the arcs for deletion are considered in a decreasing order of shifts. Figure 4 illustrates this stage of the algorithm continuing Example 1.

At the third stage the algorithm searches for the best possible path through the previously constructed and reduced graph. The path cannot include (by definition) any vertex twice, but, in this algorithm, it also cannot include both a sequence and its reverse complement at the same time (remember that sequences correspond to vertices). In the best case a path including half of the vertices is found, thus all input sequences (either in their straight or reverse complementary form) are included in the consensus sequence. In fact, the goal is to construct such a path with the maximum overall reliability score (in case there are many paths of maximum score, the least erroneous one is preferred). This problem can be perceived as a variant of the Traveling Salesman Problem (Gutin & Punnen, 2002). This is a computationally hard problem and that is why a heuristic is required. If construction of one connected path is not possible, the output of this stage is a set of disjoint paths. Hence, the output of the entire algorithm is a set of sequences (instead of one), which hopefully are subsequences of the original sequence.

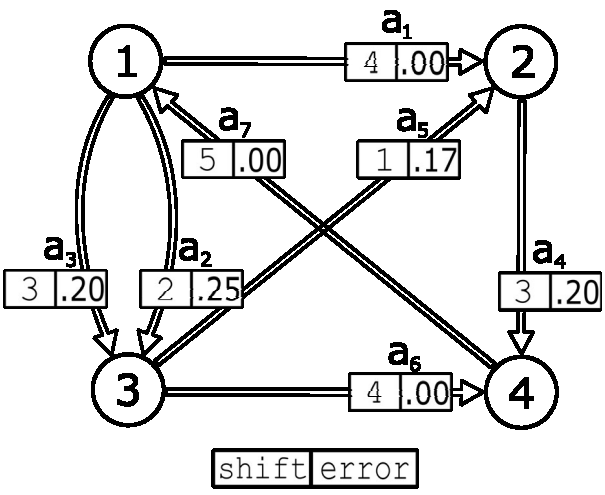

Figure 3. Result of stage 1 — the overlap graph.

As the first element of the constructed path, it is desired to choose such a vertex which has unattractive incoming arcs and thus it is not likely to be in the middle of a good path. Mathematically speaking, for each vertex v an incoming arc *a(v)* with the greatest score (or the smallest error value in case of a tie) is found.

Among all the vertices, the one with the smallest score of *a(v)* (or the greatest error value in case of a tie) is selected to be the first in the constructed path. Once such a vertex is selected, it is not to be used again. The same applies to its reverse complement. Having a part of the path determined, new vertices are appended one by one. In each step the potentially best successor for the last vertex of the path is chosen in the following way. Suppose that the last vertex of the so far constructed path is *v*. For each potential immediate successor *u* a function $f(v,u)=\frac{w(v,u)}{\lim 1(v)}+\frac{w(v,u)}{\lim 2(u)}$ is evaluated and the vertex with the highest value is chosen. *w(v,u)* is the maximum reliability score among arcs from *v* to *u*. The value of *lim1(v)* is the maximum score of an arc beginning in *v* and ending in any other vertex. The value of *lim2(u)* is the maximum score of an arc ending in *u* and beginning in any other vertex. In this way, a specific compromise between the best arc outgoing from the last node in the constructed path and the best arc incoming to a potential successor is reached. In case of equal values of function *f*, a vertex *u* with the smallest error value of the incoming arc (the one selected when computing the value of *f*) is chosen. Again, once a vertex is selected, neither it nor its reverse complement can be used in the future. It is possible that the last vertex of the so far constructed path has no successor, despite the path being not finished yet (it does not have all the vertices). This may happen if there is no coverage or a low one in some region of the original sequence, or when the algorithm gets distracted by poor-quality data. In such a case, construction of the path is left apart. A new disjoint path is started and then constructed in the same way. However, it is possible that several disjoint paths are created because of imperfect selection of the first vertex (which in fact should be somewhere in the middle of a correct path). Therefore, after creating all the disjoint paths we try to reorder them and check whether or not they fit with each other. When the end of a path constructed later matches (by means of alignment) the beginning of a path constructed earlier, they are reordered and merged into one path. Figure 5 illustrates selection of the first element of the constructed path for the data from Example 1. In this particular example, after selecting the first element, selection of next elements is straightforward and no reordering is necessary, thus they are not illustrated here.

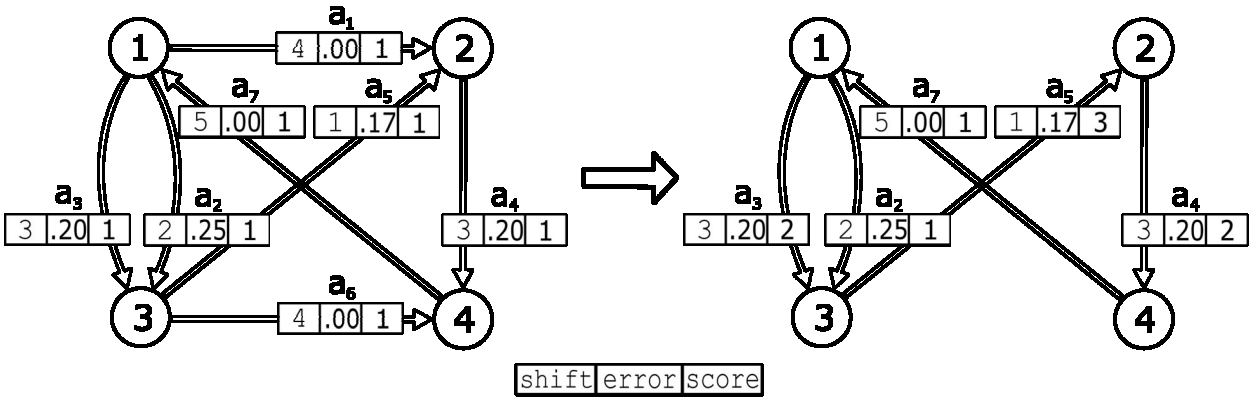

Figure 4. Result of stage 2 — the overlap graph after reduction.

	v	*a*(*v*)	score of *a*(*v*)
selected →	(1)	a_7	1
	(2)	a_5	3
	(3)	a_3	2
	(4)	a_4	2

Figure 5. First part of stage 3 — selection of the first element of the path.

The path found in an overlap graph is an input to the solution construction stage. The solution is a consensus sequence that contains all the sequences (vertices) from the path (or a set of such consensus sequences in case the construction of one connected solution was not possible). To build the consensus sequence, the input sequences are iteratively merged into one big metasequence. Unlike the alignments done in the first stage of the algorithm, this part of the method requires only one additional alignment calculation per sequence. However, since overlaps between sequences are not perfect, there is a need for the metasequence to keep some extra data associated with each position; it is the information on how many symbols confirm each position, where the symbol is either a letter representing a nucleotide or a space. The consensus sequence is then established according to the majority rule – the nucleotide which appears most frequently on a given position is selected to the consensus, or the position is skipped in the case when the space is the most frequent symbol. Merging a sequence with the so far merged metasequence requires calculation of their alignment to determine the best merge layout. The diagonal-bounded Smith-Waterman alignment algorithm is used again. This time, however, to select the best diagonal, the algorithm benefits from the knowledge of the shift between the sequences on the path. Figure 6 contains the layout of the input sequences (derived from the path constructed in stage three) and the consensus sequence (which is the output of the entire algorithm)

(1)	ACTTaGTC———-
(3)	––TtGTCCA—–
(2)	——aGTCCAtG––
(4)	———-CCAaGACT
consensus	ACTTaGTCCAaGACT

Figure 6. Result of the entire algorithm – the consensus sequence.

for the data given in Example 1 (positions without a perfect agreement are marked with smaller letters).

Computational experiment with the SARS-CoV genome

To demonstrate the efficiency and usefulness of the presented method, an application which implements it was created (it is further referred to as ASM). It was applied to analyze the shotgun data – TOR2. The file downloaded from the Canada's Michael Smith Genome Science Centre contains sequences coming from seven clones, referred to as SARS11, SARS12, SARS211, SARS212, SARS213, SARS214 and SARS215. At the beginning, SARS11 and SARS12 were excluded from further consideration, due to small size and high rate of unknown nucleotides (about 45% of N letters in the sequences). The SARS215 set has the best quality of data and thus it was mainly focused on while assembling the SARS-CoV genome. However, the rest of the clones were helpful in the finishing part and were also used in efficiency testing of the method.

Since the obtained sequences were not quality clipped, they required to be cut. After some analysis, the sequences from SARS215 were trimmed by 90 nucleotides at the beginning and 40 at the end. To achieve the best possible result, various combinations of the algorithm parameters were used, but $eb = 0.04$ and $mo = 40$ turned out to be the best in this case. The application found five main contigs, partially overlapping with one another, whose lengths ranged from about 2500 bp to about 9500 bp. The contigs covered 99.8% of the entire genome. Comparison of this result with the previously assembled sequence (AY274119.3 from NCBI) proved that the five contigs are subsequences of AY274119.3 (as expected). Moreover, they match in about 99%. This is a very good result, which allowed easy manual finishing of the assembling process. The manually constructed genome is almost identical

to the AY274119.3 sequence. There are only two small differences between them:

- nucleotide 10249 is U instead of C;
- instead of AUAUUAGGUUUU at the beginning of the sequence, there is AAUUCGCGGCCGCGUCG.

To further evaluate the quality of the result and also the efficiency of ASM, it was compared to two other, publicly available, assembling applications – Phrap (http://www.pharp.org) (one of the most popular assembling software) and CAP3 (http://genome.cs.mtu.edu/cap/cap3.html). Both were run with the same trimmed SARS215 sequences on input, as described above. Phrap produced three overlapping contigs, one about 18500 bp and two about 6000 bp long, covering also about 99.8% of the entire genome. The first two match AY274119.3 in about 98.15% and the third one in 97.2%, which is an up to 2% drop in quality comparing to ASM. CAP3 produced four contigs of lengths from 1000 bp to 11000 bp, covering barely 98% of the genome. Three of them almost perfectly matched AY274119.3, but the fourth one in about 98.5%.

As far as time efficiency is concerned, all three methods were tested on the SARS211, SARS212, SARS213, SARS214 and SARS215 sets. The sequences were trimmed as well, but sequences from sets other that SARS215 were trimmed by 150 nucleotides at the beginning and 650 at the end, because of their low quality. Table 1 presents the execution times for them. As mentioned in Materials and Methods, the test was performed on SunFire 6800. For this particular test, in order to make equal the chances for all the applications, the parallel computation abilities of ASM were disabled. However, the ability to be executed on a parallel machine is a great advantage of ASM as compared to the other methods, since it can drastically decrease the time of computations.

Table 1. Comparison of computation times of the assembling programs.

clone	input size	CAP3	Phrap	ASM
SARS211	371 kB	236 min	7 s	217 s
SARS212	275 kB	52 min	8 s	66 s
SARS213	244 kB	35 min	8 s	57 s
SARS214	152 kB	23 min	3 s	25 s
SARS215	489 kB	61 min	23 s	176 s

DISCUSSION

The ASM application is very efficient and thus it was able to assemble a good quality SARS-CoV genome in a short time. The results of the computational experiment have shown that the efficiency of the proposed algorithm and the ASM application is comparable to the Phrap's one (it is only slightly slower) and much better than the efficiency of CAP3. These results are quite promising and give basis for further research. Moreover, ASM has a unique property in this class of applications which is its distributed structure (Błażewicz *et al.*, 2004b). It means that the program can be run in the environments of distributed systems, which should result in a significant increase of computation performance. This property is especially important since parallel and distributed systems have become more available recently.

As far as the quality is concerned, in results of all three applications are quite good and allow for easy manual finishing. However, the best results are obtained using ASM, thus simplifying the job of scientists to the neces-

sary minimum. CAP3 has also very good results (contigs usually are slightly better than in the case of ASM), but the price is high – the computation time is very long and not all the genome is covered. Only Phrap was able to produce few very long contigs, covering almost the entire genome like ASM. However, their quality was not very high. To summarize, all the methods are quite good and each one is superior to the others on a particular aspect. All produce results of a quality good enough to easily do the finishing. However, only ASM managed to cover almost perfectly the entire genome, thus proving to be very useful in practice.

Manual finishing of the SARS-CoV genome, basing on the results of ASM, revealed small differences with AY274119.3. However, due to low coverage of those areas, it is impossible to determine which result is correct. It requires additional research. Anyway, these results prove that the sequence assembled by the Canada's Michael Smith Genome Science Centre is correct (allowing small variations) and at the same time verifies the presented approach in practice. The application satisfies the demand for a new, fast and reliable method of DNA sequence assembling.

Moreover, there is a growing need to make sequencing data publicly available. Free access to such data makes a comparison of methods (algorithmic and computational) and of their results possible, which could lead to a significant increase of the quality of published genomic sequences.

REFERENCES

Bains W. (1991) Hybridization methods for DNA sequencing. *Genomics.*; **11**: 294–301.

Błażewicz J, Formanowicz P, Kasprzak M, Markiewicz WT, Węglarz J. (1999a) DNA sequencing with positive and negative errors. *J Comput Biol.*; **6**: 113–23.

Błażewicz J, Formanowicz P, Kasprzak M, Markiewicz WT. (1999b) *Method of Sequencing of Nucleic Acids.* The Patent Office of the Republic of Poland, Patent Application No. P 335786.

Błażewicz J, Formanowicz P, Kasprzak M, Markiewicz WT, Węglarz J. (2000) Tabu search for DNA sequencing with false negatives and false positives. *Eur J Oper Res.*; **125**: 257–65.

Błażewicz J, Formanowicz P, Kasprzak M, Markiewicz WT, Świercz A. (2004a) Tabu search algorithm for DNA sequencing by hybridization with isothermic libraries. *Comput Biol Chem.*; **28**: 11–9.

Błażewicz J, Kasprzak M, Jackowiak P, Janny D, Jarczyński D, Nalewaj M, Nowierski B, Styszyński R, Szajowski Ł, Widera P. (2004b) *ASM – DNA Assembly Application.* Report RA-001/2004, Poznań Supercomputing and Networking Center.

Chin G, Coontz R, Helmuth L. (2004) Biology by the numbers. *Science.*; **303**: 781.

Cormen TH, Leiserson CE, Rivest RL. (1990) *Introduction to Algorithms*. MIT Press, Cambridge.

Farci P, Strazzera R, Alter HJ, Farci S, Degioannis D, Coiana A, Peddis G, Usai F, Serra G, Chessa L, Diaz G, Balestrieri A, Purcell RH. (2002) Early changes in hepatitis C viral quasispecies during interferon therapy predict the therapeutic outcome. *Proc Natl Acad Sci USA.*; **99**: 3081–6.

Figlerowicz M, Alejska M, Kurzyńska-Kokorniak A, Figlerowicz M. (2003) Genetic variability: the key problem in the prevention and therapy of RNA-based virus infections. *Med Res Rev.*; **23**: 488–518.

Gusfield D. (1997) *Algorithms on Strings, Trees, and Sequences. Computer Science and Computational Biology*. Cambridge University Press, Cambridge.

Gutin G, Punnen AP, eds. (2002) *Travelling Salesman Problem and its Variations*. Kluwer Academic Publishers, Dordrecht.

Marra MA, Jones SJ, Astell CR, Holt RA, Brooks-Wilson A, Butterfield YS, Khattra J, Asano JK, Barber SA, Chan SY, Cloutier A, Coughlin SM, Freeman D, Girn N, Griffith

OL, Leach SR, Mayo M, McDonald H, Montgomery SB, Pandoh PK, Petrescu AS, Robertson AG, Schein JE, Siddiqui A, Smailus DE, Stott JM, Yang GS, Plummer F, Andonov A, Artsob H, Bastien N, Bernard K, Booth TF, Bowness D, Czub M, Drebot M, Fernando L, Flick R, Garbutt M, Gray M, Grolla A, Jones S, Feldmann H, Meyers A, Kabani A, Li Y, Normand S, Stroher U, Tipples GA, Tyler S, Vogrig R, Ward D, Watson B, Brunham RC, Krajden M, Petric M, Skowronski DM, Upton C, Roper RL. (2003) The genome sequence of the SARS-associated coronavirus. *Science.*; **300**: 1399–404.

Myers EW, Sutton GG, Delcher AL, Dew IM, Fasulo DP, Flanigan MJ, Kravitz SA, Mobarry CM, Reinert KHJ, Remington KA, Anson EL, Bolanos RA, Chou HH, Jordan CM, Halpern AL, Lonardi S, Beasley EM, Brandon RC, Chen L, Dunn PJ, Lai ZW, Liang Y, Nusskern DR, Zhan M, Zhang Q, Zheng XQ, Rubin GM, Adams MD, Venter JC. (2000) A whole-genome assembly of *Drosophila*. *Science.*; **287:** 2196–204.

Pevzner PA. (2000) *Computational Molecular Biology. An Algorithmic Approach*. MIT Press, Cambridge, London.

Richman DD. (2001) HIV chemotherapy. *Nature.*; **410**: 995–1001.

Rota PA, Oberste MS, Monroe SS, Nix WA, Campagnoli R, Icenogle JP, Penaranda S, Bankamp B, Maher K, Chen MH, Tong SX, Tamin A, Lowe L, Frace M, DeRisi JL, Chen Q, Wang D, Erdman DD, Peret TCT, Burns C, Ksiazek TG, Rollin PE, Sanchez A, Liffick S, Holloway B, Limor J, McCaustland K, Olsen-Rasmussen M, Fouchier R, Gunther S, Osterhaus ADME, Drosten C, Pallansch MA, Anderson LJ, Bellini WJ. (2003) Characterization of a novel coronavirus associated with severe acute respiratory syndrome. *Science.*; **300**: 1394–9.

Setubal J, Meidanis J. (1997) *Introduction to Computational Molecular Biology*. PWS Publishing Company, Boston.

Smith TF, Waterman MS. (1981) Identification of common molecular subsequences. *J Mol Biol.*; **147**: 195–7.

Southern EM. (1988) United Kingdom Patent Application GB8 810400.

Waterman MS. (1995) *Introduction to Computational Biology. Maps, Sequences and Genomes*. Chapman & Hall, London.

Wilbur WJ, Lipman DJ. (1983) Rapid similarity searches of nucleic acid and protein data banks. *Proc Natl Acad Sci USA.*; **80**: 726–30.