RAIRO Operations Research
Will be set by the publisher

GENETIC AND TABU SEARCH ALGORITHMS FOR PEPTIDE ASSEMBLY PROBLEM

Jacek Błażewicz[1], Marcin Borowski[2], Piotr Formanowicz[3] and Tomasz Głowacki[4]

Abstract. Determining amino acid sequences of protein molecules is one of the most important issues in molecular biology. These sequences determine protein structure and functionality. Unfortunately, direct biochemical methods for reading amino acid sequences can be used for reading short sequences only. This is the reason, which makes peptide assembly algorithms an important complement of these methods. In this paper, a genetic algorithm solving the problem of short amino acid sequence assembly is presented. The algorithm has been tested in computational experiment and compared with an existing tabu search method for the same problem. The results clearly show that the genetic algorithm outperformed the tabu search approach.

Keywords: peptide sequencing, combinatorial optimization problem, genetic algorithm, tabu search algorithm

2009-10-28.

[1] Institute of Computing Science, Poznań University of Technology, Piotrowo 2, 60-965 Poznań, Poland; Institute of Bioorganic Chemistry, Polish Academy of Sciences, Noskowskiego 12/14, 61-704 Poznań, Poland

[2] Institute of Computing Science, Poznań University of Technology, Piotrowo 2, 60-965 Poznań, Poland

[3] Institute of Computing Science, Poznań University of Technology, Piotrowo 2, 60-965 Poznań, Poland; Institute of Bioorganic Chemistry, Polish Academy of Sciences, Noskowskiego 12/14, 61-704 Poznań, Poland

[4] Institute of Computing Science, Poznań University of Technology, Piotrowo 2, 60-965 Poznań, Poland

1 Introduction

Two of the most important groups of molecules in every living organism are *nucleic acids* (i.e. DNA and RNA) and *proteins*. DNA is used for storing and copying the genetic information. On the basis of this information, other types of molecules are built. In general, genetic information determines the structure and the functionality of any organism. Proteins compose the class of molecules responsible for most of organism's features. These molecules perform two main roles, i.e. they are building blocks of tissues and also catalyze many biochemical reactions. A protein function strongly depends on its three-dimensional structure. The structure is determined by the amino acid sequence and it may be also dependent on the environment. The problem of determining the three-dimensional protein structure on the basis of its amino acid sequence is one of the most important and challenging problems of computational and molecular biology. Clearly, the preliminary stage of the process of determining protein structure must be the reading of its amino acid sequence.

From the fact that the genetic information encoded in DNA determines the amino acid sequences of proteins, it could be concluded that there is no need for direct reading of the latter ones – it should suffice to read the DNA sequence and translate it to the amino acid one. However, in practice not always it is so easy. It happens that in the process of protein synthesis the amino acid sequence is modified by another protein. Moreover, very often for some biological and technological reasons it is easier to extract and analyze proteins than nucleic acids.

The existing methods for direct peptide (i.e. short amino acid sequences) reading (i.e. *sequencing*) are based on mass spectrometry or on Edman's degradation (cf. [10,12]). In each case only short peptides can be directly sequenced, i.e. 10–20 amino acid long peptides in the case of mass spectrometry and 50 amino acid peptides when the Edman's approach is used. Since protein sequences are usually longer, there is a need for a method which can be used to assemble the short fragments read by the direct methods. The *peptide assembly problem* was formulated as a combinatorial optimization one in [6]. Since it has been proved to be NP-hard in the strong sense [6], thus, the need arises to construct efficient heuristics for solving it. (Let us note here that the process of protein reading is similar to the one of reading DNA sequences [3,4,11,13]. However, due to the differences in lengths and the nature of biochemical procedures used in both processes, the latter is usually divided into 3 stages: *sequencing*, *assembling* and *mapping*. The corresponding combinatorial problems require different procedures than the ones for proteins.) In this paper, two of the frequently used methods for solving combinatorial problems, i.e. a *Genetic algorithm* and *Tabu search* method, adopted for solving the problem of peptide assembly, are proposed. The algorithms have been tested in an extensive computational experiment and compared. The results clearly show that the genetic algorithm outperformed the tabu search approach.

The organization of the paper is as follows. In Section 2 the peptide assembly problem is formulated. In Section 3 the Genetic algorithm is presented, while in

Section 4 the results of the computational experiment are shown. The paper ends with conclusions in Section 5.

2 Formulation of the peptide assembly problem

As mentioned in the previous section, direct peptide sequencing methods allow for reading only short amino acid sequences. In order to read a sequence of a whole protein, it should be cleaved into shorter fragments and then the fragments can be sequenced. Such a cleavage can be done using proteases, i.e. enzymes recognizing some specific amino acids and cutting a peptide sequence in the position directly following the position of the recognized amino acid. However, in this process the information about the order of the resulting shorter fragments in the whole protein is lost. The peptide assembly problem consists in recovering this information (cf. [1,5]).

Amino acid sequences can be seen as strings over some particular alphabet. Let Σ be such an alphabet corresponding to amino acids and let $C \subset \Sigma$ (Note that cardinality of Σ is 20 since there are 20 amino acids). The elements of set C, called cutters, correspond to amino acids recognizable by some proteases. If $c \in C$ then there exists some protease which is used to cut the examined protein sequence directly after c. Let us denote the protein sequence by s. A fragment obtainable from cutter c is substring z of s satisfying two conditions, i.e. c is the last symbol of z and if z starts in position i in sequence s ($i > 1$), then in position $i-1$ there is symbol c. If c occurs exactly once in z then it is said that sequence z results from a full digest of sequence s. It means that directly after all occurrences of symbol c the cuts have been made by the protease used in the biochemical experiment. If more than one occurrence of c is present in z then z resulted from a partial digest of s. A string is obtainable from C if there exists some $c \in C$ from which it is obtainable [2,6]. Besides sequences read in the sequencing process, the biochemical experiment can also provide numbers of occurrences of particular amino acids in the protein. This information is then used at the assembly stage of reading proteins.

The peptide assembly problem can be formally defined in the following way [6].

Peptide assembly problem – search version:
Instance: Multiset S of strings over alphabet Σ, set of cutters $C \subset \Sigma$ and a distribution (number of occurrences in S) D of letters from alphabet Σ, i.e. a set of pairs (x, n) for all symbols $x \in \Sigma$, where n is a positive integer.
Answer: Superstring for S satisfying D such that all elements of S are obtainable from $C \subset \Sigma$.

The problem is strongly NP-hard even in the ideal case without errors what was shown in [6]. Note, that even without the last assumption made in the proof about knowing the first and the last string of the resulting superstring, the proof is still correct, what means that the variant of the problem considered here is

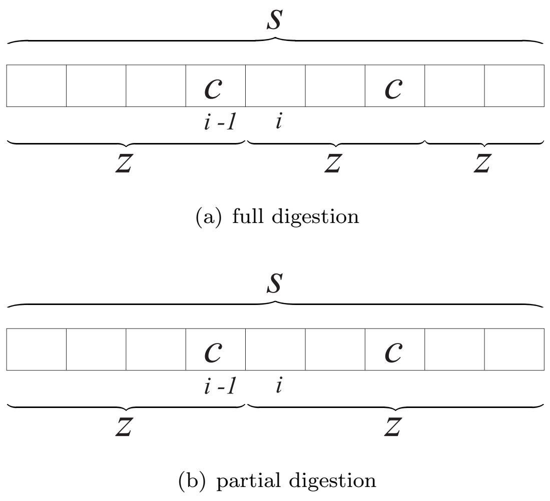


FIGURE 1. Digestions of sequence s (cutter = c)

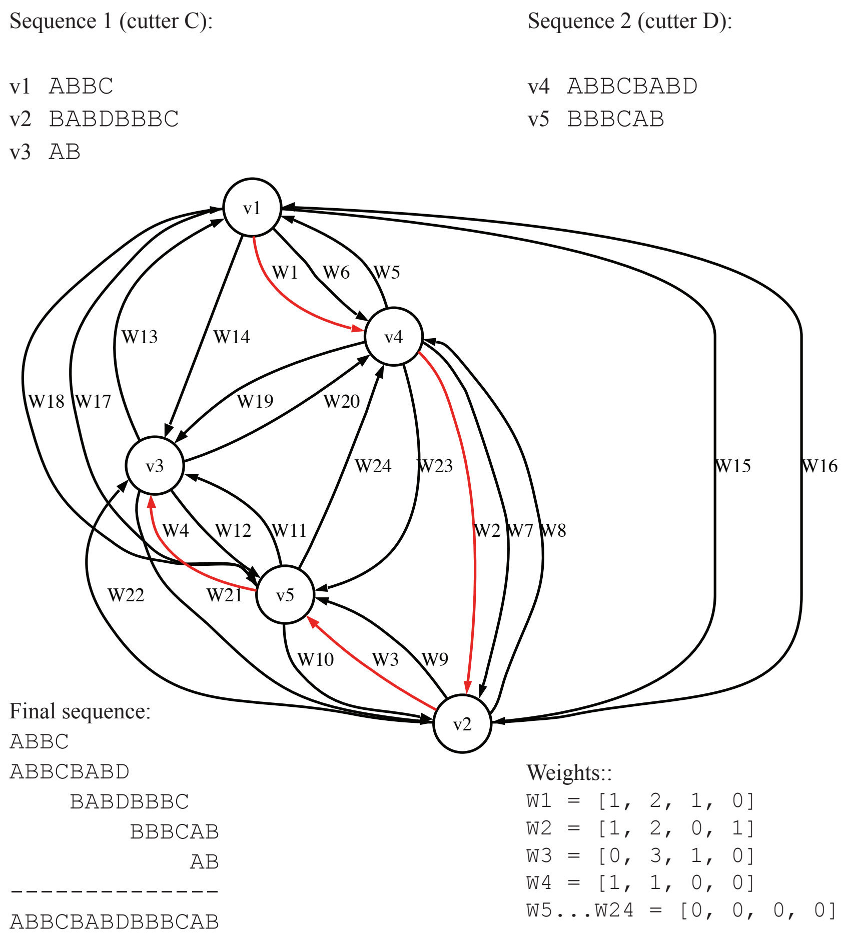
also NP-hard in the strong sense. This intractability result justifies looking for a heuristic method solving this problem.

The above formulated problem may be also expressed in terms of graph theory. This formulation is a basis for algorithms described in the next section. Each element of S may be modeled as a labeled vertex in certain graph $G = (V, A)$. The label of a vertex is a textual representation of a short peptide which the vertex corresponds to. There is arc j in $G = (V, A)$ from vertex v_1 to v_2 if and only if a suffix of a label of v_1 is equal to a prefix of a label of v_2. It is significant to mention that there is possible more than one overlapping between suffix of the label of v_1 and prefix of the label of v_2. As every potential overlapping between the vertices results in an arc in $G = (V, A)$, graph $G = (V, A)$ is a multigraph.

Weight W_j of corresponding arc j (note that W_j is a letter distribution associated with arc j, not a single value) is defined as a set of pairs (x, n) for all symbols $x \in \sum$ for a given prefix. Let us define graph $G' = (V, A')$ which is a modification of $G = (V, A)$ obtained by adding arcs of weight $(x, 0)$ between each ordered pair of vertices in $G = (V, A)$, so $A \subset A'$ (these additional arcs have been introduced to ensure that each possible solution will be a valid solution). An example graph is shown in Fig. 2. Note that graph $G' = (V, A')$ remains a multigraph.

Let us introduce E as a set of pairs (x, n) for all symbols $x \in \sum$. E is equal to a total distribution of letters for all labels related to vertices of graph $G' = (V, A')$ (note that this distribution calculated for $G = (V, A)$ is the same).

Now, we introduce vectors associated with sets of pairs that were defined above. We also propose the conversion between those sets of pairs and vectors. The latter are used since for vectors there are well defined mathematical operations useful



Sequence 1 (cutter C):

```
v1  ABBC
v2  BABDBBBC
v3  AB
```

Sequence 2 (cutter D):

```
v4  ABBCBABD
v5  BBBCAB
```

Final sequence:

```
ABBC
ABBCBABD
    BABDBBBC
        BBBCAB
            AB
--------------
ABBCBABDBBBCAB
```

Weights::

```
W1 = [1, 2, 1, 0]
W2 = [1, 2, 0, 1]
W3 = [0, 3, 1, 0]
W4 = [1, 1, 0, 0]
W5...W24 = [0, 0, 0, 0]
```

FIGURE 2. An example of graph $G' = (V, A')$

in our analysis of the problem and the algorithms. Let $\vec{D}$, $\vec{E}$ and $\vec{W}_j$ be 20-dimensional vectors that correspond to relevant set of pairs D, E and W_j. For the conversion to vectors, we assume that all sets of pairs (x, n) have been ordered alphabetically according to x. Each coordinate of these vectors is also labeled by some letter x. The value in the coordinate x of the vector is set to n. Now, having defined vectors, we can formulate a problem we resolve in graph theory. The solution of the problem is a path in graph $G' = (V, A')$ that contains all vertices of $G' = (V, A')$ and fulfills the following condition:

$$\vec{D} = \vec{E} - \sum_{j \in A'_1} \vec{W}_j$$

where $A'_1 \subseteq A'$ is a set of arcs chosen to a solution.

3 Algorithms

3.1 Genetic algorithm

Each element of the population is defined as a path in $G' = (V, A')$ which consists of all vertices of $G' = (V, A')$. The initial population is created randomly by permuting the set of vertices and, as we deal with a multigraph, choosing randomly an arc between each ordered pair of them. For each element of the population the evaluation function $f(l)$ is defined as a difference between the expected distribution $\vec{D}$ and the obtained distribution $\vec{E} - \sum\limits_{j \in A'_1} \vec{W}_j$ for a given solution $l \in L$:

$$f(l) = module(\vec{D} - \vec{E} + \sum_{j \in A'_1} \vec{W}_j)$$

where module is a length of a vector and $\vec{D}$, $\vec{E}$, $\vec{W}_{j'}$ have been defined in Section 2 and L is a population. The length was calculated in Manhattan distance. The Manhattan distance is a distance between two distributions measured along axes (i.e. dimensions). For example, the Manhattan distance, d_1 between two vectors $\vec{p}$ and $\vec{q}$ in an n-dimensional integer vector space with fixed Cartesian coordinate system can be calculated according to formula:

$$d_1(\vec{p}, \vec{q}) = \sum_{i=1}^{n} |p_i - g_i|$$

where $\vec{p} = (p_1, p_2, p_3, ..., p_n)$ and $\vec{q} = (q_1, q_2, q_3, ..., q_4)$ are vectors.

Function $f(l)$ is to be minimized. Two solutions from the population are selected randomly for recombination. To choose the low value of $f(l)$ the roulette method is used in a selection phase. The probability $P(l)$ of a selection of a certain solution l for recombination is inversely proportional to $f(l)$ and is equal to:

$$P(l) = \frac{f_{\max} - f(l)}{\sum\limits_{m \in L} (f_{\max} - f(m))}$$

where $f_{\max}$ is a maximum value of $f(l)$ for a certain population. A representation of a solution is a sequence of vertices and arcs between them: $v_1, e_1, ..., e_{n-1}, v_n$. During the recombination phase, a list of common subsequences for two chosen solutions is created. One random element of this list is taken as a part of a new solution. Thereafter randomly chosen elements of the list are added to the partial solution in the following way: insert element in a randomly selected position of the partial solution satisfying conditions that an arc which precedes element has a nonzero value (the module of the arc is nonzero) and an arc which connects the considered element to the rest of the solution is nonzero too. If there is no such a place in the solution, choose randomly a place satisfying only one of the conditions. Finally, if there are no places in the solution which allow to connect element by a nonzero value arc, choose randomly any place in the solution. If a value of f calculated for the new solution is smaller than a value of the worst solution in the population, then the worst solution is replaced by the new one.

The probability of a mutation was experimentally set to 0.05. When the mutation occurs some element of a population is chosen randomly. For the chosen element the two subsequences are randomly chosen and swapped. The new nonzero value arcs between an existing solution and these subsequences are chosen randomly.

The proposed Genetic algorithm can be formally described in the following way:

```
randomly create initial population $P(1)$
for $i = 1$ to 10000 do
begin
    select two elements $X_l$ and $X_k$ of $P(i)$ according to the roulette rule
    $S$:={all common subsequences of $X_l$ and $X_k$}
    new_solution:=randomly selected element of $S$ for $j = 2$ to size($S$) do
    begin
        $x$:=randomly selected element of $S$
        insert $x$ in randomly selected position $p$ in new_solution
        – position $p$ should satisfy the following conditions:
        substring new_solution$(p-1)$ should be overlapped with new_solution$(p)$
        and new_solution$(p)$ should be overlapped with new_solution$(p+1)$
        if it is impossible then
        begin
            insert $x$ in randomly selected position $p'$ satisfying the condition that
            new_solution$(p')$ can be overlapped with new_solution$(p'-1)$
            or with new_solution$(p'+1)$
            if it is impossible then
            begin
                insert $x$ in a randomly selected position of new_solution
            end
        end
    end
    if $f$(new_solution) $< f(X_l)$ or $f$(new_solution) $< f(X_k)$ then
    begin
        replace the worst of solutions $X_l$ and $X_k$ with new_solution
        and create population $P(i+1)$
    end
    //MUTATION
    mutate = random value$\in \langle 0; 1\rangle$
    if (mutate $\leq 0.05$)
    begin
        randomly select element $X_q$ of $P(i+1)$
        randomly select two subsequences $s_a$ and $s_b$ of $X_q$
        move $s_a$ to the position of $s_b$ and $s_b$ to the position of $s_a$ in $X_q$
        randomly select the overlappings of the moved subsequences with their
        neighbors in $X_q$ (if more than one possible overlap exists)
    end
end.
```

3.2 Tabu Search

To evaluate efficiency of the proposed Genetic algorithm, the second method solving instances of this problem have been designed - the Tabu search heuristic, which is a method being one of the most frequently used in combinatorial optimization.

The Tabu search method belongs to local search methods, formally described in [2], where the general step of an iterative procedure consists in constructing next solution j from current solution i and checking whether one should stop there or perform another step. Next solution j is chosen from the set $N(i)$ - a neighborhood of feasible solutions of current solution i. To improve the efficiency of the search process, the method should keep not only local information like current value of the objective function but also other information related to the search process. Commonly used are the best solution visited and the mechanisms to prevent the method being stuck in a local optimum - tabu list - a list of moves (decisions of choice next solution j from current solution i) already performed by the algorithm. None of the moves from tabu list can be performed unless it leads to a solution better than the best already found. The next mechanism preventing the method being stuck in a local optimum is a mechanism of random moves, which results in moving the search process to another area of the search space.

The details of the Tabu search algorithm for peptide assembly problem have been described in [2]. The main difference between the version of the algorithm described in [2] and the one used in the current paper is the definition of the objective function. Previously it was a maximization of a sum of overlaps of peptides from multiset S (spectrum) for particular permutation, while now objective function f was redefined to minimize the Manhattan distance between 20-dimensional vectors of amino acid distributions of the obtained solution and the expected one (similarly to the objective function of the proposed Genetic algorithm). Note, that the latter one can be determined in a biochemical experiment.

4 Computational experiment

The Genetic algorithm described in the previous section as well as the Tabu search method have been implemented in Java 1.5 language and run on PC Intel 2×Xeon 3.6 GHz with 4 GB RAM.

The computational experiment has been divided into three parts. In all of them real protein sequences (see Appendix) composed of 100, 150, 200, 250 and 300 amino acids have been used to generate instances of the assembly problem. 10 sequences of each length have been chosen, hence the set of initial sequences (the real protein sequences used to generate the instances) has been composed of 50 sequences.

In Part I of the computational experiment 9 problem instances have been generated on the basis of each initial sequence. First, in each of the sequences 10, 15 and 20 positions have been randomly chosen, respectively, and amino acids present in these positions have been replaced by cutters (in this stage the cutters are some artificial amino acids). This operation resulted in 3 new sequences for each initial sequence. Then, for each of the new sequences containing 10 artificial cutter positions, 1, 2 and 3 of them have been randomly selected and marked as sources of errors, i.e. it has been assumed that they will not be recognized by a protease. In the case of sequences containing 15 cutter occurrences, 1, 3 and 4 of them have been marked and in the case of sequences with 20 cutter positions, 2, 4 and 6 of them have been marked as error sources. Finally, each sequence modified in this way has been cut directly after every cutter occurrence (except the ones marked as error sources). In this way, each initial sequence resulted in 9 instances, as previously mentioned.

In Part II the initial sequences have not been modified. Instead, two amino acids, i.e. *aspartic acid* and *proline* have been chosen as cutter proteases. Then, for each of the sequences 1, 2 and 3 cutter positions have been marked and omitted during cutting procedure (analogously like in Part I). Finally, every sequence has been cut directly after these cutter occurrences, except the marked ones. In this way, every initial sequence has been used to generate 3 instances.

Part III is similar to Part II. The difference is that in this case it has been assumed that the proteases used are *endoproteinase Lys-C*, recognizing amino acid *lysine*, and *dilute acid* recognizing amino acid *asparagine*. (Note that in Part II two amino acids have been chosen as the ones recognized by some proteases and in Part III two real proteases have been chosen first and then the recognized amino acids have been located in the sequences. This may result in a difference in the number of peptide fragments obtained as a result of the cutting process.) 1, 2 and 3 cutter positions corresponding to the chosen proteases have been marked as error sources and the sequences have been cut. In this way, every initial sequence has been used to generate 3 instances.

In Part I for each combination of a sequence length, a number of substitutions and a number of error sources (number of marked cutter positions) and in Parts II and III for each combination of a sequence length and a number of errors, respectively, 10 instances have been used in the computational experiment. Each instance has been run 10 times and mean values of computation times and of the similarity of the obtained solution to the original sequence have been calculated.

The similarity has been calculated according to Needleman–Wunsch algorithm [2,9]. The algorithm compares two sequences: the one generated by a tested algorithm s_t and the original sequence s_o. The similarity of the sequences is determined according to the following formula:

$$\sigma = 100\frac{\delta - \psi}{\chi - \psi}$$

where δ is a scoring for the two sequences, being a sum of scores for all columns in an optimal alignment (1 point for a match, -1 for a mismatch or a gap), and:

$$\psi = \begin{cases} l(s_o)d + (l(s_t) - l(s_o))g & \text{if } l(s_t) > l(s_o) \\ l(s_t)d + (l(s_o) - l(s_t))g & \text{otherwise} \end{cases}$$

$$\chi = \begin{cases} l(s_t)m & \text{if } l(s_t) > l(s_o) \\ l(s_o)m & \text{otherwise} \end{cases}$$

where $l(s_t)$ and $l(s_o)$ are lengths of sequence s_t and s_o, respectively, and $m = 1$, $d = -1$, $g = -1$.

The results of Parts I, II and III of the experiment are shown in Tables 1, 2 and 3, respectively. Each entry in the tables corresponds to computations performed on 10 instances (i.e. it is a mean value of 10 means, since each instance has been run 10 times). Similarities to original sequences in Part I are greater than in case of Part II and Part III, because in this stage cut positions were generated artificially and the number of them was limited. In Parts II and III, the numbers of cuts were unknown and depended on amino acid sequences of each instance. In all cases similarity of the obtained sequences to the original one decreases when the sequence length increases, which is not a surprise.

In order to illustrate better a comparison of the two algorithms, the results of the experiments in Part III have been additionally depicted in a graphical form in Fig. 3. We see that the similarities of the obtained sequences to the original sequence for the Genetic algorithm are much higher than in the case of the Tabu search algorithm and the value of similarity for the first algorithm was never lower than ca. 80%. In the case of Tabu these similarities are going down with the increase of a sequence length, reaching 45%. Computational time of finding the results in the case of the Genetic algorithm was more or less constant and equal to ca. 2 seconds, while in case of the Tabu search, computational time grows quite fast with the increase of a sequence length.

5 Conclusions

The experiment which results have been presented in the previous section has been performed on instances containing errors. These errors correspond to the situation where not all amino acids which should be recognized by the used proteases have been really recognized. This makes the assembly problem computationally intractable. The results of the computational experiment clearly show that the Genetic algorithm outperforms the Tabu search method in the sense of solution quality and computation time. In each of the test parts, the Genetic algorithm gave better solution than the Tabu search - in some cases similarity was equal to 100%. The time of finding solution in case of the Tabu search was even over 100 times longer than in case of the Genetic algorithm - especially it can be seen in the results of Part II and III, respectively. The computational tests confirmed quite high efficiency of the algorithms. Especially, the Genetic algorithm could be useful in the protein identification process.

6 Acknowledgements

This research has been partially supported by grant NN519314635 from the Ministry of Science and Higher Education, Poland. The first author acknowledges the support of INRIA Rhone-Alpes grant.

Table 1. Results for Part I.

Sequence length	Number of substitutions	Number of errors	Tabu Search Similarity [%]	Tabu Search Time [s]	Genetic Algorithm Similarity [%]	Genetic Algorithm Time [s]
100	10	10%	77.20	2.39	98.15	0.98
100	10	20%	80.80	1.05	97.39	0.94
100	10	30%	78.70	1.29	96.86	1.08
100	15	10%	55.83	8.82	90.00	1.08
100	15	20%	56.41	5.93	90.46	1.10
100	15	30%	60.42	5.39	86.75	0.98
100	20	10%	40.41	38.01	78.33	0.94
100	20	20%	41.77	25.41	74.60	1.08
100	20	30%	49.78	14.91	78.26	1.09
150	10	10%	65.39	6.85	93.45	0.96
150	10	20%	73.85	4.30	88.83	1.08
150	10	30%	85.90	1.85	98.86	1.10
150	15	10%	48.11	19.11	90.45	1.20
150	15	20%	55.64	14.47	92.79	1.02
150	15	30%	53.80	10.79	90.76	1.06
150	20	10%	46.33	41.71	89.05	1.14
150	20	20%	55.00	23.99	91.33	1.06
150	20	30%	50.08	22.41	91.67	1.05
200	10	10%	66.62	4.73	98.10	1.12
200	10	20%	84.89	4.44	98.47	1.09
200	10	30%	95.44	1.40	100.00	1.09
200	15	10%	50.76	22.16	88.52	1.05
200	15	20%	57.38	10.26	91.68	1.09
200	15	30%	60.81	12.78	92.66	1.06
200	20	10%	42.90	68.99	91.01	1.27
200	20	20%	49.50	40.03	91.38	1.15
200	20	30%	52.79	22.31	93.96	1.00
250	10	10%	85.52	3.37	100.00	1.33
250	10	20%	74.30	5.62	93.40	1.05
250	10	30%	83.53	3.28	97.50	1.01
250	15	10%	59.89	14.50	92.09	1.18
250	15	20%	55.97	15.61	89.58	1.11
250	15	30%	72.20	8.63	99.47	1.03
250	20	10%	45.25	66.94	82.35	1.09
250	20	20%	43.51	62.48	90.73	1.16
250	20	30%	60.14	17.48	98.00	1.16
300	10	10%	76.67	11.69	92.77	1.24
300	10	20%	76.23	4.36	100.00	1.08
300	10	30%	84.23	3.44	98.98	1.25
300	15	10%	59.16	15.29	96.59	1.19
300	15	20%	63.19	12.88	95.30	1.01
300	15	30%	69.71	10.13	96.37	1.10
300	20	10%	48.12	56.94	90.52	1.28
300	20	20%	59.25	23.33	95.32	1.06
300	20	30%	50.93	28.62	95.26	1.09

References

[1] J. Błażewicz, M. Borowski, P. Formanowicz, T. Głowacki, On graph theoretical models for peptide sequence assembly, *Foundations of Computing and Decision Sciences* 30 (2005) 183–191.

[2] J. Błażewicz, M. Borowski, P. Formanowicz, M. Stobiecki. Tabu Search Method for Determining Sequences of Amino Acids in Long Polypeptides, *Lecture Notes in Computer Science* 3449 (2005) 22–32.

[3] J. Błażewicz, M. Kasprzak, Combinatorial optimization in DNA mapping - a computational thread of the Simplified Partial Digest Problem, *RAIRO - Operations Research* 39 (2005) 227-241.

TABLE 2. Results for Part II.

Sequence length	Number of errors	Tabu Search		Genetic Algorithm	
		Similarity [%]	Time [s]	Similarity [%]	Time [s]
100	1	62,55	6.35	86.46	1.92
100	2	70.86	5.76	88.18	2.29
100	3	73.13	3.80	92.36	2.68
150	1	46.72	21.44	80.58	2.58
150	2	53.52	16.23	84.70	2.44
150	3	56.77	9.05	87.62	2.12
200	1	46.36	84.31	80.16	2.93
200	2	43.22	77.04	79.78	1.76
200	3	48.22	64.15	81.53	2.97
250	1	41.36	200.60	79.13	2.07
250	2	43.42	167.90	80.24	1.76
250	3	44.10	147.35	81.60	2.18
300	1	40.53	384.89	81.05	3.00
300	2	40.64	366.26	78.86	2.93
300	3	41.12	197.10	80.46	2.69

TABLE 3. Results for Part III.

Sequence length	Number of errors	Tabu Search		Genetic Algorithm	
		Similarity [%]	Time [s]	Similarity [%]	Time [s]
100	1	81.33	6.07	82.58	1.87
100	2	85.73	1.76	87.19	1.93
100	3	65.66	2.11	88.92	1.89
150	1	70.22	15.39	83.54	1.73
150	2	73.89	14.73	81.17	2.02
150	3	74.19	12.62	85.67	1.98
200	1	56.67	86.49	80.19	1.86
200	2	58.91	58.58	81.12	1.92
200	3	62.41	51.43	84.56	2.01
250	1	49.22	150.46	81.86	1.89
250	2	49.54	154.40	79.34	1.99
250	3	54.23	93.04	83.27	1.94
300	1	45.88	289.12	83.40	1.96
300	2	47.98	289.48	80.96	2.02
300	3	44.58	250.75	81.02	2.11

[4] J. Błażewicz, P. Formanowicz, M. Kasprzak, Selected combinatorial problems of computational biology, *European Journal of Operational Research* 161 (2005) 585-597.

[5] P. Formanowicz. Selected Combinatorial Aspects of Biological Sequence Analysis, Poznań, Publishing House of Poznań University of Technology 2005.

[6] J. K. Gallant, The complexity of the overlap method for sequencing biopolymers, *Journal of Theoretical Biology* 101 (1983) 1–17.

[7] F. Glover, Tabu Search, Part I, *ORSA Journal on Computing* 1 (1989) 190–206.

[8] F. Glover, Tabu Search, Part II, *ORSA Journal on Computing* 1 (1990) 4–32.

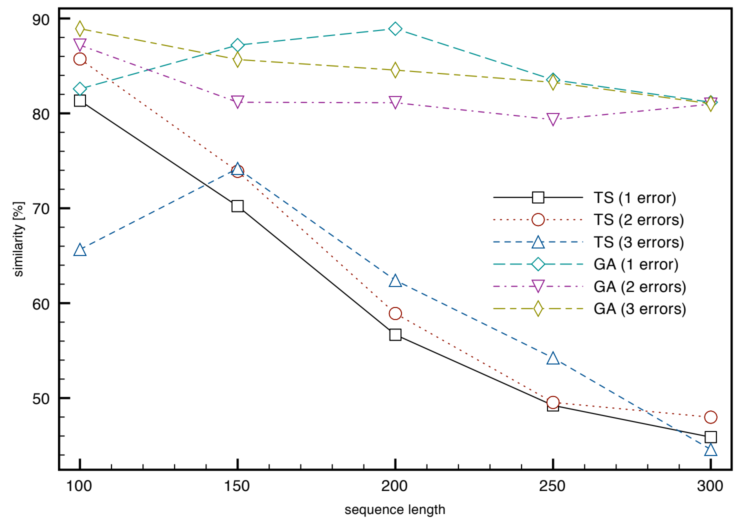
[9] S.B Needleman, C.D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, *Journal of Molecular Biology* 48 (1970) 443–453.

[10] P. A. Pevzner, *Computational molecular biology. An algorithmic approach*, Cambridge, Massachusetts, The MIT Press, 2000.

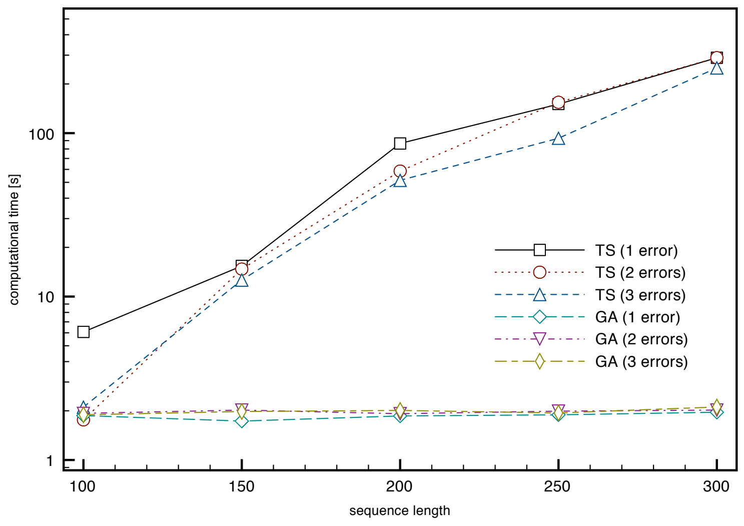
[11] J. C. Setubal, J. Meidanis, *Introduction to computational molecular biology*, Boston, PWS Publishing Co., 1996.

[12] L. Stryer, *Biochemistry*, 4th edition, New York, W.H. Freeman and Company, 1995.

[13] M. S. Waterman, *Introduction to computational biology*, London, Chapman & Hall, 1995.



(a) similarity



(b) computational time

FIGURE 3. Results for Part III (TS stands for Tabu search and GA for Genetic algorithm, respectively).

A List of test sequences (with GenBank accession numbers)

AAI22861: SARM1 protein [Homo sapiens]

AAI22867:	PTPRR protein [Homo sapiens]
AAI25270:	Chromosome 16 open reading frame 84 [Homo sapiens]
AAI25271:	FSHR protein [Homo sapiens]
AAI25273:	gi|115940697|gb|AAI25273.1|
AAI25274:	ZNF497 protein [Homo sapiens]
ABI97387:	ADAM metallopeptidase domain 33 [Homo sapiens]
ABI97388:	ATP-binding cassette, sub-family G (WHITE), member 2 [Homo sapiens]
ABI98401:	lung specific F-box and DH domain containing protein [Homo sapiens]
ABJ09587:	sodium-driven chloride bicarbonate exchanger [Homo sapiens]