The Simplified Partial Digest Problem: approximation and a graph-theoretic model[☆]

Jacek Blazewicz[a,b], Edmund K. Burke[c], Marta Kasprzak[a,b,*], Alexandr Kovalev[a], Mikhail Y. Kovalyov[d]

[a] *Institute of Computing Science, Poznan University of Technology, Piotrowo 2, 60-965 Poznan, Poland*
[b] *Institute of Bioorganic Chemistry, Polish Academy of Sciences, Noskowskiego 12, 61-704 Poznan, Poland*
[c] *School of Computer Science, University of Nottingham, Jubilee Campus, Nottingham NG8 1BB, UK*
[d] *United Institute of Informatics Problems, National Academy of Sciences of Belarus, Surganova 6, 220012 Minsk, Belarus*

Abstract

The goal of the Simplified Partial Digest Problem (SPDP) is motivated by the reconstruction of the linear structure of a DNA chain with respect to a given nucleotide pattern, based on the multiset of distances between the adjacent patterns (interpoint distances) and the multiset of distances between each pattern and the two unlabeled endpoints of the DNA chain (end distances). We consider optimization versions of the problem, called SPDP-Min and SPDP-Max. The aim of SPDP-Min (SPDP-Max) is to find a DNA linear structure with the same multiset of end distances and the minimum (maximum) number of incorrect (correct) interpoint distances. Results are presented on the worst-case efficiency of approximation algorithms for these problems. We suggest a graph-theoretic model for SPDP-Min and SPDP-Max, which can be used to reduce the search space for an optimal solution in either of these problems. We also present heuristic polynomial time algorithms based on this model. In computational experiments with randomly generated and real-life input data, our best algorithm delivered an optimal solution in 100% of the instances for a number of restriction sites not greater than 50.

[☆]This work was supported by the Marie Curie BIOPTRAIN fellowship of Alexandr Kovalev, EPSRC grant GR/S64530/01 and the Ministry of Science of Poland grant N N519 314635.

[*]Corresponding author

Email addresses: jblazewicz@cs.put.poznan.pl (Jacek Blazewicz), ekb@cs.nott.ac.uk (Edmund K. Burke), marta@cs.put.poznan.pl (Marta Kasprzak), akovalev@cs.put.poznan.pl (Alexandr Kovalev), kovalyov_my@newman.bas-net.by (Mikhail Y. Kovalyov)

Keywords: integer programming, genome mapping, combinatorial optimization, heuristics, approximation algorithms

1. Introduction to partial digesting in DNA mapping

Over the last sixty years or so, Operational Research techniques and methodologies have facilitated important scientific and managerial advances across a broad spectrum of industries and application areas [24]. This inherently inter-disciplinary tradition is as evident today as it was at the very beginning of Operational Research activity. Moreover, recent years have seen the scope of this inter-disciplinarity expand into the life sciences as the availability of computational power increases and as our understanding of the strong potential of Operational Research techniques in these areas expands. In particular, the last few years have seen a significant level of Operational Research activity in bioinformatics and related applications. For example, without understanding of Operational Research based techniques it would have been impossible to have achieved the major breakthrough of deciphering the human genome. Graph-theoretic based approaches were employed for the assembling stage of genome reading [36, 21, 28]. Another example of the impact of Operational Research on bioinformatics is the novel employment of genetic and tabu search algorithms, as well as other metaheuristic schemes, to overcome the oligonucleotide repetition drawback of the sequencing by hybridization approach [7, 38, 27, 8]. Also the problems arising around the analysis of aminoacid sequences are solved with the use of Operational Research methods [26, 1]. It is also worth noting that the EURO (European Association of Operational Research Societies) distinguished Ph.D. prize was awarded in 2006 to Marta Szachniuk for her thesis on the application of Operational Research techniques to the analysis of NMR images of RNA chains. Furthermore, [2] employs dynamic programming to establish results which deepened our understanding of the simplified partial digest problem. We build upon these achievements in the current paper to present new results which not only impact upon Operational Research but which also demonstrate the far reaching effect of Operational Research methodologies across disciplinary boundaries and, in particular, in bioinformatics and genome mapping.

An essential characteristic of a DNA molecule is its linear structure, which can be viewed as a sequence of four nucleotide bases: adenine, cytosine, guanine and thymine. Modern

measurement technologies are unable to determine the linear structure of a DNA molecule directly. *Mapping* is one of three typical existing indirect methods [3]. Clones of a DNA molecule, under mapping, are exposed to restriction enzymes that cut them at particular patterns of nucleotides called *restriction sites*. The lengths of the cut fragments and appropriate information about the cutting technique represent inputs to *restriction site analysis*, see [31], [37] or [29] for details. Typical cutting approaches in the literature are *double digest*, where two restriction enzymes are employed [37, 29, 12], and *partial digest*, where the DNA is cut by one enzyme but with different reaction times [15, 16].

The *Partial Digest Problem (PDP)* is the mathematical model for the partial digest. The output of the partial digestion and the input for PDP is to be a multiset of lengths of all possible DNA fragments, where the endpoints of a fragment are the restriction sites or the ends of the DNA chain. For n restriction sites and 2 ends of the DNA chain, this multiset should consist of $\binom{n+2}{2} = \frac{(n+1)(n+2)}{2}$ fragment lengths. As Pevzner [29] writes in his book, the partial digestion has never been the favorite mapping method in biological laboratories because of a difficulty in performing cuts for every pair of restriction sites. Although the PDP approach is sometimes used in practice (cf. works of the Nobel Prize winner Christiane Nuesslein-Volhard that reconstructed the genome of zebrafish [35]), experiments applying this technique are usually quite small: i.e. for chains containing less than 20 restriction sites [18, 20, 23, 25].

PDP has been studied in the ideal *error-free* case and in the case of experimental errors [33, 34, 11, 10]. PDP was proved to be NP-hard in the cases of measurement errors [10] and noisy data [11]. The NP-hardness of the original error-free PDP as well as a polynomial-time solution algorithm for it is not known [17].

A *simplified partial digest method* and a corresponding mathematical model, called *Simplified Partial Digest Problem (SPDP)*, was proposed in [4], and it was later studied in [2, 3, 4, 5, 6].

2. Formulation and discussion of SPDP

The output of the simplified partial digestion and the input for SPDP consists of a multiset B of distances between each two adjacent sites, which we call *interpoint distances*, and a multiset A of distances between each site and the ends of the DNA chain, which we call *end distances*. Each two adjacent sites produce one interpoint distance, which is the length of the corresponding DNA fragment, and each single site produces two end distances,

which are the lengths of the two DNA fragments, whose one endpoint is this site and the other endpoint is the end of the DNA chain. Thus, for n restriction sites, $|B| = n + 1$ and $|A| = 2n$. Since the size of the input for SPDP is much smaller than that for PDP, its solution is less affected by the measurement errors.

In this paper, we study SPDP for the error-free case such that the multisets A and B contain correct data about the end distances and the interpoint distances. SPDP can be formulated as follows, see the graphical interpretation in Fig. 1.

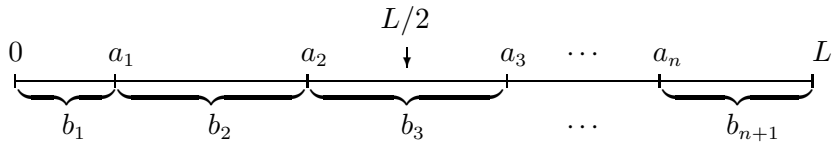

Figure 1: Graphical interpretation of SPDP.

There is a sequence $a = (a_1, a_2, \ldots, a_n)$ of integer numbers and integer number L such that $0 < a_1 < a_2 < \ldots < a_n < L$, where L is the length of the target DNA, and two multisets of positive numbers, $A(a)$ and $B(a)$, associated with it, such that

$$A(a) = \{a_j, L - a_j \mid j = 1, \ldots, n\},$$

$$B(a) = \{b_j \mid j = 1, \ldots, n + 1\},$$

where $b_j = a_j - a_{j-1},\ j = 2, 3, \ldots, n,\ b_1 = a_1,\ b_{n+1} = L - a_n$. Observe that multiset $A(a)$ contains at most two identical pairs $\{a_j, L - a_j\}$. Identical pairs, if they exist, correspond to the restriction sites that are symmetric with respect to the middle of the molecule. Given multisets $A := A(a)$ and $B := B(a)$ for an unknown sequence a, SPDP is to find a sequence of integer numbers $p = (p_1, p_2, \ldots, p_n)$ such that $0 < p_1 < p_2 < \ldots < p_n < L$ and $A(p) = A$ and $B(p) = B$.

SPDP always has at least two solutions, one corresponding to the map of the original DNA and its mirror image (congruent solution). Furthermore, SPDP may have more than two solutions. An example with two non-congruent solutions and, hence, four solutions in total, is given in Fig. 2.

The input of PDP for the above example is the multiset $\{3, 5, 10, 11, 2, 7, 8, 5, 6, 1\}$ of all fragment lengths.

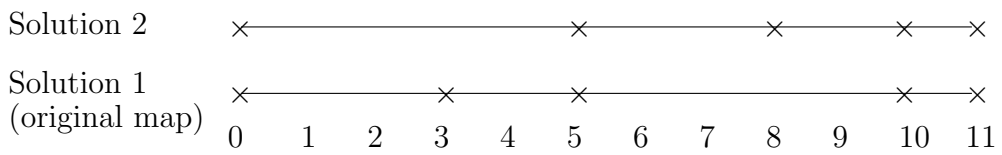

Figure 2: Two non-congruent solutions for SPDP with $n = 3$, $L = 11$, $A = \{1, 3, 5, 6, 8, 10\}$ and $B = \{1, 2, 3, 5\}$.

2.1. SPDP vs. PDP via an example

The advantage of the simplified partial digest over the partial digest, and its potential to solve practical problems of genome mapping can be demonstrated by the following example. In [18], a biochemical experiment was considered, whose purpose was to reconstruct the DNA map of the chromosome of the non-pathogenic bacterium Lactobacillus sakei 23K that occurs naturally on meat and meat products. The experiment included full and partial digestions by enzyme I-*Ceu*I, see Fig. 3.

In this figure, column 1 indicates the lengths of fragments obtained after the full digestion (C_1 to C_7), column 2 indicates the lengths of fragments obtained after the partial digestion (C_a to C_h), and column 3 indicates the molecular mass markers (λ concatemers) with sizes from 48500 to 727000 *base pairs*, which were used to determine the fragment lengths.

After the full digestion, the multiset $B = \{1104, 331, 170, 128, 98, 36, 5\}$ of seven inter-point distances $C_1, \ldots, C_7$ was obtained. Partial digestion resulted in eight more fragment lengths $C_a = 1265$, $C_b = 590$, $C_c = 550$, $C_d = 485$, $C_e = 455$, $C_f = 420$, $C_g = 265$ and $C_h = 164$. The lengths were measured in kilo base pairs with some degree of error. Not all fragment lengths were found. The absence of some fragment lengths prevented existing PDP mathematical tools from being used in the mapping process. In [18], the genome map was determined by a case specific analytical method, which cannot be extended to other cases. However, because of the measurement errors, it was not clear that the constructed DNA map is correct and unique, and how far it is from the correct map. Based on the results of the above experiment and the solution found in [18], we were able to re-construct the missing data required for the simplified partial digest method. The size of the data is considerably smaller than that needed for any other digestion model, which decreases the chance of an incorrect map construction. We obtained the following multiset of end distances: $A = \{1104, 750, 1265, 584, 485, 485, 1372, 1372, 1685, 164, 1815, 36\}$. These distances correspond to the fragment lengths C_1, $C_g + C_d$, C_a, $C_f + C_h$, C_d, C_d, $C_1 + C_3 + C_5$, $C_1 + C_3 + C_5$, $C_a + C_f$, C_h, $C_a + C_c$ and C_6, respectively. Fragment lengths $C_g + C_d$, $C_f + C_h$,

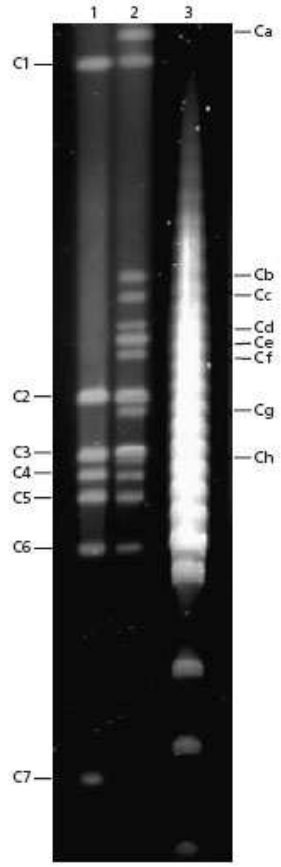

Figure 3: Digestion of Lactobacillus sakei 23K chromosome DNA by enzyme I-*Ceu*I (from [18]).

$C_1 + C_3 + C_5$, $C_a + C_f$ and $C_a + C_c$ were missing in the experiment. We assumed that the relative measurement error r in the experiment was less than 2.5% and applied algorithm Interval-ENUM for SPDP with measurement errors [2] to reconstruct the DNA map. The unique solution $p^* = (5, 36, 100, 131, 172, 325, 1104)$ was obtained, which nearly matches the solution found in [18]: $p^0 = (5, 36, 98, 128, 170, 331, 1104)$. Thus, SPDP and its mathematical solution tools provide extra confirmation of the earlier discovered structure of the bacterium Lactobacillus sakei 23K chromosome.

2.2. Foreword to SPDP complexity and tools

In [6], it was proved that SPDP is NP-hard in the strong sense, and an $O(n \log n)$ time algorithm was developed for the case where $b_j \in \{1, 2\}$, $j = 1, \ldots, n + 1$. Enumerative algorithms for SPDP were proposed in [2, 4, 5]. In [2], a dynamic programming algorithm

with $O(n^{2q})$ running time for SPDP with q distinct interpoint distances was presented. An example of the problem was given, in which the maximum number of mutually non-congruent solutions is equal to $2^{\frac{n+2}{3}-1}$, and a suite of computational experiments was provided with 3710 instances of SPDP generated using real DNA sequences taken from [19] to establish the number of non-congruent solutions. It was equal to 1, 2 and 4 for 3641 (98.14%), 64 (1.725%) and 5 (0.135%) instances, respectively, and it was never equal to 3 or exceeded 4.

In the next section, we consider optimization versions of SPDP, denoted as SPDP-Min and SPDP-Max. The goal of SPDP-Min (SPDP-Max) is to find the orderings of the pairs from the multiset A such that the multiset of the corresponding interpoint distances B^0 (being a result of this ordering) contains the minimum (maximum) number of interpoint distances not from the multiset B (from the multiset B). In other words, the number of mismatchings between the constructed multiset B^0 and the original multiset B is minimized in SPDP-Min, and the number of matchings between the multisets B^0 and B is maximized in SPDP-Max. Note that the exact multiset A is required to be constructed for these problems. Furthermore, for the considered error-free case of the problem, the optimal solution value of SPDP-Min is equal to zero and the optimal solution value of SPDP-Max is equal to $n+1$. It is easy to see that an optimal solution for SPDP-Min or SPDP-Max is a solution for SPDP and vice versa. As the basic version of SPDP (its search formulation) remains NP-hard, it is worth considering its optimization versions and developing approximation algorithms for the latter.

We are interested in the development of polynomial time approximation algorithms for SPDP-Min and SPDP-Max. The *approximation ratio* of an algorithm is defined as the number ρ such that $\max\{\frac{F^0}{F^*}, \frac{F^*}{F^0}\} \leq \rho$ for any problem instance, where F^* is the optimum value (here we assume $F^* > 0$, $F^0 > 0$) and F^0 is the value of the solution delivered by the algorithm. The corresponding algorithm is called the *ρ-approximation algorithm*. It guarantees relative error $\varepsilon = \rho - 1$ in case of minimization, and $\varepsilon = 1 - 1/\rho$ in case of maximization. Clearly, $\rho \geq 1$. Furthermore, the smaller ρ is, the better is the quality of the corresponding solution.

In the next section, we show that SPDP-Min cannot be approximated in pseudo-polynomial time with any finite ρ, unless $\mathcal{P} = \mathcal{NP}$. Therefore, SPDP-Min does not belong to the class APX of optimization problems that allow polynomial time approximation algorithms with approximation ratio bounded by a constant [14]. We also show that SPDP-Max cannot be approximated in pseudo-polynomial time with $\rho < 1 + \frac{1}{n}$, unless $\mathcal{P} = \mathcal{NP}$, but it can

be approximated in $O(n \log n)$ time with $\rho = 1 + \frac{n}{n+2} < 2$. In Section 4, we suggest a graph-theoretic model for SPDP and an algorithm based on this model, which reduces the search space for solutions of SPDP. The aim is to find a path between two specified vertices of a *weighted directed graph (digraph)*. Section 5 presents three graph-theoretic algorithms for finding an approximate solution of SPDP, in which all interpoint distances are from the multiset B but their multiset does not necessarily coincide with the multiset B, and all end distances are from the multiset A but their multiset does not necessarily coincide with the multiset A. A suite of computational experiments, described in Section 6, demonstrates that, for instances of SPDP generated using real DNA sequences taken from [19] with $20 \leq n \leq 50$, and for randomly generated instances with $n = 50$, two of the proposed heuristic algorithms always delivered an exact solution. Section 7 contains a brief summary of the results and suggestions for future research.

3. Worst-case analysis of approximation algorithms for SPDP-Min and SPDP-Max

In this section, we discuss the worst-case efficiency of approximation algorithms for SPDP-Min and SPDP-Max. We start with simple proofs and continue with an $O(n \log n)$ time approximation algorithm and its worst-case analysis.

3.1. Simple approximability proofs

Proposition 1. *There does not exist a pseudo-polynomial time ρ-approximation algorithm with any finite ρ for SPDP-Min, unless $\mathcal{P} = \mathcal{NP}$.*

Proof. Let $F^{\min} = 0$ denote the optimal solution value of SPDP-Min. Assume that there exists a pseudo-polynomial algorithm H for this problem, which delivers a solution with value $F^H \leq \rho \cdot F^{\min}$, where $1 \leq \rho < \infty$. Apply algorithm H for any instance of SPDP. By the definition, $F^H \leq \rho \cdot F^{\min} = 0$. Hence, algorithm H solves SPDP in pseudo-polynomial time, which is impossible (unless $\mathcal{P} = \mathcal{NP}$) because SPDP is NP-hard in the strong sense. □

Proposition 1 implies that SDPD-Min does not belong to the class APX of optimization problems [14].

Proposition 2. *There does not exist a pseudo-polynomial time ρ-approximation algorithm with $\rho < 1 + \frac{1}{n}$ for SPDP-Max, unless $\mathcal{P} = \mathcal{NP}$.*

Proof. Let $F^{\max} = n + 1$ denote the optimal solution value for SPDP-Max. Assume that there exists a pseudo-polynomial algorithm G for this problem, which delivers a solution with value F^G such that $F^G \geq \frac{F^{\max}}{\rho} > \frac{n+1}{1+1/n} = n$. Since F^G is an integer number, we deduce $F^G \geq n + 1$. Hence, algorithm G solves SPDP in pseudo-polynomial time, which is impossible, unless $\mathcal{P} = \mathcal{NP}$. □

Proposition 3. *SPDP-Max can be approximated in $O(2^c n)$ time with the worst-case performance guarantee $\rho = \frac{n+1}{c+1}$, where c is a given positive integer number.*

Proof. We prove this proposition by describing an $O(2^c n)$ time approximation algorithm with the indicated value of ρ. The algorithm can be outlined as follows.

In $O(cn)$ time, select $c+1$ smallest values a_j in the multiset A and construct 2^{c+1} partial solutions to SPDP by assigning each selected value a_j as the point a_j (to the left hand part of the interval $[0, L]$) or as the point $L - a_j$ (to the right hand part of the interval $[0, L]$). One of these partial solutions can be extended to be a complete feasible solution of SPDP. Such a partial solution has at least $c + 1$ interpoint distances from the original multiset B. By assigning the remaining values in the multiset A arbitrarily to the left hand part or the right hand part of the interval $[0, L]$, in $O(n)$ time extend every constructed partial solution to a complete solution of SPDP. Such a complete solution is feasible with regard to the original multiset A but it can be infeasible with regard to the original multiset B. Among 2^{c+1} constructed complete solutions, select the one with the largest number of the interpoint distances from the original multiset B. It is clear that this number is at least $c + 1$. Hence, we have found in $O(2^c n)$ time a solution to SPDP-Max with the objective value F^0 such that $F^{\max}/F^0 \leq \frac{n+1}{c+1}$. □

3.2. An $O(n \log n)$ time approximation algorithm

We now describe an $O(n \log n)$ time $(1 + \frac{n}{n+2})$-approximation algorithm, denoted as SWITCH, for SPDP-Max. This algorithm is a modification of the exact $O(n \log n)$ time algorithm in [6], which was developed for the case of interpoint distances $b_j \in \{1, 2\}$. Observe that $\{a_j, L - a_j\} \subset A$ implies that point a_j or symmetric point $L - a_j$ is present in each optimal solution of SPDP-Max. Thus, there are two potential symmetric locations (left and right ones) of a point associated with each pair $\{a_j, L - a_j\} \subset A$, $j = 1, \ldots, n$. In the algorithm SWITCH, these potential locations are denoted by labels "∘", and the locations chosen by the algorithm are denoted by labels "×". Assume that $a_1 \leq \cdots \leq a_{2n}$. Algorithm

SWITCH considers n smallest end distances $a_1, \ldots, a_n$ in this order and constructs a sequence of labels $S = (l_0 = \times, l_1, \ldots, l_n, r_n, \ldots, r_1, r_0 = \times)$, where labels l_j and r_j correspond to the left and right potential locations, respectively, of the point associated with the pair of end distances $\{a_j, L - a_j\} \subset A$. Labels $\times$ in the beginning and the end of the sequence correspond to the points 0 and L, respectively. In the sequel, we will not distinguish between labels $\times$ and the corresponding points in the interval $[0, L]$.

Let $D(A)$ be a subset of the multiset A such that $a_j \in D(A)$ if and only if number a_j, $j \leq n$, appears twice in A, i.e., if symmetric points a_j and $L - a_j$ (or one point $a_j = L/2$) are present in any optimal solution of SPDP-Max. Algorithm SWITCH initializes sequence S such that $l_j = r_j = \times$ if $a_j \in D(A)$ and $l_j = r_j = \circ$ otherwise. Thus, it determines the obligatory symmetric points in the interval $[0, L]$. Then, it iteratively modifies the initial sequence by replacing one of the two labels $l_j = \circ$ and $r_j = \circ$ by label $\times$ for $j = 1, \ldots, n$, i.e., it determines asymmetric points in the interval $[0, L]$. Consider iteration j. Let T_l denote the distance between point 0 and the point $\times$ to the left of l_j closest to it, and let T_r denote the distance between point L and the point $\times$ to the right of r_j closest to it. If $a_j \in D(A)$, then $l_j = r_j = \times$. In this case, if any of the generated interpoint distances $a_j - T_l$ and $a_j - T_r$ is from B, then the algorithm removes it from the multiset B in order not to use it twice in the solution under construction and passes to iteration $j + 1$. Assume $a_j \notin D(A)$. Notice that $T_l = a_{j-1}$ or $T_r = a_{j-1}$ or $T_l = T_r = a_{j-1}$. Let $T_l = a_{j-1}$, which implies $l_{j-1} = \times$. If $a_j - T_l \in B$, then the algorithm re-sets $l_j = \times$ and removes distance $a_j - T_l$ from the multiset B. If $a_j - T_l \notin B$, then it re-sets $r_j = \times$, and if $a_j - T_r \in B$, removes distance $a_j - T_r$ from the multiset B. The case $T_r = a_{j-1}$ is handled similarly. A formal description of the algorithm is given below.

Algorithm SWITCH

Step 1. Represent multiset B as a self-balancing binary search tree of numbers $b_j \in B$. Sort numbers in the multiset A in non-increasing order such that $a_1 \leq a_2 \leq \ldots \leq a_{2n}$. Calculate the set $D(A)$. Initialize sequence $S = (\times, l_1, \ldots, l_n, r_n, \ldots, r_1, \times)$ such that $l_j = r_j = \times$ if $a_j \in D(A)$ and $l_j = r_j = \circ$ otherwise, $j = 1, \ldots, n$. Set $T_l = T_r = 0$, $Side =$ `Left` and $j = 1$.

Step 2. If $a_j \in D(A)$, perform the following computations. If $a_j - T_l \in B$, then remove $a_j - T_l$ from B. The removal can be done in $O(\log n)$ time. If $a_j - T_r \in B$, then remove $a_j - T_r$ from B. Re-set $T_l = T_r = a_j$ and go to Step 3. If $a_j \notin D(A)$, perform the following

computations.

If $Side = \texttt{Left}$ and $a_j - T_l \in B$, then re-set $l_j := \times$ and remove $a_j - T_l$ from B. Re-set $T_l := a_j$. Do not re-set $Side$. Go to Step 3.

If $Side = \texttt{Left}$ and $a_j - T_l \notin B$, then verify whether $a_j - T_r \in B$. If $a_j - T_r \in B$, then remove $a_j - T_r$ from B. Irrespectively of the latter inclusion, re-set $r_j := \times$, $Side := \texttt{Right}$ and $T_r := a_j$. Go to Step 3.

If $Side = \texttt{Right}$ and $a_j - T_r \in B$, then re-set $r_j := \times$ and remove $a_j - T_r$ from B. Re-set $T_r := a_j$. Do not re-set $Side$. Go to Step 3.

If $Side = \texttt{Right}$ and $a_j - T_r \notin B$, then verify whether $a_j - T_l \in B$. If $a_j - T_l \in B$, then remove $a_j - T_l$ from B. Irrespectively of the latter inclusion, re-set $l_j := \times$, $Side := \texttt{Left}$ and $T_l := a_j$.

Step 3. If $L/2 \notin A$ and $j = n$ or $L/2 \in A$ and $j = n - 1$, then output sequence S and stop. Otherwise, re-set $j := j + 1$ and go to Step 2.

Since the number of iterations of Step 2 does not exceed n, the running time of algorithm SWITCH is $O(n \log n)$.

An application of algorithm SWITCH for the example in Fig. 2 is illustrated in Fig. 4.

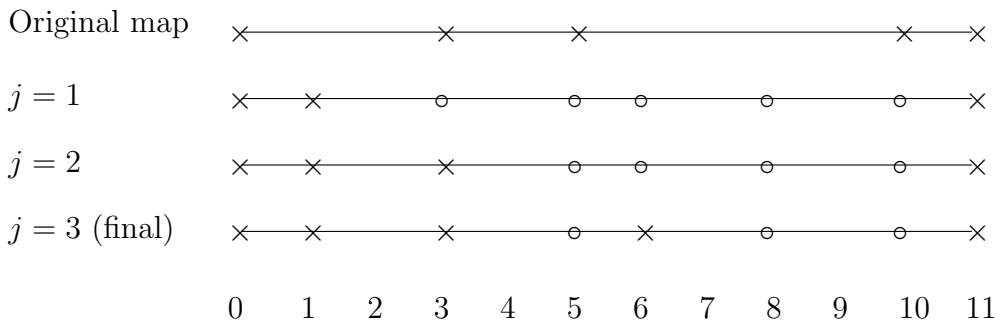

Figure 4: Application of SWITCH for SPDP-Max with $n = 3$, $L = 11$, $A = \{1, 3, 5, 6, 8, 10\}$ and $B = \{1, 2, 3, 5\}$.

For the considered example, algorithm SWITCH finds an optimal solution, which is not the original DNA map.

Theorem 1. *Algorithm SWITCH is a* $(1 + \frac{n}{n+2})$*-approximation algorithm for SPDP-Max.*

Proof. We will consider the case $L/2 \notin A$. The case $L/2 \in A$ can be handled similarly. Let sequence of labels $S^* := (\times, l_1^*, \ldots, l_n^*, r_n^*, \ldots, r_1^*, \times)$ represent an arbitrary optimal solution

to SPDP-Max or its mirror image, which is also optimal and in which (symmetric) labels l_j^* and r_j^* are interchanged for each $j = 1, \ldots, n$.

Consider an arbitrary iteration j of algorithm SWITCH. Denote by T_l^* the distance between point 0 and the point $\times$ to the left of l_j^* closest to it in S^*. Denote by T_r^* the distance between point L and and the point $\times$ to the right of r_j^* closest to it in S^*. Consider the following two cases with regard to the values T_l, T_r determined with regard to the sequence S constructed by the algorithm, and T_l^*, T_r^*:

Case C1: $T_l = T_l^*$ and $T_r = T_r^*$.

Case C2: $T_l = T_l^*$ and $T_r \neq T_r^*$, or $T_r = T_r^*$ and $T_l \neq T_l^*$.

Since at least one of the two symmetric potential locations of each point $\times$ is occupied in S^* and S, and since we do not distinguish between the considered optimal sequence and its mirror image, it is easy to notice that, in each iteration j of algorithm SWITCH, case C1 or case C2 takes place. Furthermore, in either case, we have $l_{j-1} = l_{j-1}^* = \times$ or $r_{j-1} = r_{j-1}^* = \times$. We will analyze transitions between these cases in the course of algorithm SWITCH with regard to the number of the generated interpoint distances from B and not from B in S. Consider $j = 1$. We have $S = (\times, \ldots, \times)$ and $S^* = (\times, \ldots, \times)$. Thus, case C1 takes place. Now consider iteration $j \geq 2$ and assume that case C1 or case C2 takes place in iteration $j - 1$.

If case C1 takes place in iteration $j - 1$, then only the following three cases can happen in iteration j of algorithm SWITCH:

C1.1 ($C1 \rightarrow C1$ transition): $a_j \in D(A)$, see Fig. 5 for an illustration. Case C1 is

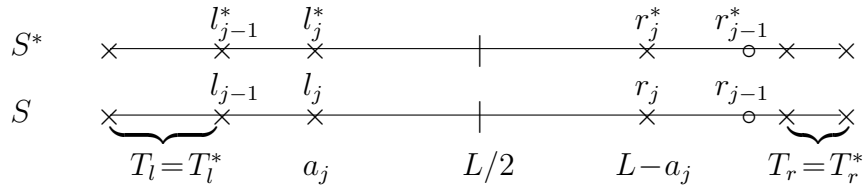

Figure 5: Case C1.1.

maintained for the pair of sequences (S^*, S) in iteration j. The two generated interpoint distances for S are from the multiset B.

C1.2 ($C1 \rightarrow C1$ transition): $a_j \notin D(A)$ and a) $(l_j^*, r_j^*) = (l_j, r_j) \neq (\times, \times)$, or b) $l_{j-1}^* = r_{j-1}^* = l_{j-1} = r_{j-1} = \times$, see Fig. 6 and 7.

Under condition a), case C1 is maintained for the pair of sequences (S^*, S). Under condition b), case C1 is maintained for the pair of sequences (S^*, S) or $(Mirror(S^*), S)$,

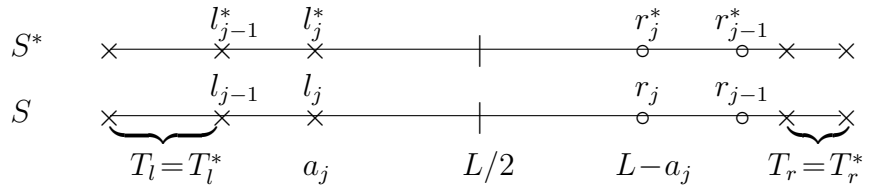

Figure 6: Case C1.2,a).

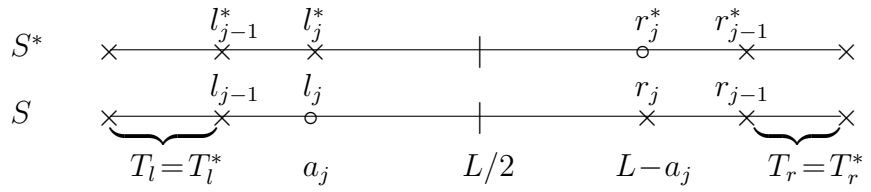

Figure 7: Case C1.2,b).

where $Mirror(S^*)$ is the mirror image of S^*. The only generated interpoint distance for S is from B.

C1.3 ($C1 \rightarrow C2$ transition): $a_j \notin D(A)$ and $(l^*_j, r^*_j) \neq (l_j, r_j)$, see Fig. 8. Case C2 appears

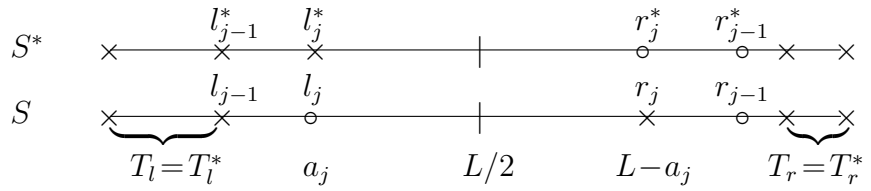

Figure 8: Case C1.3.

for the pair of sequences $(Mirror(S^*), S)$ in iteration j. The only generated interpoint distance might be or might be not from B. If interpoint distance $a_j - T^*_l \in B$ or $a_j - T^*_r \in B$ is not generated in S, then all copies of this distance should have been generated earlier. As we will see below, this generation can only happen in $C2 \rightarrow C1$ transition.

If case C2 takes place in iteration $j-1$, then again there are only the following three cases to consider in iteration j of algorithm SWITCH:

C2.1 ($C2 \rightarrow C1$ transition): $a_j \in D(A)$, see Fig. 9. In this case, $(l^*_j, r^*_j) = (l_j, r_j) = (\times, \times)$. Case C1 appears for the pair of sequences (S^*, S) in iteration j. One of the two generated interpoint distances for S is from the multiset B and the other distance may or may not be from B.

C2.2 ($C2 \rightarrow C2$ transition): $a_j \notin D(A)$ and a) $l_{j-1} = l^*_{j-1} = \times$, $(l^*_j, r^*_j) = (l_j, r_j) = (\times, \circ)$,

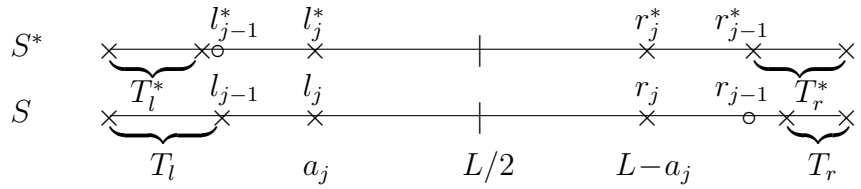

Figure 9: Case C2.1.

or b) $r_{j-1} = r^*_{j-1} = \times$, $(l^*_j, r^*_j) = (l_j, r_j) = (\circ, \times)$, see Fig 10 and 11. Case C2 is maintained

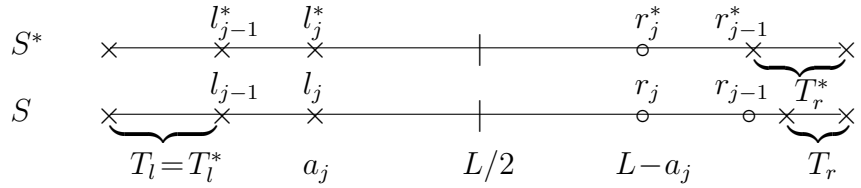

Figure 10: Case C2.2,a).

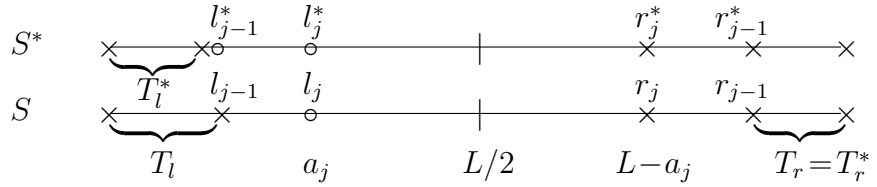

Figure 11: Case C2.2,b).

for the pair of sequences (S^*, S) in iteration j. The only generated interpoint distance for S is from B.

C2.3 ($C2 \to C1$ transition): $a_j \notin D(A)$ and a) $l_{j-1} = l^*_{j-1} = \times$, $(l^*_j, r^*_j) = (l_j, r_j) = (\circ, \times)$, or b) $r_{j-1} = r^*_{j-1} = \times$, $(l^*_j, r^*_j) = (l_j, r_j) = (\times, \circ)$, see Fig 12 and 13.

Case C1 appears for the pair of sequences (S^*, S) in iteration j. The only generated interpoint distance may or may not be from B.

Thus, for the transitions $C1 \to C1$ and $C2 \to C2$, only interpoint distances from B can be generated. For the transition $C1 \to C2$, the worst case is C1.3, in which the only generated interpoint distance is not from B, and one of the earlier transitions is $C2 \to C1$ which generates interpoint distance from B. Furthermore, each such transition $C2 \to C1$ implies at most one transition $C1 \to C2$ which generates interpoint distance not from B. For the transition $C2 \to C1$, the worst case is C2.1, in which the only generated interpoint distance is not from B. The first transition is obligatorily $C1 \to C1$ because $l^*_0 = r^*_0 = l_0 = r_0 = \times$.

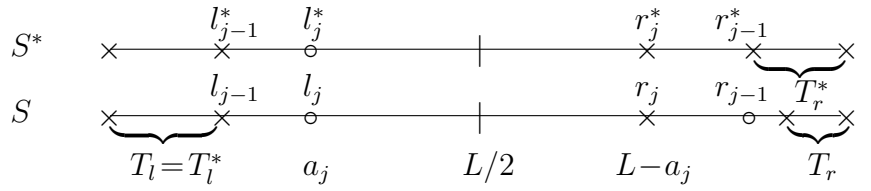

Figure 12: Case C2.3,a).

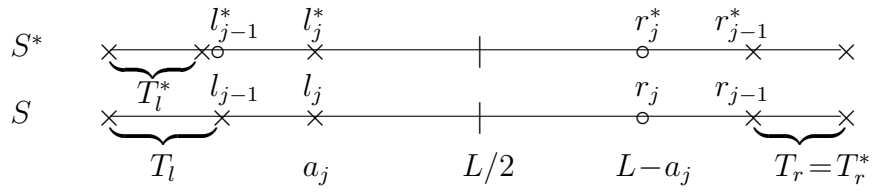

Figure 13: Case C2.3,b).

We deduce that the worst chain of transitions between cases $C1$ and $C2$ in the course of algorithm SWITCH is $C1 \to C1 \to (C2 \to C1) \to (C2 \to C1) \cdots (C2 \to C1)$, where the first two transitions $C1 \to C1 \to C2$ generate two interpoint distances from B, and the last transition $C2 \to C1$ generates one interpoint distance not from B. Let k be the number of the remaining transitions in the above chain. They have the property that each transition, which generates an interpoint distance not from B, has a counterpart transition, which generates an interpoint distance from B. Then, in the worst case, algorithm SWITCH will generate at least $k/2+2$ interpoint distances from B among all $k+3 = n+1$ generated interpoint distances. Thus, its approximation ratio is $\rho = \frac{k+3}{k/2+2} = 1 + \frac{n}{n+2}$. □

Since $\rho < 2$, Theorem 3 implies that SPDP-Max belongs to the class APX of optimization problems.

An open research issue is whether there exists a polynomial time ρ-approximation algorithm for SPDP-Max such that $1 + \frac{1}{n} \leq \rho < 1 + \frac{n}{n+2}$. The existence of a *Polynomial Time Approximation Scheme (PTAS)* (i.e., an approximation algorithm that delivers a solution with any given relative error ε in time polynomial in n) is an open question as well. Notice that the strong NP-hardness of SPDP and the fact that the optimum value of SPDP-Max is a polynomial of n imply that SPDP-Max cannot have a *Fully Polynomial Time Approximation Scheme (FPTAS)*, unless $\mathcal{P} = \mathcal{NP}$. An FPTAS differs from a PTAS in that its running time must be polynomial in n and $1/\varepsilon$.

4. A graph-theoretic model for SPDP

In this section, we present a graph-theoretic model for SPDP and describe its size reduction.

4.1. Notations and definitions

Given an instance of SPDP, let us construct a weighted digraph $G = (V, U)$, where V is a set of vertices and $U \subset \{(i, j)|i \in V, j \in V\}$ is a set of arcs. An example of digraph G is given in Fig. 14.

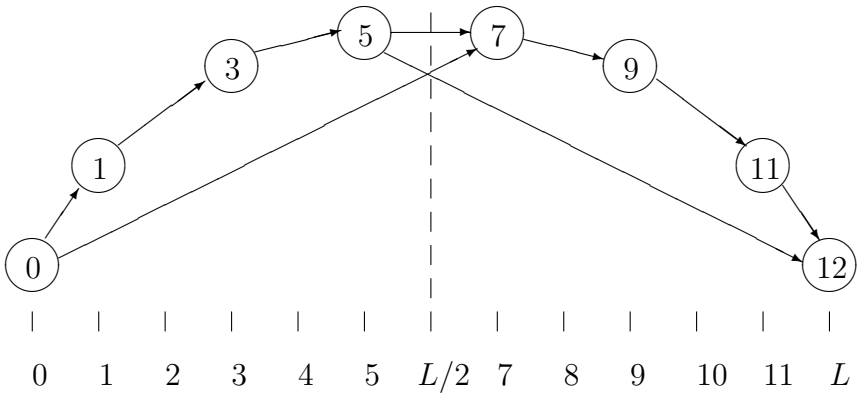

Figure 14: Digraph $G = (V, U)$ for SPDP with $L = 12$, $n = 4$, $A = \{1, 11, 3, 9, 5, 7, 5, 7\}$ and $B = \{1, 2, 2, 2, 7\}$.

The set of vertices V comprises two specified vertices 0 and L, which correspond to the two ends 0 and L, respectively, of the interval $[0, L]$, and two vertices a_j and $L - a_j$, which correspond to the pair $\{a_j, L - a_j\} \in A$ for each $j = 1, \ldots, n$. We do not make copies of the vertices a_j and $L - a_j$ if there are two identical pairs $\{a_j, L - a_j\} \in A$. Thus, set V contains at most $2n + 2$ vertices. It can be computed in $O(n)$ time. We call vertices $a_j \in V$ and $L - a_j \in V$ *symmetric*. In order to avoid using multilevel indices, we shall skip indices in the vertex notations. Introduce a set of vertices

$$V_{Double} := \{i, L - i| \text{ two copies of } i \text{ (and } L - i) \text{ are present in the multiset } A\}.$$

We define the set of arcs U as follows. There exists an arc $(i, j) \in U$ if and only if $i \in V$, $j \in V$, $i < j$, and one of the following two conditions is satisfied: a) $j \neq L - i$, $j - i \in B$, or b) $j = L - i$, $i \in V_{Double}$. Thus, there is no arc $(i, L - i)$ if $i \notin V_{Double}$. In Fig. 14, there

are no arcs $(1, 11)$ and $(3, 9)$ because $V_{Double} = \{5, 7\}$. We determine the weight of each arc $(i, j) \in U$ as $d_{i,j} = j - i$. Given a set of arcs X, denote the multiset of all its arc weights as $D(X)$. Thus,

$$D(U) = \{d_{i,j} | (i, j) \in U\}.$$

Since $|U| \leq |V|^2$, $|V| \leq 2n + 2$ and $|B| = n + 1$, the set U and the multiset $D(U)$ can be computed in $O(n^3)$ time. We define a path P between vertices 0 and L in digraph G as a sequence of vertices

$$P = (0, i_1, i_2, \ldots, i_k, L)$$

such that $(0, i_1) \in U$, $(i_k, L) \in U$ and $(i_r, i_{r+1}) \in U$ for $r = 1, \ldots, k - 1$. We denote by $\{P\}$ the set of vertices in path P. With the above path, we associate the multiset of its arc weights,

$$D(P) = \{d_{0,i_1}, d_{i_1,i_2}, \ldots, d_{i_{k-1},i_k}, d_{i_k,L}\},$$

and the multiset of the distances between *internal* vertices from the set $\{P\}\backslash\{0, L\}$ and vertices 0 and L,

$$A(P) = \{i_j, L - i_j | j = 1, \ldots, k\}.$$

We call a path P *feasible* if $A(P) = A$ and $D(P) = B$. It is easy to see that each feasible path $P = (0, i_1, i_2, \ldots, i_n, L)$ corresponds to a feasible solution of SPDP, in which i_r is the distance between the end of the molecule corresponding to the point 0 and the r-th restriction site, $r = 1, \ldots, n$. Thus, SPDP reduces to finding feasible paths in digraph G.

It is convenient to use the following definition of a feasible path. A path $P = (0, i_1, i_2, \ldots, i_k, L)$ is feasible if and only if it satisfies properties (a)-(d) given below:

(a) $k = n$;

(b) $V_{Double} \cup \{0, L\} \subseteq \{P\}$;

(c) $D(P) = B$;

(d) for any $i \in V\backslash(V_{Double} \cup \{0, L\})$ either $i \in \{P\}$ or $L - i \in \{P\}$.

Property (d) says that a feasible path should include exactly one vertex of each pair of symmetric vertices $\{a_j, L - a_j\} \subseteq V\backslash(V_{Double} \cup \{0, L\})$.

A similar graph can be used to represent the results of tandem mass spectrometry experiments for the de novo peptide sequencing problem, see [9]. Although the graphs in [9] and

in this paper have the same structure, problems formulated on these graphs are different. Specifically, a path in our problem is feasible if and only if the multiset of arc weights is equal to the multiset B, while a path in the problem in [9] is feasible if arc weights are from the multiset B. Another difference is that we allow some symmetric vertices to be both present in a feasible path, while the model in [9] allows at most one of them to be present in a feasible path. The approach in [9] can be modified to be a heuristic for SPDP. However, the required modifications will lead to the approaches presented in this paper.

4.2. Reducing graph G

Observe that if path $P = (0, i_1, i_2, \ldots, i_n, L)$ is feasible, then its *symmetric path* $P' = (0, L - i_n, L - i_{n-1}, \ldots, L - i_1, L)$ is feasible as well. In order to avoid constructing feasible symmetric paths, it is sufficient to remove from the set $V \backslash (V_{Double} \cup \{0, L\})$ an arbitrary vertex i, thus leaving its symmetric vertex $L - i$ as a member. Let $i_{\min} = \min\{i | i \in V \backslash (V_{Double} \cup \{0, L\})\}$. We remove vertex $L - i_{\min}$ from the set V and all its *incident* arcs of the form $(i, L - i_{\min})$ or $(L - i_{\min}, j)$ from the set U. In the sequel, the removal of any vertex will automatically imply the removal of all its incident arcs. In Fig. 14, vertex 11 and arcs $(9, 11)$ and $(11, 12)$ will be removed. Now, due to property (d), every feasible path should include vertex $i_{\min}$.

Along with the sets V and U, we shall use the *adjacency list representation* of digraph G [13]. More specifically, with each vertex $i \in V$, we shall associate the set of immediate successors, $After(i) = \{j | (i, j) \in U\}$, the set of immediate predecessors, $Before(i) = \{j | (j, i) \in U\}$, and their cardinalities $Card_A(i) = |After(i)|$ and $Card_B(i) = |Before(i)|$. All the sets $After(i)$ and $Before(i)$ and numbers $Card_A(i)$ and $Card_B(i)$, $i \in V$, can be computed in $O(n^3)$ time.

In digraph G, there can exist *redundant* vertices and arcs that do not belong to any path connecting vertices 0 and L, and *preventing* arcs that, if included into a path, prevent vertices of the set $V_{Double} \cup \{i_{\min}\}$ from being included into the same path. An arc (i, j) is a preventing one if there exists vertex $i^D \in V_{Double} \cup \{i_{\min}\}$ such that $i < i^D < j$. Notice that redundant vertices and arcs may appear after the vertex $L - i_{\min}$ has been removed from G.

We now describe an algorithm, called REMOVE, which removes redundant vertices and arcs and preventing arcs from the digraph G. The algorithm consists of Steps 1-4. Preventing arcs are removed in Step 1. Redundant vertices and arcs that do not connect to vertex L or to vertex 0 are taken away in Steps 3 and 4.

Algorithm REMOVE

Step 1. Let $V_{Double} \cup \{i_{\min}\} = \{i_r^D | r = 1, \ldots, q\}$, where $0 < i_1^D < i_2^D < \ldots < i_q^D < L$. For each $(i, j) \in U$, perform the following computations.

If $\min\{i_r^D | i_r^D > i, r = 1, \ldots, q\} < j$, remove preventing arc (i, j) from the set U, vertex i from the set $Before(j)$ and vertex j from the set $After(i)$. Update the cardinalities of these sets $Card_A(i)$ and $Card_B(j)$ accordingly.

Step 2. Let $V = \{i_j | j = 1, \ldots, k\}$, where $i_1 = 0$, $i_2 < i_3 < \ldots < i_{k-1}$, $i_k = L$. For each $i_j \in V, j = 2, \ldots, k$, perform the following computations. If $Card_B(i_j) = 0$ then for all $t \in After(i_j)$, remove i_j from $Before(t)$ and re-set $Card_B(t) := Card_B(t) - 1$.

Step 3. For each $i_j \in V, j = k-1, \ldots, 1$, perform the following computations. If $Card_A(i_j) = 0$ then for all $t \in Before(i_j)$, remove i_j from $After(t)$ and re-set $Card_A(t) := Card_A(t) - 1$.

Step 4. For each $i_j \in V, j = 2, \ldots, k-1$, remove i_j if $Card_A(i_j) = 0$ and $Card_B(i_j) = 0$.

The running time of algorithm REMOVE is $O(n^2 \log |V_{Double}|)$. This algorithm reduces the search space for a (feasible) solution of SPDP.

From now on, we assume that there is no preventing arc in digraph G, and that every vertex and arc belongs to at least one path from 0 to L. It follows that every path from 0 to L contains all vertices of the set $V_{Double} \cup \{i_{\min}\}$, i.e., property (b) is satisfied for all paths from 0 to L and all paths are mutually asymmetric.

5. Finding paths in digraph *G*

This section describes three approximation algorithms for SPDP-Max. The algorithms search for a path in digraph G, which is close to a feasible path. They are presented in increasing order of their complexity. Assume that the set of vertices is $V = \{i_1, \ldots, i_v\}$, where $i_1 = 0$, $i_v = L$ and $i_j < i_{j+1}$, $j = 1, \ldots, v-1$.

5.1. A single path algorithm with no dominance

Our first algorithm, called PATH, constructs a path from vertex 0 to vertex L in digraph G, which satisfies property (a), i.e., it contains exactly $n + 1$ arcs. Property (b) is automatically satisfied for such a path. The algorithm does not guarantee the feasibility of this path because properties (c) and (d) can be violated. However, if there is exactly one path

in digraph G satisfying properties (a) and (b), like in Fig. 14 with vertex 11 removed (to avoid symmetric paths), then algorithm PATH solves the corresponding SPDP.

Algorithm PATH is a dynamic programming algorithm that constructs paths from vertex 0 to vertex L by scanning vertices of the set V in increasing order. With a path from vertex 0 to vertex i_j, which we refer to as $(0, i_j)$-path, we consider the number of arcs in this path, denoted as k. We require $k \leq n+1$. If there are several $(0, i_j)$-paths with the same number of arcs k, only one of them is chosen for further expansion. It is clear that if there is a $(0, i_j)$-path with k arcs, which can be extended to a path including $n+1$ arcs, then any $(0, i_j)$-path with k arcs can be extended to such a path. Therefore, we associate a single *dominating* $(0, i_j)$-path with each number k. There is no specific rule for choosing dominating paths in algorithm PATH.

Let $Pred(i_j, k)$ denote an immediate predecessor of vertex i_j in the dominating $(0, i_j)$-path with k arcs. We set $Pred(i_1, 0) = 0$ and $Pred(i_j, k) = -1$ if there is no $(0, i_j)$-path with k arcs. Algorithm PATH calculates values $Pred(i_j, k)$ and outputs a $(0, L)$-path with $n+1$ arcs.

Algorithm PATH

Step 1. Set $Pred(i_1, 0) = 0$ and $Pred(i_j, k) = -1$, $(i_j, k) \neq (i_1, 0)$, $j = 1, \ldots, v$, $k = 0, 1, \ldots, j-1$. Re-set $j := 1$.

Step 2. For all $i_t \in Before(i_{j+1})$ and $k \in \{0, 1, \ldots, t-1\}$ such that $Pred(i_t, k) \neq -1$, re-set $Pred(i_{j+1}, k+1) := i_t$.

If $j + 1 = v$, go to Step 3. If $j + 1 \leq v - 1$, re-set $j := j + 1$ and repeat Step 2.

Step 3. Construct a $(0, L)$-path $P^0 = (i_1^0, i_2^0, \ldots, i_{n+2}^0)$, where $i_1^0 = 0$, $i_{n+2}^0 = L$ by the following recursive computations: $i_j^0 = Pred(i_{j+1}^0, j)$, $j = n+1, n, \ldots, 1$.

Algorithm PATH runs in $O(n^3)$ time. The path P^0 delivered by this algorithm can be checked for feasibility in $O(n)$ time.

5.2. A single path algorithm with dominance

Our second algorithm, denoted by PATH-F, employs a specific rule for selecting dominating $(0, i_j)$-paths. It can be outlined as follows. With each $(0, i_j)$-path P with k arcs, we associate a digraph, denoted as $G(P)$, which is obtained from the original digraph by eliminating vertices of the set $\{P\} \backslash \{i_j\}$ and vertices $L - i$ such that $i \in \{P\}$ and $i \notin V_{Double} \cup \{0\}$.

A feasible path extended from P cannot include, in its extension, the eliminated vertices. Then we apply algorithm PATH to verify whether, in digraph $G(P)$, there exists a path (i_j, L) including $n+1-k$ arcs. If it does not exist, then path P cannot be extended to a feasible path. We exclude it from further consideration. Let us assume that the above mentioned path with $n+1-k$ arcs exist. Denote it as PE. We verify whether the complete path (P, PE) is feasible. If it is feasible, algorithm PATH-F outputs (P, PE) as a feasible solution and stops. If it is not feasible, it is not known whether path P can be extended to a feasible path or not. We include P in the set of candidates for a dominating $(0, i_j)$-path with k arcs, denoted as $S(i_j, k)$, and store its extension PE with it. After all $(0, i_j)$-paths with k arcs have been considered, a dominating path among them, denoted as $P^*_{(i_j,k)}$, is selected from the set $S(i_j, k)$ such that its extension, denoted as PE^*, has the minimum number of occurrences of symmetric vertices not from the set $V_{Double} \cup \{L\}$. We store complete path $P^0_{(i_j,k)} := (P^*_{(i_j,k)}, PE^*)$ as a *candidate solution*. After the set $S(i_v, n+1)$ has been constructed, algorithm PATH-F outputs a candidate solution with the maximum number of arcs having weights from the set B among all available candidate solutions. For initialization, we set $S(i_1, 0) = \{P^0\}$, where P^0 is the output of algorithm PATH.

Algorithm PATH-F

Step 1. Set $P^*_{(i_1,0)} = (i_1)$, $G(P^*_{(i_1,0)}) = G$, $P^0_{(i_1,0)} = P^0$ and $P^*_{(i_j,k)} = (-1)$, $(i_j, k) \neq (i_1, 0)$, $j = 1, \ldots, v$, $k = 0, 1, \ldots, j-1$. Set $S(i_j, k) = \phi$, $j = 2, \ldots, v$, $k = 0, 1, \ldots, j-1$. Re-set $j := 1$.

Step 2. For all $i_t \in Before(i_{j+1})$ and $k \in \{0, 1, \ldots, t-1\}$ such that $P^*_{(i_t,k)} \neq (-1)$, re-set $S(i_{j+1}, k+1) := S(i_{j+1}, k+1) \cup P'$, where $P' = (P^*_{(i_t,k)}, i_{j+1})$. If $i_{j+1} \in V_{double}$, then set $G(P') = G(P^*_{(i_t,k)})$. Otherwise, construct digraph $G(P')$ by eliminating from digraph $G(P^*_{(i_t,k)})$ vertex $L - i_{j+1}$.

For $k \in \{0, 1, \ldots, j-1\}$ such that $S(i_{j+1}, k+1) \neq \phi$, perform the following computations.

For each $P \in S(i_{j+1}, k+1)$, find in digraph $G(P)$ a (i_{j+1}, L)-path with $n+1-k$ arcs by applying algorithm PATH. If such an extension path was not found, then eliminate P from $S(i_{j+1}, k+1)$. If such an extension path, denoted as PE, was found, then check path (P, PE) for feasibility. If it is feasible, output this path and stop. If it is not feasible, do not eliminate path P from $S(i_{j+1}, k+1)$. Pass to consideration of another path from $S(i_{j+1}, k+1)$ as described above. After considering all paths from $S(i_{j+1}, k+1)$, select a path with the minimum number of occurrences of symmetric vertices not from $V_{Double} \cup \{L\}$

in its extension as the dominating path $P^*_{(i_{j+1},k+1)}$. Store path $P^0_{(i_{j+1},k+1)} := (P^*_{(i_{j+1},k+1)}, PE^*)$ as a candidate solution, where PE^* is the extension of $P^*_{(i_{j+1},k+1)}$.

If $j+1 = v$, go to Step 3. If $j+1 \le v-1$, re-set $j := j+1$ and repeat Step 2.

Step 3. Select from $S(i_v, n+1)$ a path with the maximum number of arcs having weights from the set B, output it and stop.

Algorithm PATH-F runs in $O(n^6)$ time because $|S(i_j,k)| \le |Before(i_j)|$, $k = 1,\dots,n$, $\sum_{j=2}^{v} |Before(i_j)| = |U| \le O(n^2)$, and for each $P \in S(i_j,k)$, $k = 1,\dots,j-1$, algorithm PATH with running time $O(n^3)$ is applied once.

5.3. A multiple paths algorithm with dominance

Our third algorithm, denoted as PATH(x) where x is a positive integer number, is a modification of algorithm PATH. It constructs at most x paths, where x is the number of $(0,L)$-paths $P^{(h)} = (i_1^{(h)},\dots,i_{n+2}^{(h)})$, $h = 1,\dots,x$, satisfying properties (a) and (b). It then selects the best path among them. In the beginning, sets $SetPred(i_j,k)$, $j = 1,\dots,v$, $k = 0,1,\dots,j-1$, are calculated, where $SetPred(i_j,k)$ is a set of immediate predecessors of vertex i_j in all $(0,i_j)$-paths with k arcs. Path $P^{(1)}$ is constructed backwards by setting $i_{n+2}^{(1)} = L$ and then by selecting $i_j^{(1)} \in SetPred(i_{j+1}^{(1)}, j)$, $j = n+1, n, \dots, 1$. Path $P^{(h)}$, $h \ge 2$, is obtained from path $P^{(h-1)}$ by replacing at least one vertex in the head of $P^{(h-1)}$ by an appropriate new vertex. For each path $P^{(h)}$, we have $i_{n+2}^{(h)} = L$ and $i_j^{(h)} \in SetPred(i_{j+1}^{(h)}, j)$, $j = n+1, n, \dots, 1$. Vertices assigned to the paths are labeled to avoid constructing identical paths. In the beginning, all vertices are unlabeled. We denote $s_{i_j,k} = |SetPred(i_j,k)|$.

Algorithm PATH(x)

Step 1. Calculate $s_{i_1,0} = 1$ and $s_{i_j,k} = 0$, $(i_j,k) \ne (i_1,0)$, $j = 1,\dots,v$, $k = 0,1,\dots,j-1$. Re-set $j := 1$.

Step 2. For all $i_t \in Before(i_{j+1})$ and $k = 0,1,\dots,t-1$ such that $s_{i_t,k} \ne 0$, add i_t to $SetPred(i_{j+1}, k+1)$ and re-set $s_{i_{j+1},k+1} := s_{i_{j+1},k+1} + 1$.

If $j+1 = v$, go to Step 3. If $j+1 \le v-1$, re-set $j := j+1$ and repeat Step 2.

Step 3. Construct $(0,L)$-paths $P^{(h)} = (i_1^{(h)}, i_2^{(h)}, \dots, i_{n+2}^{(h)})$, where $i_1^{(h)} = 0$, $i_{n+2}^{(h)} = L$, $1 \le h \le x$, by the recursive computations in Step 4. Initialize $h := 1$ and $j := n+1$.

Step 4. If all vertices in the set $SetPred(i_{j+1}^{(h)}, j)$ are labeled and $j = n + 1$, then all $(0, L)$-paths satisfying properties (a) and (b) have been constructed. Select a feasible path among them, output it and stop.

If all vertices in the set $SetPred(i_{j+1}^{(h)}, j)$ are labeled and $j < n + 1$, then remove labels of all vertices in the set $SetPred(i_{j+1}^{(h)}, j)$, re-set $j := j + 1$ and repeat Step 4.

If not all vertices in the set $SetPred(i_{j+1}^{(h)}, j)$ are labeled, then select an unlabeled vertex $i_t \in SetPred(i_{j+1}^{(h)}, j)$ such that 1) $i_t \in V_{double} \cup \{0, i_{\min}\}$ or 2) $i_t \notin V_{double} \cup \{0, i_{\min}\}$ and $L - i_t \notin \{i_{j+1}^{(h)}, i_{j+2}^{(h)}, \ldots, i_{n+2}^{(h)}\}$. If such a vertex does not exist, select an arbitrary unlabeled vertex $i_t \in SetPred(i_{j+1}^{(h)}, j)$. In the latter case, property (d) will be violated for path $P^{(h)}$. The rule of selecting vertex i_t aims at satisfying property (d). Label selected vertex and set $i_j^{(h)} = i_t$.

If $j \geq 2$, then re-set $j := j - 1$ and repeat Step 4.

If $j = 1$, then $(0, L)$-path $P^{(h)}$ has been constructed. If $h = x$, go to Step 5. If $h < x$, set $P^{(h+1)} = P^{(h)}$, re-set $h := h + 1$ and repeat Step 4.

Step 5. Select $(0, L)$-path $P^{(h)}$, $1 \leq h \leq x$, with the maximum number of arcs having weights from the set B, output it and stop.

Since $s_{i_j,k} \leq 2n + 1$ for all i_j and k used in the algorithm PATH(x), Steps 1-3 and 5 require $O(n^3)$ time. Step 4 requires $O(xn^3)$ time. Therefore, the overall time complexity of the algorithm is $O(xn^3)$.

Notice that the outputs of PATH, PATH-F and PATH(x) for the same digraph G can be all different because ties in Step 2 of PATH and PATH-F and ties in Step 4 of PATH(x) can be settled differently.

PATH, PATH-F and PATH(x) can be used to find an approximate solution of SPDP, for which all interpoint distances are from the multiset B but the multiset of these distances does not necessarily coincide with B, and all end distances are from the multiset A but their multiset does not necessarily coincide with A. However, computational experiments (described in Section 6) demonstrated that, for instances of SPDP generated using real DNA sequences taken from [19] with $20 \leq n \leq 50$, and for randomly generated instances with $n = 50$, algorithms PATH-F and PATH(n^2) always delivered an exact solution.

The algorithm PATH(x) can be modified to construct all non-congruent solutions of SPDP. Only Step 4 of this algorithm needs to be modified for these purposes, while Step 5 should be removed. The key element for the modification is an observation that any feasible

path from 0 to L in the digraph G is of the form

$$P = (i'_1, \ldots, i'_{n+2}), \text{ where } i'_{n+2} = L \text{ and } i'_j \in SetPred(i'_{j+1}, j),\ j = n+1, n, \ldots, 1. \quad (1)$$

The set of non-congruent solutions of SPDP corresponds to the set of feasible paths P satisfying relations (1). The modification of PATH(x) is to construct the set of paths satisfying these relations and then eliminate from this set paths that do not comply with properties (c) and (d).

We have $i'_{n+1} \in I_{n+1} := SetPred(L, n+1)$, $i'_n \in I_n := \cup_{i \in I_{n+1}} SetPred(i, n)$, $i'_{n-1} \in I_{n-1} := \cup_{i \in I_n} SetPred(i, n-1)$, $\ldots$, $i'_3 \in I_3 := \cup_{i \in I_4} SetPred(i, 3)$, $i'_2 = i_{\min}$, $i'_1 = 0$.

The total number of paths satisfying (1) is equal to 2^K, where $K = 2 + \sum_{j=3}^{n+1} |I_j|$. Therefore, the total number of non-congruent solutions of SPDP does not exceed 2^K.

6. Computational experiments

In this section we present the results of a set of computational experiments with the algorithms SWITCH, PATH-F and PATH(x), and their comparison with the $O(n2^n)$ time enumerative algorithm ENUM in [2]. We did not test algorithm PATH because it is a part of algorithm PATH-F. Algorithm REMOVE was used as a part of either algorithm PATH-F and PATH(x). All the considered algorithms were implemented in C++ with the use of the STL library and The Boost Graph library [32]. The tests were run on a portable PC with Intel Pentium M 2GHz processor and 480Mb of RAM under the Windows XP operational system. All random numbers were generated by using uniform distribution.

The algorithms were compared against their average running times and average solution quality (the latter makes sense for approximation algorithms only). The average solution quality of the approximation algorithms was evaluated by three parameters: 1) the average percentage of exact solutions found (entry $\%Exact$ in the tables below); 2) the average percentage of correct interpoint distances from multiset B found (entry $\%Correct\ b_j$ in the tables below) and 3) the average percentage of correct end distances found from multiset A (entry $\%Correct\ a_j$ in the tables below). The best result in each experiment is indicated in bold.

In the first set of experiments, algorithms ENUM, SWITCH, PATH-F, PATH(1) and PATH(n^2) were run for various values of n. Given n, we randomly generated interpoint distances $b_j \in (100, 2000)$, $j = 1, \ldots, n+1$, that formed the multiset B. We further assumed

Table 1: Average run time on random data.

n	Average run time, sec				
	ENUM	SWITCH	PATH-F	PATH(1)	PATH(n^2)
50	< **0,001**	< **0,001**	0,002	0,001	0,004
100	0,002	< **0,001**	0,012	0,011	0,128
200	0,005	< **0,001**	0,48	0,084	4,21
300	0,014	< **0,001**	7,23	0,25	15,4
400	0,05	< **0,001**	47,7	0,5	32,9
500	0,42	< **0,001**	219	0,87	58,4
600	17,22	**0,001**	> 10 min	1,56	126
700	> 10 min	**0,001**	> 10 min	2,04	174
800	> 10 min	**0,001**	> 10 min	3,17	328
900	> 10 min	**0,001**	> 10 min	4,23	494

Table 2: Percentage of exact solutions found on random data.

n	Average $\%Exact$			
	SWITCH	PATH-F	PATH(1)	PATH(n^2)
50	58%	**100%**	78%	**100%**
100	10%	**100%**	14%	96%
200	0%	**56%**	0%	0%
300	0%	**14%**	0%	0%
400	0%	0%	0%	0%
500	0%	0%	0%	0%
600	0%	–	0%	0%

that $o_j = \sum_{i=1}^{j} b_i$ represents the end distance between point 0 and a point corresponding to the restriction site j in a DNA chain. The multiset A was generated accordingly. Algorithms ENUM, SWITCH, PATH-F, PATH(1) and PATH(n^2) were run for 50 instances for every value of n. The results of the first set of experiments are given in Tables 1-4. Algorithm PATH-F was not run for $n \geq 600$ because it exceeded a 10 minute time limit.

Our second set of experiments was performed on data associated with real DNA chains taken from the nucleotide database, [19]. In this experiment, we cut 43 DNA molecules containing 2,000-200,000 base pairs by 168 enzymes. Then we divided distinct pairs (DNA molecule, restriction enzyme) into groups depending on the size n of instances of SPDP derived from these pairs. Given a sequence of nucleotides and a restriction enzyme, the multisets A and B were determined as if we performed an ideal biochemical experiment. Tables 5-8 present the results of our second set of experiments. Algorithm PATH-F was not

Table 3: Average percentage of correct interpoint distances found on random data.

n	Average $\%Correct\ b_j$			
	SWITCH	PATH-F	PATH(1)	PATH(n^2)
50	97,7%	**100%**	96,6%	**100%**
100	95,3%	**100%**	79,1%	99,9%
200	90,9%	**98,4%**	65%	83,2%
300	87,4%	**97%**	61,8%	71,2%
400	84,6%	**94,3%**	61,9%	67,3%
500	81,5%	**92,7%**	62,2%	67,9%
600	**79,9%**	–	63,8%	66,7%
700	**78,3%**	–	65%	67,3%
800	**76,9%**	–	65,8%	69,7%
900	**75,8%**	–	66,3%	69,3%

Table 4: Average percentage of correct end distances found on random data.

n	Average $\%Correct\ a_j$			
	SWITCH	PATH-F	PATH(1)	PATH(n^2)
50	**100%**	**100%**	96,9%	**100%**
100	**100%**	**100%**	80,1%	99,9%
200	**100%**	99,1%	65,4%	85,2%
300	**100%**	98,8%	61,1%	72,6%
400	**100%**	97,6%	61,1%	68,1%
500	**100%**	97,4%	61,3%	68,7%
600	**100%**	–	63,3%	67,5%
700	**100%**	–	64,3%	68,2%
800	**100%**	–	65,7%	71,7%
900	**100%**	–	66,8%	72,3%

Table 5: Average run time on data from GenBank.

Size of instances	Number of instances	Average run time, sec				
		ENUM	SWITCH	PATH-F	PATH(1)	PATH(n^2)
$20 \leq n \leq 50$	1060	< **0,001**	< **0,001**	0,001	0,001	0,001
$50 \leq n \leq 100$	557	0,001	< **0,001**	0,012	0,006	0,043
$100 \leq n \leq 150$	174	0,004	< **0,001**	0,37	0,04	1,1
$150 \leq n \leq 200$	115	0,06	< **0,001**	4,88	0,15	4,52
$200 \leq n \leq 250$	62	0,014	< **0,001**	36	0,33	8,7
$250 \leq n \leq 300$	84	60,4	< **0,001**	147	0,54	14,9
$300 \leq n \leq 350$	63	> 15 min	< **0,001**	667	0,92	26,7
$350 \leq n \leq 400$	91	> 15 min	**0,001**	> 15 min	1,63	34,4

Table 6: Percentage of exact solutions found on data from GenBank.

Size of instances	Average $\%Exact$			
	SWITCH	PATH-F	PATH(1)	PATH(n^2)
$20 \leq n \leq 50$	90%	**100%**	94,7%	**100%**
$50 \leq n \leq 100$	58,3%	**98,7%**	68,5%	93,3%
$100 \leq n \leq 150$	12,5%	**84,5%**	10,8%	51,3%
$150 \leq n \leq 200$	0%	**49,6%**	0%	0,32%
$200 \leq n \leq 250$	0%	**13,8%**	0%	0%
$250 \leq n \leq 300$	0%	0%	0%	0%
$300 \leq n \leq 350$	0%	0%	0%	0%
$350 \leq n \leq 400$	0%	–	0%	0%

run for $n \geq 350$ because it exceeded a 15 minute time limit.

Our experiments show that, as n grows, the quality of approximate solutions decreases. However, for $20 \leq n \leq 50$ and the data from GenBank and for $n = 50$ and random data, algorithms PATH-F and PATH(n^2) always delivered an exact solution. For $n = 100$ and random data, algorithm PATH-F always delivered an exact solution. The fastest of our approximation algorithms is SWITCH. For $20 \leq n \leq 50$ and data from GenBank, it found 90% of exact solutions and the average percentage of correct interpoint distances was 99%.

In the experiments with larger n ($600 \leq n \leq 900$) and random instances, algorithm SWITCH returned the best average results. Moreover, it did it with the shortest computational time. It always reached more than 75% of correct interpoint distances with 100% of correct end distances. It seems that, for data coming from real DNA sequences, SWITCH could be recognized as the best algorithm on average. It is still the fastest and produces exact end distances. However, in terms of $\%Correct\ b_j$, algorithm PATH-F generated the best

Table 7: Average percentage of correct interpoint distances found on data from GenBank.

Size of instances	Average %$Correct\ b_j$			
	SWITCH	PATH-F	PATH(1)	PATH(n^2)
$20 \leq n \leq 50$	99%	**100%**	99,1%	**100%**
$50 \leq n \leq 100$	97,5%	**99,9%**	93,9%	99,3%
$100 \leq n \leq 150$	94,6%	**98,6%**	77,8%	94%
$150 \leq n \leq 200$	90,1%	**96,3%**	68,6%	83,4%
$200 \leq n \leq 250$	88,2%	**93,3%**	67,9%	77,1%
$250 \leq n \leq 300$	83,5%	**88,5%**	66,9%	72%
$300 \leq n \leq 350$	80,9%	**86,3%**	67,8%	71,4%
$350 \leq n \leq 400$	**79,7%**	–	69,8%	72,1%

Table 8: Average percentage of correct end distances found on data from GenBank.

Size of instances	Average %$Correct\ a_j$			
	SWITCH	PATH-F	PATH(1)	PATH(n^2)
$20 \leq n \leq 50$	**100%**	**100%**	99,2%	**100%**
$50 \leq n \leq 100$	**100%**	99,9%	94,5%	99,5%
$100 \leq n \leq 150$	**100%**	99,2%	78,5%	95%
$150 \leq n \leq 200$	**100%**	98%	68,8%	85,7%
$200 \leq n \leq 250$	**100%**	96,6%	68,6%	79,1%
$250 \leq n \leq 300$	**100%**	94,7%	67,9%	74%
$300 \leq n \leq 350$	**100%**	92,3%	69,3%	73,8%
$350 \leq n \leq 400$	**100%**	–	72,8%	75,6%

solutions. Therefore, the latter can also be considered as a good tool for solving real-world SPDP instances.

Another observation from the experimental data is that algorithms PATH-F and PATH(1) solve randomly generated instances of the same size n faster than instances based on real data. It is also true for algorithm PATH(n^2), but to a lesser extent. The reason might be that interpoint distances in real data instances are not uniformly distributed. An analysis of real data instances in our experiments shows that the interpoint distances follow an exponential distribution rather than a uniform one, and most of the distances are relatively small. This leads to larger graphs for real data instances because there are more chances that an additional arc between any two vertices will appear, thus making the model more difficult to solve. The difference in the running times on real data and randomly generated instances of the algorithm PATH(n^2) is smaller because the time needed for graph construction takes a smaller part of its total running time.

7. Conclusions

In this paper, we have shown that SPDP-Min cannot be approximated in pseudo-polynomial time with any finite approximation ratio ρ, and SPDP-Max cannot be approximated in pseudo-polynomial time with $\rho < 1 + \frac{1}{n}$, unless $\mathcal{P} = \mathcal{NP}$. An approximation algorithm, called SWITCH, for SPDP-Max has been presented, which provides $\rho = 1 + \frac{n}{n+2}$, and runs in $O(n \log n)$ time. A graph-theoretic model for SPDP has been suggested, and an algorithm, called REMOVE, based on this model has been developed. This reduces the search space for solutions of SPDP, and runs in $O(n^2 \log n)$ time. Three graph-theoretic polynomial time algorithms for finding an approximate solution of SPDP have been presented. Computational experiments demonstrated that, for instances of SPDP generated using real DNA sequences from GenBank with $20 \leq n \leq 50$, and for randomly generated instances with $n = 50$, two of these algorithms always delivered an exact solution.

Two theoretical research issues related to the approximation of SPDP are left open: the existence of a polynomial time ρ-approximation algorithm for SPDP-Max such that $1 + \frac{1}{n} \leq \rho < 1 + \frac{n}{n+2}$, and the existence of a PTAS for this problem. Resolving these issues is of interest for future research. Furthermore, the development of heuristic algorithms for SPDP that behave well on real-life data has a practical value. Here, one could use an interesting evolutionary approach as discussed in [22]. In the general area of partial digest

models and methods, the computational complexity of the error-free Partial Digest Problem is yet to be established [17].

References

[1] J. Blazewicz, M. Borowski, P. Formanowicz, T. Glowacki, Genetic and tabu search algorithms for peptide assembly problem, RAIRO — Operations Research 44 (2010) 153–166.

[2] J. Blazewicz, E.K. Burke, M. Kasprzak, A. Kovalev, M.Y. Kovalyov, Simplified Partial Digest Problem: enumerative and dynamic programming algorithms, ACM/IEEE Transactions on Computational Biology and Bioinformatics 4 (2007) 668–680.

[3] J. Blazewicz, P. Formanowicz, M. Kasprzak, Selected combinatorial problems of computational biology, European Journal of Operational Research 161 (2005) 585–597.

[4] J. Blazewicz, P. Formanowicz, M. Kasprzak, M. Jaroszewski, W.T. Markiewicz, Construction of DNA restriction maps based on a simplified experiment, Bioinformatics 17 (2001) 398–404.

[5] J. Blazewicz, M. Jaroszewski, New algorithm for the Simplified Partial Digest Problem, Lecture Notes in Bioinformatics 2812 (2003) 95–110.

[6] J. Blazewicz, M. Kasprzak, Combinatorial optimization in DNA mapping — a computational thread of the Simplified Partial Digest Problem, RAIRO — Operations Research 39 (2005) 227–241.

[7] J. Blazewicz, C. Oguz, A. Swiercz, J. Weglarz, DNA sequencing by hybridization via genetic search, Operations Research 54 (2006) 1185–1192.

[8] C. Blum, M. Yabar Valles, M.J. Blesa, An ant colony optimization algorithm for DNA sequencing by hybridization, Computers & Operations Research 35 (2008) 3620–3635.

[9] T. Chen, M.-Y. Kao, M. Tepel, J. Rush, G. M. Church, A dynamic programming approach to de novo peptide sequencing via tandem mass spectrometry, in: Symposium on Discrete Algorithms, 2000.

[10] M. Cieliebak, S. Eidenbenz, Measurement errors make the Partial Digest Problem NP-Hard, LATIN (2004) 379–390.

[11] M. Cieliebak, S. Eidenbenz, P. Penna, Noisy data make the Partial Digest Problem NP-hard, Lecture Notes in Bioinformatics 2812 (2003) 111–123.

[12] M. Cieliebak, S. Eidenbenz, G.J. Woeginger, Double Digest revisited: complexity and approximability in the presence of noisy data, COCOON (2003) 519–527.

[13] T.H. Cormen, C.E. Leiserson, R.L. Riverst, Introduction to Algorithms, The MIT Press, McGraw-Hill Book Company, 1990.

[14] P. Crescenzi, L. Trevisan, On approximation scheme preserving reducibility and its applications, in: Proceedings of 14th Conference on Foundations of Software Technology and Theoretical Computer Science 880 (1994) 330–341.

[15] K. Danna, D. Nathans, Specific cleavage of simian virus 40 DNA by restriction endonuclease of Hemophilus Influenzae, in: Proceedings of the National Academy of Sciences of the United States of America 68 (1971) 2913–2917.

[16] K.J. Danna, G.H. Sack, D. Nathans, Studies of simian virus 40 DNA. VII. A cleavage map of the SV40 genome, Journal of Molecular Biology, 78 (1973) 363–376.

[17] A. Daurat, Y. Gerard, M. Nivat, Some necessary clarifications about the Chords' Problem and the Partial Digest Problem, Theoretical Computer Science 347 (2005) 432–436.

[18] A. Dudez, S. Chaillou, L. Hissler, R. Stentz, M. Champomier-Verges, C. Alpert, M. Zagorec, Physical and genetic map of the Lactobacillus sakei 23K chromosome, Microbiology 148 (2002) 421–431.

[19] Genetic Sequence Database GenBank, http://www.ncbi.nlm.nih.gov/Genbank/index.html, 2008.

[20] J. Grimwood, L. Gordon, A. Olsen and others, The DNA sequence and biology of human chromosome 19, Nature 428 (2004) 529–535.

[21] S. Heber, M. Alekseyev, S.H. Sze, H. Tang, P.A. Pevzner, Splicing graphs and EST assembly problem, Bioinformatics 18 (2002) 181–188.

[22] L. Jourdan, C. Dhaenens, E-G. Talbi, Evolutionary Feature Selection for Bioinformatics, in: Computational Intelligence In Bioinformatics, IEEE Computer Society Press (2007) 117–140.

[23] S. Keis, J.T. Sullivan, D.T. Jones, Physical and genetic map of the Clostridium saccharobutylicum (formerly Clostridium acetobutylicum) NCP 262 chromosome, Microbiology 147 (2001) 1909–1922.

[24] M.W. Kirby, Operational Research in War and Peace: The British Experience from the 1930s to 1970, Imperial College Press, London, 2003.

[25] T. Kuwahara, M.R. Sarker, H. Ugai, S. Akimoto and others, Physical and genetic map of the Bacteroides fragilis YCH46 chromosome, FEMS Microbiology Letters 207 (2002) 193–197.

[26] P. Lukasiak, J. Blazewicz, M. Milostan, Some operations research methods for analyzing protein sequences and structures, Annals of Operations Research 175 (2010) 9–35.

[27] A. Nikolakopoulos, H. Sarimveis, A metaheuristic approach for the sequencing by hybridization problem with positive and negative errors, Engineering Applications of Artificial Intelligence 21 (2008) 247–258.

[28] P.A. Pevzner, H. Tang, M.S. Waterman, An Eulerian path approach to DNA fragment assembly, In: Proceedings of the National Academy of Sciences 98 (2001) 9748–9753.

[29] P.A. Pevzner, Computational Molecular Biology. An Algorithmic Approach, MIT Press, Cambridge, MA, 2000.

[30] R.J. Roberts, G.A. Wilson, F.E. Young, Recognition sequence of specific endonuclease BamH.I from Bacillus amyloliquefaciens H, Nature 265 (1977) 82–84.

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

[32] J.G. Siek, L-Q. Lee, A. Lumsdaine, The Boost Graph Library: User Guide and Reference Manual, Addison-Wesley, 2001.

[33] S.S. Skiena, W.D. Smith, P. Lemke, Reconstructing sets from interpoint distances, in: Proceedings of the 6th ACM Symposium on Computational Geometry (1990) 332–339.

[34] S.S. Skiena, G. Sundaram, A partial digest approach to restriction site mapping, Bulletin of Molecular Biology 56 (1994) 275–294.

[35] M. Sonawane, Y. Carpio, R. Geisler, H. Schwarz, H.-M. Maischein, Ch. Nuesslein-Volhard, Zebrafish penner/lethal giant larvae 2 functions in hemidesmosome formation, maintenance of cellular morphology and growth regulation in the developing basal epidermis, Development 132 (2005) 3255–3265.

[36] J.C. Venter, M.D. Adams, E.W. Myers and others, The sequence of the human genome, Science 291 (2001) 1304–1351.

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

[38] J.-H. Zhang, L.-Y. Wu, Y.-Y. Zhao, X.-S. Zhang, An optimization approach to the reconstruction of positional DNA sequencing by hybridization with errors, European Journal of Operational Research 182 (2007) 413-427.