Dynamic programming for LR-PCR segmention of bacterium genomes

Rumen Andonov
University of Valenciennes
Le Mont Houy 59313 Valenciennes, France
randonov@univ-valenciennes.fr

Dominique Lavenier, Philippe Veber
IRISA, Campus de Beaulieu,
35042 Rennes Cedex, France
lavenier@irisa.fr

Nicola Yanev
University of Sofia,Bulgaria
choby@math.bas.bg

Abstract

Bacterium genome plasticity can efficiently be studied by Long-Range PCR: genomes of different strains are split into hundreds of short segments which, after LR-PCR amplification, are used to sketch profiles. The segments have : (1) to cover the entire genome, (2) to overlap each other, and (3) to be of nearly identical size. This paper adresses the problem of finding a list of segments satisfying these constraints "as much as possible". Two algorithmes based on dynamic programming approach are presented. They differ on the optimization criteria for measuring the quality of the covering. The first one considers the maximal deviation of the segment lengths relatively to an ideal *length. The second one automatically* finds *a segment length which minimizes the maximal deviation.*

1 Introduction

A practical way to study the plasticity of bacterium genomes without systematically sequencing all the available strains is to exploit the LR-PCR (Long Range Polymerase Chain Reaction) technique. The genomes of the strains are split into a large number of short segments before performing a LR-PCR on each of them. Depending on the reorganization, the deletion or the insertion of certain genomic zones, it is expected that a few segments will not be amplified. Thus, a *profile* – or a signature – can be assigned to each strain. It represents the set of amplified and non amplified segments. The final step is to perform a global analysis of all the profiles. This strategy, recently tested by Ohnishi *et al.* [2] to study the genome diversity of *E. coli*, is explained on Fig. 1.

This strategy first implies to determine the set of segments which will cover the genome. A strain whose genome is entirely sequenced is chosen as reference. Then, potential PCR primers are localized on the genome since they specify the position where the segments start and end. Having these data, the goal is to cover the genome with overlapping segments of nearly identical size, knowing that the segments locations are constrained by the position of the primers.

Actually, the distribution of the primer sites along the bacterium genome is non-uniform. There may have large regions (a few Kbp) without primer sites or, on the contrary, very dense regions of primer sites. In addition, some regions are forbidden: they correspond to repeated zones, bacteriophage sequences, or mobile elements such as transposons. As some of these regions are longer than the expected length of the covering segments, the circular genome is cut into a few number of linear pieces, called domains (see Fig. 2).

Thus, the problem of segmenting a complete bacterial genome is reduced to cover each domain with segments of nearly identical size. Along a domain, there are specific positions called *primer sites*. The overlapping segments can start and end *only* at these particular positions. If we assume, for the sake of simplicity, that a solution is made of a list S of N segments, and that each segment can take only P different positions, then the number of possibilities is equal to P^N. Finding the best one when N is large is clearly a combinatorial problem (in real application, $N > 100$).

More formally, the problem can be formulated as follows. Given a *domain*, i.e. a DNA sequence ranging from a few 100 Kpb to a few Mbp, together with all potential primer positions, we need to cover it with a sequence of overlapping segments of nearly identical size. Such a covering will be called a *segmentation* if the segments satisfy the following conditions:

- The length of any segment varies in the range $[\underline{L}, \overline{L}]$.
- The length of the overlap between any two consecutive

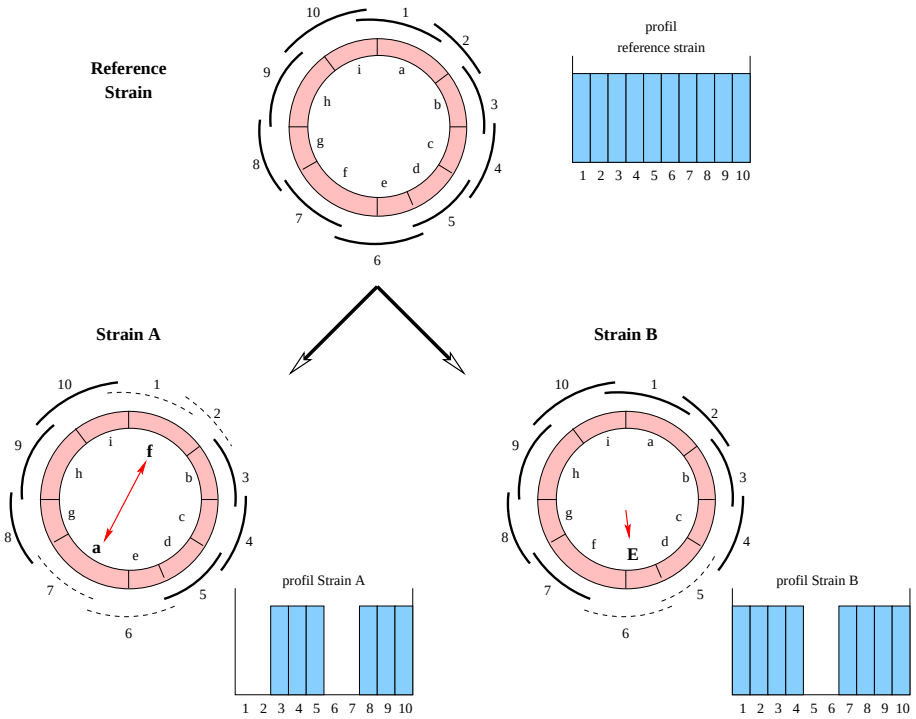

Figure 1. Strategy to study the plasticity of a circular bacterial genome: a reference strain is fully covered with overlapping segments. The two extremities of each segment are characterized by starting and ending primers. On the reference strain, the LR-PCR amplifies all the segments. On strain A the zones a and f have swapped, preventing the amplification of segments 1, 2, 6 and 7. On strain B, zone e is modified: segments 5 and 6 cannot be amplified.

segments varies in the range $[\underline{O}, \overline{O}]$.

- The distance from the beginning of the domain to the starting-primer of the first segment is no more than D_s. The distance from the ending-primer of the last segment to the end of the domain is no more than D_e.

Two cases of this problem have been considered. In the first one we search for a sequence S of overlapping segments, each one of size as close as possible to a *given* ideal length L. In the second case, the value of L is *unknown* and we look for a couple (L^*, S), where L^*, $\underline{L} \leq L^* \leq \overline{L}$, and such that the sequence S is of of minimal error with respect to L^*.

For each case we: (i) formulate a suitable combinatorial optimization model; (ii) program dedicated algorithm for solving these models; (iii) analyze the complexity of the proposed algorithms. We are not aware of other algorithms from the literature to have been used for this purpose. This paper focusses on the algorithmic aspects of the problem. The reader interested in the genomic aspects can find more details in the accompanying paper [5].

Organization of the paper is as follows. The formal statement of the problem and definitions are given in section 2. Section 3 is dedicated to the first case of the problem, while section 4 considers the second case. Numerical results and complexity analysis are provided in section 5.

2 Graph problem formulation

The formal statement of the problem is as follows. Let be given: i) a nucleotide sequence D containing D_L elements (called domain); ii) a set S^l of starting-primer sites; iii) a set S^r of ending-primer sites. We can then define the set F of *feasible segments*, i.e. couples of starting and ending primers, $f = [b, e], \quad b < e$, such that:

- $b \in S^l, \quad e \in S^r$.
- the length $l(f) = e - b$ satisfies $\underline{L} \leq l(f) \leq \overline{L}$.

Let us denote by F_s (resp. F_t) the set of segments which can begin (resp. end) a segmentation. This means that if $f = [b, e] \in F_s$ then $b \leq D_s$ and that if $f = [b, e] \in F_t$ then $D_L - e \leq D_e$.

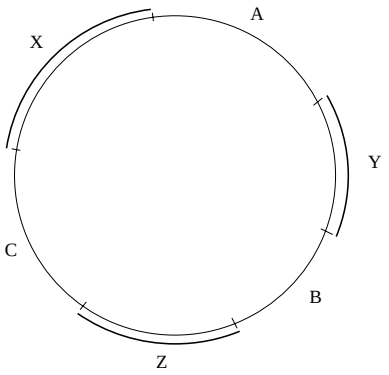

Figure 2. Regions X,Y and Z are forbidden. The length of each one of them is larger than the size of the covering segments. The problem of segmenting a full genome is therefore transformed in segmenting three linear pieces denoted here as A, B and C and called domains. Any domain is associated to the solution of an independent subproblem.

Definition 2.1. The segment f' *is compatible with* the segment f (denoted as $f \prec f'$), iff f' starts to the left of the ending-primer site of f and the length of the overlap is in $[\underline{O}, \overline{O}]$.

Definition 2.2. A sequence $S = f_1, f_2, \ldots f_k$ of feasible segments will be referred to as a *covering sequence (segmentation)* if $f_1 \in F_s, f_k \in F_t$ and $f_i \prec f_{i+1}$.

Definition 2.3. A *covering graph* of the nucleotide sequence is a directed graph $G(V, A)$:

• the node set $V = F \cup \{s, t\}$, where s and t two additional vertices.

• the arc set

$$\begin{aligned} A = \quad & \{(f, f') \in F \times F \ : f \prec f'\} \\ \cup \quad & \{(s, f) \in \{s\} \times F_s\} \\ \cup \quad & \{(f, t) \in F_t \times \{t\}\} \end{aligned}$$

Remark 2.1. Note that the covering graph $G(V, A)$ is without circuits because of the binary relation "is compatible with". The non-directed version of this graph is a subgraph of the so called interval graph (see chapter 1.5.4 [4]) over the set of feasible intervals.

3 The case when the segment length L is given

In this section we assume that an *ideal* length L is given and we define a cost function $C_L(f)$ on F as: $\forall f \in F \quad C_L(f) = |l(f) - L|$. The problem to solve can be considered as a $\boldsymbol{minmax}$ (bottleneck) variant of the classical Shortest Path Problem (**SPP**), if the length of a path $\boldsymbol{r = s, v_1, \ldots v_k, t}$ is determined by $\boldsymbol{C_L(r) = \max_{v_i \in r} C_L(v_i)}$.

One can easily see an one-to-one correspondence between covering sequences and the directed paths from $\boldsymbol{s}$ to $\boldsymbol{t}$ in $\boldsymbol{G}$. In this context the length of a path can be viewed as the error of the segmentation associated to this path. If we denote by $\boldsymbol{R}$ the set of paths from $\boldsymbol{s}$ to $\boldsymbol{t}$, the problem to be solved is $\boldsymbol{\min_{r \in R} C_L(r) = C^*_L}$. An instance of the problem is given on Fig. 3, while its corresponding covering graph is depicted on Fig. 4.

For a graph $\boldsymbol{G}$ without circuits, a dynamic programming recurrence gives an algorithm linear in $\boldsymbol{A}$. Let us denote by $\boldsymbol{d_i}$ the length (in sense of max instead of sum) of the shortest path from $\boldsymbol{s}$ to $\boldsymbol{v_i}$ and let $\boldsymbol{\Gamma^{-1}(v)}$ be the set of all predecessors of $\boldsymbol{v}$. Then obviously we have :

$$\boldsymbol{d_i = \min_{v_j \in \Gamma^{-1}(v_i)} \max\{d_j, C_L(v_i)\}} \qquad (1)$$

which leads to the algorithm **SPP** given below.

The **SPP** algorithm takes as input a graph and generates a list of segments. The vertices of **G** are topologically sorted before processed. This means that vertices are arranged on a line in such a way that all arcs are from left to right.

Complexity Analysis

If the graph is represented by the predecessors of each vertex, then the algorithm **SPP** has complexity $\boldsymbol{O(|A|)}$. This follows from the observation that $\boldsymbol{|A|}$ equals the sum of indegrees of the vertices and from the fact that the complexity of the topological sort is $\boldsymbol{O(|A|)}$ (see [4] chapter 3.3.4).

Remark 3.1. For our problem the indices of the vertices are naturally induced by appearance of the starting-primers, i.e. the graph is already topologically sorted.

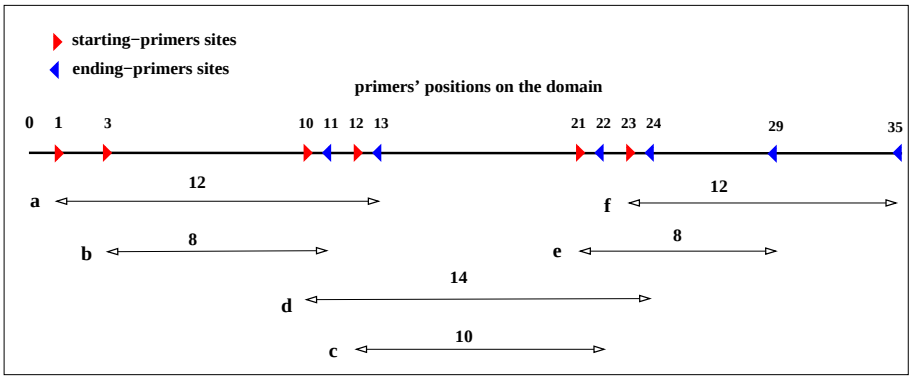

Figure 3. An instance of the problem where : $D_L = 35, S^l = \{1, 3, 10, 12, 21, 23\}, S^r = \{11, 13, 22, 24, 29, 35\}, D_s = 3, D_e = 6$, $(\underline{L}, \overline{L}) = (6, 14)$, $L = 10$, $(\underline{O}, \overline{O}) = (1, 3)$. **For the sake of simplicity we do not consider the entire set** F**, but a subset of it containing the feasible segments** a, b, c, d, e, f **with lengths respectively** $(12, 8, 10, 14, 8, 12)$.

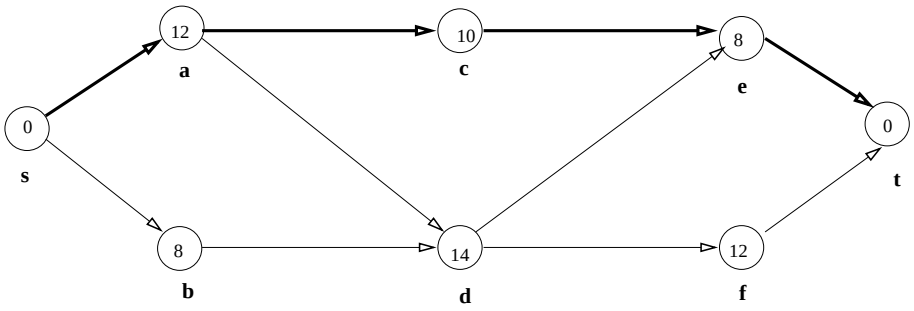

Figure 4. The covering graph corresponding to Fig. 3. The circles contain the segments lengths. When the algorithm SPP is applied to this graph, it finds an optimal path $(s - a - c - e - t)$ **which contains segments with lengths 12,10 and 8. The error with respect to 10 equals 2.**

Algorithm SPP(G) {Search for the shortest $s - t$ path in **G** using a DP recurrence}
input : directed graph $G(V, A)$; the given length L;
output : the length of the optimal path–$d_{|V|}$, the optimal path contained in π;
sort : reindex the vertices of **G** by a topological sort
initialize : $d_1 = 0,\ \pi[1] = 0$;
body : **for** i=2 **to** $|V|$ **do**
compute $d_i = \min_{v_j \in \Gamma^{-1}(v_i)} \max\{d_j, C_L(v_i)\} = \max\{d_k, C_L(v_i)\}$
set $\pi[i] = k$; (note that $k =$ **arg** $\min_{v_j \in \Gamma^{-1}(v_i)} \max\{d_j, C_L(v_i)\}$
endfor
print : $c \leftarrow \pi[|V|]$;
while $c > 0$ **do**
print c;
$c \leftarrow \pi[c]$;
endwhile

4 Searching for the optimal segment length in the interval $[\underline{L}, \overline{L}]$

Up to now, the error of the segmentation was measured by the maximal deviation of the segments from a *given* ideal length L. Usually, this length is taken as the middle of the interval $[\underline{L}, \overline{L}]$ (i.e. $L = (\underline{L} + \overline{L})/2$) and is in fact a kind of simplification of the problem. Note for example that on Fig. 3 there is a feasible path (a, d, f). The deviation in the lengths of the corresponding segments, $(12, 14, 12)$, is very small. In fact the error with respect to $L = 13$ is one and this path is definitely a good candidate for the LR-PCR technique. However, it cannot be discovered in the framework of the above described model.

For these reasons, in this section, we make a step further toward a quite natural generalization of the problem by considering L as a parameter and looking for L^* such that the best segmentation with respect to it is of minimal error. This will change the original problem $\min\limits_{r \in R} C_L(r) = C^*_L$ to the problem $\min\limits_{L} \min\limits_{r \in R} C_L(r) = C^*$. In order to put the later one in more tractable form we can exclude L from the model in the following way: For an arbitrary $(s-t)$-path $r = sv_0 \dots v_n t$ let $c^r_{min} = \min\limits_{v_i \in r}\{l(v_i)\}$ and $c^r_{max} = \max\limits_{v_i \in r}\{l(v_i)\}$. (Recall that $l(i)$ is the length of the i^{th} segment). Then the following assertion is true:

Theorem 4.1. *The minimal error of the segmentation given by a path r is $0.5(c^r_{max} - c^r_{min})$ and it is attained at the length $L^*(r) = 0.5(c^r_{max} + c^r_{min})$.*

If we call $c^r_{max} - c^r_{min}$ *spread* of the path r then according to the theorem an equivalent reformulation of the above-mentioned problem is simply to **find the $(s-t)$-path in G of minimal spread**, which is to find $\triangle^* = c_{max} - c_{min} = \min\limits_{r \in R}\{c^r_{max} - c^r_{min}\}$

Now, let us associate to any vertex $i \neq s$ of the covering graph a set $\mathcal{A}_i$ defined as follows:

$$\mathcal{A}_i = \{(c^{si}_{\min}, c^{si}_{\max}) : si \text{ being a path from } s \text{ to } i\} \quad (2)$$

In this way a list $\mathcal{A}_i$ contains diverse spreads corresponding to *all* possible $(s-i)$-paths. The solution is the minimal spread in the list $\mathcal{A}_t$. An intuitive construction of the lists $\mathcal{A}_i$ is illustrated on Fig. 5, while formally they are computed by the recurrences (3).

$$\mathcal{A}_i = \begin{cases} \{(\overline{L}, \underline{L})\} & \text{if } i = s \\ \bigcup\limits_{j \in \Gamma^-(t)} \mathcal{A}_j & \text{if } i = t \\ \bigcup\limits_{j \in \Gamma^-(i)} \{l(i) \triangleright (l,u) : (l,u) \in \mathcal{A}_j\} & \text{otherwise} \end{cases} \quad (3)$$

where $e \triangleright (l,u)$ denotes $(\min(l,e), \max(u,e))$.

Remark 4.1. Note that the recurrence (3) is correct since the covering graph is without circuits.

Defined in this way, the set $\mathcal{A}_t$ contains the pair (c^r_{min}, c^r_{max}) for *any* r being a path from s to t. If the vertices of the graph are topologically sorted, the recurrence (3) can be computed by a single traverse of the graph. The rest of the algorithm is now straightforward: select from $\mathcal{A}_t$ the couple (l,u) with minimal spread, delete vertices with length not in the interval $[l,u]$. *Any* of the $(s-t)$-paths in the reduced graph is optimal.

This algorithm is in fact a simple enumeration procedure and the size of the sets $\mathcal{A}_i$ could be very large. For these reasons we introduce an operation (say * operation) which leads to a significant reduction in these sets size. The * operation retains only those couples which are eligible for continuation, i.e. mutually non inclusive and is more precisely defined as follows:

$$\mathcal{A}^* = \mathcal{A} \setminus \{(l,u) \in \mathcal{A} : \exists (l',u') \in \mathcal{A}, \quad [l',u'] \subset [l,u]\} \quad (4)$$

The recurrence (3) is respectively modified:

$$\mathcal{A}^*_i = \begin{cases} \{(\overline{L}, \underline{L})\} & \text{if } i = s \\ \left(\bigcup\limits_{j \in \Gamma^-(t)} \mathcal{A}^*_j \right)^* & \text{if } i = t \\ \left(\mathcal{A}^i_j\right)^* & \text{otherwise} \end{cases} \quad (5)$$

where $\mathcal{A}^i_j = \bigcup\limits_{j \in \Gamma^-(i)} \{l(i) \triangleright (l,u) : (l,u) \in \mathcal{A}^*_j\}$.

The * operation removes from $\mathcal{A}_i$ only pairs (l,u) which are obviously non optimal, because of (5), and we therefore *do not* lose solution. The algorithm **SITA**, is described below.

Complexity analysis

Let $\mathcal{A}^*$, $\mathcal{B}^*$ be two sets such that any $e \in \mathcal{A}^*$ (resp. any $e \in \mathcal{B}^*$) is a minimum in respect to the inclusion relation. Note that in this case we can define the following total order relation in $\mathcal{A}^*$ (resp. $\mathcal{B}^*$).

$$(l,u) \prec (l',u') \text{ iff } (l < l') \wedge (u < u') \quad (6)$$

If we assume now that $\mathcal{A}^*$ and $\mathcal{B}^*$ are sorted according to (6), then applying sort-merge alike algorithm we can realize the operation $(\mathcal{A}^* \bigcup \mathcal{B}^*)^*$ in $O(\max(|\mathcal{A}^*|, |\mathcal{B}^*|))$ operations (interval comparisons). Also note that the result is directly sorted according to (6). Using this observation, we can easily prove that the complexity of the **SITA** algorithm

Algorithm SITA(G)
input: directed graph $G(V, A, C)$;
output: minimal spread (c_{min}, c_{max});
initialization: $\mathcal{A}_s^* \leftarrow \{(\overline{L}, \underline{L})\}$;
topological_sort(**G**);
for i=2 **to** $|V|$ **do**
for all $v_j \in \Gamma^-(v_i)$ **do** $\mathcal{A}_i^* \leftarrow \left(\bigcup \mathcal{A}_j^*\right)^*$ **enddo** ;
enddo ;
$(c_{min}, c_{max}) \leftarrow \arg \min_{(l,u)\in\mathcal{A}_t^*} (u - l)$;

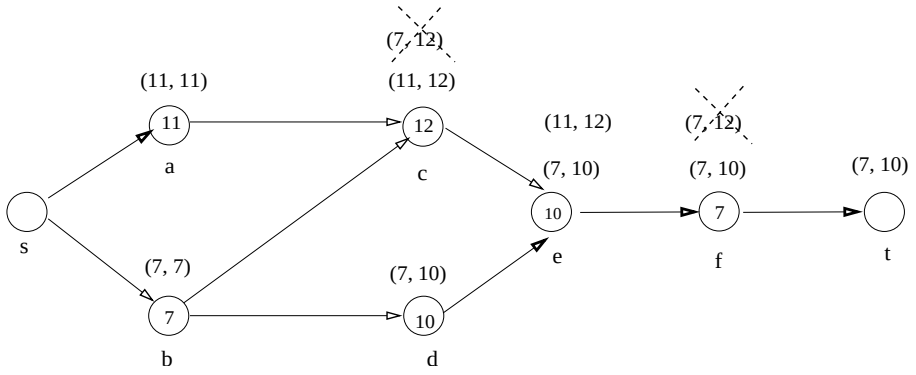

Figure 5. This graph illustrates the links between six segments (a,b,c,d,e,f) and the two artificial vertices s and t. The circles contain the lengths of the corresponding segments. Above any vertex i is given the set $\mathcal{A}_i$ as defined in (2). Note that no one of the elements of the list $\mathcal{A}_e$ can be eliminated. Although the element $(7, 10)$ appears less interesting than $(11, 12)$, its elimination leads to a loss of the solution. In contrast, at vertex c we can eliminate $(7, 12)$ since $[11, 12] \subset [7, 12]$ without loosing the solution. Respectively, at vertex f we can eliminate $(7, 12)$ since $[7, 10] \subset [7, 12]$. This elements reduction corresponds to the * operation defined in (5).

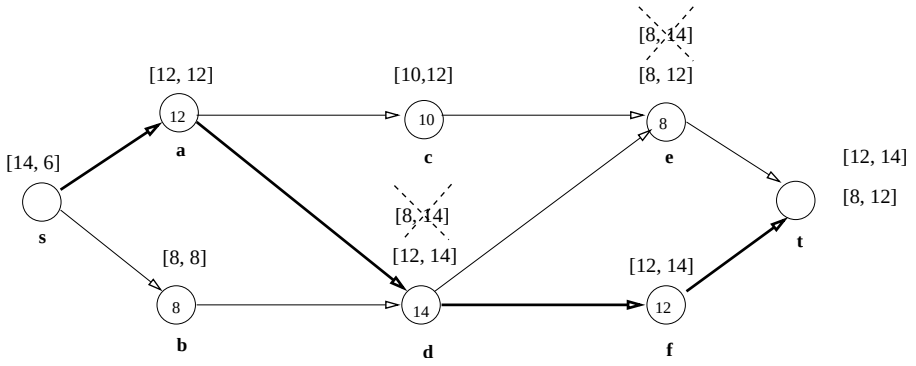

Figure 6. The graph illustrates the behavior of the algorithm SITA on the problem instance depicted on Fig. 3. It finds that an optimal spread of size 2 exists and that the associate length L^* equals 13. The obtained optimal path is $(s - a - d - f - t)$. It contains segments with lengths 12, 14 and 12. Note that run on the same graph, the SPP algorithm was misled by the value of L^* and returned a path of spread 4.

is $O(C|A|)$, where C is the maximum number of eligible intervals and all of them are in the interval $[\underline{L}, \overline{L}]$. The inequality $C \leq (\overline{L} - \underline{L})/2$ can be easily verified.

5 Computational experiments

The **SPP** and **SITA** algorithms are general purpose in sense of underlying graphs, but the primary goals were to use them for the interval graphs discussed in the introduction. That is why all runs are done on graphs, corresponding to domains of varying lengths with uniformly distributed primers. Thus the lack of sufficient biological material is compensated by a randomly generated genomes and despite some mismatches with the reality they could serve well for measuring the computational analysis of their efficiency.

Recalling that the basic parameters are: the *length* D_L of the studied genome domain, the *number* n of primers in this domain, the allowing length from $\underline{L}$ to $\overline{L}$ for the segments and overlap from $\underline{O}$ to $\overline{O}$, it seems more convenient to express the computational complexity of the algorithms as a function of these parameters. Towards this end, the following mixture of probabilistic and deterministic arguments are used below.

If we denote by δ the average density of the primers in the domain we obviously have $\delta = \frac{n}{D_L} < 1$. Now, for any starting-primer in the domain we have on average $(\overline{L} - \underline{L})\delta$ compatible primers (i.e. each one of can built a different segment beginning with the same starting-primer). Thus, the total number of segments in the domain is $O((\overline{L} - \underline{L})\delta n)$. Similarly, for a given segment, there are on average $(\overline{O} - \underline{O})\delta$ potential primers to begin a compatible segment; for any of these starting-primers, there are on average $(\overline{L} - \underline{L})\delta$ potential ending-primers. The total number of pairs of compatible segments (remember it corresponds to $|A|$ in the graph terminology) is therefore $O((\overline{L} - \underline{L})^2(\overline{O} - \underline{O})\delta^3 n)$.

Therefore, we obtain that the algorithms proposed in this paper are *linear* in respect to the number of primers in this domain. More precisely, the average bounds for the maximum number of operations are: $O(|A|) = O((\overline{L} - \underline{L})^2(\overline{O} - \underline{O})\delta^3 n)$ for the **SPP** algorithm and $O(C|A|) = O((\overline{L} - \underline{L})^3(\overline{O} - \underline{O})\delta^3 n)$ for **SITA** algorithm. As we already mentioned $C = (\overline{L} - \underline{L})/2$ is a theoretical upper bound for the lengths of the lists associated with the vertices. It was quite intricated (but not unexpected) to observe how huge is the gap between this bound and the real ones (less then 10 in all runs depicted on Fig. 7). One can easily show that this bound is achieved for example on a graph of 11 vertices and costs from $\underline{L} = 10$ to $\bar{L} = 20$. If the pairs entering the vertex of cost 15 are $[10, 16], [11, 17], [12, 18], [13, 19], [14, 20]$ then all $(C = (\overline{L} - \underline{L})/2)$ of them will survive the seep of * operation. But whatever is the graph with such costs there is no corresponding nucleotide sequence. In this sense, this theoretical upper estimate for C is indeed very pessimistic and unlikely to be reached in real life. For instance, real life values for the parameters are: 1 Mbp for the length of the domains ; 5000 to 15000 for the number of primers ; 10 Kbp $\pm$ 1000 for the length of the segments ; 1 Kbp $\pm$ 500 for the overlap and $\delta < 10^{-2}$ (density). In all our runs with these real life parameters we observed that $C < 10$. In practice, the algorithm is fast and can segment whole genomes in very short time.

6 Conclusion

In this paper we pose and answer two questions about covering a genome by a sequence of overlapping segments. The quality of the covering is measured according to two criteria:

- the maximal deviation of the segment's length from a given length is minimal;
- the maximal spread between the longest and the shortest segment is minimal.

We propose two algorithms: **SPP** for solving the former problem and **SITA** for solving the later one. They take as input the set of starting and ending-primers, the genome domain to split into segments, and the parameters corresponding to the segment length and the overlap size. The result is an optimal list of segments satisfying the corresponding criterion. The algorithm **SITA** has been implemented using the Objective CAML language. It is part of a package called **GenoFrag** which also includes another software, jointly developed with the INRA[1] microbiology team, to generate the set of primers[2]. Actually, this software acts as a pipeline of filters fed by a complete genome: each filter, dedicated to some specific features, discard all the primers which do not satisfy user-specified constraints–GC-content, thermodynamic stability, hairpin loop size, etc. (see [5] for more details).

We tested the two algorithms on the *Staphylococcus Aureus* [1], a Gram-positive pathogenic bacterium. Primers were generated from the N315 *S. Aureus* strain using different filters. The largest domain represents 1.3 Mbp with an average primer density of 0.006. The computation time for generating the optimal list of overlapping segments on a standard Linux machine (PC running at 1.6 Ghz with 256 Mbytes of memory) does not exceed one minute. This is a

[1] Laboratoire d'hygiène alimentaire,UMR STLO, INRA, ENSAR, 65 rue de Saint Brieuc, 35042 Rennes, France

[2] **GenoFrag** contains also a software for solving the problem when the length L is given. The complexity of the underlying algorithm is slightly weaker (logarithmic factor) than **SPP**, since it focuses on graphs with circuits. We do not present it here for the lack of space. The interested reader can find its description in [6].

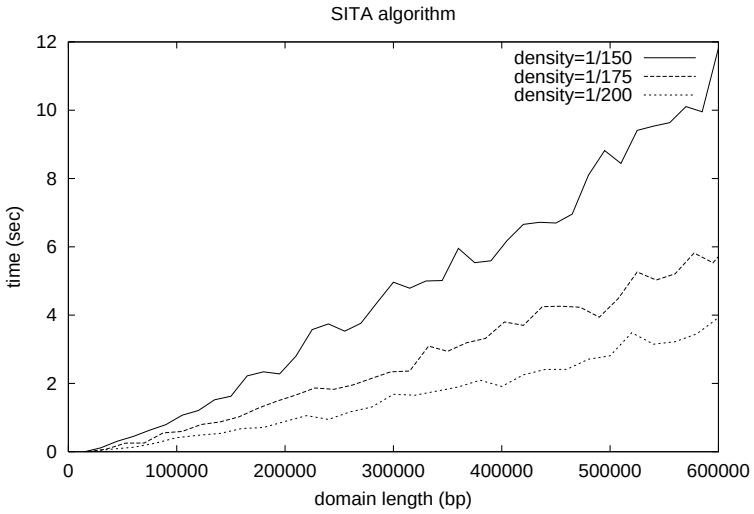

Figure 7. Execution time (user time) for the algorithm SITA, run on randomly generated genomes of increasing length, (*i.e* primers are uniformly distributed over the domain), but of fixed primer density for each curve. We performed our computational experiments on a Pentium 4 (1.6 Ghz) machine on Linux. Each point on the curves is the average of ten runs.

very fast process compared to the space of all potential solutions. Furthermore, as explained in the previous section, the complexity of the algorithms is linear in respect to the genome size.

Thanks to this property, the use of these algorithms is definitely not restricted to small genomes, but can be applied to significantly larger ones.

7 Acknowledgement

This work was partially supported by the French-Bulgarian exchange research program (PAI) RILA'2003 No 06320NM.

We are grateful to Yves Leloir, Michel Gauthier and Nauri Benzacour from the INRA-Rennes microbiology team, introducing the problem to us and for providing us with all data concerning the *Staphylococcus aureus* bacterium.

Special thanks are also due to Jacques Nicolas and Vincent Poirriez for helpful discussions.

References

[1] Kuroda et al., Whole genome sequencing of meticellin-resistant *Staphylococcus aureus*, Lancet, No 357, 2001.

[2] M. Ohnishi, J. Terajima, K. Kurokawa, K. Nakayama, T. Murata, K. Tamura, Y. Ogura, H. Watanabe, T. Yayashi, Genomic diversity of enterohemorrhagic *Escherichia coli O157* revealed by whole genome PCR scanning, Proc. Natl. Acad. Sci. USA, No 99, 2002.

[3] N. Christofides, Graph Theory. An algorithmic approach, Academic Press, London, 1975

[4] M. Gondran and M. Minoux, Graphs and Algorithms, John Willey & Sons, 1984

[5] B. Zakour, M. Gautier, R. Andonov, D. Lavenier, M. Cocher, P. Veber, A. Sorokin, Y. Le Loir, GenoFrag: software to design primers optimized for whole genome scanning by long-range PCR amplification, Nucleic Acids Research, 2004, Vol.32, No. 1, pages 17-24.

[6] R. Andonov, D. Lavenier, N. Yanev, P. Veber, Combinatorial approaches for segmenting bacterium genomes, RR IRISA 1536, Campus de Beaulieu, 35042 Rennes Cedex, France, may 2003. http://www.irisa.fr/bibli/publi/pi/2003/1536/1536.html