HAL
open science

Combinatorial approaches for segmentingbacterium genomes

Rumen Andonov, Nicola Yanev, Dominique Lavenier, Philippe Veber

To cite this version:

Rumen Andonov, Nicola Yanev, Dominique Lavenier, Philippe Veber. Combinatorial approaches for segmentingbacterium genomes. [Research Report] RR-4853, INRIA. 2003. inria-00071730

HAL Id: inria-00071730

https://inria.hal.science/inria-00071730

Submitted on 23 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Combinatorial approaches for segmenting bacterium genomes

R. Andonov N. Yanev D. Lavenier P. Veber

N°4853

Juin 2003

THÈME 3

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--4853--FR+ENG

Combinatorial approaches for segmenting bacterium genomes

R. Andonov * N. Yanev † D. Lavenier ‡ P. Veber §

Thème 3 — Interaction homme-machine,
images, données, connaissances

Projets Symbiose

Rapport de recherche n° 4853 — Juin 2003 — 18 pages

Abstract: Bacterium genome plasticity analysis can efficiently be performed by Long-Range PCR (Polymerase Chain Reaction). The first step requires to split the genome into hundreds of short overlapping segments which, after amplification, are used to sketch the profile of different bacterium strains. The segments to be amplified are determined using a reference bacterium strain. They have to cover the entire genome and to be of nearly identical size. In this report, we use an interval graph for presenting the segments and their position. We show that the most appropriate choice for such segments is equivalent to solving a kind of Shortest Path Problem on a subgraph of the interval graph called covering graph. We propose two optimization models for this problem. These models correspond to two criteria for measuring the quality of the covering; (i) the maximal deviation of the segments lengths from a *given* ideal length should be minimal, (ii) to *find* a length such that that maximal deviation from it is minimal. For each criterium we derive two algorithms of linear complexity in respect to the number of arcs in the graph.

Key-words: combinatorial optimization, network flow, genome plasticity

(Résumé : tsvp)

This work is partially supported by the french-bulgarian project RILA'2003 "Programme d'actions integrées (PAI)" and was performed during a visit of N. Yanev to the SYMBIOSE team, IRISA, Rennes

* University of Valenciennes, Rumen.Andonov@univ-valenciennes.fr
† University of Sofia, Bulgaria, choby@math.bas.bg
‡ Dominique.Lavenier@irisa.fr
§ University of Rennes 1

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Approches combinatoires pour la segmentation de génomes bactériens

Résumé : La plasticité des génomes bactériens peut efficacement être étudiée par LR-PCR (*Long Range Polymerase Chaine Reaction*). La première étape consiste à découper le génome en une multitude de courts segments qui se chevauchent et qui, après amplification – ou non –, représentent le profile de différentes souches bactériennes. Les segments sont déterminés par rapport à une souche de référence. Ils doivent recouvrir l'ensemble du génome et être de taille pratiquement identique. Dans ce rapport, un graphe d'intervalles est utilisé pour représenter les segments et leur position. Nous montrons que le meilleur choix pour déterminer une liste optimale de segments est équivalent à la résolution d'une variante du problème du plus court chemin (*Shortest Path Problem*) dans le graphe d'intervalles. La qualité du recouvrement d'une suite de segments est mesurée selon deux critères : (i) le minimum de la déviation maximale des taille des segments par rapport à une longueure donnée ; (ii) le minimum de l'écart maximal entre le plus long et le plus court segment. Pour chaque critère, nous proposons deux algorithmes de complexité linéaire par rapport au nombre des arcs du graphe.

Mots-clé : optimisation combinatoire, graphe d'intervalles, problème de flot, plasticité des génomes

1 Introduction

A practical way to study the plasticity of bacterium genomes without systematicaly 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 by the LR-PCR. Thus a *profile* corresponding to the amplified – or non amplified – segments will be assigned to each strain. 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.

The segments to be amplified are determined using a reference strain. The goal is to cover the genome with overlapping segments of nearly identical size, knowing that the segments locations are constrained by starting and ending-primers. The distribution of the primer sites along the bacterium genome is non-uniform. There may be 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 greater than the expected size of the segments, the genome is cut into a few number of linear segments, called domains.

Thus, the problem of segmenting a complete bacterial genome is reduced to split each domain into segments of nearly identical size. Along a domain, there are specific positions corresponding to all possible primer sites. The overlapping segments can only start and end at these positions. If we assume, for the sake of simplicity, that a solution is made of a list 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, $N > 100$).

In this paper, we consider diverse approaches for solving this problem. 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. These segments have to satisfy the following conditions:

- The length of any segment varies in the interval $[\underline{L}, \overline{L}]$.
- The length of the overlap between any two segments varies in the interval $[\underline{O}, \overline{O}]$.
- The distance from the begin of the domain to the starting-primer of the first segment has to be no more than D_s. The distance from the ending-primer of the last segment to the end of the domain has to be no more than D_e.

Two cases of this problem have been considered. In the first one we search for a sequence of overlapping segments, each one of size in the interval $[\underline{L}, \overline{L}]$ and as close as possible to a

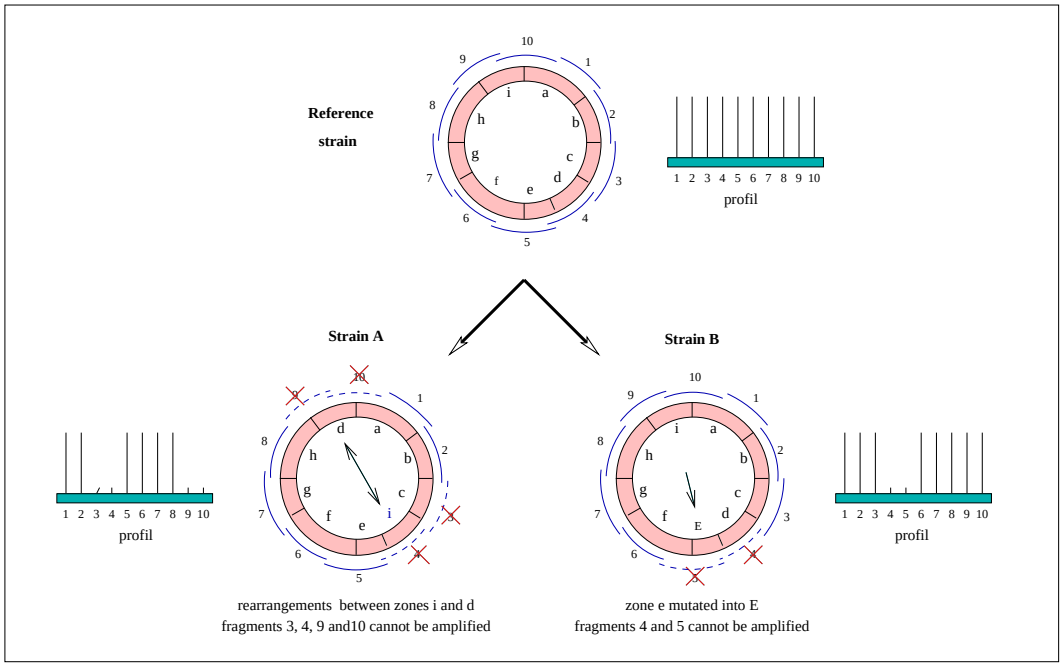

Figure 1: Strategy to study the plasticity of a circular bacterial genome: a reference strain is fully covered with overlap segments. The two extremities of each segments are characterized by two primers. To work properly, the LR-PCR requires these primers to be spaced by a fixed distance (for example 10 Kbp). On the reference strain, the LR-PCR amplifies all the segments, giving a reference profile as depicted above. When the LR-PCR is applied on different strains, some segments will not be amplified depending on the genome variations.

given ideal size L. In the second case, L is considered as an *unknown* and we look for L^*, $\underline{L} \leq L^* \leq \overline{L}$, such that the best segmentation with respect to it is of minimal error. For each case we: (i) formulate a suitable combinatorial optimization model; (ii) program dedicated graph algorithms for solving these models; (iii) analyze the complexity of these algorithms. We are not aware of other algorithms from the literature to have been used for this purpose.

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 are provided in section 5.

2 Graph problem formulation

The formal statement of the problem is as follows: for a given nucleotide sequence, a set S^l of starting-primer sites, and a set S^r of ending-primer sites we can define the set F of *feasible segments*, i.e. the segments $f = [b, e], \quad b < e$ such that:

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

The binary relation "is compatible with" : $f \prec f'$ iff f' starts to the left of the ending-primer site of f and the length of the overlap falls in $[\underline{O}, \overline{O}]$. Let us denote by F_s (resp. F_t) the set of segments which can begin (resp. end) a segmentation.

Definition 2.1. An arbitrary sequence $f_1, f_2, \ldots f_k$ 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 2.2. The *covering graph* of the nucleotide sequence is a directed graph $G(V, A)$:

- the node set $V = F \cup \{s, t\}$, with two additional vertices s and t.
- the arc set

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

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 the segment length L is given and we define the cost function $C_L(f)$ on F as: $\forall f \in F \quad C_L(f) = |l(f) - L|$. This is a 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 $\boldsymbol{G}$. An instance of the problem is given on Fig. 2, while its corresponding covering graph is depicted on Fig. 3. The arcs weights on Fig. 3 are computed according to equation (2) and will be used in section 3.3. If we denote by $\boldsymbol{R}$ the set of paths from $\boldsymbol{s}$ to $\boldsymbol{t}$, the problem to be solved is $\boldsymbol{\min_{r \in R} C_L(r) = C_L^*}$.

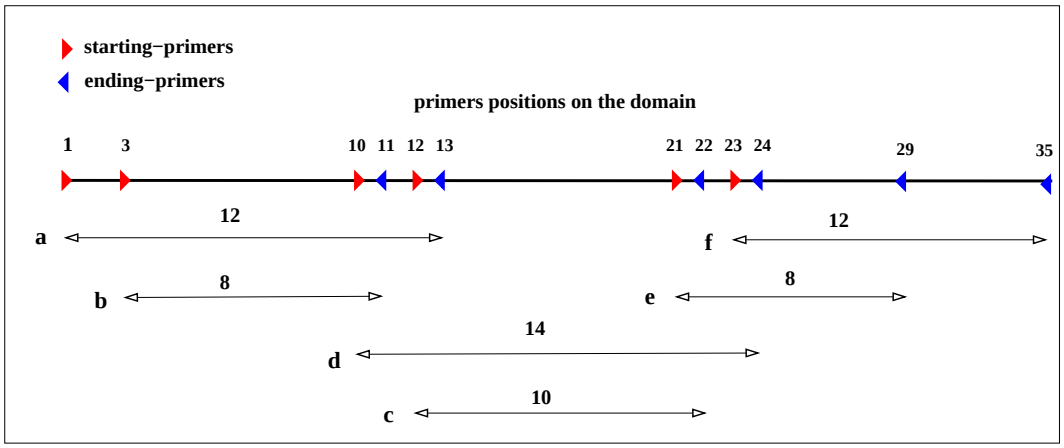

Figure 2: The constraints are $(\underline{L}, \overline{L}) = (6, 14)$, $L = 10$, $(\underline{O}, \overline{O}) = (1, 3)$, $(D_s, D_e) = (3, 7)$ and the domain length D_L equals 35. For the sake of simplicity we do not consider the entire set F, but its subset S containing the feasible segments a, b, c, d, e, f with lengths respectively $(12, 8, 10, 14, 8, 12)$.

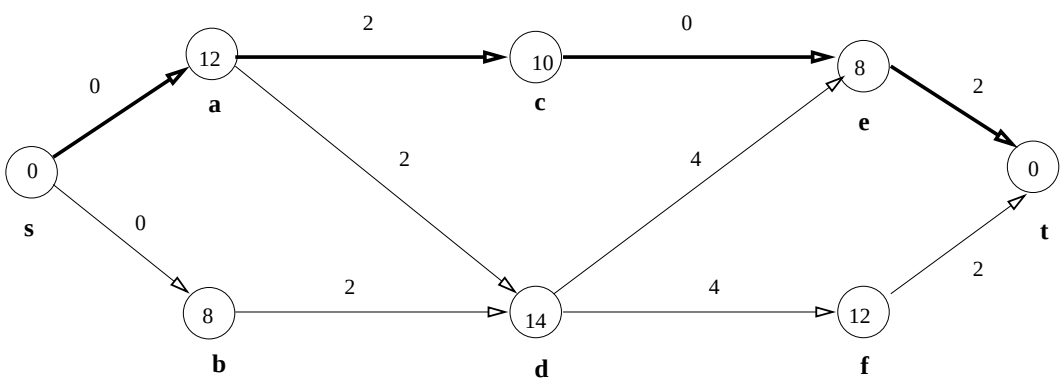

Figure 3: The covering graph corresponding to Fig. 2. The circles contain the segments lengths. When algorithm **SPP-DIH** from section 3.1 or algorithm **SPP** from section 3.2 is applied to this graph, it finds an optimal path $(s - a - c - e - t)$ which contains segments with lengths 12,10 and 8. The error in respect to 10 equals 2.

3.1 An approach using dichotomy

The typical approach for solving such problems is to "catch" C_L^* by sequence of decreasing intervals (brackets) and to exclude from consideration all v_i whose weights fall outside the brackets. In order to save computations and to avoid unnecessary checks, instead of "absolute" costs $C_L(f)$ it is better to use "relative" costs $C'(f)$, defined by: $C'(f) = i$, if $C_L(f)$ is the i-th minimal cost in the set of all costs. One could trivially verify that bracketing of C'^* is equivalent to bracketing of C_L^*. Let us denote by **DFS(s)** a Depth-First-Search procedure, which discovers all vertices of G reachable from s. A call of **DFS(s)** returns *true* if a $(s-t)$-path is found and *false* otherwise. The following algorithm **SPP-DIH(G)** (DIchotomic SPP) finds C'^* by at most $\log_2 k$ calls of DFS(s), where $k = \text{card}\{C_L(f) \quad f \in V\}$.

Algorithm SPP-DIH(G) {A dichotomic search for the shortest $s-t$ path in **G**}
input: directed graph $G(V, A, C')$
output: C'^*-the optimal "length"; acyclic graph **T**, rooted at s.
initialize: $E_l = 0$; $E_r = k$
while $E_l \neq E_r$ **do**
    temporary remove all vertices with cost greater than $\frac{E_l+E_r}{2}$
    if $DFS(s)$
    then $E_r \leftarrow \frac{E_l+E_r}{2}$ and remove all temporary removed vertices
    else $E_l \leftarrow \frac{E_l+E_r}{2}$ and restore all temporary removed vertices
endwhile
set $C'^* = E_l$;
print C'^*, T

Complexity Analysis At the final call ($E_l = E_f$) the **SPP-DIH(G)** procedure builds (on a user request) an acyclic graph **T** where *all paths* from s to t are optimal solutions. The vertices cost transformation could be done by sorting the costs $C_L(v_i)$ into ascending order, i.e. in $O(|V| \log |V|)$ time. Since any call of **DFS(s)** requires $O(|V| + |A|)$ time units, the overall complexity is $O(\log k(|V| + |A|)) = O(|A| \log k)$. When the **SPP-DIH(G)** algorithm uses "absolute" costs $C_L(f)$, its complexity is $O(|A| \log E)$ where E is the maximum allowed error. Though E could be less than $|V|$, using the relatives costs C' instead of C guarantees that **DFS** works on different G on each call.

Remark 3.1. A different approach is used in [3] chapter 8.2 where an algorithm is proposed for finding the path of maximal capacity (the minimal capacity over the arcs of the path). The algorithm is based on a Ford-Fulkerson's alike max-capacity=min-cut (relation when the cut capacity is equal to the maximal capacity over the arcs of the cut). For our problem, the min-max relation should be reversed and with each arc a capacity according to (2) from

section 3.3 should be associated. The complexity of this algorithm is not given in [3], but it is definitely superior to the complexity of the **SPP(G)** algorithm.

3.2 Dynamic programming approach (DP)

When the graph G has no circuit, a DP recurrence gives an algorithm linear in A. Let us denote by $d(i)$ the length (in sense of max instead of sum) of the shortest path from s to v_i and let $\Gamma^{-1}(v)$ be the set of all predecessors of v. Then obviously we have :

$$d_i = \min_{v_j \in \Gamma^{-1}(v_i)} \max\{d_j, c_L(v_i)\} \tag{1}$$

which leads to the following algorithm :

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

[a] topological sort of a graph means to arrange the vertices on a line in such a way that all arcs are from left to right

[b] Note that $k = \arg \min_{v_j \in \Gamma^{-1}(v_i)} \max\{d_j, c_L(v_i)\}$

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

Remark 3.2. 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.

3.3 Network flow formulation

In this section we associate a binary variables x_{ij} with any arc $(i,j) \in A$ of the covering graph $G(V,A)$ and we also provide the arcs with lengths as follows:

$$\begin{cases} C_L(s) = C_L(t) = 0 \\ \forall (i,j) \in A \quad C_L(i,j) = C_L(i) \\ \text{for each path } r = sv_0 \dots v_n t \quad C_L(r) = \max\limits_{i=0\dots n-1} C_L(v_i, v_{i+1}) \end{cases} \tag{2}$$

We can give the following network flow formulation:

$$\text{Minimize } \xi \tag{3}$$

such that:

$$\sum_{j \in \Gamma^+(i)} x_{ij} - \sum_{j \in \Gamma^-(i)} x_{ji} = 0, \forall i \in V \text{ and } i \notin \{s,t\} \tag{4}$$

$$\sum_{i \in \Gamma^+(s)} x_{si} = 1 \tag{5}$$

$$\sum_{i \in \Gamma^-(t)} x_{it} = 1 \tag{6}$$

$$\forall (i,j) \in A \quad C_L(i,j) x_{ij} \leq \xi \tag{7}$$

Into this model, $\Gamma^+(i)$ is the set of arcs out-going from vertex i and $\Gamma^{-1}(i)$ is the set of in-going arcs. Constraints (4), (5) and (6) are network flow representation of the paths from s to t. Constraints (7) serve to make the objective function linear. (3)-(7) is the most natural mixed integer programming formulation of the problem. Note that the constraints (7) are the only non network flow type constraints. Because of these constraints solving directly the model (3)-(7) is not efficient.

We used the ILOG CPLEX [5] solver for this purpose and we clearly observed very poor performance. The way to make this model easily solvable (in polynomial time) is the following. Let us solve (3)-(7) but considering ξ as a constant (say set $\xi = T$). Then, the CPLEX presolver sets $x_{ij} = 0$, when $C_L(i,j) > T$. Otherwise, the constraints (7) have no impact on the binary variables x_{ij}. In this manner the model (3)-(7) is transformed into pure network flow problem which is solved in polynomial time (an integer solution is found simply by solving the relaxed problem) [1]. Note that the solution indicates if a path from s

[1] We observed however, that this does not happen if the CPLEX presolver is set off, since in this case the constraints (7) continue to perturb the network flow formulation (4)-(6).

to t exists or not in respect to the value of T. To perform a dichotomic search on T, by at most $\log_2 T$ solutions of the corresponding network flow model, is equivalent then to solve the model (3)-(7). We do not present here the time for solving it since it is significantly higher than the time given in section 5 which is obtained by dedicated algorithms. Note however that these results have been obtained in a very short time (only for a day) and without writing any code. They served to validate our ideas and to provide the biologists with first results.

4 Searching for the optimal 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. 2 there is a feasible path (a, d, f). The deviation in the lengths of corresponding segments, $(12, 14, 12)$, is very small, 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 $\min_{r \in R} C_L(r) = C_L^*$ to the problem $\min_{L} \min_{r \in R} C_L(r) = C^*$. 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 $r = sv_0 \ldots v_n t$ let $c^r_{min} = \min_{v_i \in r}\{l(v_i)\}$ and $c^r_{max} = \max_{v_i \in r}\{l(v_i)\}$. (Recall that $l(i)$ is the length of the ith segment). Then the following assertion is true:

Theorem 4.1. *The minimal error of segmentation given by 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 $\triangle^* = c_{max} - c_{min} = \min_{r \in R}\{c^r_{max} - c^r_{min}\}$ (In case of multiple optimal solution $\triangle^*$ is represented by the one obtained with the minimum c_{min}). In order to find this optimal value and its respective optimal path (segmentation) we propose two polynomial algorithms. The first one is based on a direct application of the **SPP(G)** algorithm while the second is based on a dynamic programming like approach.

4.1 MUltiple Traverse Algorithm (MUTA)

In the sequel we assume that the **SPP(G)** is properly adjusted to return the couple $(c_*(G), c^*(G))$ where $c_*(G)$ and $c^*(G)$ are resp. the minimal and the maximal cost of the vertices in some "shortest" path in G (note that the length of the path $r = (v_0, v_1, ..., v_r)$, where $v_0 = s$ and $v_r = t$, is given by $\max_{v_i \in r}\{l(v_i)\}$ and that all shortest paths have the same $c^*(G)$ but not necessarily the same $c_*(G)$). Formally, $c^*(G) = \min_{p \in R} \max_{v_i \in p}\{l(v_i)\} = \max_{v_i \in q}\{l(v_i)\}$, where q is an optimal path, and $c_*(G) = \min_{v_i \in q}\{l(v_i)\}$. Now, let k be the minimal number of subsets $S_j \subset V$, s.t. $l(v_i) = l_j, v_i \in S_j$ (this guarantees that $l(v_i) \neq l(v_m)$ if $v_i \in S_j$, $v_m \in S_h$ and $j \neq h$). Let $L_{set} = \{l_1, \ldots, l_k\}$ be ordered by $l_i < l_{i+1}, i = 1, 2, ..., k-1$. Denote by $G_u = (V_u, A_u)$ the subgraph of G with $V_u = \{v_i \in V, l(v_i) \geq u\}$. One could easily prove the corectness of the following assertions:

i. $c_{max} = c^*(G_{c_{min}})$, i.e. if c_{min} is known, the minimal spread problem can be solved by a single call to **SPP(G)** algorithm.

In order to find c_{min} we can use :

ii $c_{min} = \arg\min_{u \in L_{set}} \{c^*(G_u) - u\}$, i.e. the problem is solved by at most k (cardinality of L_{set}) calls to **SPP(G)** algorithm (to find $c^*(G_u)$), applied to a sequence of subgraphs G_u with decreasing cardinality of the vertex set.

The number of calls could be significantly reduced by using the following *skip*|*stop* criteria:

iii_a [*skip*] $c_{min} \notin [u, c_*(G_u))$, i.e. there is no need to apply **arg min** procedure from **ii** to all $u \in L_{set}$
and

iii_b [*stop*] if $c^*(G_u) = -\infty$ (i.e. if u is such that there is no path from s to t in G_u, then the current c_{min} is the optimal one).

We will call **MUTA** the realization of **ii** in terms of calls to the **SPP(G)** algorithm and taking into account **iii**. The input is graph G and the number $u = (l_1, \ldots, l_k)$, used for discarding all vertices of G with cost less than u. The $O(k|A|)$ complexity follows from the cardinality of L_{set} and the complexity of the **SPP(G)** algorithm.

4.2 SIngle Traverse Algorithm (SITA)

We begin this section with an example on Fig. 4 and we give the formal description of the algorithm afterwords. Let us associate to any vertex $i \neq s$ of the covering graph a set $\mathcal{A}_i$ defined as follows:

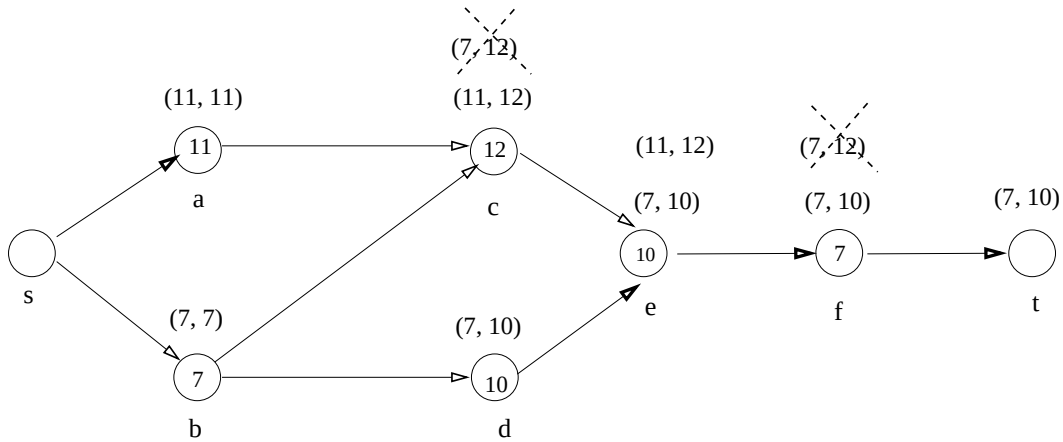

Figure 4: This graph illustrates the links between six segments (a,b,c,d,e,f) and the two artificial vertices s and t. The circles contain the lengths of the corresponding segments. Above any vertex i is given the set $\mathcal{A}_i$ as defined in (8). Note that no one of the elements of the list $\mathcal{A}_e$ 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 (10).

$$\mathcal{A}_i = \{(l,u) \mid \exists r \text{ a path from } s \text{ to } i \text{ such that } c^r_{\min} = l, c^r_{\max} = u\} \tag{8}$$

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. 4, while formally they are computed by the recurrences (9).

$$\mathcal{A}_i = \begin{cases} \{(\overline{L}, \underline{L})\} & \text{if } i = s \\ \bigcup\limits_{j \in \Gamma^-(t)} \{(l,u) \mid (l,u) \in \mathcal{A}_j\} & \text{if } i = t \\ \bigcup\limits_{j \in \Gamma^-(i)} \{(\min(l, l(i)), \max(u, l(i))) \mid (l,u) \in \mathcal{A}_j\} & \text{otherwise} \end{cases} \tag{9}$$

Remark 4.1. Note that the recurrence (9) is correct since the covering graph is acyclic.

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. A covering graph being acyclic, any vertex can be visited only once all its

predecessors have been evaluated. Since the covering graph is without circuits, this property permits to compute these sets 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 $[l, u]$. *A*ny 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:

$$\mathcal{A}^* = \mathcal{A} \setminus \{(l, u) \in \mathcal{A} \mid \exists (l', u') \in \mathcal{A}, [l', u'] \subset [l, u]\} \tag{10}$$

The recurrence (9) is respectively modified:

$$\mathcal{A}_i^* = \begin{cases} \{(\overline{L}, \underline{L})\} & \text{if } i = s \\ \left(\bigcup_{j \in \Gamma^-(t)} \mathcal{A}_j^* \right)^* & \text{if } i = t \\ \left(\bigcup_{j \in \Gamma^-(i)} \{(\min(l, l(i)), \max(u, l(i))) \mid (l, u) \in \mathcal{A}_j^*\} \right)^* & \text{otherwise} \end{cases} \tag{11}$$

The * operation removes from $\mathcal{A}_i$ only pairs (l, u) which are obviously non optimal, because of (10), and we therefore *do not* lose solution. The algorithm **SITA**, is described below.

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 ;
$(c_{min}, c_{max}) \leftarrow \arg\min_{(l,u) \in \mathcal{A}_t^*} (u - l)$;

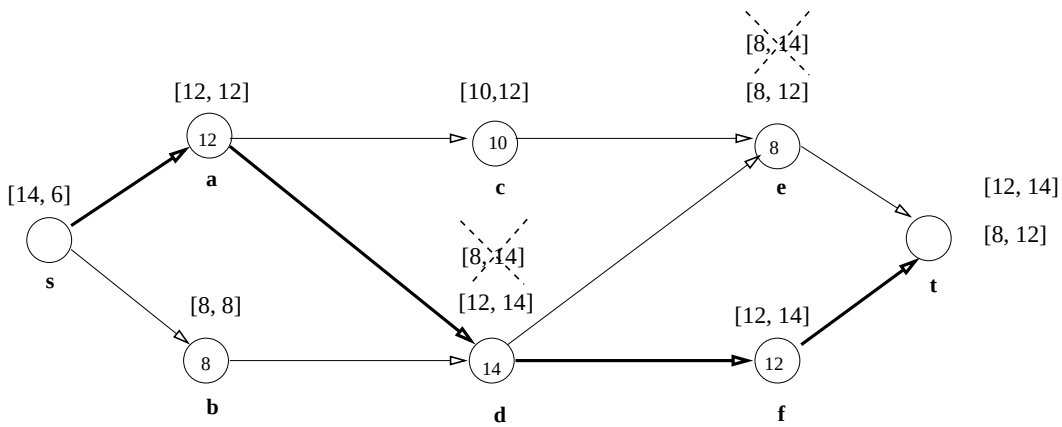

Figure 5: The graph illustrates the behavior of the algorithm **SITA** on the problem instance depicted on Fig. 2. It finds that an optimal spread of size 2 exists and that the associate length L^* equals **13**. The obtained optimal path is $(s-a-d-f-t)$. It contains segments with lengths 12, 14 and 12.

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. $\mathcal{B}^*$).

$$(l, u) \prec (l', u') \text{ iff } (l < l') \wedge (u < u') \tag{12}$$

If we assume now that $\mathcal{A}^*$ and $\mathcal{B}^*$ are sorted according to (12), than 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 (12). Using this observation, we can easily prove that the complexity of the **SITA** algorithm 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 above mentioned 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's 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.

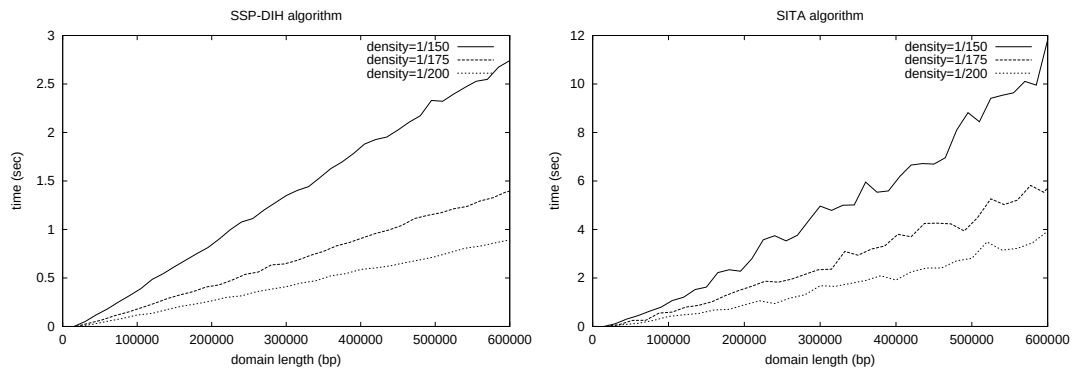

Figure 6: Execution time (user time) for both algorithms, run on randomly generated genomes of increasing length, (*i.e* primers are uniformly distributed over the domain), but of fixed primer density for each curve. We performed our computational experiments on a Pentium 4 (1.6 Ghz) machine on Linux. Each point on the curves is the average of ten runs. The linearity in n is apparent.

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 all 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, $O((\overline{L} - \underline{L})|A|) = O((\overline{L} - \underline{L})^3(\overline{O} - \underline{O})\delta^3 n)$ for **MUTA** 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. 6. 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 $[\mathbf{10,16}], [\mathbf{11,17}], [\mathbf{12,18}], [\mathbf{13,19}], [\mathbf{14,20}]$ then all ($\boldsymbol{C} = (\overline{\boldsymbol{L}} - \underline{\boldsymbol{L}})/\mathbf{2}$) of them will survive the seep of * operation. But whatever is the graph with such costs there is no corresponding domain. The complexity of **MUTA** algorithm should be taken with some precautions because of its quite loose derivation (the complexity of all repetitive calls to **SPP** algorithm is taken at its maximum). In this sense (the value of $\boldsymbol{C}$) only the complexity of **SITA** algorithm is sensitive to the underlying graph, i.e. it could be faster on the set of interval graphs than on its complement. Of course, this needs justification (as well the efficiency of **MUTA** in comparison to **SITA**) on randomly generated graphs (instead of domains).

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 $\boldsymbol{\delta} < \mathbf{10^{-2}}$ (density). Note however that the upper estimate for $\boldsymbol{C}$ is indeed very pessimistic and unlikely to be reached in real life: in all our runs we observed that $\boldsymbol{C} < \mathbf{10}$. In practice, the algorithms are fast and can segment whole genomes in very short time.

6 Conclusion

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

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

We propose four algorithms: **SPP-DIH** and **SPP** for solving the first problem, and **MUTA** and **SITA** for solving the second one. The input of the algorithms is a so-called covering graph which is a subgraph of the interval graph used for presenting the feasible segments (intervals) and their position over the domain. The algorithms are of different complexity and, as a rule, a higher complexity corresponds to a wider range of applications. For instance, **SPP-DIH** works on arbitrary graphs, while for **SPP** the input graph is without circuits. **MUTA** and **SITA** are of higher complexity than the **SPP** algorithm, but the problem they solve is of seemingly higher complexity. Both **MUTA** and **SITA** work only over graphs without circuit, but **MUTA** is expected to be more stable on arbitrary graph without circuits, while **SITA** is supposed to work preferably on interval graphs. For example, its complexity $\boldsymbol{O}(|\boldsymbol{A}||\boldsymbol{V}|)$ is obtained using the relation $\boldsymbol{C} < |\boldsymbol{V}|$, but this upper bound is very rough (in all our experiments on interval graphs $\boldsymbol{C} < \mathbf{10}$). Table 1 recapitulates the characteristics of the proposed algorithms.

problem	algorithm	graph $G = (V, A)$ arbitrary	graph $G = (V, A)$ no-circuit	complexity
L-given	**SPP-DIH**	X	X	$\|A\| \log \|V\|$
	SPP		X	$\|A\|$
L-unknown	**MUTA (SPP-DIH)**	X	X	$\|A\|\|V\| \log \|V\|$
	MUTA (SPP)		X	$\|A\|\|V\|$
	SITA		X	$\|A\|\|V\|$

Table 1: Recapitulation of the domain of applicability and the complexity of the proposed algorithms

The **SPP-DIH** and **SITA** algorithms have been implemented using Objective CAML language. 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. For both programs, the result is an optimal list of segments satisfying either the ideal length criterion (**SPP-DIH** algorithm), or the minimal spread criterion (**SITA** algorithm).

These two algorithms are part of a package called **GenoFrag** which also includes another software, jointly developed with the INRA[2] microbiology team, to generate the set of primers. 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.).

Both algorithms have been tested 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 very fast process compared to the space of all potential solutions. Furthermore, as explained in the previous section, the complexity of both 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

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.

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

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] ILOG CPLEX 7.0 reference manual, www.ilog.com

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399