Rearrangement of DNA fragments: a branch-and-cut algorithm[1]

C. E. Ferreira[a] C. C. de Souza[b] Y. Wakabayashi[a]

[a] *Instituto de Matemática e Estatística*
Universidade de São Paulo
e-mail: [cef,yw]@ime.usp.br

[b] *Instituto de Computação*
Universidade Estadual de Campinas
cid@dcc.unicamp.br

Abstract

We consider a problem that arises in the process of rearrangement of DNA fragments. We present a graph theoretical formulation of the problem and mention some extensions. We show this problem to be $\mathcal{NP}$-hard. A 0/1-integer linear programming formulation of the problem is given and some results of a branch-and-cut algorithm based on this formulation are discussed.

Key words: *Integer Programming, Computational Biology, Branch-and-Cut, Polyhedral Combinatorics.*

1 Introduction

Let Σ be a finite alphabet and $\mathcal{C}$ a collection of strings over Σ. Let ℓ and k be positive integers. If s is a string in $\mathcal{C}$, then we denote by $\ell{-}last(s)$, resp. $\ell{-}first(s)$, the substring of s consisting of the *last*, resp. *first*, ℓ characters of s.

Given two strings s and t such that $\ell{-}last(s) = \ell{-}first(t)$, and t has length n, then we denote by $s|_\ell t$ the string obtained by concatenating s with the $n-\ell$ last characters of t. For example, if $s =$ ACTGTCA and $t =$ TCAGGGT then $s|_3 t$ denotes the string ACTGTCAGGGT. The notation $s_1|_{\ell_1} s_2|_{\ell_2} \ldots |_{\ell_{m-1}} s_m$ is used to denote the string s'_{m-1} obtained as follows. First, we obtain the string $s'_1 := s_1|_{\ell_1} s_2$, then for $i = 2, \ldots, m-1$ we let $s'_i := s'_{i-1}|_{\ell_i} s_{i+1}$. Thus, for s and t as above, and $x =$ GTAACC, the notation $s|_3 t|_2 x$ stands for the string ACTGTCAGGGTAACC.

If a sequence $s = \langle s_1, s_2, \ldots, s_m \rangle$ of strings in $\mathcal{C}$ has the property that for any two of its consecutive strings, say s_j and s_{j+1}, there exists an integer $\ell_j \geq k$ such that $\ell_j{-}last(s_j) =$

[1]This research has been partially supported by CNPq (Project ProComb of ProTeM–II–CC; Proc. 680065/94) and CNPq individual research grants (proc. no. 300752/94-6, 300883/94-3 and 304527/89-0 resp.). The two authors from University of São Paulo have also been supported by FAPESP (Project "Structural and Algorithmic Aspects of Combinatorial Structures", Proc. 96/4505–2).

ℓ_j–*first*(s_{j+1}), and neither s_j is a substring of s_{j+1} nor s_j is a substring of s_{j+1}, then the string $z := s_1|_{\ell_1} s_2|_{\ell_2} \ldots |_{\ell_{m-1}} s_m$ is called a *k-contig* (on $\mathcal{C}$). The sequence s is called a *skeleton* of the k-contig z. If u and v are strings in $\mathcal{C}$, such that u is a substring of v (eventually $u = v$) and v is in a skeleton of a k-contig z, then we say that u is *covered* by z.

For each positive integer k, we define the *Minimum k-Contig Problem*, denoted by MkCP, as follows. Given a collection $\mathcal{C}$ of strings over an alphabet Σ, find a collection $\mathcal{C}'$ of k-contigs with the property that each string in $\mathcal{C}$ is covered by precisely one k-contig in $\mathcal{C}'$, and $\mathcal{C}'$ has the smallest possible cardinality.

This problem occurs in the reconstruction of DNA fragments. An usual strategy to determine the sequence of the bases in a DNA molecule (which can be seen as a string over the alphabet {A, T, C, G}) can be described as follows. First, many copies of this molecule are produced and, afterwards, these copies are (by means of chemical substances) broken into several small pieces that we are able to handle. Then, the problem is *how to "glue" the small pieces in the correct way to reconstruct the original sequence?* (see [M92] and [MS96]).

Many different approaches have been used to solve the DNA Fragment Assembly Problem (see [KM95] for a good survey on the subject). A usual strategy is to apply algorithms designed for the shortest common superstring problem. Kececioglu proposes in his Ph.D. thesis [K91] a natural graph theoretical model to this problem. In this paper we suggest a very similar formulation and the use of polyhedral techniques to develop a branch-and-cut algorithm for the problem. We also present some computational results obtained with this approach.

The MkCP can be applied in the context of DNA Fragment Assembly. The idea is to obtain – for a given positive integer k and a collection $\mathcal{C}$ of pieces – a collection $\mathcal{C}'$ of k-contigs in such a way that the original molecule is completely covered by the k-contigs in $\mathcal{C}'$. A collection $\mathcal{C}'$ with the smallest possible cardinality gives a best possible approximation to the original molecule.

Example 1.1: Let $k = 2$ and $\mathcal{C}$ consist of the following strings:

1. CCTAATGCTT	5. TGTTTAGCCTG	9. TGCGTTTTGTGC
2. TGTTTAGCCTGCGT	6. CCTGCGTTTTGTGCC	10. GACGTAGACA
3. CTGTGTTTAGCCT	7. TTTTGTCCAT	
4. GTAGACAACCCTGTG	8. TTTTGTCCATC	

An optimal solution for the corresponding minimum 2-contig problem consists of a single 2-contig

$$\text{GACGTAGACAACCCTGTGTTTAGCCT} \ldots \text{CTAATGCTTTTGTCCATC},$$

having as a skeleton the sequence $\langle 10, 4, 3, 2, 6, 1, 8 \rangle$. Note that string 5 is a substring of 2, string 7 a substring of 8, and string 9 is a substring of 6.

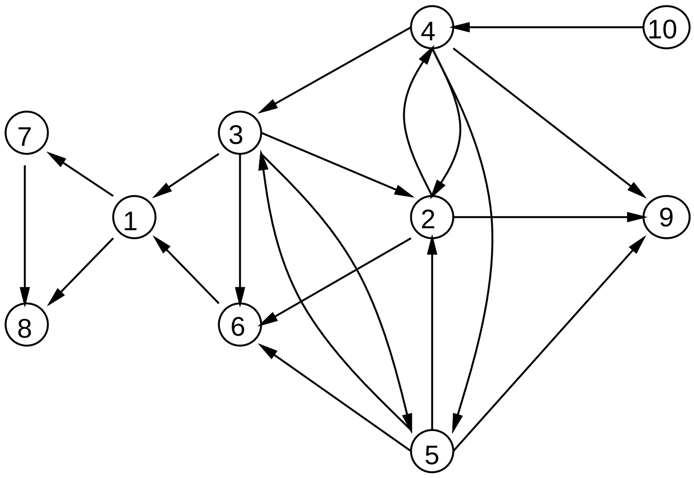
The real problem is surely much more complicated than that, since it involves reverse complements, errors and other issues. We discuss some of these aspects in the sequel.

2 A model using graph theory

Consider an instance of the MkCP consisting of a collection $\mathcal{C}$ of strings. Let $G = (V, A)$ be a directed graph constructed as follows: each node $i \in V$ corresponds to a string $s_i \in \mathcal{C}$, and there is an arc $(i, j) \in A$ if and only if there exists $\ell \geq k$ such that ℓ*–last*$(s_i) = \ell$*–first*(s_j).

It is easy to see that any directed path in G corresponds to a skeleton of a k-contig. Thus, if we find in G a collection of node-disjoint directed paths covering all nodes of G, we have a collection of k-contigs covering exactly once each string in $\mathcal{C}$. There may exist, however, nodes corresponding to strings that are substrings of others and they need not be covered by the collection of paths. These nodes are called *Steiner nodes* and the other ones are called *terminals*. Thus a smallest set of node-disjoint directed paths covering the terminals of G gives a solution for the given instance of the MkCP.

Example 2.1: For the instance given in Example 1.1 the corresponding graph is the following. Nodes 5, 7 and 9 are Steiner nodes.



A similar formulation for the problem has been proposed by Kececioglu [K91], [KM95], the main difference lying in the treatment of the Steiner nodes. Kececioglu's model includes special edges when a string is contained in another one. In this case the correspondence between paths and k-contigs (or its skeletons) is not given. His model does not include either the idea of Steiner nodes.

In the practical application we are interested in, some difficulties arise when the strings are handled. We know that a DNA molecule can be viewed as two parallel strings over { A, C, G, T }, where the second string is called the *reverse complement.* Given a string x over { A, C, G, T }, its reverse complement $\bar{x}$ is the string obtained from x by exchanging each occurence of the caracters A,C,G,T by T,G,C,AXS, respectively, and then reversing the order of the obtained sequence. For example, the reverse complement of the string CCTAATGCTT is AAGCATTAGG. Thus, after a molecule is broken into pieces, one cannot say whether a piece is from the first string or from the reversed one.

Now, suppose we are given a collection $\mathcal{C}$ of strings (some of them can be reversed) and an integer k. Our goal is to find a collection $\mathcal{C}'$ of k-contigs with minimum cardinality, such that, for each string, either it or its reverse complement is covered by a k-contig in $\mathcal{C}'$. We refer to this problem as MkCP^r.

We present now an extension of the model to treat reverse complements. Consider a collection $\mathcal{C}$ of strings and a positive integer k. Initially, for each string $s \in \mathcal{C}$ take its reverse complement $\overline{s}$. Let $\mathcal{C}^r$ be the union of $\mathcal{C}$ with the reverse complements. We construct a directed graph $G^r = (V, A)$ in the following way. Each node $i \in V$ corresponds to a string $s_i \in \mathcal{C}^r$ and there is an arc $(i, j) \in A$ if and only i and j are not reverse complements and there exists $\ell \geq k$ such that $\ell{-}last(s_i) = \ell{-}first(s_j)$. Nodes corresponding to strings which are substrings of others are called Steiner nodes. The objective is to cover every terminal, or its reverse complement, but not both, by a path. Thus, a smallest set of node-disjoint directed paths covering all terminals or their reverse complements gives a solution to this problem.

There are several other versions of the problem that could be of interest. For instance, we may allow that a string in $\mathcal{C}$ (or $\mathcal{C}^r$) can be used more than once to form k-contigs. The graph model is the same but now we are looking for a collection of node-disjoint walks (or trails) covering the nodes of the graph. Steiner nodes are also admitted. Other versions allow concatenation of strings which differ by at most ϵ characters in their final and initial positions, provided that this occurs in a substring of size at least k. These variants are of interest in practical applications since errors may occur during the determination of the substring. For instance, a string ACCCTGCCAT can be wrongly read as ACC**GA**GCCAT or ACC-TGCCA-. We can also handle this problem by giving weights to the arcs in such a way that bigger weights are given when we have more confidence in the concatenation. This weight can be given by using, for instance, the edit distance of the substrings. Then, we search for a collection of paths that covers all terminals and has maximum weight.

In this paper we concentrate our attention to the unweighted versions of the problem (although our approach can be used to handle the weights), with and without considering reverse complements.

3 $\mathcal{NP}$-completeness of MkCP

The version of the problem we are considering here is $\mathcal{NP}$-hard. We prove that the corresponding decision version of the MkCP is $\mathcal{NP}$-complete, even if no Steiner node is allowed.

To show this we first prove the following result.

Lemma 3.1 *Let $G = (V, \mathcal{A})$ be a bipartite directed graph with maximum degree 3 and without cycles of length 2. Then there is a collection $\mathcal{C}$ of strings over an alphabet Σ, such that $|\mathcal{C}| = |V|$ and to each node $i \in V$ corresponds a string $s_i \in \mathcal{C}$ with the property that*

$$(i, j) \in \mathcal{A} \text{ if and only if there exists } \ell \geq 1 \text{ such that } \ell{-}last(s_i) = \ell{-}first(s_j). \qquad (*)$$

Furthermore, the collection $\mathcal{C}$ can be constructed in polynomial time in the size of G.

Proof. Since G is a bipartite graph with maximum degree 3, the arcs of G can be covered by 3 matchings, say M_1, M_2 and M_3.

The construction of the collection $\mathcal{C}$ of strings s_i ($i \in V$), is done in 3 steps, each corresponding to a matching. In step 1 we consider the matching M_1 and assign labels (of length 2) to all nodes of G; in step 2, we consider M_2, and extend these labels to others (of length 3 or 4); and in step 3 we consider M_3 and extend only the labels assigned to the nodes covered by M_3. For $i = 1, 2, \ldots, n$, the string s_i corresponds to the label assigned to the node i, after considering these 3 steps.

The idea behind the constrution is the following. Let us say that a pair of nodes (i, j) is *good* if the labels assigned to i and j, say s'_i and s'_j, satisfy the property that there exists an $\ell \geq 1$ such that $\ell-last(s'_i) = \ell-first(s'_j)$. Thus, our goal is to label the nodes in such a way that all arcs in G become good. For that, in step 1 we label the nodes so that all arcs of M_1 become good (with $\ell = 1$); in step 2 the arcs of M_2 become good (with $\ell = 2$) while the arcs of M_1 remain good (with $\ell = 1$); and in step 3 the arcs of M_3 become good (with $\ell \in \{2, 3, 4\}$) while the arcs in $M_1 \cup M_2$ remain good.

Furthermore, since we want property $(*)$ to hold, we have to make sure that for the final labels, whenever a pair (i, j) is not an arc of G then it is not a good pair.

Let $V = \{1, 2, \ldots, n\}$, and suppose that the arcs of M_1 and of M_3 are all named, each with a different character of length 1 (these characters being different from the characters corresponding to the nodes in V, which we call numbers, and are also assumed to be of length 1).

- Step 1

Consider the matching M_1 and assign labels to the nodes covered by M_1, as follows.

For each arc $a = (i, j)$ in M_1, assign the label $L_i a$ to the node i, and the label aL_j to the node j. Here we are assuming that L_i (a node name) is a string of length 1 and $L_i \neq L_j$ if $i \neq j$.

If there is a node i that is not covered by M_1, then assign to it the label $L_i\$$, where \$ is a new special character (the same for every such node), indicating that the node i is not covered by M_1.

- Step 2

Consider now the matching M_2. Change the labels assigned to the nodes covered by M_2, according to the following rule.

Suppose (i, j) is an arc in M_2 with tail i labelled AB and head j labelled CD. Note that at this stage, each of the symbols A, B, C, D consists of one character (throughout this proof we assume they stand for one character).

Extend the label AB to $AiCB$ (that is, insert the string iC between A an B). Extend the label CD to $CBjD$ (that is, insert the string Bj between C and D).

If there is a node i labelled AB that is not covered by M_2 then give this node the label AiB.

We remark here that if a label contains a character that is a number (corresponding to a node), this character should be seen as a marker that divides the current label into 2 substrings. This marker indicates that in the next step, the insertion (if any, to extend this label) will occur either immediately after, or immediately before this marker. The substring with lenght 2 that is formed in this step is called an *inseparable pair*. For example, in the label $AiCB$ (resp. $CBjD$) the substring CB is an inseparable pair (no character can be inserted between C and B in the next step).

- Step 3

Consider now the matching M_3. If a node is not covered by M_3 then we leave its label unchanged; otherwise, we change its label in the following way.

Let $b = (i, j)$ be an arc of M_3. Suppose the node i is labelled $\alpha i\beta$, and the node j is labelled $\gamma j\delta$, where α and β (resp. γ and δ) are strings having length 1 or 2 (both have length 1 if the corresponding node is not covered by M_2, otherwise, at most one of them has length 2).

Extend the label $\alpha i\beta$ to $\alpha ib\gamma\beta$ (that is, insert the string $b\gamma$ between i and β). Extend the label $\gamma j\delta$ to $\gamma\beta bj\delta$ (that is, insert the string βb between γ and j).

Note that this rule is similar to the rule in step 2. There, we perform the insertion in the middle of the existing label (there is no need of a marker as we know there are only 2 characters), and we insert the node character followed (resp. preceded) by the appropriate character. In step 3 the insertion is performed after the marker (for tails) or before the marker (for heads), and we insert the character corresponding to the arc in M_3 followed (resp. preceded) by the appropriate characters.

The following observations may be helpful to clarify the labelling process. We also give an illustrative example in the sequel.

(1) After step 3 every node has a label of length between 3 and 7.

(2) For $i = 1, 2 \ldots, n$, the character i that occurs in the label s_i assigned to the node i does not occur in any other label.

(3) A character corresponding to an arc in M_3 occurs only in the labels assigned to the nodes incident to this arc.

(4) In the labelling process the *first* and the *last* character remain unchanged. That is, once a label, say AB, is given to a node (in step 1), the final label of this node will start with A and will end with B.

(5) The labels assigned to nodes i covered by M_3 have one of the following forms: $\alpha i b \gamma \beta$ or $\alpha \gamma b i \beta$, where b is the arc of M_3 incident to the node i.

The labels assigned to nodes i not covered by M_3 have one of the forms: $AiBC$, $ABiC$ or AiB.

Let us show that the strings s_i assigned to the nodes of G satisfy property $(*)$.

By construction, it follows immediately that all arcs in G are good, that is,

$$\text{if } \ (i,j) \in A \ \text{ then there exists } \ \ell \geq 1 \text{ such that } \ \ell\text{--}last(s_i) = \ell\text{--}first(s_j).$$

It remains to show the converse of the above statement, that is,

$$\text{if } \ (i,j) \notin A \ \text{ then } \ell\text{--}last(s_i) \neq \ell\text{--}first(s_j), \text{ for every } \ell \geq 1. \qquad (+)$$

From the observation (5), it follows that if $\ell \geq 5$ then for every string s_i, both $\ell\text{--}last(s_i)$ and $\ell\text{--}first(s_j)$ contain the character i and/or a character corresponding to an arc of M_3. Thus, from the observations (2) and (3), we can conclude that $(+)$ holds for $\ell \geq 5$.

It is easy to see that for any two nodes, say i and j, $1\text{--}last(s_i) = 1\text{--}first(s_j)$ if and only $(i,j) \in M_1$.

Thus, it remains to show that $(+)$ holds for $2 \leq \ell \leq 4$. For that, take a node i, and consider the following 3 cases.

Case 1. The string s_i has length 3 or 4

In this case, s_i has one of the following forms: $AiBC$, $ABiC$ or AiB.

It is immediate that, from the observation (2), we can conclude that $(+)$ holds for $3 \leq \ell \leq 4$.

If $s_i = AiBC$ then there exists a unique node that has a label starting with BC. This is the node adjacent to i by an arc of M_2 (that leaves i). Thus, in this case, $(+)$ holds for $\ell = 2$. If $s_i = ABiC$ or $s_i = AiB$, from observation (2) we conclude that $(+)$ holds for $\ell = 2$.

Case 2. The string s_i has the form $\alpha i b \gamma \beta$

We have four cases to analyse, according to the length of the strings γ and β:

Case 2.1. $\gamma = AB$ and $\beta = CD$;

Case 2.2. $\gamma = AB$ and $\beta = C$;

Case 2.3. $\gamma = A$ and $\beta = BC$;

Case 2.4. $\gamma = A$ and $\beta = B$.

The following claims hold, assuming the existence of the string s_i.

(a) There is a unique node whose label starts with $\gamma\beta$; this is the node adjacent to i by an arc of M_3 that leaves i.

(b) Let $\gamma = AB$. If $\beta = CD$ then both γ and β are inseparable pairs, and in this case there is no label starting with BC. If $\beta = C$ then if there exists a node whose label starts with BC, this is the node that is adjacent to i by an arc of M_2.

(c) If there exists a node whose label starts with β, and β has length 2, then this node is unique and is the one that is adjacent to i by an arc of M_2 that leaves i.

The proof of these claims can be obtained by analysing how the labels are generated. From these claims we can conclude that (+) holds for each of the 4 cases above.

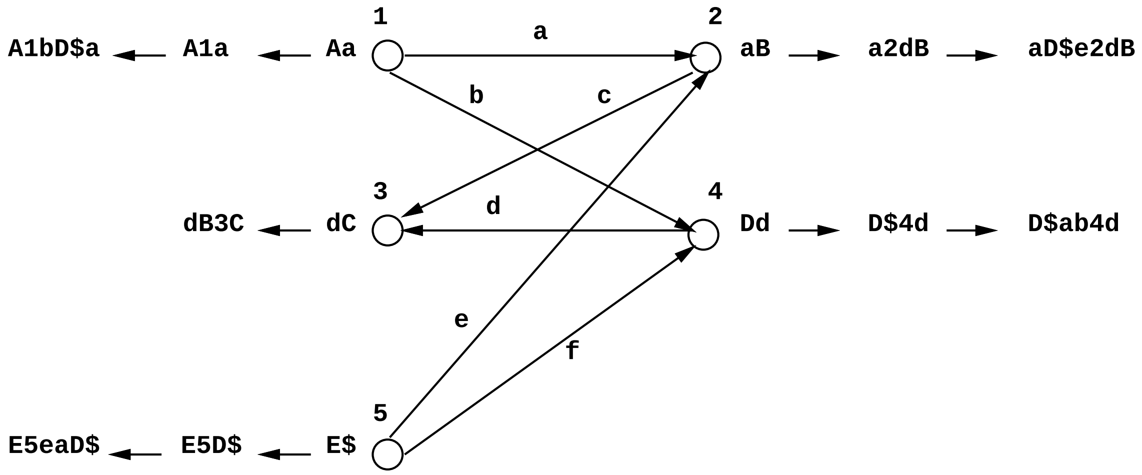
Case 3. The string s_i has the form $\alpha\gamma b i\beta$

In this case the proof is simple, as the only not straigthforward case is when $\ell = 2$ and β has length 2. But in this case, we know that β is an inseparable pair formed in step 2, and there is a unique node adjacent to i by an arc of M_2 (leaving i).

The analyses above complete the proof that the collection $\mathcal{C}$ satisfies $(*)$. Since it is immediate that the construction of $\mathcal{C}$ can be carried out in polynomial time in the size of the input graph, the proof of the lemma is now complete.

□

Example: Let G be the graph defined in the figure below. The node names corresponding to the nodes 1, 2, 3, 4 and 5 are A, B, C, D and E, respectively. The arc names are indicated in the figure. Consider that $M_1 = \{a, d\}$, $M_2 = \{c, f\}$ and $M_3 = \{b, e\}$. We show in the figure, the node labels after each step of the construction.



Theorem 3.2 *For each integer $k \geq 1$ the decision version of MkCP is $\mathcal{NP}$-complete.*

Proof. Let DMkCP denote the decision version of MkCP: *given an integer $p \geq 1$, decide whether a given collection of strings can be covered by at most p k-contigs.* Clearly, this problem is in $\mathcal{NP}$.

Let us consider first the case $k = 1$, which we denote here by DM1CP.

We show that the following problem can be reduced to DM1CP: *given an integer $p \geq 1$ and a directed graph with maximum degree 3, decide whether this graph can be covered by at most p node-disjoint paths.* This problem is $\mathcal{NP}$-complete, since for $p = 1$ this is the Hamiltonian Path Problem (see [GJ79]).

It is immediate that the problem above remains $\mathcal{NP}$-complete when the input graph is bipartite with maximum degree 3 and with no cycles of length 2. By Lemma 3.1, given such a graph G, we can construct in polynomial time (in the size of G) a collection $\mathcal{C}$ of strings that constitutes an instance of DM1CP.

Since the collection $\mathcal{C}$ satisfies property $(*)$, it follows that the input graph G can be covered by at most p node-disjoint paths if and only if $\mathcal{C}$ can be covered by at most p k-contigs. Thus, DM1CP is $\mathcal{NP}$-complete.

Now it remains to show that DMkCP is $\mathcal{NP}$-complete for each integer $k \geq 2$. For that, we prove two claims.

Claim 1. Let $k \geq 1$. If DMkCP is $\mathcal{NP}$-complete then DMtCP is $\mathcal{NP}$-complete for $t = 2 * k$.

> (Proof of Claim 1) Given an instance of DMkCP consisting of a collection $\mathcal{C}$ of strings s_i, we construct an instance of DMtCP consisting of a collection $\mathcal{C}'$ of strings s_i', as follows. The collection $\mathcal{C}'$ has exactly $|\mathcal{C}|$ strings, and each string s_i' is obtained from s_i by replacing each character in s_i with two copies of this character. That is, if $s_i = ABCB$, then $s_i' = AABBCCBB$.

It is easy to see that the collection $\mathcal{C}$ can be covered by at most p k-contigs if and only the collection $\mathcal{C}'$ can be covered by at most p $(2*k)$-contigs.

Claim 2. Let $k \geq 2$. If DMkCP is $\mathcal{NP}$-complete then DMtCP is $\mathcal{NP}$-complete for $t = 2*k-1$.

(Proof of Claim 2) Let be given an instance of DMkCP consisting of a collection of $\mathcal{C}$ strings s_i. We construct an instance of DMtCP consisting of a collection $\mathcal{C}'$ of strings, with precisely $|\mathcal{C}|$ strings s'_i, defined as follows. Each s'_i is obtained from s_i by inserting a new character, say \$, between every two consecutive characters in s_i. We take the same character \$ for all strings s'_i. Thus, if if $s_i = ABCB$ then $s'_i = A\$B\$C\$B$; and if $s_j = CCCA$ then $s'_j = C\$C\$C\$A$. Note that if s_i has length q then s'_i has length $2*q-1$.

It is not difficult to see that the collection $\mathcal{C}$ can be covered by at most p k-contigs if and only the collection $\mathcal{C}'$ can be covered by at most p $(2*k-1)$-contigs.

As we have proved that DM1CP is $\mathcal{NP}$-complete, using the two claims above we can conclude that DMkCP is $\mathcal{NP}$-complete for every $k \geq 1$.

$\square$

Remark. We note that for a graph with n nodes, the construction of the strings in $\mathcal{C}$ in Lemma 3.1 requires an alphabet Σ with at most $3n+1$ characters. These strings can be encoded to strings of length $O(\log(n))$ over an alphabet with 4 (or even 2) characters. However, with such an encoding, we cannot assure that DMkCP remains $\mathcal{NP}$-complete for each $k \geq 1$ (at least using the results we have presented).

4 Integer Programming Formulations for MkCP and MkCP^r

In this section we show integer programming formulations for MkCP and MkCP^r. These formulations are based on flow techniques. We consider the directed graph $G = (V, A)$ defined in the last section and add two new nodes, a source s and a sink t, and arcs linking s to all nodes in V, and arcs linking all nodes in V to t. We associate with each arc $(i, j) \in A$ a variable x_{ij} with the following interpretation:

$$x_{ij} = \begin{cases} 1, & \text{if the arc } (i,j) \text{ is in a path;} \\ 0, & \text{otherwise.} \end{cases}$$

Moreover, for all $i \in V$, we let s_i (resp. t_i) be the variable corresponding to the arc (s, i) (resp. (i, t)).

Thus, a 0/1-ILP formulation for the problem is given by

$$
(P) \qquad
\begin{array}{rlll}
z = & \min \displaystyle\sum_{i \in V} s_i & & \\
& \displaystyle\sum_{j \in \delta^-(i)} x_{ji} + s_i = \sum_{j \in \delta^+(i)} x_{ij} + t_i & \text{for all } i \in V, & (1) \\
& \displaystyle\sum_{j \in \delta^+(i)} x_{ij} + t_i = 1 & \text{for all } i \in Z, & (2) \\
& \displaystyle\sum_{j \in \delta^+(i)} x_{ij} + t_i \leq 1 & \text{for all } i \in V \setminus Z, & (3) \\
& \displaystyle\sum_{e \in E(C)} x_e \leq |C| - 1 & \text{for all } ? \neq C \subseteq V, & (4) \\
& x_{ij} \in \{0,1\} & \text{for all } (i,j) \in A, & (5) \\
& s_i, t_i \in \{0,1\} & \text{for all } i \in V, & (6)
\end{array}
$$

where $E(C) := \{(i,j) \in A \mid i,j \in C\}$, $\delta^-(u) := \{v \mid (v,u) \in A\}$ and $\delta^+(u) := \{v \mid (u,v) \in A\}$.

The first set of inequalities means that the solution x must be a feasible flow, i.e., for every node that is not a source or a sink the flow conservation law must be satisfied. Inequalities (2) and (3), respectively, guarantee that the terminals must be covered, and the Steiner nodes may be covered by at most one path. Inequalities (4) eliminate the possibility of choosing arcs that induce a cycle.

It is not difficult to check that a 0/1-vector x satisfies (1) to (6) if and only if the set of arcs $a \in A$ with $x_a = 1$ induces a collection of node-disjoint paths in G that covers all nodes in Z. Moreover, since we minimize the number of arcs leaving the node s, the number of such paths is minimized.

It should be noted that the number of constraints (4) is exponential. However, the separation problem corresponding to these inequalities, called *subtour elimination constraints*, can be solved in polynomial time (see [PG85]. This means that the optimum value of the relaxed LP (substituting constraints (5) by $0 \leq x_{ij} \leq 1$ for all $(i,j) \in A$ and (6) by $0 \leq s_i \leq 1$, $0 \leq t_i \leq 1$ for all $i \in V$) can be calculated in polynomial time (this follows from a result due to Grötschel, Lovász and Schrijver [GLS88]).

This formulation can be extended for the model with reverse complements. For that, consider the graph $G^r = (V, A)$ as defined in the last section, and include nodes s and t linked to all nodes in V, as above. Let us represent the reverse complement of i by $\bar{i}$. We can now formulate MkCP^r as follows.

$$
(P^r) \quad
\begin{array}{lll}
z^r = \min & \sum_{i \in V} s_i + s_{\bar{i}} & \\
& \sum_{j \in \delta^-(i)} x_{ji} + s_i = \sum_{j \in \delta^+(i)} x_{ij} + t_i & \text{for all } i \in V, \quad (1') \\
& \sum_{j \in \delta^+(i)} x_{ij} + t_i + \sum_{j \in \delta^+(\bar{i})} x_{\bar{i}j} + t_{\bar{i}} = 1 & \text{for all } i \in Z, \quad (2') \\
& \sum_{j \in \delta^+(i)} x_{ij} + t_i + \sum_{j \in \delta^+(\bar{i})} x_{\bar{i}j} + t_{\bar{i}} \le 1 & \text{for all } i \in V \setminus Z, \quad (3') \\
& \sum_{e \in E(C)} x_e \le |C| - 1 & \text{for all } ? \ne C \subseteq V, \quad (4') \\
& x_{ij} \in \{0,1\} & \text{for all } (i,j) \in A, \quad (5') \\
& s_i, t_i \in \{0,1\} & \text{for all } i \in V. \quad (6')
\end{array}
$$

Inequalities $(2')$ and $(3')$ treat the nodes and its reverse complements together. Constraints $(2')$ guarantee that each terminal is covered by exactly one path, and it reaches either the node or its reverse complement, but not both. Constraints $(3')$ assure that a Steiner node or its reverse complement is used by at most one path.

Similarly to the formulation for MkCP, inequalities $(4')$ avoid the existence of subtours in the solution. Moreover, these inequalities can be lifted in this version, as we show in the following lemmas.

Lemma 4.1 *Let C be a subset of nodes in V that induces a violated subtour elimination inequality. Then, there is no node i such that $\{i, \bar{i}\} \subseteq C$.*

Proof. Immediate consequence from $(1')$, $(2')$ and $(3')$. □

In the next lemma, the notation $[S : S']$ for node sets S and S', stands for the set of arcs going from S to S'.

Lemma 4.2 *Let C be a subset of nodes in V and consider the corresponding subtour elimination inequality*

$$\sum_{e \in E(C)} x_e \le |C| - 1.$$

Then, the following lifted inequality is also satisfied by all feasible solutions of MkCP^r, formulated as above.

$$\sum_{e \in E(C)} x_e + \sum_{e \in E(\overline{C})} x_e + \sum_{e \in [C:\overline{C}]} x_e + \sum_{e \in [\overline{C}:C]} x_e \le |C| - 1,$$

where $\overline{C}$ denotes the set of reverse complements of the nodes in C.

Proof. Consider an arbitrary feasible solution S for MkCP^r whose incidence vector is given by x^S. Let C_1 be the set of nodes in C incident to an arc e in $E(C) \cup [\overline{C} : C]$ with $x_e^S = 1$. Similarly, let C_2 be the set of nodes in C whose reverse complements are in $\overline{C}$ and are incident to the arcs in $E(\overline{C}) \cup [C : \overline{C}]$. Constraints $(2')$ and $(3')$ guarantee that $C_1 \cap C_2 = \emptyset$, and therefore $|C_1| + |C_2| \leq |C|$. Moreover, $\sum_{e \in E(C)} x_e^S \leq |C_1| - 1$ and $\sum_{e \in E(\overline{C})} x_e^S \leq |C_2| - 1$.

Now, let p be the number of arcs in the set $[\overline{C} : C] \cup [C : \overline{C}]$ with $x_e^S = 1$, that is:

$$\sum_{e \in [C:\overline{C}]} x_e^S + \sum_{e \in [\overline{C}:C]} x_e^S = p.$$

For each arc (u, v) contributing to this summation, it follows that the nodes $\overline{u}$ and $\overline{v}$ cannot be incident to any other arc in S (as the constraints $(1')$ to $(3')$ have to be satisfied).

Thus, $\sum_{e \in E(C)} x_e^S + \sum_{e \in E(\overline{C})} x_e^S \leq |C_1| + |C_2| - 2p - 1$ (if both C_1 and C_2 are nonempty this bound is $|C_1| + |C_2| - 2p - 2$).

Summing up these valid inequalities we obtain the inequality:

$$\begin{aligned} \textstyle\sum_{e \in E(C)} x_e^S + \sum_{e \in E(\overline{C})} x_e^S + \sum_{e \in [C:\overline{C}]} x_e^S + \sum_{e \in [\overline{C}:C]} x_e^S &\leq |C_1| + |C_2| - 2p - 1 + p \\ &\leq |C| - p - 1 \\ &\leq |C| - 1. \end{aligned}$$

□

5 Branch-and-cut algorithms

The method we have used to tackle both problems MkCP and MKCP^r is based on the linear programming relaxation combined with *branch-and-bound* and cutting-planes, the so-called *branch-and-cut* technique.

For that, we consider the polyhedron defined as the convex hull of the feasible (integer) solutions of (P), which we denote by $P_k(G)$, that is,

$$P_k(G) := \text{ conv } \{x \in \mathbb{R}^A |\ x \text{ satisfies (1) to (6)}\}.$$

Analogously, we consider the polyhedron corresponding to MkCP^r:

$$P_k^r(G) := \text{ conv } \{x \in \mathbb{R}^A |\ x \text{ satisfies } (1') \text{ to } (6')\ \}.$$

Both polyhedra are not full-dimensional, since their descriptions include equations. In this paper we do not mention results concerning classes of facet-defining inequalities for both polyhedra. We have found such an inequality which we could not generalize and these studies might be addressed in a future paper. In our present implementation of a branch-and-cut algorithm we have used only the inequalities presented in the last section.

As we have observed before, although there exists an exponential number of inequalities of type (4) (resp. (4′)) they can be separated efficiently (see [CP80]). We have implemented a separation heuristic for these inequalities, based on contractions of the graph, as described in [PG85]. In the case of inequalities (4′) these are then lifted as indicated in Lemma 4.2.

It is interesting to note that, if the graph is acyclic, then there are no violated inequalities of type (4), and the corresponding polytope $P_k(G)$ is integral, i.e., has only 0/1-vertices. In this case, since linear programming can be solved in polynomial time, the problems is easy.

The idea of the approach is to start with a relaxation of the polyhedron $P_k(G)$ (resp. $P_k^r(G)$) and to solve iteratively better approximations of this polyhedron, obtained by using facet-defining or, at least, valid inequalities that are violated by the optimal solution of the current relaxation. When we are not able to find any violated valid inequality we fix the value of some variables and proceed in a branch-and-bound fashion.

We have implemented two versions of the algorithm: one for the model without reverse complements, and one that handles it. In both cases we begin with the LP given by constraints (1), (2) and (3) (resp. (1′), (2′) and (3′)), and use, as mentioned above, a separation heuristic for the subtour elimination constraints. In the version with reverse complements the subtour elimination inequality is lifted, as indicated in Lemma 4.2. Whenever no violated inequality is found by the separation routine we perform a branching step.

The value of the current LP relaxation is used by a primal heuristic. The idea is to use the arcs with bigger values in the current solution and try to cover all terminals.

We have tested both versions with two types of instances. In the first type the original string is 5000 characters long and has been generated randomly on the alphabet {A, T, C, G}. Each substring has length between 500 and 700 characters. Each substring is generated by choosing randomly its length and also the position it starts in the original string. For the model allowing reverse complements the sequence is reversed with probability 50%. The second type of instances corresponds to real data, given by DNA molecules. For the problem with reverse complements, we use some of the instances presented in the DIMACS Challenge 95.

5.1 Instances of MkCP

For the version without reverse complements we are able to solve instances with up to 200 nodes within 1s (in a Sun Sparc 1000), and in many cases the graph is acyclic, and therefore the first LP is sufficient to provide the optimal solution. We have obtained similar results for the random and the real instances.

Table 1 summarizes the results for the random instances. In the first column we indicate the number of nodes and arcs in the graph. In the second column we indicate the value of k of the corresponding MkCP problem. Columns 3 and 4 show the number of LPs solved and the number of nodes in the branch-and-bound tree, respectively. Finally, in the last column we present the CPU time spent to solve the problems in a SPARC 1000. We use CPLEX to solve the LP in each iteration.

We have also tested our approach with some real instances (obtained from the author of

# nodes	# arcs	k	# LP	# BB nodes	CPU Time (sec.)
10	11	3	1	1	0.10
20	47	10	1	1	0.15
30	105	5	1	1	0.15
50	330	10	1	1	0.11
50	333	5	23	4	0.47
70	589	10	1	1	0.14
70	598	5	1	1	0.17
80	798	5	1	1	0.16
100	1266	5	23	4	0.90
200	4566	5	1	1	0.32

Table 1: Computational results of the version without reverse complements.

# nodes	# arcs	k	# LP	# BB nodes	CPU Time (sec.)
273	358	12	1	1	0.15
273	358	10	1	1	0.16
273	366	8	1	1	0.23
273	392	6	1	1	0.19
273	595	4	142	12	5.40

Table 2: Computational results for instances arising from `hss`.

[M92]). The DNA sequence (denoted by `hss`) has length 10532 and has been cut into 273 pieces. We have been able to solve to optimality different instances of the problem, obtained by using different values of k. See Table 2.

Our explanation to the fact that the problems are not difficult to solve is that the corresponding graphs are, in most cases, acyclic, and therefore, the solution of the LP relaxation is integral. Further tests we have carried out with random graphs (with many cycles) indicate that the problem becomes much more difficult to solve, even for small instances.

5.2 Instances of MkCP^r

We have obtained similar results for the version with reverse complements when testing with random instances. The results are presented in Table 3.

We have got memory overflow problem when we have tested the version with reverse complements, as in this version the number of nodes of the graph is twice as large. We considered four instances of DIMACS benchmark; these are described in Table 4.

We have considered the problem for $k = 10$. Note that the number of arcs in the graphs goes from 10331 to 35774. Since we could not solve some of these instances because of insufficient

# nodes	# arcs	k	# LP	# BB nodes	CPU Time (sec.)
10	30	3	1	1	0.19
20	118	3	1	1	0.11
30	280	3	12	3	0.30
50	682	5	153	26	3.89
70	1336	5	1	1	0.22
80	1732	5	1355	246	1:30.48
100	2587	5	22	4	1.21

Table 3: Computational results of the version with reverse complements.

Name	# pieces
`b1496`	1811
`b2126`	1504
`l247`	1704
`l518`	1910

Table 4: DIMACS Challenge benchmark.

memory space (we use a Sun Sparc 1000 with 702 MBytes memory), we have decided to generate many subinstances of the original ones, in order to detect how far we could go with our code to solve practical instances. These instances have been generated by choosing a random subgraph of the complete instance, with a certain percentage of the total number of nodes. Tables 5, 6, 7 and 8 summarize the results we have obtained. The second column of these tables shows the value of the optimal solution of the problem.

The computational experiments carried out lead us to the following observations. The value of the lower bound of the first LP relaxation is already the value of the optimal solutions for all instances we have tested. We have spent most of the time to prove that this was indeed the case. Thus, with better primal heuristics we could possibly obtain better performances. This stresses our belief that the formulation we propose for the problem can be used satisfactorily to test primal heuristics. For instances arising from `b2296`, `l518` and `l247` the value of the

# nodes	# arcs	sol.	# LP	# BB nodes	CPU Time (sec.)
342	85	136	1	1	0.18
722	411	228	1	1	0.26
1074	892	308	56	20	2.05
1436	1540	no sufficient memory			

Table 5: Computational results for `b1496` (# nodes=3622, # edges=10331).

# nodes	# arcs	sol.	# LP	# BB nodes	CPU Time (sec.)
286	186	40	1	1	0.21
608	874	45	1	1	0.21
892	1950	30	1	1	0.24
1206	3426	25	526	162	49.96
1486	5152	17	1676	516	7:23.46
1804	7640	no sufficient memory			

Table 6: Computational results for `b2296` (# nodes=3008, # edges=20940).

# nodes	# arcs	sol.	# LP	# BB nodes	CPU Time (sec.)
362	320	39	1	1	0.19
770	1398	42	1	1	0.27
1138	3070	23	21	8	1.18
1524	5744	13	26	10	2.42
1888	8700	7	51	20	6.32
2282	12964	no sufficient memory			

Table 7: Computational results for `l518` (# nodes=3820, # edges=35774).

# nodes	# arcs	sol.	# LP	# BB nodes	CPU Time (sec.)
320	226	43	1	1	0.11
688	1040	51	1	1	0.21
1012	2136	40	1	1	0.29
1690	6176	15	1	1	0.51
2040	8958	3	1	1	0.80
2220	10748	no sufficient memory			

Table 8: Computational results for `l247` (# nodes=3408, # edges=25990).

optimal solution of the subinstance decreases when the graph becomes more dense. This is an indication that the approach can be used iteratively with decreasing values of k to provide a solution to the Fragment Assembly Problem. The only instance for which our approach has not performed well was `b1496`, whose graph has some heavily connected components (note that the number of arcs is small).

6 Conclusions and future research

The computational results we have presented in this paper show that the ILP formulation we suggest for MkCP and MkCP^r can be satisfactorily used to solve medium size instances of the problems. Our aim is to increase the size of the instances we are able to solve to optimality. For that, we need other classes of valid inequalities to improve the lower bounds in each node of the branch-and-cut tree and better primal heuristics. Moreover, polyhedral investigations on facet-defining inequalities for the polyhedra we have defined is one of our goals for future work. It is still a rather long way until we are able to solve instances with the size of interest for the computational biology community. DIMACS instances could not be satisfactorily handled because of insufficient memory space, but the results we were able to derive allow us to say that the approach can be useful for determining the structure of the original DNA molecule for "real world" examples. The DNA molecules that the community of computational biologists intend to determine have length of millions of bases; and typically, these molecules are broken into several hundreds of pieces. Another possible direction to continue research is to extend the model to allow the use of the same piece several times (i.e. covering the graph by node-disjoint walks) and to handle errors.

7 Acknowledgement

We thank J. Meidanis for bringing this problem into our attention and making available some of his data sets (as the instance `hss`). The discussions we had during the elaboration of this work were very stimulating.

References

[CP80] H. Crowder, M.W. Padberg, "Solving large-scale symmetric traveling salesman problems to optimality", *Management Science* 26, 495-509 (1980).

[GJ79] M.R. Garey and D.S. Johnson, *Computer and Intractability - A Guide to the Theory of NP-Completeness*, Freeman, New York (1979).

[GLS88] M. Grötschel, L. Lovász and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin (1988).

[K91] J.D. Kececioglu, "Exact and Approximation Algorithms for DNA Sequence Reconstruction", *PhD. Thesis*, University of Arizona, Tucson (1991).

[KM95] J.D. Kececioglu and E.W. Myers, "Combinatorial Algorithms for DNA Sequence Assembly", *Algorithmica* 13, 7-51 (1995).

[M92] J. Meidanis, "Algorithms for Problems in Computational Genetics", *Ph.D. Thesis*, University of Wisconsin-Madison (1992).

[MS96] J. Meidanis and J.C. Setubal, *Computational Molecular Biology*, PWS, Boston (to appear in 1997).

[NW88] G.L. Nemhauser and L.A. Wolsey, "Integer and Combinatorial Optimization", Wiley, (1988).

[PG85] M.W. Padberg and M. Grötschel, "Polyhedral Aspects of the Travelling Salesman Problem II : Computations", in E.L. Lawler, J.K. Lenstra and A.H.G. Rinooy Kan eds., *The Travelling Salesman Problem*, Wiley, New York (1985).