Further thoughts on the syntenic distance between genomes

Nadia Pisanti [◇] Marie-France Sagot [†‡*]

[◇] Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy.

[‡] Institut Gaspard-Monge, Université de Marne-la-Vallée, France

[*] Inria Rhône-Alpes, Laboratoire de Biométrie et Biologie Évolutive,
Université Claude Bernard (LyonI)
43, Bd du 11 Novembre 1918
69622 - Villeurbanne cedex, France

Corresponding author: pisanti@di.unipi.it, tel.:(39)(050)22 12 773, fax:(39)(050)22 12 726.

Abstract

The syntenic distance between two multi-chromosomal genomes is the minimum number of fusion (union), fission (split) and translocation (exchange) operations necessary to transform a genome into another. Genomes are therefore seen as unlabeled bags of genes and the syntenic distance as the number of moves needed to obtain the same bags. This paper introduces the following new results concerning syntenic distance:

- Using a direct proof, we solve a conjecture by Liben-Nowell [1] for the syntenic diameter size, that is, for the length of the longest optimal move sequence for an instance of size n, showing that it is, in fact, not $2n-3$ but $2n-4$ (an independent and different proof of this was provided by Kleinberg and Liben-Nowell in [2]). The structure of the proof further enables us to characterize the minimum number of translocations an optimal sequence of moves must have. This has important consequences for algorithmical purposes. The result is then generalized to what we call bidimensional syntenic diameter.
- When the initial instance has more than one component, we suggest a way of performing inter-component moves, something that was not done in previous algorithms. We show however that, even in simple cases with more than two components involved, an optimal solution for a general instance of the problem using such moves is $\mathcal{NP}$-hard to obtain.
- We introduce, we believe, a simpler, more general and efficient algorithm for obtaining a move sequence that is optimal for the cases that Liben-Nowell and Kleinberg called in [3] nested synteny, exact and linear (this last one is the case addressed in [3]).
- We sketch an extension of the algorithm which we conjecture may optimally solve the connected nested synteny (without requiring linearity).
- We prove an inclusion theorem for the general synteny that is an extension of the one shown for linear synteny in [3].

Keywords: Genome rearrangement, genome distance, syntenic distance, nested synteny, syntenic diameter, discrete algorithms, computational biology.

1 Introduction

Similarity between biological sequences is frequently measured by counting point mutations (substitutions, deletions and insertions). However, mutations happen also at a larger scale. Whole fragments of genomes, involving sometimes thousands of DNA bases, often containing one or more genes, may thus be deleted, moved about, reversed, copied (duplicated) or exchanged (between chromosomes in the case of multi-chromosomal genomes). One may therefore wish to measure intra or inter-species distances based on such larger scale rearrangement events happening at a genomic level, instead of on the history of point mutations.

Methods for measuring such distances may be classified into three main categories depending on what type of rearrangement events are allowed and what kind of data is given as input to the algorithm. Data basically consist of genome representations (or representations of significant parts of it, such as chromosomes). The representations come under the form of either one or more sequences of genes (this will be the case when the order of the genes along the chromosome is taken into account), or sets of genes (when that order is ignored). Each sequence or set corresponds to a chromosome. The types of rearrangement events allowed by the various approaches may of course depend on the data considered. Most *rearrangement distances* take the gene order into account and allow for one or more type of event among deletion, duplication, reversal, inter or intra-chromosome transposition and inter-chromosome translocation [4] [5] [6] [7] [8] [9]. In the case of a *breakpoint distance* [10], the gene order is considered as well, but the type of rearrangement events that may have changed it is not fixed. Finally, the *syntenic distance* [11], which is exclusively reserved for multi-chromosomal genomes, ignores the gene order. A genome in this case consists of a number of unordered sets, each one corresponding to a chromosome and containing elements that are the genes belonging to that chromosome). Furthermore, only three types of rearrangement events are considered. These are *fusion* (of two sets into one), *fission* (of one set into two), and *translocation* (two sets exchange part of their contents). Hence, the only information that syntenic distance is able to capture is the *synteny* of genes (two genes are *syntenic* if they belong to the same chromosome). Syntenic distance can not keep track of intra-chromosomal events, as no information about topological properties within chromosomes (the relative order of genes, their distances from one another and their sizes) is stored in the sets representation. Obtaining the syntenic distance requires, as usual, determining the minimum number of operations among the three just mentioned that are necessary to transform a first genome into a second.

Syntenic distance is proved to be $\mathcal{NP}$-hard in [12] and a 2-approximation algorithm is suggested (algorithm $\mathcal{H}$). The bound is tight. This bound is the same for Liben-Nowell's algorithm $\mathcal{F}'$ [1] which is an improvement of Ferretti's $\mathcal{F}$ algorithm [11]. Liben-Nowell also proves that, assuming the instance is partitioned into *components* with distinct genes, no approximation ratio better than 2 can be reached as long as only intra-component moves are performed, which is what all algorithms proposed so far do. Both in [12] and [1], many interesting properties of syntenic distances are proved and some particular classes of instances, such as *linear synteny* and *exact synteny*, are examined. For one particular class of linear synteny, it is proved in [3] that under certain conditions (requiring inclusion properties among the sets of syntenic genes, resulting in what is called a *nested synteny*), the distance is computable in polynomial time. Finally, in [1], the *syntenic diameter*, that is, the length of the longest optimal move sequence for an instance of size n, is discussed. It is conjectured that its size is $2n-3$.

This paper presents some new theoretical and algorithmical results concerning nested synteny and general synteny. We start by giving some definitions and recalling earlier results (Section 2). Finding an optimal solution for the syntenic distance, in particular for nested synteny, is deeply dependent on being able to calculate the syntenic diameter (see [1]). We therefore provide a solution to a conjecture by Liben-Nowell [1] for the syntenic diameter size, that is, for the length of the longest optimal move sequence for an instance of size n. We use a direct proof to show that the size is, in fact, not $2n-3$ but $2n-4$ (an independent and different proof of this was provided by Kleinberg and Liben-Nowell in [2]). The structure of the proof further enables us to characterize the minimum number of translocations an optimal sequence of moves must have. This has important consequences for algorithmical purposes. We generalize the result to include what we call *bidimensional syntenic diameter*. When the initial instance has more than one component, we then suggest a way for performing inter-component moves, something that was not done in previous algorithms (Section 4). We show however that, even in simple cases with more than two components involved, an optimal solution for a general instance of the problem using such moves is $\mathcal{NP}$-hard to obtain. We follow this by introducing an algorithm which is, we believe, simpler, more general and efficient than in [3] for obtaining a move sequence that is optimal for the exact and linear cases when the instance is nested. In addition, we sketch an extension which we conjecture may optimally solve the connected general nested case (Section 5). The algorithm time complexity is $O(k^2+kn+k\log k)$ where k and n are the respective number of chromosomes in the two genomes and $k \leq n$. Finally, concerning general synteny, we prove an inclusion theorem equivalent to the one shown in [3] for linear synteny (Section 6).

2 Definitions and Previous Results

A *genome* is a collection of k sets (chromosomes), each chromosome being a subset of a set of n objects (genes). A genome mutates through one among the following three possible moves involving the sets: *fusion*: $\{S,T\} \rightarrow S' = S \cup T$, *fission*: $S \rightarrow \{S', S''\}$ where $S = S' \cup S''$, and *translocation*: $\{S,T\} \rightarrow \{S',T'\}$ with $S \cup T = S' \cup T'$. Given two genomes $\mathcal{G}_1$ and $\mathcal{G}_2$ over the same gene set Σ, the *syntenic distance* between $\mathcal{G}_1$ and $\mathcal{G}_2$, denoted by $\mathcal{D}(\mathcal{G}_1, \mathcal{G}_2)$, is the minimum number of moves needed to transform $\mathcal{G}_1$ into $\mathcal{G}_2$. Following the syntenic distance literature, we adopt the compact representation of the problem: suppose that genomes $\mathcal{G}_1$ and $\mathcal{G}_2$ contain k and n sets respectively (these correspond to the number of chromosomes in each). The *compact representation* $\mathcal{G}_2'$ of $\mathcal{G}_2$ with respect to $\mathcal{G}_1$ is defined as follows: replace the j^{th} set $\mathcal{G}_{2,j}$ of $\mathcal{G}_2$ for $1 \leq j \leq n$ by the set $\mathcal{G}'_{2,j} = \cup_{x \in \mathcal{G}_{2,j}} \{i \mid x \in \mathcal{G}_{1,i}, 1 \leq i \leq k\}$. Let $\mathcal{G}_1'$ be defined in a symmetrical way as the compact representation of $\mathcal{G}_1$ with respect to $\mathcal{G}_2$. Computing $\mathcal{D}(\mathcal{G}_1, \mathcal{G}_2)$ is the same as computing the distance between $\mathcal{G}_1'$ and $\{\{1\},\{2\},\ldots,\{n\}\}$, which is also the same as computing the distance between $\mathcal{G}_2'$ and $\{\{1\},\{2\},\ldots,\{k\}\}$ (this latter is called the *dual* problem of the former). Notice that, since the former distance equals $\mathcal{D}(\mathcal{G}_2, \mathcal{G}_1)$, we have that $\mathcal{D}(\mathcal{G}_1, \mathcal{G}_2) = \mathcal{D}(\mathcal{G}_2, \mathcal{G}_1)$, that is, the syntenic distance is symmetrical (reflexivity and triangular inequality also hold, and the syntenic distance is therefore a metric). Observe that we can assume without loss of generality that $n \geq k$ since we can always consider the dual problem instead. We denote by $\mathcal{S}(n,k)$ an instance of a synteny problem in the compact representation where there are n elements and k initial sets, and by $\mathcal{D}(\mathcal{S}(n,k))$ the minimum number of moves required to solve $\mathcal{S}(n,k)$. Two sets $\mathcal{S}_1$ and $\mathcal{S}_2$ are *connected* iff $\mathcal{S}_1 \cap \mathcal{S}_2 \neq \emptyset$. In that case, they belong to the same *component*. For an element ℓ, *count(ℓ)* denotes the number of chromosomes in which ℓ appears.

Definition 1 [Canonical move sequence] *Let $\mathcal{G}_1, \mathcal{G}_2$ be an instance of synteny. A move sequence $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ such that $m = \mathcal{D}(\mathcal{G}_1, \mathcal{G}_2)$ and all fusions appear before all translocations, which themselves appear before all fissions is called a* canonical *move sequence.*

Definition 2 [Non-redundant move sequence] *Let $\mathcal{G}_1, \mathcal{G}_2$ be an instance of synteny. A move sequence $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ containing no moves that produce two sets with a non empty intersection is called a* non-redundant *move sequence.*

In [12] and [1] the following is proved.

Proposition 1 [Optimal canonical and non-redundant move sequence] *For any instance of synteny, there is always an optimal move sequence that is non-redundant canonical.*

Theorem 1 [Monotonicity] *Let $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$ and $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ be two collections of sets where for all $1 \leq i \leq k$, we have $\mathcal{T}_i \subseteq \mathcal{S}_i$. Let $n = |\bigcup_i \mathcal{S}_i|$ and $n' = |\bigcup_i \mathcal{T}_i|$. Let $\mathcal{S}(n,k) = \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$ and $\mathcal{T}(n',k) = \mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$. Then $\mathcal{D}(\mathcal{S}(n,k)) \geq \mathcal{D}(\mathcal{T}(n',k))$.*

We say that a move *solves an element x* if it produces the set $\{x\}$, and x appears nowhere else anymore.

There exists a special class of synteny instance, called *exact synteny* [12], which results in a conceptually simpler problem than the general synteny (although in [12] the $\mathcal{NP}$-hardness of syntenic distance is proved using an exact instance of the problem, and therefore also exact synteny is $\mathcal{NP}$-hard). Exact instances have the property that at each step a *splitting move* can be done. A splitting move is a move which increases by 1 the number of components. Observe that a translocation which solves an element is a special case of a splitting move.

Definition 3 *An instance of synteny is* connected *if it consists of one component only. A connected instance $\mathcal{S}(n,k)$ of the synteny problem is* exact *if $n = k$ and $D(\mathcal{S}(n,k)) = n-1$.*

It is easy to see that in an exact synteny instance, all moves are splitting moves. The reason is that initially there is one component, at the end there are n of them, and each move increases by at most one the number of components.

In [3], Liben-Nowell and Kleinberg considered a special class of instances which they called *nested synteny*:

Definition 4 *An instance of synteny $\mathcal{S}(n,k) = \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$ is nested if, for all $i \neq j$, either (1) $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$, (2) $\mathcal{S}_i \subseteq \mathcal{S}_j$ or (3) $\mathcal{S}_j \subseteq \mathcal{S}_i$*

Again in [3], a special case of move sequence is considered. This is called a *linear move sequence.* For an instance $\mathcal{S}(n,k)$, the sequence starts with k–1 moves that are either fusions or translocations that solve an element. The remaining moves are all fissions. In other words, there is a suitable order of the initial k sets such that the linear move sequence just consists of fusing the first two sets, or translocating if an element has count 1 or 2, for $k-1$ times, and then fissioning the remaining unique set of unsolved elements. In [12] it has been shown that, even if linear syntenic distance d' has little biological motivation, calculating it is interesting as we are guaranteed that within a component we have $d \leq d' \leq d + \log_{4/3}(d)$ where d is the unconstrained distance.

Finally, an instance $\mathcal{S}(n,k)$ is *linear exact* if it can be solved with $max\{n,k\} - 1$ moves.

3 Syntenic Diameters

3.1 Syntenic Diameter

In [1], the *syntenic diameter* problem is addressed. The idea is to find the maximum possible distance between two genomes of a given size. Thanks to the monotonicity result, we can identify the instance of size n that requires the highest number of moves to be solved. This is the synteny instance consisting of n copies of the set $\{1, 2, \dots n\}$. This instance was denoted $\mathcal{K}_n(n, n)$ by Liben-Nowell [1]. Where there is no ambiguity, we simplify this to $\mathcal{K}(n, n)$. Liben-Nowell conjectured that $D(\mathcal{K}(n, n)) = 2n - 3$. As will be seen later, finding an optimal solution for the synteny distance, in particular in the case of a nested synteny, is deeply dependent on being able to provide an optimal move sequence for an instance of the diameter. This is what we do now, proving that $\mathcal{K}(n, n) = 2n$–4 differently from what was conjectured by Liben-Nowell (an independent and different proof of this was given by Kleinberg and Liben-Nowell in [2]. Their proof does not contain a result we prove and which we need for the algorithm presented later).

We start by giving a generalisable counter example that shows how to solve $\mathcal{K}(n, n)$ in $2n$–4 steps: from the initial $\mathcal{K}(n, n)$, we perform $n - 4$ fusions so that we obtain 4 times the set $\{1, 2, \dots, n\}$. We then do 2 translocations in order to get $\{1, 2\}, \{1, 2\}, \{3, 4, \dots, n\}, \{3, 4, \dots, n\}$, and 2 more translocations to obtain $\{1\}, \{2\}, \{3\}, \{4, \dots, n\}$. Finally, with $n - 4$ fissions we get the desired result. Overall, this takes $n - 4 + 2 + 2 + n - 4 = 2n - 4$ moves to solve $\mathcal{K}(n, n)$. Let us observe that there exists other optimal move sequences made of translocations only. Since there is no way to solve $\mathcal{K}(3, 3)$ in only $2 \cdot 3 - 4 = 2$ moves or less, we have proved that:

Proposition 2 *$D(\mathcal{K}(n, n)) \leq 2n - 4$, $\forall\, n \geq 4$.*

In fact $2n - 4$ is also a lower bound, and hence equality holds as we show at the end of the section. We start by proving some intermediate results.

Lemma 1 *In an optimal move sequence solving $\mathcal{K}(n, n)$, the number m_1 of fusions is equal to the number of fissions, and the number of translocations equals the optimal solution of $\mathcal{K}(n - m_1, n - m_1)$ when only translocations are allowed.*

Proof Consider a non-redundant optimal move sequence $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_m)$ for $\mathcal{K}(n, n)$ in *canonical* order, that is we have m_1 fusions followed by m_2 translocations, followed by m_3 fissions, as shown below:

$$\mathcal{K}_n(n, n) = \mathcal{S}_0 \xrightarrow[m_1 \text{ fusions}]{} \mathcal{S}_1 \xrightarrow[m_2 \text{ translocations}]{} \mathcal{S}_2 \xrightarrow[m_3 \text{ fissions}]{} \mathcal{S}_3$$

where sets $\mathcal{S}_1$ and $\mathcal{S}_2$ are intermediate instances and $\mathcal{S}_3 = \{\{1\}, \{2\}, \dots, \{n\}\}$ is the final instance. Let k_i denote the number of sets in instance $\mathcal{S}_i$. Moreover, let a_i denote the total amount of elements (that is, the total sum of count(ℓ_i) for all the ℓ_i's) in $\mathcal{S}_i$. Finally, let $b_i = a_i - k_i$ be the number of commas that one would need to separate elements in the sets (this is the total number of fissions one would have to do to each set to solve it independently of the others). At the beginning, we have $\mathcal{S}_0$ with $k_0 = n$, $a_0 = n^2$, and $b_0 = n(n - 1)$. After m_1 fusions, we obtain $\mathcal{S}_1$ with $k_1 = n - m_1$, $a_1 = n(n - m_1)$, and $b_1 = (n - 1)(n - m_1)$. We therefore have n–m_1 copies left of the set $\{1, 2, \dots, n\}$.

We then perform m_2 translocations generating $\mathcal{S}_2$ with $k_2=k_1=n-m_1$ (since translocations do not change the number of sets), $a_2=n$ (only the final n elements remain since fission is the only operation still allowed to get to the n singleton sets of the solution), and $b_2=a_2-k_2=m_1$ as we know that $b_i=a_i-k_i$ for any i. The number b_2 of commas between these final n elements in the k_2 sets of $\mathcal{S}_2$ coincides exactly with the number m_3 of fissions we have to do to transform $\mathcal{S}_2$ into $\mathcal{S}_3$ with $k_3=n$, $a_3=n$, and $b_3=0$. We therefore have that $m_3=b_2=m_1$.

Let us now examine the m_2 translocations which transform the instance $\mathcal{S}_1$ into $\mathcal{S}_2$. Consider the $n-m_1$ distinct sets of the instance $\mathcal{S}_2$, and apply the following renaming to the elements in $\mathcal{S}_1$ and $\mathcal{S}_2$ (obtaining respectively $\mathcal{S}_1'$ and $\mathcal{S}_2'$): for each set i of $\mathcal{S}_2$ ($i=1,\ldots,n-m_1$), rename x_i all elements that end up in set i in $\mathcal{S}_2$. In this way, we obtain an instance $\mathcal{S}_1'$ consisting of $n-m_1$ times the set $\{x_1,x_2,\ldots x_{n\text{–}m_1}\}$ which is just the instance $\mathcal{K}(n-m_1,n-m_1)$, and another instance $\mathcal{S}_2'=\{\{x_1\},\{x_2\},\ldots,\{x_{n\text{–}m_1}\}\}$ which is the solution for $\mathcal{S}_1'$. An optimal move sequence leading to such a solution consists indeed in the m_2 translocations that, in the solution for $\mathcal{K}(n,n)$, transform $\mathcal{S}_1$ into $\mathcal{S}_2$. We therefore have that the m_2 translocations which transform the instance $\mathcal{S}_1$ into $\mathcal{S}_2$ correspond to an optimal solution, containing only translocations, for the instance $\mathcal{K}(n\text{–}m_1,n\text{–}m_1)$. □

Lemma 2 *If only translocations are performed, then* $D(\mathcal{K}(n,n))\geq 2n-4$.

Proof Let us consider an arbitrary optimal move sequence for $\mathcal{K}(n,n)$ containing only translocations. We now show that this contains at least $2n-4$ moves.

Since each element has initial count n and a move may decrease such count by at most 1, no element can be solved in the first $n-2$ moves. At best, we end up with some elements having count 2 which may therefore be solved in subsequent steps.

We claim now that, at the end of the first n–2 moves, all elements with count 2, let us say we have p of them (p may be 0), will find themselves in at most four sets $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$ and $\mathcal{S}_4$. Indeed, in order to obtain a count of 2 for elements $x_1,\ldots,x_p$ which all appear n times at the beginning, each of the first $n-2$ moves must involve two sets that both contain $x_1,\ldots,x_p$. Hence, the first n–4 operations are of the type $\{x_1,\ldots,x_p,\ U_1\},\{x_1,\ldots,x_p,\ U_2\}\longrightarrow\{x_1,\ldots,x_p,U_3\},\ \{U_4\}$, where U_1 or U_2 (but not both) may be empty, U_3 may be empty but never U_4 (otherwise, the operation would not be a translocation). After these translocations, the elements $x_1,\ldots,x_p$ have all count 4. In the next two operations, we may either leave the p elements together (performing again moves like the one just shown), or we may split them by doing the following move: $\{x_1,\ldots,x_p,\ U_1\},\{x_1,\ldots,x_p\ U_2\}\longrightarrow$ $\{x_1,\ldots,x_h,U_3\},\ \{x_h+1,\ldots,x_p,U_4\}$, where $1\leq h<p$ and U_1, U_2, U_3 and U_4 may, this time, all be empty. In the case where the last two moves split the p elements with count 2 among 4 sets, the elements in all the other $n-4$ sets will have count at least 4 (since such sets were translocated $n-4$ times only).

Let $\mathcal{S}(n,k)$ be the instance of synteny obtained and c the number of its components. We have that $D(\mathcal{S}(n,k))\geq n-c$ [12]. The bound is absolute: no optimal move sequence, whether it contains inter-component moves (as shall be considered later) or not, can do less operations than that. This gives us a lower bound on the number of steps we still need to perform after the $n-2$ translocations just described. If c is 1, then we are done, since we would need at least $n-1$ further moves giving a total of at least $(n-2)+(n-1)=2n-3$ instead of the $2n-4$ that we are capable of producing. If c is 2, we need at least $n-2$ more moves after the $n-2$ initial ones, giving the total lower bound of $2n-4$. Finally, if c is greater than 2, we have just shown above that only 2 (say the first and the second) of the c components may contain all the elements with count 2, if any such exist. The other components all contain elements with higher counts. Let us call *intra-component moves* those involving only sets that belong to a same component. Moves that involve sets belonging to two different components are called *inter-component moves*. If only intra-component moves were done, each of these other components would require at least one extra move with respect to $n_i - 1$, where n_i is the number of elements of component i. If inter-component moves were allowed, $(\sum_{j=3}^{c}(n_j - 1)) + (c-2)$ would be a lower bound on the number of moves required. The $(c-2)$ extra moves in relation to an exact instance with $(c-2)$ components come from the fact that an inter-component move (that is, the first move between two different components) does not decrease the count of any element. Even if an inter-component move enabled to solve an element, at least $(c-2)$ moves would therefore not decrease the counts of any element. Since each of these other components has at least one element which has count at least 4, overall at least $(c-2) = (\sum_{j=3}^{c} 1)$ extra moves will be "wasted" for getting nearer to the solution. Hence, we need at least

$$(n-2)+\sum_{i=1}^{c} D(\mathcal{S}(n_i,k_i)) \geq n-2+(n_1-1)+(n_2-1)+\sum_{j=3}^{c}((n_j-1)+1)=2n-4.$$

moves in this case also, where k_i is the number of sets in component i. □

A direct consequence of Lemma 2 is that the diameter is exactly $2n-4$ for all $n \geq 4$ in the case where neither fusions nor fissions are performed. The question remains open whether by using also fusions and fissions a $\mathcal{K}(n,n)$ instance could be solved in even less moves. Such question will be answered by Theorem 2, but we can already state the following result.

Proposition 3 *The m_2 translocations of Lemma 1 correspond to an optimal solution for $\mathcal{K}(n-m_1, n-m_1)$.*

Proof The proof follows directly from Lemma 1 and Lemma 2. □

We can now prove the following theorem.

Theorem 2 *$D(\mathcal{K}(n,n)) = 2n-4$, $\forall\ n \geq 4$, and the maximum number of fusions allowed in an optimal move sequence is $n-4$.*

Proof If the number f of fusions is equal to 0, then the result follows immediately from Lemma 2. Let us now suppose that there are $f > 0$ fusions. By Lemma 1, any canonical optimal move sequence contains also f fissions and $D(\mathcal{K}(n-f, n-f))$ translocations (leading to $n-f-1$ singleton sets and a set with $f+1$ elements that have to be fissioned). We prove by induction on n that in this case also, we have $D(\mathcal{K}(n,n)) = 2n-4$ for all $n \geq 4$.

The base case is $\mathcal{K}(4,4)$ for which it holds that $D(\mathcal{K}(4,4)) = 4 = 2n{-}4$. The inductive step consists in showing that $D(\mathcal{K}(n,n)) = 2n-4$ holds if it holds for all n_0 such that $4 \leq n_0 < n$. By hypothesis (since $f \geq 1$ and therefore $n-f < n$), we know that $D(\mathcal{K}(n-f, n-f)) = 2(n-f)-4 = 2n-4-2f$. We therefore have that $D(\mathcal{K}(n,n)) = D(\mathcal{K}(n-f,n-f)) + 2f = 2n-4$.

It is important to observe here that $D(\mathcal{K}(n,n)) = 2n-4$ holds if $n \geq 4$. As a consequence, if we fuse more than $n-4$ times, we would have that the $D(\mathcal{K}(n-f,n-f))$ translocations would be more than $2(n-f)-4$, and then we would no longer obtain an optimal move sequence for $\mathcal{K}(n,n)$. Therefore, we must have at most $n-4$ fusions. $\Box$

As a consequence of Theorem 2, since there are at most $2n-8$ fusions and fissions, we have that:

Corollary 1 *In an optimal move sequence for $\mathcal{K}(n,n)$ with $n \geq 4$, there are always at least 4 translocations.*

If $n=1,2$, we have trivially that $D(\mathcal{K}(n,n))=n-1$. For $n=3$, it can be checked that $D(\mathcal{K}(n,n))=n=3$.

3.2 Bidimensional Syntenic Diameter

The $\mathcal{K}(n,n)$ instance is a particular case of $\mathcal{K}(n,m)$, which consists of m copies of the set $\{1,2,\ldots,n\}$. We call instances of $\mathcal{K}(n,m)$ with $n \neq m$, *bidimensional syntenic diameter* since in this case the input size depends on two parameters. The value of $D(\mathcal{K}(n,m))$ will therefore be a function of both n and m. Notice that $D(\mathcal{K}(n,k))$ gives an upper bound for $D(\mathcal{S}(n,k))$. When $n \neq m$, it is no longer true that the number of fusions equals the number of fissions. In this case in fact, we have the following.

Lemma 3 *Let n_1 be the number of fusions in an optimal solution of $D(\mathcal{K}(n,m))$. We then have that the number of fissions is $n_3 = n - m + n_1$.*

Proof Again, let us consider a canonical optimal move sequence solving $D(\mathcal{K}(n,m))$. Initially there are m sets, which become $m-n_1$ after the fusions. Translocations do not change the number of sets, and fissions only increase it. Since we must have n sets left at the end, we have that $m-n_1 \leq n$. Furthermore, the number of fissions are exactly $n-(m-n_1)$. $\Box$

Theorem 3 $D(\mathcal{K}(n,m)) = m+n-4$ *for all $n,m \geq 4$. Furthermore, at most $max\{n,m\}-4$ of these operations may be fusions and $n-4$ may be fissions.*

Proof We have to analyse separately various subcases:

$4 \leq m \leq n$

In this case, the $\mathcal{K}(n,m)$ is just an intermediate case of the optimal solution we gave for $\mathcal{K}(n,n)$. Therefore, m–4 fusions followed by 4 translocations and $n-4$ fissions as in the solution for $\mathcal{K}(n,n)$ represent an optimal move sequence for $\mathcal{K}(n,m)$. We thus have $D(\mathcal{K}(n,m)) = m-4+4+n-4 = m+n-4$.

$m > n$

In this case, we have an instance with more sets than elements, and we consider its dual, which contains n copies of $\{1,2,\ldots,m\}$, bringing us back to the previous case.

We therefore conclude that $D(\mathcal{K}(n,m)) = m+n-4$ for all $n,m \geq 4$. The results concerning upper bounds on the number of fusions and fissions follow directly from those of $\mathcal{K}(n,n)$ and from

Lemma 3: as in the case of $\mathcal{K}(n,n)$, we must have at least 4 translocations. The number n_1 of fusions are thus at most $m-4$ if $m \geq n$, and at most $n-4$ otherwise. Finally, since we have at least 4 sets after the translocations and the goal is to end up with n sets, then we may have at most $n-4$ fissions. □

If $m < 4$, we split into three possible cases.

$m = 1$

We have one copy of $\{1, 2, \ldots, n\}$ and clearly the only thing to do is $n-1$ fissions, giving $D(\mathcal{K}(n,1)) = n-1$, $\forall\, n$.

$m = 2$

Here the best thing to do is to translocate the two copies of $\{1, 2, \ldots, n\}$ into $\{1, 2, \ldots, h\}$ and $\{h+1, \ldots, n\}$ for any $1 \leq h \leq n$ (or to fuse if $n = 1$, in which case we are done), and then do $n-2$ fissions. We therefore have $D(\mathcal{K}(n,2)) = n-1$ if $n \geq 2$, (and $D(\mathcal{K}(1,2)) = 1$ for $n = 1$).

$m = 3$

It is easy to check that we can not make it in less that n moves (for example, fusing first and then doing as for $m = 2$). Thus $D(\mathcal{K}(n,3)) = n$. Again, $n = 1$ is an exception as $D(\mathcal{K}(1,3)) = 2$.

Finally, if $n < 4$ and $m \geq 4$:

$n = 1$

We have m copies of the set $\{1\}$, and clearly $m-1$ fusions is the move sequence to do. Therefore $D(\mathcal{K}(1,m)) = m-1$, $\forall\, m$.

$n = 2$

Fusing the initial m copies of $\{1,2\}$ $m-2$ times, and then translocating $\{1,2\}$ and $\{1,2\}$ into $\{1\}$ and $\{2\}$ is the optimal move sequence, giving $D(\mathcal{K}(2,m)) = m-1$.

$n = 3$

The move sequence consisting in $m-3$ fusions followed by the 3 translocations which solve $\mathcal{K}(3,3)$ is optimal (although not unique), hence $D(\mathcal{K}(3,m)) = m$.

Finally, let us observe that Lemma 1 and Lemma 3 can be extended to any instance $\mathcal{S}(n,k)$ with $k \leq n$ in the sense that the number m_3 of fissions is always $m_3 = n-k+m_1$ where m_1 is the number of fusions.

4 The Multi-Component Case

4.1 General Observations

All algorithms for syntenic distance presented in the literature so far perform moves involving only sets that belong to a same component, that is, intra-component moves.

As mentioned in the introduction, Liben-Nowell has shown in [1] that, in the case of instances with more than one component, no approximation ratio better than 2 may be reached as long as only intra-component moves are performed. In fact, there are instances with $c>1$ components for which the concatenation of optimal *local* solutions for the c components leads to a *global* solution that is close to twice the optimal. An example is the instance composed of the set $\{1,2,\ldots,n-1\}$ and $n-1$ times the set $\{n\}$. Clearly, the only thing to do within component $\{\{1,2,\ldots,n-1\}\}$ is to fission $n-2$ times. Inside the other component, we need to fuse $n-2$ times the copies of $\{n\}$ in order to have only one left. The overall number of moves is $2n-4$ while the instance could have been solved with $n-1$ translocations by performing, for example, first the translocation $\{1,2,\ldots,n-1\},\{n\} \to \{2,3,\ldots n\},\{1\}$, and then $n-3$ more translocations of the kind $\{i,\ldots,n-1,n\},\{n\} \to \{i\},\{i+1,\ldots,n\}$ using one of the copies of $\{n\}$ at each step while solving $i \in \{2,\ldots,n-1\}$. At the end, the only two remaining sets are $\{n-1,n\}$ and $\{n\}$. One more translocation is then enough for solving the instance. This gives a total of $1+n-3+1=n-1$ moves, which represents almost half the number we would need if we did not perform the inter-component moves. The reason is that almost each translocation of the globally optimal move sequence has the same effect as a fission in one component (an element with count one is solved) plus a fusion in the other (one of the redundant copies of the set $\{n\}$ disappears). The only exception to this concerns the first translocation between two different components. Such a translocation has the same effect as a fission but fuses no elements together. This last is an important observation and will be used in Section 4.3 to show that optimizing by performing inter-component moves is an $\mathcal{NP}$-hard problem.

4.2 Optimizing in the Special Case of Two Components by Performing Inter-Component Moves

We start by showing how, in the special case of just two components, a move sequence may be optimized by performing inter-component moves.

Theorem 4 *Let $\mathcal{S}(n,k)$ be an instance composed of:*

- *a component $\mathcal{C}$ which contains a set of g sets – denoted by $\mathcal{S}_1,\ldots,\mathcal{S}_g$ – which have to be fused all together in an optimal local move sequence, and such that the set is maximal (no other set needs to be fused with these) and unique (there exists no other set of sets in $\mathcal{C}$ needing to be fused);*
- *a component $\mathcal{C}'$ consisting of the single set $\{x_1,\ldots,x_f\}$.*

Then performing inter-component moves saves $min\{f,g\}-2$ moves.

Proof Let $\mathcal{C}'=\{x_1,x_2,\ldots,x_f\}$ and $\mathcal{C}$ contain the sets $\mathcal{S}_1,\ldots,\mathcal{S}_g$ that must be fused together. Our goal is to obtain the sets $\{x_1\},\{x_2\},\ldots,\{x_f\}$ and the set $\mathcal{S}=\mathcal{S}_1\cup\ldots\cup\mathcal{S}_g$. We observe that the global solution consisting of the union of the two separate intra-component solutions that reach this goal takes $(f-1)+(g-1)=f+g-2$ moves overall.

Let us first consider the case $g\leq f$. The following g inter-component moves can be performed:

1. $\{x_1,\ldots,x_f\},\mathcal{S}_1 \to \{x_1\},\{x_2,\ldots,x_f\}\cup\mathcal{S}_1$
2. $\{x_2,\ldots,x_f\}\cup\mathcal{S}_1,\mathcal{S}_2 \to \{x_2\},\{x_3,\ldots,x_f\}\cup\mathcal{S}_1\cup\mathcal{S}_2$
$\vdots$ $\quad\ldots\quad\ldots\quad\ldots$

g. $\{x_g,\ldots,x_f\} \cup \mathcal{S}_1 \cup \mathcal{S}_2 \cup \ldots \cup \mathcal{S}_{g-1}, \mathcal{S}_g \to \{x_g\}, \{x_{g+1},\ldots,x_f\} \cup \mathcal{S}_1 \cup \mathcal{S}_2 \cup \ldots \cup \mathcal{S}_g$

after which the only unsolved part left is the set $\{x_{g+1},\ldots,x_f\} \cup \mathcal{S}$. With f–g more fissions, we reach our declared goal. This gives a total of $g+f-g=f$ moves. We therefore saved $g-2=\min\{f,g\}-2$ moves. Moreover, if we do not fission immediately, the $f-g$ elements left in component $\mathcal{C}$ can be used later on for a further optimization.

Conversely, if $g>f$, we can perform the f translocations:

1. $\{x_1,\ldots,x_f\}, \mathcal{S}_1 \to \{x_1\}, \{x_2,\ldots,x_f\} \cup \mathcal{S}_1$,
2. $\{x_2,\ldots,x_f\} \cup \mathcal{S}_1, \mathcal{S}_2 \to \{x_2\}, \{x_3,\ldots,x_f\} \cup \mathcal{S}_1 \cup \mathcal{S}_2$,
$\vdots$ $\quad\ldots\quad\ldots\quad\ldots$
f. $\{x_f\} \cup \mathcal{S}_1 \cup \mathcal{S}_2 \cup \ldots \cup \mathcal{S}_{f-1}, \mathcal{S}_f \to \{x_f\}, \mathcal{S}_1 \cup \mathcal{S}_2 \cup \ldots \cup \mathcal{S}_f$,

after which all the x_i are solved and there remain only the $g-f$ sets $\mathcal{S}_{f+1},\ldots,\mathcal{S}_g$ which still need to be fused with the set $\mathcal{S}_1 \cup \mathcal{S}_2 \cup \ldots \cup \mathcal{S}_f$. Hence, $g-f$ more fusions are required for a total of g moves. In this case, we saved $f-2$ moves, which equals $\min\{f,g\}-2$ again. □

In the case where inter-component moves such as those of Theorem 4 are performed, we say that components $\mathcal{C}$ and $\mathcal{C}'$ have been *paired*. We claim that such inter-component moves are the only ones that make us save operations with respect to the case where only intra-component moves are performed.

4.3 Finding an Optimal Components Pairing is $\mathcal{NP}$-hard

Usually, while components such as $\mathcal{C}'$ of Theorem 4 will contain an unique set which needs to be fissioned into singletons, components such as $\mathcal{C}$ may contain more than one set of sets which need to be fused. In a more general way, an instance will be composed of components where both fusion and fission operations need to be performed. The general *optimal pairing problem* consists in finding the pairing (in the sense given at the end of Section 4.2) of the components of an instance that will result in an optimal move sequence.

For some classes of multi-component instances, it is possible to characterize an optimal move sequence when inter-component operations are allowed. An example of this is expressed in the following proposition.

Proposition 4 *Let $\mathcal{S}(n,k)$ be an instance composed of:*

- *$c \geq 2$ components $\mathcal{C}_1,\cdots,\mathcal{C}_c$ where each $\mathcal{C}_i$ contains g_i times the set $\{y_i\}$;*
- *$d \geq 1$ components $\mathcal{C}'_1,\cdots,\mathcal{C}'_d$ with $\mathcal{C}'_j=\{x^j_1,\cdots,x^j_{f_j}\}$.*

If inter-component moves are allowed, then an optimal move sequence solving $\mathcal{S}(n,k)$ will be one which, at the same time*:*

- *performs the maximum number of pairings between components $\mathcal{C}_1,\cdots,\mathcal{C}_c$ and components $\mathcal{C}'_1,\cdots,\mathcal{C}'_d$;*

- *translocates each component $\mathcal{C}_i$ for $1 \leq i \leq c$ with a minimum number of different components among $\mathcal{C}'_1, \cdots, \mathcal{C}'_d$.*

Proof Derives immediately from the observation at the end of Section 4.1 that the first translocation done between two different components is the only one which saves no operation in relation to doing only intra-component moves. Indeed, such a translocation replaces a fission only instead of a fission and a fusion as do the other (new) translocation operations in the inter-component optimization procedure. An optimal move sequence must therefore economize as much as possible on such translocations while doing a maximum number of pairings. □

We now show that even for such a class of multi-component instances, the optimal pairing problem is $\mathcal{NP}$-hard. We prove this by reducing 3-Partition to the optimal pairing problem.

We start by recalling the 3-Partition problem which has been proved to be $\mathcal{NP}$-hard in the strong sense [13].

Definition 5 [3-Partition problem] *Given a finite set A of $3m$ elements, a bound $B \in \mathbb{Z}^+$, and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$ such that $B/4 < s(a) < B/2$ and $\sum_{a \in A} s(a) = mB$, check whether A can be partitioned into m disjoint sets $A_1, A_2, \ldots, A_m$ such that, for $1 \leq i \leq m$, $\sum_{a \in A_i} s(a) = B$.*

We now state and prove the $\mathcal{NP}$-hardness of our problem.

Theorem 5 *If $\mathcal{P} \neq \mathcal{NP}$, then finding an optimal component pairing for inter-component moves for the class of restricted synteny instances given in Proposition 4 is not solvable in polynomial time.*

Proof Let us consider an arbitrary instance of the 3-Partition problem. We thus have a finite set A containing $3m$ elements, a bound $B \in \mathbb{Z}^+$, a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$ such that $B/4 < s(a) < B/2$ and $\sum_{a \in A} s(a) = mB$. We build an instance $\mathcal{S}(n = 3m + Bm, k = (3+1)m)$ in the following way. The instance will contain:

- $3m$ sets $\mathcal{C}_1, \ldots, \mathcal{C}_{3m}$ having each $s(C_i)$ copies of a same element y_i with $B/4 < s(C_i) < B/2$ for a $B \in \mathbb{Z}^+$;
- m distinct components $\mathcal{C}'_1, \ldots, \mathcal{C}'_m$ each containing an unique set with B distinct elements $\{x^i_1, \ldots, x^i_B\}$ which need to be fissioned.

By Proposition 4, obtaining an optimal move sequence for instance $\mathcal{S}(n, k)$ requires finding an optimal pairing of the $\mathcal{C}_1, \ldots, \mathcal{C}_{3m}$ and $\mathcal{C}'_1, \cdots, \mathcal{C}'_m$ components of the instance. This is a pairing which makes the maximum number of inter-component translocations yet which, at the same time, translocates each of $\mathcal{C}_1, \ldots, \mathcal{C}_{3m}$ with the minimum number of $\mathcal{C}'_j$ components for $1 \leq j \leq m$. Since the number $s(C_i)$ of elements in each component $\mathcal{C}_i$ is such that $B/4 < s(C_i) < B/2$, the minimum will be reached if we can translocate exactly 3 components among $\mathcal{C}_1, \ldots, \mathcal{C}_{3m}$ with each component $\mathcal{C}'_1, \cdots, \mathcal{C}'_m$. Finding whether this is realisable requires solving the 3-Partition problem. This ends the proof. □

5 Nested Synteny

5.1 General Observations

Let $\mathcal{N}(n,k)$ be a nested synteny instance. In [3], Liben-Nowell and Kleinberg optimally solved such an instance in the case of a linear move sequence by using a job-scheduling algorithm [14] as a black box in their own program. Their idea is the following. Let us consider a single component nested synteny instance $\mathcal{N}(n,k)$. Let us denote by $\tilde{D}(\mathcal{N}(n,k))$ the length of the optimal linear move sequence for $\mathcal{N}(n,k)$. They first prove that $\tilde{D}(\mathcal{N}(n,k)) = \max\{n,k\}+d-1$ where d is the minimum number of elements that must be added to the biggest set in the component in order to obtain a linear exact instance. This result deals with the general case where we may also have $k>n$. To facilitate, from now on we shall assume that $k\leq n$. Finding a linear solution for $\mathcal{N}(n,k)$, that is calculating $\tilde{D}(\mathcal{N}(n,k))$, requires therefore only finding the value of d. They further prove that the instance will be linear exact if, and only if, the graph model of the instance which they give as input to the job-scheduling algorithm results in an output that is equal to 0. The value of d is determined by performing a binary search between 0 and k (we never need more than k elements), interrogating the job-scheduling algorithm each time. The case where the instance is not linear exact is easy to determine because too many or too few extra elements were given to it. The job-scheduling algorithm serves therefore as an oracle answering the question: is the instance linear exact? However, such algorithm had been initially elaborated for solving a more general problem in another area. Therefore, we introduce below a much simpler algorithm that enables to answer the same question (*i.e.* "Is the instance linear exact?") in a more efficient way. Later, we suggest how Liben-Nowell and Kleinberg's idea for the linear case could be extended in polynomial time to address the case of a connected general (that is, not necessarily linear) nested synteny. We conjecture the proposed approach is optimal.

5.2 Exact Nested Synteny

5.2.1 Tree Representation of the Problem

Let us consider a directed weighted tree $\mathcal{T} = (V \cup V', E \cup E')$. Nodes $v \in V$ are the n sets $\mathcal{S}_i$ in $\mathcal{N}(n,k)$. Nodes $v' \in V'$ are dummy nodes whose purpose will appear clear in a short while.

An edge $e \in E$ leaves a node $\mathcal{S}_i$ (we shall describe below where it leads to) if there exists at least one $\mathcal{S}_j$ such that either of the two is satisfied:

- $\mathcal{S}_i \supset \mathcal{S}_j$ and there exists no set $\mathcal{S}_k$ such that $\mathcal{S}_i \supset \mathcal{S}_k \supset \mathcal{S}_j$;
- $\mathcal{S}_i = \mathcal{S}_j$ and $i < j$ and there exists no $i < k < j$ such that $\mathcal{S}_i = \mathcal{S}_k$.

If there exists only one $\mathcal{S}_j$ satisfying one of the above conditions in relation to $\mathcal{S}_i$, edge e links $\mathcal{S}_i$ directly to $\mathcal{S}_j$. If there exists more than one $\mathcal{S}_j$, we add a dummy node $v' \in V'$, and edge e will now link node $\mathcal{S}_i$ to v'. We also place an edge $e' \in E'$ from v' to node $\mathcal{S}_j$ for all $\mathcal{S}_j$s satisfying either condition.

Since this is a nested instance, $\mathcal{T}$ is a tree if there is only one component, or a forest otherwise. Let us call $r_1, \ldots, r_c$ the roots of the forest for c the number of components. Each edge $e \in E$ connecting a set $\mathcal{S}_i$ directly to a set $\mathcal{S}_j$ is weighted by an integer, namely $w_e = |\mathcal{S}_i - \mathcal{S}_j| - 1$. Edges $e \in E$ leading to a dummy node are weighted $w_e = |\mathcal{S}_i - \cup\mathcal{S}_j| - 1$ for all $\mathcal{S}_j$ satisfying one of the two subset conditions given above. Finally, edges $e' \in E'$ have weight zero. An example is given in Figure 1.

We now enrich the tree by placing some information on the nodes in V.

Definition 6 *Let us call* frontier *node, a node $v \in V$ such that the edge leading to v is not negative while the one leaving v is less than zero. We shall also consider all leaves as frontier nodes.*

To each frontier node $v \in V$ is attached a pair of integers (v_b, v_t) (b for "bad" and t for "total") such that:

- $v_t = \max\{\sum w_e, 0\}$ for all w_e (positive and negative) along the path from v to the son of its first ancestor that is also a frontier node; if there is no ancestor, then $v_t = 0$;
- $v_b = (\sum w_e)$ for all $w_e < 0$ along the same path as above; if there is no ancestor or there is no negatively weighted edge e, then $v_b = 0$.

Values v_b and v_t indicate, respectively, the number of moves that would solve no element and the number of elements whose count becomes one (this last number can never be negative) by performing an optimal move sequence between two frontier nodes along a same branch of $\mathcal{T}$ (these are v and the first ancestor of v which is a frontier node). Such values are equivalent to the notion of *profit* in [3]. In the case of an exact synteny, the idea is to traverse the tree $\mathcal{T}$ by stepping on frontier nodes only (in fact, $\mathcal{T}$ could be contracted to contain only such nodes), possibly jumping from arc to arc, always choosing as next stepping stone the one that allows us to earn the maximum number of elements with count one (possibly none).

Theorem 6 *Building the tree $\mathcal{T}$ for an instance $\mathcal{N}(n,k)$ takes $O(k^2 + kn)$ time and kn space.*

Proof Let us assume that the input data is correct (that is, the nested condition is satisfied) and that the cardinalities of each set have been pre-calculated (this takes at most kn time). Building $\mathcal{T}$ takes then at most k^2 time because each comparison of two sets can be done in constant time (using kn space – each set is represented as an array of dimension n, or $k^2 \log n$ – each set is represented as a balanced tree). Indeed, for each set u in turn by order of decreasing cardinality (sorting the sets takes at most $O(k \log k)$ time), we need to consider one of its elements only and ask whether it is in one of the nodes already in $\mathcal{T}$ starting with the root. If it is in one of the nodes, let us say v, then u is either identical with v (just check cardinality), or it is included in it and we recursively compare u with the sons of v. Once the tree $\mathcal{T}$ is built, adding information to the edges and nodes can be done by just traversing the tree depth-first, each operation taking constant time. □

5.2.2 Algorithm

Once we have represented the instance by means of a tree such as $\mathcal{T}$, an algorithm for optimally solving the exact nested synteny is the following:

1. **while** the root has at least one son **do**
2. let w_e be the weight of the edge e leaving r
3. **if** $w_e < 0$ **then**
4. **if** find son v of r that has an element x whose count is 2 (there must be one if exact)
5. translocate v with r solving x
6. attach node z (maybe dummy) that is son of v to r
7. **else** return: "this is not an exact instance of synteny"
8. **else if** $w_e \geq 0$ **then**
9. **if** find descendant frontier node $v \in V$ with information (v_b, v_t) such that $|v_b| \leq w_e$ and $new_{w_e} = v_t$ is maximum (there must be one if exact)
10. **if** there are two or more with maximum value v_t **then**
11. **repeat**
12. pick the one with a descendant frontier node u such that $|v'_b| + |u_b| \leq w_e + v'_t$ and $new_{w_e} = v'_t + u_t$ is maximum where v' is the first frontier ancestor of u
13. **until** u is unique or all u's are leaves, in which case pick any of them
14. $v := u$
15. translocate with r all sets (v included) along the path from r to v (in order) preferentially solving an element with count 2, otherwise one with count 1
16. detach the node from V that is a first descendant of r and an ancestor of v
17. attach at its place node z that is son of v
18. **if** z is a dummy node **then**
19. attach to a dummy node son of r the sons of z
20. give weight new_{w_e} to the edge e from r to its son
21. **else** ($|v_b| > w_e$) return: "this is not an exact instance of synteny"

Theorem 7 *The algorithm has a time complexity in $O(k \log k)$.*

Proof Calculating the syntenic distance can be done in $O(k \log k)$ time if we maintain priority queues for the frontier nodes and, for each frontier node, a pointer to its position in the queue. We need in fact one queue for each possible distance between a frontier node u and the root r of the tree (where the distance corresponds to the number of frontier nodes in the unique path between u and r). We need to update the queues for all nodes below z in line 17. This can be done in $O(k \log k)$ time. Finding node v in lines 9-13 with the priority queues takes the same time ($O(logk)$ per queue, and there can be up to $O(k)$ queues). A smart implementation of the tree allows to recover an optimal move sequence corresponding to the distance without increasing the asymptotic complexity of the algorithm. This can be done, for instance, by coloring at construction time the elements in the set corresponding to a node u that do not belong to any of the sets associated with the children of u. The algorithm has therefore a time complexity in $O(k \log k)$. $\square$

Theorem 8 *The algorithm solves the exact nested synteny optimally.*

Proof If there are no dummy nodes, translocating the root with its unique son is the best we can do as was shown by Liben-Nowell and Kleinberg in [3], and indeed this is what the algorithm will do at each step.

Suppose now that there are dummy nodes. Let us assume there is one dummy node d just below the root r (we can assume this without loss of generality since if this was not the case, we know how we can reach this situation by doing translocations between the root and its unique son as indicated above). Let w_e be the weight of the arc between r and d. We must consider two cases:

$w_e < 0$
: All elements present in the root are also present in one of its sons, and at least one of these elements must have count at most 2 (otherwise the instance would not be exact). The algorithm solves each of these elements in turn and optimality is preserved.

$w_e \geq 0$
: Let $v_1, v_2, \ldots, v_k$ be the first frontiers node that are descendants of r (there may be just two). Since none of them has a frontier node for ancestor, all have $(v_b, v_t) = (0, 0)$. Whether line 10 of the algorithm is executed or not, for all nodes u until the last, condition $|v'_b| + |u_b| \leq w_e + v'_t$ (let us call it α) will be satisfied and u_t will always be maximum. After u is identified (and assigned to v), we translocate with r all nodes in the path from r to v, each time solving one element. This is possible thanks to condition α. Clearly, solving first any element with count equal to 2 preserves optimality, as does solving any element having count equal to 1 if none with count 2 exists. Moreover, since the v chosen had maximum value v_t, in the end, the edge leaving r will receive a value new_{w_e} that is also maximum. This means that all the other moves we could have done instead remain feasible. □

5.3 Linear Nested Synteny

Using the same idea as in Liben-Nowell and Kleinberg (sketched in Section 5.1), we can employ the above algorithm as a black box for optimally solving the linear nested synteny in $O(k^2 + kn + c\, k \log^2 k)$ time where c is the number of components. We recall that the $(k^2 + kn)$ term comes from the construction of $\mathcal{T}$, which needs to be done once only. We remind also that the optimal linear move sequence in the case of multiple components is defined as just the concatenation of all local optimal move sequences. The instance will be linear exact if the algorithm runs to conclusion, possibly leaving some sets which need to be fissioned (unless the instance is exact which may be the case only if $n = k$). However, these sets must not contain any of the extra elements given to it. If any one does, this means we utilized too many extra elements. If, on the other hand, the algorithm is not able to run to conclusion (because at some point we have that either $w_e < 0$ or $|v_b| > w_e$), this means we need more extra elements. These two observations are what allow us to search for the value of d such that $\tilde{D}(\mathcal{N}(n, k)) = \max\{n, k\} + d - 1$ in a dichotomous way.

5.4 General Nested Synteny

5.4.1 Compulsory Translocations

We show now how the above idea can be extended to deal with a general nested synteny. We use for this the concept of *good* and *bad* moves.

The difficulty in calculating $D(\mathcal{N}(n, k))$ without the linearity constraint comes from the fact that when, at a given step, we meet the situation where the only moves we can make are *bad* ones (basically in our model, either when $w_e < 0$ and there is no element whose count is 2, or when $|v_b| > w_e \geq 0$), then we have no means to decide which bad move we should choose without trying all possible complete move sequences. This does not happen when all moves until the next frontier node are *good*, that is, each solves an element as it happens with an exact instance of synteny. In that case, it is easy to decide which is the best move to make at any given step as was shown above. Liben-Nowell and Kleinberg's nice idea for the linear case is to create a situation where we have to face good moves only. This is done through the use of extra elements which allow to transform the bad moves (in the case, the fusion operations) into translocations. The solution requires just finding how many extra elements are required.

Fusions are the only bad moves (as defined above) one can make in the linear case. The situation of a general nested synteny is more complicated in the sense there are now three possible bad moves one can make. These are:

- translocations that can not be substituted by a fusion otherwise the move sequence will not be optimal anymore;
- translocations that can be replaced by fusions without affecting the total number of operations needed to solve the instance;
- fusions that could be replaced by translocations without affecting the total number of operations needed to solve the instance.

Two examples of the first type are given by the diameter $\mathcal{K}(n,n)$ for $n \geq 4$ and by the instance $\mathcal{N}(8,8)$ composed of the sets $\{1,\ldots,8\}$, $\{1,\ldots,8\}$, $\{1,\ldots,8\}$, $\{1,\ldots,8\}$, $\{1,\ldots,4\}$,$\{1,\ldots,4\}$, $\{5,\ldots,8\}$,$\{5,\ldots,8\}$. The two instances can be solved optimally by using, respectively, four and ten translocations. Among these translocations, respectively two and six solve no element. However, if extra elements were added to the instances to transform such translocations that do not solve an element into ones that do, we would not obtain an optimal move sequence. In fact, these translocations have the property that they can not be replaced by fusions. We call them *compulsory translocations*. We say that a $\mathcal{S}(n,k)$ instance is an *exact diameter instance* if any optimal move sequence contains only translocations that solve an element and compulsory translocations (corresponding to the first type above).

On the other hand, there are translocations that solve no element and could be replaced by fusions. Such translocations, as well as all fusions, can be replaced by translocations solving extra elements given to the original instance without increasing the number of steps needed to arrive at a solution. Having those extra elements can only enable us (if we have enough of them) to decide, as for the linear case, what is the next move to do.

Deciding which translocations are of the first type, that is, compulsory, and which are not is one of the main problems we must therefore treat. We know how to address it in the case of a diameter instance, $\mathcal{K}(n,n)$, and of a bidimensional diameter instance, $\mathcal{K}(n,m)$. We consider next the general case.

5.4.2 Instance $\mathcal{K}(n,m)$ with *Descendants*

The general case of a nested synteny corresponds in fact to a $\mathcal{K}(n,m)$ *with descendants*, that is to a $\mathcal{K}(n,m)$ which contains also one or more subsets of $\{1,2,\ldots,n\}$ (we call *children*), plus possibly proper subsets or twins of these (we call them *grandchildren*), and so on. We denote such cases by $\mathcal{K}(n,m,c)$ where c indicates the number of distinct maximal proper subsets of the m sets $\{1,2,\ldots,n\}$. In the tree $\mathcal{T}$ associated with $\mathcal{K}(n,m,c)$, we have m copies of the root, of which the lowest one has either $c = 1$ son or is connected to a dummy node having $c > 1$ children (observe that each such child is itself the root of a $\mathcal{K}(n', m', c')$ instance for appropriate values of n', m', c'). If a linear move sequence is considered, then just fusing at least $m-2$ times is the only thing required, as until then no element can be solved and thus no translocations performed. We suggest that in general a shorter (no longer linear) move sequence may be found. We conjecture such sequence is optimal.

As an example, consider the instance $\{\{1,2,3,4\},\{1,2,3,4\},\{1,2\},\{3,4\}\}$. No element has count 2 or less, so a linear move sequence would require a fusion as a first step. Assume that the fusion involves the two identical sets (we shall see in Proposition 9 that this is a reasonable assumption), so that we are left with the three sets $\{1,2,3,4\},\{1,2\},\{3,4\}$. From now on, all elements can be solved, but at least 3 more moves are necessary to terminate, leading to a total of 4 moves. If linearity had not been required, the first move could have been the translocation $\{1,2,3,4\},\{1,2,3,4\} \rightarrow \{1,2\},\{3,4\}$, resulting in the four sets $\{1,2\},\{1,2\},\{3,4\},\{3,4\}$. Two more translocations ($\{1,2\},\{1,2\} \rightarrow \{1\},\{2\}$ and $\{3,4\},\{3,4\} \rightarrow \{3\},\{4\}$) would then have been enough to terminate, leading to a total of 3 moves only. This latter is the optimal (not linear) move sequence. Observe that the translocation we did represents a compulsory one. Intuitively, the reason for which performing first a translocation may result in a shorter solution is that the translocation can split a set into two, which correspond to the children. The problem is thus split into two subcomponents.

Let us consider another example. This is represented by the instance composed of twice the set $\{1,2,\ldots,n\}$ plus sets that are children of this one and where all elements have count 3 (hence the union of the children equals the father and there are no granchildren). If we start by performing a fusion, $n-1$ more moves will be required to solve the instance. Indeed, at each step after the fusion, only at most one element can be solved, until the last move that solves two of them. If, instead, we start by splitting the instance into two components, performing for this the move $\{1,2,\ldots,n\},\{1,2,\ldots,n\} \rightarrow \{1,2,\ldots,h\},\{h{+}1,\ldots,n\}$ with a suitable choice for h, the children can be partitioned into those that are children of $\{1,2,\ldots,h\}$ and those that are children of $\{h{+}1,\ldots,n\}$ (this is always feasible after possibly renumbering the elements). After such a move, we are left with two separate components, consisting respectively of h and $n-h$ elements. Each one is of the same type we would have obtained if we had performed a fusion: a father with children that are a partition of the father, and no grandchildren. However, only $(h-1)+(n-h-1) = n-2$ moves are required now to finish solving the instance. This is one move less than if we had started with a fusion. Splitting the instance into two subcomponents therefore helps.

Similarly, if m in $\mathcal{K}(n,m,c)$ is greater than 4, then splitting the problem into 4 subcomponents (performing only $m-4$ fusions followed by 4 translocations), will in some cases represent an even better choice. An example is the instance $\{\{1,\ldots,8\}$, $\{1,\ldots,8\}$, $\{1,\ldots,8\}$, $\{1,\ldots,8\}$, $\{1,2\}$, $\{3,4\}$, $\{5,6\}$, $\{7,8\}\}$.

In general, when there are descendants (that is $c \neq 0$), the best will often be to solve $\mathcal{K}(n,m,c)$ by starting to transform the m copies of the set $\{1,\ldots,n\}$ into a suitable d-partition of it. The instance is thus split into d subcomponents. We shall see that the value of d as well as the choice of which elements go into which partition depends upon the value of m, the number of children c, and whether there are grandchildren or not. Before examining this, we start by giving some intuition for the conjecture that we shall state later.

First, we need to prove a proposition.

Proposition 5 *Transforming m copies of the set $\{1, \ldots, n\}$ into a d-partition of $\{1, \ldots, n\}$ takes $m + d - 4$ moves for all $m, d \geq 4$.*

Proof Let $\mathcal{S}_1, \ldots, \mathcal{S}_d$ be the d sets we wish to produce. These sets must be pairwise disjoint and their union give $\{1, \ldots, n\}$. Rename x_i all elements that end up in set S_i for all $i = 1 \ldots d$. Such a renaming transforms the problem into that of solving a $\mathcal{K}(d, m)$ instance, which we know requires $m + d - 4$ moves (Lemma 3). These moves are those needed to perform the tranformation we wish to obtain. If we could make it with less moves, by applying the renaming backwards we would be able to find a better than $m + d - 4$ solution for $\mathcal{K}(d, m)$, which is a contradition. □

Let us now consider as above the simple case of a $\mathcal{K}(n, m, c)$ instance with m equal to 2 and where each element has count exactly 3. We have seen two examples where the best to do is to split the instance into, respectively, 2 and 4 subcomponents. It can be checked that splitting into three subcomponents is not convenient, even if there are 3 children. This is because the split into 3 subcomponents can never be done in an efficient way. The reason is the same for which the size of $2n-4$ for the diameter does not hold for $n=3$. If there are more than 4 children, then splitting the instance according to the number of children (and therefore in more than 4 subcomponents) does not help either: the move gained because of the creation of one more component (as a result of a translocation which solves two elements simultaneously) is lost during the splitting process, as shown by the Proposition 5.

Suppose now that m is 2 in the instance $\mathcal{K}(n, m, c)$ but that the union of the children represents this time only a proper subset of the father set. In this case there are, say, f elements that have count 2 that the split process will have to assign to one of the new subcomponents. After the split, such elements have count 1. If all the other $n - f$ elements have initial count 3 (which will become 2 after the split), that is, if there are no grandchildren, then it does not matter to which subcomponent the elements with initial count 2 are assigned after the split process. Otherwise, that is, in the general case where there are grandchildren in the instance $\mathcal{K}(n, m, c)$, then it is not indifferent to which subcomponent will go the elements that, after the split, will get a count of 1. The reason is that such elements may prove useful in the same way as happened in inter-component moves as seen in Section 4. Indeed, if they are assigned to a subcomponent that will not need them, additional operations will be required even if inter-component moves are later performed. Furthermore, we have seen that deciding which inter-component moves should be done in an optimal move sequence is an $\mathcal{NP}$-hard problem. The best to do seems therefore to split from the instance only those subcomponents which represent exact diameter instances and to leave all the other elements, including the f which will have count 1 after the split, into a single remaining subcomponent.

These observations are the motivation for the following conjecture.

Conjecture 1 *Let $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_c$ be the c children in a $\mathcal{K}(n, m, c)$ instance for $n, m \geq 4$ and let d be the number of sets into which $\{1, 2, \ldots, n\}$ could be partitioned. We have that:*

if $c \geq 3$ *then* $d=$ *min* $\{4$, *number of children that in $\mathcal{T}$ become, with the split part containing them, an exact diameter instance*$+1\}$, *and*

if $c = 1(2)$ *then* $d=$ *min* $\{1(2)$, *number of children that in $\mathcal{G}$ become, with the split part containing them, an exact diameter instance*$+1\}$.

where we call split part *of the root $\mathcal{S}$ of a $\mathcal{K}(n, m, c)$ instance, one of the sets into which $\mathcal{S}$ is partitioned. In both cases, the additional* 1 *stands for the component that does not correspond to an exact diameter.*

As a consequence, we have that in an optimal move sequence for $\mathcal{K}(n, m, c)$ with $n, m \geq 4$ and $c \geq 1$, we need at least 4 translocations if $c \geq 3$ and at least 2 otherwise ($c = 1, 2$).

We now have all the tools we need for suggesting an algorithm for the general nested synteny. We merge the results concerning the diameters with a modified version of the black box for the exact nested case in order to obtain a general (that is, not linear) solution. We use the same idea as for solving a linear nested synteny (Section 5.3), that is, we determine by a binary search between 0 and k the minimum number of extra elements we must give to the instance to make it, not an exact instance this time, but an exact diameter instance. The main difference is that, when meeting a $\mathcal{K}(n, m)$ or $\mathcal{K}(n, m, c)$ instance, the extra elements must be used only where a fusion would have been possible or needed, never where a translocation was absolutely required (see Theorem 3 and Section 5.4.2).

Lines 7 and 21 are the only things that change in the algorithm given in Section 5.2.2. They become:

7. **else** (at this point, we have a $K(n, m, c)$ instance)
7a.   **if** ($m > 4$) **or** ($2 < m \leq 4$ **and** $c = 1, 2$) **then** return: "this is not an exact diameter instance"
7b. **else** choose the next set of compulsory translocations (following Lemma 1) as the one which leads to the maximum v_t (iterate if there are more than two)

21. **else** (we have a $K(n, m, c)$ instance with $m = (|v_b| - w_e) > 0$)
21a.   **if** ($m > 4$) **or** ($2 < m \leq 4$ **and** $c = 1, 2$) **then** return: "this is not an exact diameter instance"
21b. **else** choose the next set of compulsory translocations (following conjecture 1) as the one which leads to the maximum v_t (iterate if there are more than two)

With these additions, we obtain a new black box which allows us to determine, by a binary search, the number of moves in what we conjecture is an optimal move sequence for solving a connected general nested synteny. This is the sum of the number of "good" translocations (which solve an element from the original instance), the number of compulsory translocations, the minimum number of translocations that solve an extra element and, finally, the number of fissions involving only elements from the original instance.

6 An Inclusion Theorem for the General Case

In the case of a general synteny, sets are not necessarily nested. For some pairs $(\mathcal{S}_i, \mathcal{S}_j)$ however, we may still have that either $\mathcal{S}_i \subseteq \mathcal{S}_j$ or $\mathcal{S}_j \subseteq \mathcal{S}_i$. The following result states that also in the general synteny case, there always exists an optimal move sequence that treats supersets first.

Theorem 9 *If $\mathcal{S}(n, k) = \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$ and $\mathcal{S}_i \subseteq \mathcal{S}_j$ with $\mathcal{S}_i \neq \emptyset$, then there exists an optimal move sequence in which $\mathcal{S}_j$ is used before $\mathcal{S}_i$.*

Proof If $\mathcal{S}_i = \mathcal{S}_j$, then the theorem is trivially true. Let us now consider the case where $\mathcal{S}_i \subsetneq \mathcal{S}_j$. Let a canonical optimal move sequence of $\mathcal{S}(n,k)$ be $\sigma = \sigma_1, \sigma_2, \ldots \sigma_p, \ldots, \sigma_q, \ldots, \sigma_m$. Let σ_p (resp. σ_q) be the first move using $\mathcal{S}_i$ (resp. $\mathcal{S}_j$) with $p < q$ (if such a move sequence does not exist, then we are done). We show that we can modify σ in such a way that we obtain a move sequence σ' that has also length m and uses $\mathcal{S}_j$ before $\mathcal{S}_i$. We have to treat only the cases where σ_p and σ_q are both fusions, or σ_q is a translocation and σ_p is either a translocation or a fusion. Indeed, if σ_q were a fission, we would have that the first operation made on the set $\mathcal{S}_j$ would be to fission at least one of its elements, that is to solve it. However, this contradicts the hypothesis that there exists the set $\mathcal{S}_i \neq \mathcal{S}_j$ sharing some elements with $\mathcal{S}_j$.

Let σ_h be the first move of the original move sequence where products of σ_p are involved in the same operation as products of σ_q. Notice that σ_h has to be either a fusion or a translocation because it takes two sets as an input. Notice that $h \geq q$. We have to consider two cases.

$h = q$

This means that the move σ_q involves $\mathcal{S}_j$ (which until then remains untouched and can not therefore be a product of σ_p), and another set, say $\mathcal{T}$, that is a product of σ_p, directly or indirectly (that is, is a product of products of σ_p or etc).

If σ_p is a fusion, let $\mathcal{S}$ be the set such that σ_p is the operation $(\mathcal{S}, \mathcal{S}_i) \to \mathcal{U} = \mathcal{S} \cup \mathcal{S}_i$. The question is whether this step can be replaced with $\sigma_{p'} : (\mathcal{S}, \mathcal{S}_j) \to \mathcal{U}' = \mathcal{S} \cup \mathcal{S}_j$ (leaving $\mathcal{S}_i$ instead of $\mathcal{S}_j$ untouched). We have that $\mathcal{U}' = \mathcal{U} \cup \mathcal{S}_j$, that is, it is $\mathcal{U}$ with additional elements that were in $\mathcal{S}_j$ but not in $\mathcal{S}_i$ nor in $\mathcal{S}$. Let $\mathcal{W} = (\mathcal{S}_j \setminus \mathcal{S}_i) \setminus \mathcal{S}$. We have $\mathcal{U}' = \mathcal{U} \cup \mathcal{W}$. We assume $\mathcal{W} \neq \emptyset$, otherwise the problem is trivial. Any other possible fusion that followed σ_p and preceeded σ_q in the original move sequence, and that used $\mathcal{U}$, can be replaced with a fusion that uses $\mathcal{U}'$ instead. For simplicity, we name again $\mathcal{U}'$ the result of these possible further fusions: in all these cases, we are talking about a set that in the original move sequence was a set $\mathcal{U}$, and that now contains also $\mathcal{W}$, resulting in something that we keep on naming $\mathcal{U}'$. If there were some translocations in σ between move σ_p and move σ_q that used $\mathcal{U}$, this would result in the same case as we treat next.

If σ_p is a translocation, we may start by observing that σ_q is also a translocation. Let us call $\mathcal{U}$ and $\mathcal{X}$ the two sets resulting from the translocation σ_p. If by $\mathcal{S}$ we denote the set with which $\mathcal{S}_i$ is now translocated in σ_p, the sets $\mathcal{U}$ and $\mathcal{X}$ form a partition of the set $\mathcal{S} \cup \mathcal{S}_i$ so that, if we use $\mathcal{S}_j$ instead of $\mathcal{S}_i$, we just have a bigger set $\mathcal{S} \cup \mathcal{S}_j$ to partition. This is what we do in the operation σ'_p which again replaces σ_p. Let us assume further that the extra elements $\mathcal{W} = (\mathcal{S}_j \setminus \mathcal{S}_i) \setminus \mathcal{S}$ land all together in one of the sets resulting from σ'_p, that is, $\mathcal{W}$ is not split by σ'_p (note that both $\mathcal{S}$ and $\mathcal{W}$ may be different from those of the previous case but since they represent analogous situations for the purpose of this proof, we use the same names). Let $\mathcal{U}'$ be the set such that $\mathcal{U}' = \mathcal{U} \cup \mathcal{W}$ where $\mathcal{U}$ was the set resulting from σ_p. Again here, any other possible translocation that followed σ_p and preceeded σ_q in the original move sequence, and that used $\mathcal{U}$, can be replaced with a fusion that uses $\mathcal{U}'$ instead. The case of a translocation is thus similar to the case of a fusion.

We show now how in both cases (whether σ_p was a fusion or a translocation), optimality is preserved when using σ'_p rather than σ_p, provided we suitably modify also move σ_q (which coincides with σ_h) in the following way. We have observed above that, if $h = q$, then we have that the set $\mathcal{T}$ such that σ_q took as input $\mathcal{S}_j$ and $\mathcal{T}$ must be a product of σ_p. It is therefore enough to place $\mathcal{W}$ in $\mathcal{T}$ and leave it there ($\mathcal{T}$ is thus what we called set $\mathcal{U}$, and $\mathcal{U}'$ is the union of $\mathcal{T}$ and $\mathcal{W}$), and to replace σ_q (which took $\mathcal{S}_j$ and $\mathcal{T}$ as input) with the move σ'_q which takes $\mathcal{S}_i$ and $\mathcal{U}' = \mathcal{T} \cup \mathcal{W}$ instead. Since $\mathcal{S}_j \cup \mathcal{T} = \mathcal{S}_j \cup \mathcal{U} = \mathcal{S}_i \cup \mathcal{U}'$, the result of σ'_q can be the same as the result of σ_q, and hence the remaining of the move sequence can remain as it was.

$h > q$

Let us call again $\mathcal{T}$ the set with which $\mathcal{S}_j$ is either fused or translocated in move σ_q. If $h > q$, we have that move σ_q of the original sequence does not correspond to an operation between a product (or product of products) of σ_p and $\mathcal{S}_j$. Set $\mathcal{T}$ is therefore either also untouched from the beginning (like $\mathcal{S}_j$), or it is among the products of other moves $\sigma_{t_1}, \ldots, \sigma_{t_l}$ with $1 \leq t_i < q$ and $t_i \neq p$ that are completely independent from σ_p. We can therefore build a new move sequence σ' which is equal to σ except that the order of the moves is changed in such a way that all the moves $\sigma_{t_1}, \ldots, \sigma_{t_l}$ as well as move σ_q happen before σ_p. In this way, we again obtain a move sequence that has the same length as σ and uses $\mathcal{S}_j$ before $\mathcal{S}_i$.

These transformations to the move sequence can be performed independently for all pairs $\mathcal{S}_i$ and $\mathcal{S}_j$ such that $\mathcal{S}_i \subsetneq \mathcal{S}_j$ and $\mathcal{S}_i$ is used first. None of these changes creates a new pair of the same type, hence the process will eventually finish and result in a move sequence which satisfies the desired property while remaining optimal. This proves the theorem. □

Acknowledgements

The authors wish to thank prof. Fabrizio Luccio, Valentina Ciriani and Marco Aldinucci from the University of Pisa for a careful reading of part of this paper. They also would like to thank D. Liben-Nowell for having kindly sent them a preliminary version of his and J. Kleinberg's paper [3] accepted at CPM2000. Finally, they gratefully thank the two referees for a few corrections and many suggestions which greatly helped improve this paper.

References

[1] D. Liben-Nowell. On the Structure of Syntenic Distance. In M.Crochemore and M.Paterson, editors, *Combinatorial Pattern Matching*, pages 50–65, 1999.

[2] D. Liben-Nowell and J. Kleinberg. The syntenic diameter of the space of n-chromosome genomes. In *Conference on Gene Order Dynamics, Comparative Maps, and Multigene Families (DCAF)*, 2000.

[3] D. Liben-Nowell and J. Kleinberg. Structural Properties and Tractability Results for Linear Synteny. In R.Giancarlo and D.Sankoff, editors, *Combinatorial Pattern Matching*, pages 248–263, 2000.

[4] V. Bafna and P. Pevzner. Sorting by transpositions. *SIAM Journal on Discrete Mathematics*, 11(2):224–240, 1998.

[5] D. A. Christie. Sorting permutations by block-interchanges. *Information Processing Letters*, 60(4):165–169, 1996.

[6] S. Hannenhalli. Polynomial-time algorithm for computing translocation distance between genomes. *Discrete Applied Mathematics*, 71:137–151, 1996.

[7] A. Caprara. Sorting by reversals is difficult. In *1st ACM International Conference on Computational Molecular Biology, (RECOMB 97)*, pages 75–83, 1997.

[8] S. Hannenhalli and P. Pevzner. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In *27th ACM Symposium on Theory of Computing (STOC 95)*, pages 178–189, 1995.

[9] S. Hannenhalli and P. Pevzner. Transforming man into mice (polynomial algorithm for genomic distance problem). In *36th Annual IEEE Symposium on Foundation of Computer Science (FOCS 95)*, pages 581–592, 1995.

[10] D. A. Christie. *Genome Rearrangement Problem.* PhD thesis, University of Glasgow, 1998.

[11] V. Ferretti, J. H. Nadeau, and D. Sankoff. Original Synteny. In *Combinatorial Pattern Matching*, pages 159–167, 1996.

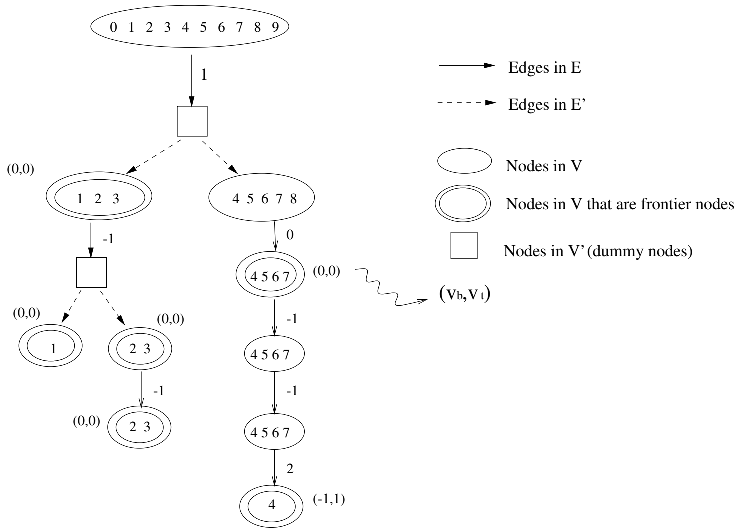
[12] B. DasGupta, T. Jiang, S. Kannan, M. Li, and E. Sweedyk. On the Complexity and Approximation of Syntenic Distance. In *1st ACM International Conference on Computational Molecular Biology, (RECOMB 97)*, pages 99–108, 1997.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* Freeman, 1979.

[14] H. M. Abdel-Wahab and T. Kameda. Scheduling to Minimize Maximum Cumulative Cost Subject to Series-Parallel Precedence Contraints. *Operations research*, 26(1):141–158, 1978.

Figure legend

Figure 1 Tree representation of a nested instance of the synteny problem.



0 1 2 3 4 5 6 7 8 9
1
(0,0)
1 2 3
4 5 6 7 8
-1
0
4 5 6 7
(0,0)
(Vb,Vt)
(0,0)
(0,0)
1
2 3
-1
4 5 6 7
-1
(0,0)
2 3
-1
4 5 6 7
2
4
(-1,1)
Edges in E
Edges in E'
Nodes in V
Nodes in V that are frontier nodes
Nodes in V' (dummy nodes)