HP Distance via Double Cut and Join Distance

Anne Bergeron[1], Julia Mixtacki[2], and Jens Stoye[3]

[1] Dépt. d'informatique, Université du Québec à Montréal, Canada.
bergeron.anne@uqam.ca
[2] International NRW Graduate School in Bioinformatics and Genome Research,
Universität Bielefeld, Germany. julia.mixtacki@uni-bielefeld.de
[3] Technische Fakultät, Universität Bielefeld, Germany.
stoye@techfak.uni-bielefeld.de

Abstract. The genomic distance problem in the Hannenhalli-Pevzner theory is the following: Given two genomes whose chromosomes are linear, calculate the minimum number of inversions and translocations that transform one genome into the other. This paper presents a new distance formula based on a simple tree structure that captures all the delicate features of this problem in a unifying way.

1 Introduction

The first solution to the genomic distance problem was given by Hannenhalli and Pevzner [6] in 1995. Their distance formula, called the general *HP distance*, requires preprocessing steps such as *capping* and *concatenation* and involves seven parameters. In the last decade, different authors pointed to problems in the original formula and in the algorithm given by Hannenhalli and Pevzner. Their algorithm was first corrected by Tesler [9]. In 2003, Ozery-Flato and Shamir [8] found a counter-example and modified one of the parameters of the distance formula. Very recently, another correction was presented by Jean and Nikolski [7]. Unfortunately, the last two recent results have not resulted in simpler presentations of the material, nor are they implemented in software tools yet. The only available tool is *GRIMM* implemented by Glenn Tesler [10].

In contrast to this rather complicated distance measure, Yancopoulos *et al.* [11] presented a general genome model that includes linear and circular chromosomes and introduced a new operation called *double cut and join* (or shortly DCJ) operation. In addition to inversions and translocations, the DCJ operation also models transpositions and block-interchanges. Beside the simple distance computation, the sorting algorithm is also basic and efficient [4].

In this paper we will show how the rearrangement model considered in the HP theory can be integrated in the more general DCJ model. Specifically, the HP distance can be expressed as

$$d_{HP} = d_{DCJ} + t$$

where t represents the extra cost of not resorting to unoriented DCJ operations. The extra cost can easily be computed by a tree data structure associated to a genome.

The next section recalls the results on the DCJ distance. In Section 3, we establish the conditions under which the two distances are equal. The general case is treated in Section 4, where we introduce the basic concepts and the tree needed for the computation of the HP distance, and we give a new proof and formula for the Hannenhalli-Pevzner theorem. Section 5 presents the conclusion.

2 The Double Cut and Join (DCJ) Model

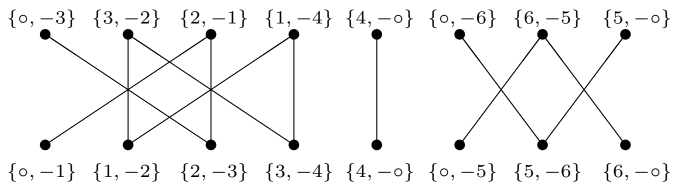
Let A and B be two linear multi-chromosomal genomes on the same set of N genes. A linear chromosome will be represented by an ordered sequence of signed genes, flanked by two unsigned telomere markers:

$$(\circ, g_1, \ldots, g_n, \circ).$$

An *interval* $(l, \ldots, r)$ in a genome is a set of consecutive genes or telomere markers within a chromosome; the set $\{l, -r\}$ is the set of *extremities* of the interval – note that $\circ = -\circ$. An *adjacency* is an interval of length 2, an adjacency that contains a telomere marker is called a *telomere*. Each gene g is the extremity of two adjacencies, one as $+g$, and one as $-g$, in both genomes A and B. This remark yields the following basic construct:

Definition 1. *The* adjacency graph $AG(A, B)$ *is a graph whose vertices are the adjacencies of genomes A and B. Each gene g defines two edges, one connecting the two adjacencies of genome A and B in which g appears as extremity +g, and one connecting the two adjacencies in which g appears as extremity −g.*

Since adjacencies that are telomeres have only one gene, the vertices of the adjacency graph will have degree one or two, thus the graph is a union of *paths* and *cycles*. Paths of odd length, called *odd paths*, connect telomeres of different genomes, and paths of even length, the *even paths*, connect telomeres of the same genome. For example, the adjacency graph of the genomes $A = \{(\circ, 3, 2, 1, 4, \circ), (\circ, 6, 5, \circ)\}$ and $B = \{(\circ, 1, 2, 3, 4, \circ), (\circ, 5, 6, \circ)\}$ has two odd paths, one cycle and two even paths:



A *DCJ operation* applied to two adjacencies of the same genome disconnects the incident edges of the adjacency graph, and reconnects them in one of the possible other ways. The *DCJ distance* between genomes A and B, $d_{DCJ}(A, B)$, is the minimum number of DCJ operations necessary to transform genome A into genome B. We have:

Theorem 1 **([4]).** *Let A and B be two genomes defined on the same set of N genes, then we have*

$$d_{DCJ}(A, B) \ = \ N - (C + I/2)$$

where C is the number of cycles and I the number of odd paths in $AG(A, B)$. An optimal sorting sequence can be found in $O(N)$ time.

A DCJ operation that reduces the DCJ distance by 1 is called *DCJ-sorting.* Using Theorem 1, we have the following property of DCJ-sorting operations, using the fact that a DCJ operation acts on at most two paths or cycles, and produces at most one new path or cycle:

Corollary 1. *A DCJ-sorting operation acts on a single path or cycle, or on two even paths of the adjacency graph.*

Some DCJ operations can create intermediate circular chromosomes, even if both genomes A and B are linear, and we will want to avoid them in the HP model. The following definition is a generalization of a classical concept in rearrangement theory, *oriented operations*:

Definition 2. *A DCJ-sorting operation is* oriented *if it does not create circular chromosomes.*

For two linear genomes, oriented operations are necessarily inversions, translocations, fusions or fissions. These operations are also called *HP operations*, and the HP distance between two genomes $d_{HP}(A, B)$ is the minimum number of HP operations needed to transform genome A into genome B. Since DCJ operations are more general than HP operations, we always have the following lower bound:

Proposition 1. *For two linear genomes A and B, we have that $d_{DCJ}(A, B) \leq d_{HP}(A, B)$.*

3 Components and Oriented Sorting

In this section, we introduce the notion of *components.* They roughly correspond to the classical concept of components, but in the context of adjacency graphs, we prove that they are unions of paths and cycles.

3.1 Basic Definitions

Definition 3. *Given two genomes A and B, an interval $(l, \ldots, r)$ of genome A is a* component *relative to genome B if there exists an interval in genome B:*
a) with the same extremities,
b) with the same set of genes, and
c) that is not the union of two such intervals.

Example 1. Let

$A = \{(\circ, 2, 1, 3, 5, 4, \circ), (\circ, 6, 7, -11, -9, -10, -8, 12, 16, \circ), (\circ, 15, 14, -13, 17, \circ)\}$,
$B = \{(\circ, 1, 2, 3, 4, 5, \circ), (\circ, 6, 7, 8, 9, 10, 11, 12\circ), (\circ, 13, 14, 15, \circ), (\circ, 16, 17, \circ)\}$.

The components of genome A relative to genome B are: $(\circ, 2, 1, 3)$, $(3, 5, 4, \circ)$, $(\circ, 6)$, $(6, 7)$, $(-11, -9, -10, -8)$, $(7, -11, -9, -10, -8, 12)$, $(\circ, 15, 14, -13)$ and $(17, \circ)$.

Note that components of length 2 are the same adjacencies in both genomes, possibly up to flipping of a chromosome. These are called *trivial components.*

Two components are *nested* if one is included in the other and their extremities are different. As the following lemma shows, two components cannot share a telomere:

Lemma 1. *If* $(\circ, \ldots, r_1)$ *and* $(\circ, \ldots, r_1, \ldots, r_2)$ *are two components, then* $r_1 = r_2$*, and if* $(l_1, \ldots, l_2, \ldots, \circ)$ *and* $(l_2, \ldots, \circ)$ *are two components then* $l_1 = l_2$.

Proof. Suppose that $(\circ, \ldots, r_1)$ and $(\circ, \ldots, r_1, \ldots, r_2)$ are two components. Since the corresponding intervals in genome B, $(\circ, \ldots, r_1)$ and $(\circ, \ldots, r_2)$, share the same gene content, the interval $(r_1, \ldots, r_2)$ shares the same gene content in both genomes, thus $(r_1, \ldots, r_2)$ is a component, and $(\circ, \ldots, r_1, \ldots, r_2)$ is the union of two components, a contradiction. The other statement has a similar proof. □

It is further known that two components can not properly overlap on two or more elements. We thus have the following generalization of a statement from [5]:

Proposition 2. *Two components are either disjoint, nested, or overlap on exactly one gene.*

Proposition 2 implies that components can be partially ordered by inclusion, and that overlapping components will have the same parent. An adjacency *properly belongs* to the smallest component that contains it.

Definition 4. *The adjacency graph of a component* $\mathcal{C}$ *is the subgraph of the adjacency graph of genomes* A *and* B *induced by the adjacencies that properly belong to* $\mathcal{C}$.

An important property of the adjacency graph is the following:

Proposition 3. *The adjacency graph of a component is a union of paths and cycles of the adjacency graph of genomes* A *and* B.

Proof. Let $\mathcal{C} = (l, \ldots, r)$ be a component. Since it has the same gene content and the same extremities as the corresponding interval in genome B, all edges of the adjacency graph that are within the interval $(l, \ldots, r)$ in genome A will also be within the interval $(l, \ldots, r)$ in genome A. Thus all these edges form a union of paths and cycles of the adjacency graph of genomes A and B.

Each component that is nested in $\mathcal{C}$ is also a union of paths and cycles of the adjacency graph of genomes A and B, and none of them contains an adjacency that properly belongs to $\mathcal{C}$. We can thus remove them without compromising the connectivity of the adjacency graph of $\mathcal{C}$. □

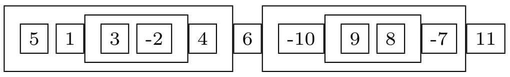
3.2 Oriented Sorting

Since orientation of genes is relative, we can always assume that all genes in a chromosome of genome B are positive and in increasing order. The proper adjacencies of a component $\mathcal{C} = (l, \ldots, r)$ induce a block partition in the corresponding chromosomes of genomes A and B. If we label the blocks in the chromosome of genome B with numbers from 1 to k, the corresponding blocks of the chromosome in genome A will be a signed permutation $(p_1, \ldots, p_k)$ of these integers $\{1, \ldots, k\}$. We will call this permutation – or it reverse – the *permutation associated to the component* $\mathcal{C}$.

Consider for example the following two genomes

$$A = \{(\circ\ 5, 1, 3, -2, 4, 6, -10, 9, 8, -7, 11\ \circ)\}$$
$$B = \{(\circ\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9, 10, 11\ \circ)\}$$

The associated components can easily be seen in the following diagram:



The component $(\circ, \ldots, 6)$ consists of three blocks: the gene 5, the block $(1, \ldots, 4)$ and the gene 6. Thus, the permutation associated to the component $(\circ, \ldots, 6)$ is $(2, 1, 3)$. For the other three non-trivial components, the associated permutations are $(1, 3, -2, 4)$, $(-4, 3, 2, -1)$ and $(1, -2, 3)$.

When the permutation associated to a component has both positive and negative signs, then it is well known from the sorting by inversion theory that the component can be optimally sorted by DCJ-sorting inversions. Components whose associated permutations have only positive elements can sometimes be optimally sorted by DCJ-sorting inversions. For example, consider the pair of genomes:

$$A = (\circ, 4, 3, 2, 1, \circ) \text{ and } B = (\circ, 1, 2, 3, 4, \circ),$$

whose associated permutation is $(4, 3, 2, 1)$. Its DCJ distance is 4, and it can be optimally sorted by inverting each of the four genes. However, we have:

Lemma 2. *If all elements of the permutation associated to a component have the same sign, then no inversion acting on one of its paths or cycles can create a new cycle.*

Proof. By eventually flipping the chromosome, we can assume that all the elements of the permutation are positive. Suppose that an inversion is applied to two adjacencies $(+i, +j)$ and $(+k, +l)$ in a single path or cycle of the component, and that this creates a new cycle. The new adjacencies will be $(+i, -k)$ and $(-j, +l)$, where at most one of $+i$ and $+l$ can be a telomere. If both of these new adjacencies belong to the same path or cycle, there was no creation of a new cycle. Suppose that the adjacency $(+i, -k)$ belongs to the new cycle, then all other adjacencies of this cycle existed in the original component, and are composed of positive elements. This, however, is impossible by the construction of the adjacency graph. $\square$

Definition 5. *A component is* oriented *if there exists an oriented DCJ-sorting operation that acts on vertices of its adjacency graph, otherwise it is* unoriented*.*

Oriented components are characterized by the following:

Proposition 4. *A component is oriented if and only if either its associated permutation has positive and negative elements, or its adjacency graph has two even paths.*

Proof. If the associated permutation has positive and negative elements, then there is at least one change of signs between blocks labeled by consecutive integers. There thus exists an inversion that creates an adjacency in genome B, thus a new cycle, and the inversion is DCJ-sorting. If there are two even paths, then one must be a path from genome A to genome A, and the other must be a path from genome B to genome B. An inversion in genome A that acts on one adjacency in each path creates two odd paths, thus is DCJ-sorting.

In order to show the converse, suppose that all elements of the associated permutation are positive, and all paths are odd. By Corollary 1, a DCJ-sorting operation must act on a single path or cycle. This operation cannot be a translocation or a fusion since all paths and cycles of a component are within a chromosome. This operation cannot be an inversion, since inversions that create new cycles are ruled out by Lemma 2, inversions acting on a single odd path cannot augment the number of odd paths, and inversions acting on cycles never create paths. Finally, this operation cannot be a fission: a fission acting on a cycle creates an even path; and a fission acting on an odd path must circularize one of the chromosome parts in order to be DCJ-sorting, otherwise it would be split into an even path and an odd path. □

Proposition 4 implies that, in the presence of unoriented components, we have $d_{DCJ}(A,B) < d_{HP}(A,B)$, since all DCJ-sorting operations will create circular chromosomes. On the other hand, well known results from the Hannehalli-Pevzner theory show that, when all components admit a sorting inversion, then it is possible to create a new cycle at each step of the sorting process with HP operations, without creating unoriented components. The same type of result can be obtained in this context, and we give it in the Appendix. Thus we have:

Theorem 2. *Given two linear genomes A and B, $d_{HP}(A,B) = d_{DCJ}(A,B)$ if and only if there are no unoriented components.*

4 Computing the General HP Distance

In this section we will show that, given the DCJ distance d_{DCJ}, one can express the Hannenhalli-Pevzner distance d_{HP} in the form

$$d_{HP} = d_{DCJ} + t,$$

where t represents the additional cost of not resorting to unoriented DCJ operations. First, we describe how to destroy unoriented components in Section 4.1 and after that, in Section 4.2, we compute the additional cost from the inclusion and linking tree of the unoriented components.

4.1 Destroying Unoriented Components

Destroying unoriented components is done by applying a DCJ operation either on one component in order to orient it, or on two components in order to merge them, and possibly others, into a single oriented component. By using the nesting and linking relationship between components, one can minimize the number of operations necessary to destroy unoriented components.

When two components overlap on one element, we say that they are *linked.* Successive linked components form a *chain.* A chain that cannot be extended to the left or right is called *maximal.* We represent the nesting and linking relations between components of a chromosome in the following way:

Definition 6. *Given a chromosome X of genome A and its components relative to genome B, define the forest F_X by the following construction:*

1. *Each non-trivial component is represented by a round node.*
2. *Each maximal chain that contains non-trivial components is represented by a square node whose (ordered) children are the round nodes that represent the non-trivial components of this chain.*
3. *A square node is the child of the smallest component that contains this chain.*

Now, we define a tree associated to the components of a genome by combining the forests of all chromosomes into one rooted tree:

Definition 7. *Suppose genome A consists of chromosomes $\{X_1, X_2, \ldots, X_K\}$. The tree T associated to the components of genome A relative to genome B is given by the following construction:*

1. *The root is a round node.*
2. *All trees of the set of forests $\{F_{X_1}, F_{X_2}, \ldots, F_{X_K}\}$ are children of the root.*

The round nodes of T are *painted* according to the following classification:

1. The root and all nodes corresponding to oriented components are painted *black.*
2. Nodes corresponding to unoriented components that do not contain telomeres are painted *white.*
3. Nodes corresponding to unoriented components that contain one or two telomeres are painted *grey.*

The tree associated to the components of the genomes A and B of Example 1 is shown in Fig. 1. Note that grey nodes are always children of the root.

The following two propositions are general remarks on components and are useful to show how to destroy unoriented components.

Proposition 5. *A translocation acting on two (unoriented) components cannot create new (unoriented) components.*

Proposition 6. *An inversion acting on two (unoriented) components $\mathcal{A}$ and $\mathcal{B}$ creates a new component $\mathcal{D}$ if and only if $\mathcal{A}$ and $\mathcal{B}$ are included in linked components.*

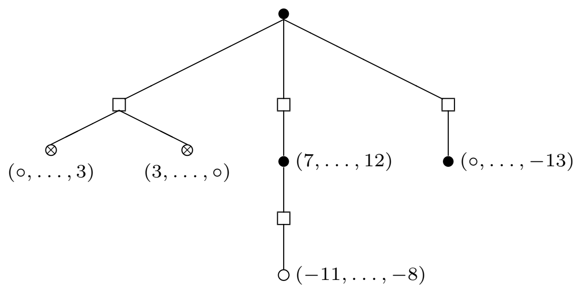


Fig. 1. The tree T associated to the genomes A and B of Example 1 has two grey leaves, one white leaf and one black leaf.

Now, we have all necessary results to get rid of unoriented components. The following two propositions are straightforward generalizations of well-known results from the inversion theory [2]. We will start by looking at one single unoriented component.

Proposition 7. *If a component $\mathcal{C}$ is unoriented, any inversion between adjacencies of the same cycle or the same path of $\mathcal{C}$ orients $\mathcal{C}$, and leaves the number of cycles and paths of the adjacency graph of $\mathcal{C}$ unchanged.*

Orienting a component as in Proposition 7 is called *cutting* the component. Note that this operation leaves the DCJ distance unchanged, and does not create new components.

It is possible to destroy more than one unoriented component with a DCJ operation acting on two unoriented components. The following proposition describes how to *merge* several components, and the relations of this operation to paths in the tree T.

Proposition 8. *A DCJ operation acting on adjacencies of two different unoriented components $\mathcal{A}$ and $\mathcal{B}$ destroys, or orients, all components on the path from $\mathcal{A}$ to $\mathcal{B}$ in the tree T, without creating new unoriented components.*

If the DCJ operation acts on two odd paths, thus on grey components, then merging the two components can be done without changing the number of odd paths, and the DCJ distance is unchanged. If the DCJ operation involves at least one cycle, then merging two components decreases the number of cycles by one, and the DCJ distance will increase by 1 in the resulting pair of genomes.

4.2 Unoriented Sorting

Let T be the tree associated to the components of genome A relative to genome B, and let T' be the smallest subtree of T' that contains all the unoriented components, that is, the white and grey nodes.

Definition 8. *A* cover *of T' is a collection of paths joining all the unoriented components, such that each terminal node of a path belongs to a unique path.*

A path that contains two or more white or grey components, or one white and one grey component, is called a *long* path. A path that contains only one white or one grey component, is a *short* path.

The *cost* of a cover is defined to be the sum of the costs of its paths, where the cost of path is the increase in DCJ distance caused by destroying the unoriented components along the path. Using the remarks following Propositions 7 and 8, we have:

1. The cost of a short path is 1.
2. The cost of a long path with just two grey components is 1.
3. The cost of all other long paths is 2.

An *optimal* cover is a cover of minimal cost. Define t as the cost of any optimal cover of T'. We first establish that t is the difference between the two distances, using the following terminology:

Definition 9. *Given genomes A and B, we call a DCJ operation applied to genome A*

- proper*, if it decreases $d_{DCJ}(A, B)$ by one, i.e. $\Delta(C + I/2) = 1$,*
- improper*, if $d_{DCJ}(A, B)$ remains unchanged, i.e. $\Delta(C + I/2) = 0$, and*
- bad*, if it increases $d_{DCJ}(A, B)$ by one, i.e. $\Delta(C + I/2) = -1$.*

Theorem 3. *If t is the cost of an optimal cover of T', the smallest subtree of T that contains all the unoriented components of genome A relative to genome B, then:*

$$d_{HP}(A, B) = d_{DCJ}(A, B) + t.$$

Proof. First, we will show that $d_{HP}(A, B) \leq d_{DCJ}(A, B) + t$. Consider any cover of the tree T'. Let

- ww be the number of long paths with only white components,
- wg be the number of long paths with white and grey components,
- gg be the number of long paths with only grey components,
- w be the number of short paths with one white component,
- g be the number of short paths with one grey component.

Clearly, we have that the cost t' of this cover is $t' = 2ww + 2wg + gg + w + g$.

Suppose that the adjacency graph $AG(A, B)$ has C cycles and I odd paths. Applying $ww + wg$ bad DCJ operations and $gg + w + g$ improper DCJ operations yields a genome A'. Since each bad DCJ operation merges two cycles or one cycle and a path, the number of cycles in $AG(A', B)$ is $C - ww - wg$. Note that the number of odd paths remains unchanged. Therefore, by Theorem 2, we have that

$$\begin{aligned} d_{HP}(A, B) &\leq d_{HP}(A', B) + ww + wg + gg + w + g \\ &= N - (C + \frac{I}{2}) + 2ww + 2wg + gg + w + g \\ &= d_{DCJ}(A, B) + t'. \end{aligned}$$

Thus, since the above equation is true for any cover, we have: $d_{HP}(A,B) \leq d_{HP}(A,B)+t$.

The fact that $d_{HP}(A,B) \geq d_{HP}(A,B)+t$ is a consequence of the fact that an optimal sorting with HP operation necessarily induces a cover of T' since all unoriented components are eventually destroyed. $\square$

It remains to establish a closed formula for t. A first easy but significant result on the size of t is the following lower bound. Let w be the number of white leaves and g be the number of grey leaves in T'. Since destroying a white leaf costs at least 1 and destroying a grey leaf costs at least 1/2, and t is an integer, we have:

$$w + \left\lceil \frac{g}{2} \right\rceil \leq t.$$

It is quite remarkable, as was observed in the original paper on HP distance [6], that this bound is at most within one rearrangement operation from the optimal solution.

A branch in a tree is called a *long branch* if it has two or more unoriented components. A tree is called a *fortress* if it has an odd number of leaves, all of them on long branches. A standard theorem of the sorting by inversion theory states that the minimal cost to cover a tree that is not a fortress is ℓ, the number of leaves of the tree, and $\ell+1$ in the case of a fortress [2].

We have first:

Theorem 4. *Let w be the number of white leaves and g be the number of grey leaves in T', the smallest subtree of T that contains all the unoriented components of genome A relative to genome B. If the root of T' has more than one child with white leaves, then the minimal cost of a cover of T' is:*

$t = w + \lceil \frac{g}{2} \rceil$ *if the smallest subtree T'' that contains all the white leaves of T' is not a fortress, or g is odd,*

$t = w + \lceil \frac{g}{2} \rceil + 1$ *otherwise.*

Proof. If the subtree T'' is not a fortress then it admits a cover of cost w, and pairing the maximum number of grey nodes yields a cover of T' costing $w + \lceil \frac{g}{2} \rceil$. If the subtree T'' is a fortress, then one of its white leaves is not paired with another leaf since the number of leaves is odd. A cover of T' can be obtained by pairing this white leaf with a grey leaf, which exists if g is odd. The resulting cost will be again $w + \lceil \frac{g}{2} \rceil$ which equals the lower bound and thus the cover is optimal.

If the subtree is a fortress and g is even, we can construct a cover costing $(w+1)+g/2$, using the cover of the fortress and pairing the grey nodes. To show that this cost is minimal, suppose that k grey nodes are paired with k white nodes, the remaining white and grey paired separately. If k is even, then the cost of such a cover would be $(w-k+1)+(g-k)/2+2k$, which is greater than or equal to $(w+1)+g/2$. If k is odd, then the cost of this cover is $(w-k)+(g-k+1)/2+2k$, which is again greater than or equal to $(w+1)+g/2$. $\square$

When all the white leaves belong to a single child of the root, the situation is more delicate. Define a *junior fortress* as a tree with an odd number of white leaves, all of them on long branches, except one that is alone on its branch, called the *top* of the fortress. We have the following:

Theorem 5. *Let $w > 0$ be the number of white leaves and $g > 0$ be the number of grey leaves in T', the smallest subtree of T that contains all the unoriented components of genome A relative to genome B. If the root of T' has only one child c with white leaves then the minimal cost of a cover of T' is:*

$$\begin{array}{ll} t = w + \lceil \frac{g}{2} \rceil & \textit{if } g \textit{ is odd and the subtree } T_c \textit{ that is rooted at } c \\ & \textit{is neither a fortress nor a junior fortress,} \\ t = w + \lceil \frac{g}{2} \rceil + 1 & \textit{otherwise.} \end{array}$$

Proof. Suppose first that $g = 1$, then the only grey leaf either belongs to T_c or not. In the first case, this grey leaf must be the child c implying that T_c is not a junior fortress. If T_c is not a fortress, then there exists a cover with minimal cost equal to the number of leaves of T_c, which is given by $w + \lceil \frac{g}{2} \rceil$, since $g = 1$. If T_c is a fortress, then the minimal cost of a cover is $w + \lceil \frac{g}{2} \rceil + 1$.

In the other case, i.e. the grey leaf does not belong to T_c, then if T_c is a fortress or a junior fortress, the whole tree T' is a fortress with $w + \lceil \frac{g}{2} \rceil$ leaves, yielding a cost of $w + \lceil \frac{g}{2} \rceil + 1$. Otherwise, if T_c is neither a fortress nor a junior fortress, then T' can not be a fortress, and hence can be destroyed with cost $w + 1 = w + \lceil \frac{g}{2} \rceil$.

The same argumentation holds for any $g > 1$ if g is odd.

Now, we consider the case $g = 2$. If T_c is a fortress, two of the white leaves in T_c can be paired with the two grey leaves outside T_c at cost 4. This eliminates the two grey leaves, two of the long white branches, and the branch containing c. The remaining $w - 2$ long branches are paired at cost $w - 2$. Together, this gives a cover of cost $4 + w - 2 = w + \lceil \frac{g}{2} \rceil + 1$. This is optimal since the cost of T' is the same as for T_c. If T_c is not a fortress, we do not need to pair white and grey leaves. T_c can be covered with cost $w + 1$ and the g grey leaves are paired with cost $\lceil \frac{g}{2} \rceil$, giving again a total cost of $w + \lceil \frac{g}{2} \rceil + 1$.

If $g > 2$ and g is even, it is always possible to pair the grey leaves, as long as there are more than two left, and then apply the case $g = 2$. This gives the same cost $w + \lceil \frac{g}{2} \rceil + 1$. □

For example, the genomes A and B of Example 1 have $N = 17$ genes. The adjacency graph $AG(A, B)$ has $C = 3$ cycles and $I = 6$ odd paths. After removing the dangling black leaf, the tree T' has $g = 2$ grey leaves and $w = 1$ white leaf (see Fig. 1). Therefore, by Theorem 5, we have $t = 2$ and thus

$$d_{HP}(A, B) = N - (C + \frac{I}{2}) + t = 17 - (3 + 3) + 2 = 13.$$

5 Conclusion

In this paper, we have given a simpler formula for the Hannenhalli-Pevzner genomic distance equation. It requires only a few parameters that can easily be computed directly from the genomes and from simple graph structures derived from the genomes. Traditionally used concepts that were sometimes hard to access, like weak-fortresses-of-semi-real-knots, are bypassed.

References

1. A. Bergeron. A very elementary presentation of the hannenhalli-pevzner theory. In *Proceedings of CPM 2001*, volume 2089 of *LNCS*, pages 106–117. Springer Verlag, 2001.
2. A. Bergeron, J. Mixtacki, and J. Stoye. Reversal distance without hurdles and fortresses. In *Proceedings of CPM 2004*, volume 3109 of *LNCS*, pages 388–399. Springer Verlag, 2004.
3. A. Bergeron, J. Mixtacki, and J. Stoye. On sorting by translocations. *J. Comput. Biol.*, 13(2):567–578, 2006.
4. A. Bergeron, J. Mixtacki, and J. Stoye. A unifying view of genome rearrangements. In *Proceedings of WABI 2006*, volume 4175 of *LNBI*, pages 163–173. Springer Verlag, 2006.
5. A. Bergeron and J. Stoye. On the similarity of sets of permutations and its applications to genome comparison. In T. Warnow and B. Zhu, editors, *Proceedings of COCOON 2003*, volume 2697 of *LNCS*, pages 68–79. Springer Verlag, 2003.
6. S. Hannenhalli and P. A. Pevzner. Transforming men into mice (polynomial algorithm for genomic distance problem). In *Proceedings of FOCS 1995*, pages 581–592. IEEE Press, 1995.
7. G. Jean and M. Nikolski. Genome rearrangements: a correct algorithm for optimal capping. *Inf. Process. Lett.*, 104:14–20, 2007.
8. M. Ozery-Flato and R. Shamir. Two notes on genome rearrangements. *J. Bioinf. Comput. Biol.*, 1(1):71–94, 2003.
9. G. Tesler. Efficient algorithms for multichromosomal genome rearrangements. *J. Comput. Syst. Sci.*, 65(3):587–609, 2002.
10. G. Tesler. GRIMM: Genome rearrangements web server. *Bioinformatics*, 18(3):492–493, 2002.
11. S. Yancopoulos, O. Attie, and R. Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346, 2005.

A Proof of Theorem 2

Components whose both extremities are genes, often called *real* components, are well studied in the context of sorting a single chromosome with the same flanking genes [1]. We have the following:

Proposition 9 ([1]). *An oriented real component has an oriented DCJ-sorting operation that does not create new unoriented components.*

Components that contain one or two telomere are called *semi-real*.

Proposition 10. *An oriented semi-real component whose associated permutation is oriented can be sorted with oriented DCJ-sorting operations.*

Proof. We will show that such components can be embedded in oriented real components with the same DCJ distance. Then, we can sort the component with oriented DCJ-sorting operations. The basic idea is the following: if the component has one telomere, add an extra gene 0 or k to the associated permutation. This transforms the – only – odd path into a cycle and preserves the DCJ distance. If the component has two telomeres – it spans a whole chromosome – flip the chromosome as necessary in order to "close" each odd path into a cycle. It is then easy to show that the DCJ distance is preserved. □

Proposition 11. *A semi-real component whose adjacency graph has even paths can be sorted with oriented DCJ-sorting operations.*

Proof. First, note that the semi-real component $\mathcal{C} = (l, \ldots, r)$ has two even paths. Consider the permutation $(p_1, \ldots, p_k)$ associated to component $\mathcal{C}$. If the permutation is oriented, then it is possible to sort the component with oriented DCJ-sorting operations by Proposition 10.

Now, if the permutation is unoriented, then all genes p_1 to p_k have the same sign. There exist two possible fissions: fission F_1 creating telomere $(k, \circ)$ and fission F_2 creating $(\circ, 1)$. It can be shown that one of these two fissions does not create new unoriented components. □

Definition 10. *A DCJ operation creating the adjacency (a, b) of B, where a and b are genes, is called* interchromosomal*, if (a, x) and (y, b) belong to different chromosomes in A.*

1. *If $x \neq \circ$ and $y \neq \circ$, the DCJ operation is a translocation.*
2. *If $x = \circ$ or $y = \circ$, the DCJ operation is a semi-translocation.*
3. *If $x = \circ$ and $y = \circ$, the DCJ operation is a fusion.*

The next proposition is the key, it says that for any interchromosomal DCJ operation that creates an unoriented component there always exists an alternative interchromosomal DCJ-sorting operation that does not. This statement, already proven in the context of sorting by translocations in [3], can be shown similarly for the general case.

Proposition 12. *Given two linear genomes A and B, if an interchromosomal DCJ operation creates an unoriented component, then there exists another interchromosomal DCJ-sorting operation that does not.*

Theorem 2. *Given two linear genomes A and B, $d_{HP}(A, B) = d_{DCJ}(A, B)$ if and only if there are no unoriented components.*

Proof. The "if" part comes from the fact that we can sort a genome without unoriented components with DCJ-sorting operations (Propositions 10, 11, 12), adding the fact that semi-real components whose graphs have even paths can be "destroyed" by fissions. The "only if" part comes from the fact that if there are unoriented components, then $d_{DCJ}(A, B) < d_{HP}(A, B)$, since we showed in Proposition 4 that all DCJ-sorting operations create circular chromosomes in these cases. □