Bridging Disparate Views on the DCJ-Indel Model for a Capping-Free Solution to the Natural Distance Problem

Leonard Bohnenkämper

Faculty of Technology and Center for Biotechnology (CeBiTec), Bielefeld University, Germany

Abstract

One of the most fundamental problems in genome rearrangement is the (genomic) distance problem. It is typically formulated as finding the minimum number of rearrangements under a model that are needed to transform one genome into the other. A powerful multi-chromosomal model is the Double Cut and Join (DCJ) model.

While the DCJ model is not able to deal with some situations that occur in practice, like duplicated or lost regions, it was extended over time to handle these cases. First, it was extended to the DCJ-indel model, solving the issue of lost markers. Later ILP-solutions for so called *natural genomes*, in which each genomic region may occur an arbitrary number of times, were developed, enabling in theory to solve the distance problem for any pair of genomes. However, some theoretical and practical issues remained unsolved.

On the theoretical side of things, there exist two disparate views of the DCJ-indel model, motivated in the same way, but with different conceptualizations that could not be reconciled so far.

On the practical side, while the solutions for natural genomes typically perform well on telomere to telomere resolved genomes, they have been shown in recent years to quickly loose performance on genomes with a large number of contigs or linear chromosomes. This has been linked to a particular technique increasing the solution space superexponentially named *capping*.

Recently, we introduced a new conceptualization of the DCJ-indel model within the context of another rearrangement problem. In this manuscript, we will apply this new conceptualization to the distance problem. In doing this, we uncover the relation between the disparate conceptualizations of the DCJ-indel model. We are also able to derive an ILP solution to the distance problem that does not rely on capping and therefore significantly improves upon the performance of previous solutions for genomes with high numbers of contigs while still solving the problem exactly. To the best of our knowledge, our approach is the first allowing for an exact computation of the DCJ-indel distance for natural genomes with large numbers of linear chromosomes.

We demonstrate the performance advantage as well as limitations in comparison to an existing solution on simulated genomes as well as showing its practical usefulness in an analysis of 11 *Drosophila* genomes.

2012 ACM Subject Classification Applied computing → Bioinformatics

Keywords and phrases Comparative Genomics, Genome Rearrangement, Double-Cut-And-Join, Indels, Integer Linear Programming, Capping

Digital Object Identifier 10.4230/LIPIcs.WABI.2023.22

Supplementary Material *Text*: `https://doi.org/10.6084/m9.figshare.23552247.v1`
Software: `https://gitlab.ub.uni-bielefeld.de/gi/ding-cf`
archived at `swh:1:dir:d01607c38f4a3c75816eaca253b5a4f7131143ac`
Software: `https://gitlab.ub.uni-bielefeld.de/gi/ffs-dcj`
archived at `swh:1:dir:e727f21413734ee5cde8d042ae941fc537099eef`

Acknowledgements I thank my supervisor Jens Stoye and Marília D. V. Braga for their helpful comments in discussions regarding notation, terms and overall structure of the paper. I furthermore thank Daniel Doerr for making me switch from water to lava and all members of the genome informatics group of Bielefeld University for their advice on how to present my research in this manuscript.

1 Introduction

In genome rearrangement studies, genomes are analyzed on a high level. Most often, the basic unit used is therefore not nucleotides, but oriented genetic *markers*, such as genes. The most fundamental problem in theoretical studies of genome rearrangements is the *distance problem*, which asks to provide the minimum number of rearrangements needed to transform one genome into the other under a restricted set of operations, also called a *model*.

In early approaches, such as the inversion model [12], solutions to the distance problem focused primarily on unichromosomal data, in which each marker appeared exactly once in each genome. These assumptions limited the applications of the models to real biological data, which often contained multiple chromosomes and a wide variety of marker distributions. Since then, researchers have sought to enable models to handle more realistic data. A major breakthrough was the DCJ-model introduced by Yancopoulos et al. in 2005 [18], a simple model that was nonetheless capable of handling multiple chromosomes. In 2010, Braga, Willing and Stoye extended the DCJ-model to the DCJ-indel model, enabling it to handle markers unique to one genome [5]. An independent, equivalent conceptualization of the same DCJ and indel operations was developed by Compeau in 2012 [7], although the precise relationship of the two conceptualizations remained unclear [8]. We refer to these views as the BWS- and Compeau-conceptualization respectively.

In 2020, the BWS-conceptualization was combined with previous results by Shao et al. [17] in [4] to yield the performant ILP solution `ding` for genome pairs with arbitrary distributions of markers, the so called *natural genomes*. In theory, `ding` enables the computation of the rearrangement distance between any pair of genomes available today.

However, `ding` uses a technique known as *capping*, which transforms linear chromosomes into circular ones during solving time. As described in [15], capping increases the solution space of ILPs like `ding` super-exponentially in the number of linear chromosomes. Since many assemblies available today are not resolved on a chromosome level and instead fragment into sometimes thousands of contigs, this renders distance computation infeasible yet again for many available genomes today. In [15], Rubert and Braga develop a heuristic solution to reduce the search space spanned by capping. Nonetheless, no exact solutions for the DCJ-indel distance problem of natural genomes avoiding capping exist as of yet.

In this work, we apply a new view on the DCJ-indel model developed in [3] to the distance problem. Using this, we are able to bridge the gap between the BWS- and Compeau conceptualizations in Section 3.1. Furthermore, this new conceptualization lends itself to a distance formula which is simple enough to be developed into a capping-free ILP (Section 4), which we then evaluate in Section 5 to show its performance advantage over `ding`.

2 Problem Definition

For this work, we use the same notation as in our previous work. Therefore large parts of this section are adapted from [3]. We conceptualize a genome $\mathbb{G}$ as a graph $(X_{\mathbb{G}}, M_{\mathbb{G}} \cup A_{\mathbb{G}})$. Its vertices $X_{\mathbb{G}}$ are the beginnings m^t and ends m^h of markers $m := \{m^t, m^h\} \in M_{\mathbb{G}}$, which we refer to as *extremities*. The genome's *adjacencies* $A_{\mathbb{G}}$ are undirected edges $\{m^x, n^y\} \in A_{\mathbb{G}}$, which signify that the extremities m^x and n^y are neighboring on the same chromosome. As a shorthand notation, we write ab for an adjacency $\{a, b\}$. Both $A_{\mathbb{G}}$ and $M_{\mathbb{G}}$ are required to form a matching on $X_{\mathbb{G}}$.

Because of that requirement, each path in $\mathbb{G}$ is simple and alternates between markers and adjacencies. A component of a genome is thus either a linear or circular simple path. We refer to them as linear and circular *chromosomes* respectively. The extremities in which

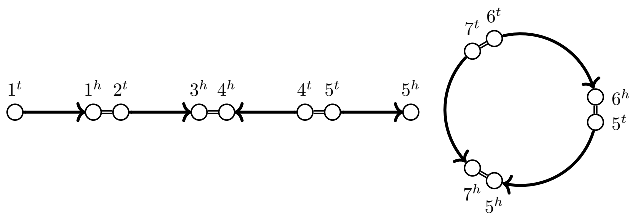

Figure 1 Genome of 7 markers with one linear and one circular chromosome. Markers drawn as arrows, adjacencies drawn as double lines.

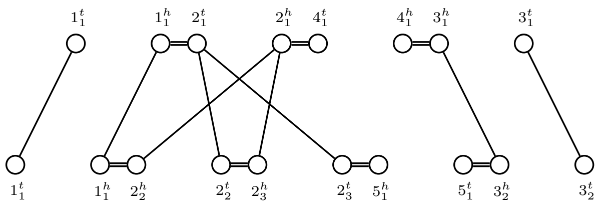

Figure 2 MRD for two genomes on an unresolved homology ($\equiv_1$) with families $\{1_1, 1_2\}$, $\{2_1, 2_2, 2_3\}, \{3_1, 3_2\}, \{4_1\}, \{5_1\}$.

a linear chromosome ends are called *telomeres*. Additionally, we refer to a subpath of a chromosome of which the first and last edge are markers as a *chromosome segment* (called a *marker path* in [3]). An example of a genome is given in Figure 1.

In our model, each marker is unique, thus there are no markers shared between genomes. Therefore, in order to calculate a meaningful distance between genomes, we borrow a concept from biology, namely *homology*. Homology can be modeled as an equivalence relation on the markers, i.e. $m \equiv n$ for some $m, n \in M_{\mathbb{G}}$. We call the equivalence class $[m]$ of a marker m its *family*. We also extend the equivalence relation to the extremities with $m^t \equiv n^t$ and $m^h \equiv n^h$ if and only if $m \equiv n$, but require no head being equivalent to any tail, i.e. $m^t \not\equiv n^h \forall m, n \in M_{\mathbb{G}}$. We can then extend the equivalence relation to adjacencies as $ab \equiv cd$ if and only if both of the extremities are equivalent, i.e. $a \equiv c \wedge b \equiv d$ or $a \equiv d \wedge b \equiv c$.

To illustrate our concept of homology, we introduce the Multi-Relational Diagram (MRD), a graph data structure here that is also useful for the distance computation. We deviate from the definition in [4] by omitting indel edges from our definition.

▶ **Definition 1.** *The MRD of two genomes $\mathbb{A}, \mathbb{B}$ and a homology relation $(\equiv)$ is a graph $\mathcal{MRD}(\mathbb{A}, \mathbb{B}, \equiv) = (V, E)$ with $V = X_{\mathbb{A}} \cup X_{\mathbb{B}}$ and two types of edges $E = E_\gamma \cup E_\xi$, namely adjacency edges $E_\gamma = A_{\mathbb{A}} \cup A_{\mathbb{B}}$ and extremity edges $E_\xi = \{\{x, y\} \in X_{\mathbb{A}} \times X_{\mathbb{B}} \mid x \equiv y\}$.*

We give an example of a MRD in Figure 2. We see that in that example, 4_1 and 5_1 have no homologues in the other genome respectively. We refer to such markers as *singular*. Additionally, We call a circular or linear chromosome consisting only of singular markers a circular or linear *singleton*.

Note also that the family $\{2_1, 2_2, 2_3\}$ in this example has more than just one marker per genome. We call markers of such families *ambiguous*. We refer to a homology, in which no markers are ambiguous as *resolved*. In order to determine the precise nature of rearrangements occurring between two genomes, it is helpful to find a *matching* between the markers of two genomes.

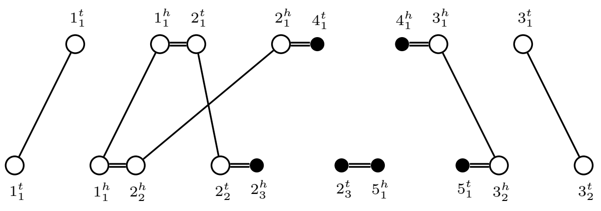

Figure 3 MRD for two genomes on a resolved homology ($\overset{\star}{\equiv}_1$) with families $\{1_1, 1_2\}$, $\{2_1, 2_2\}, \{2_3\}, \{3_1, 3_2\}, \{4_1\}, \{5_1\}$. Extremities of singular markers (called lava vertices from Section 3 onward) are filled black. ($\overset{\star}{\equiv}_1$) is a (maximal) matching on ($\equiv_1$) of Figure 2.

▶ **Definition 2.** *A* matching *($\overset{\star}{\equiv}$) on a given homology ($\equiv$) is a resolved homology for which holds $m \overset{\star}{\equiv} n \implies m \equiv n$ for any pair of markers m, n.*

We call two genomes $\mathbb{A}, \mathbb{B}$ *equal* under a homology ($\equiv$), if there is a matching ($\overset{\star}{\equiv}$) on ($\equiv$), such that each marker and adjacency of $\mathbb{A}$ has one equivalent in $\mathbb{B}$ under $\overset{\star}{\equiv}$ and vice versa.

We note that when the homology is resolved, in the MRD at most one extremity edge connects to each vertex. Because the adjacencies form a matching on the extremities, the resulting MRD consists of only simple cycles and paths. We therefore call such MRDs *simple*. We note that a simple MRD fits the definition of a simple rearrangement graph as studied in Section 3 of [3]. An example of a simple MRD is given in Figure 3.

Rearrangements in our transformation distance are modeled by the *Double-Cut-And-Join (DCJ)* operation. A DCJ operation applies up to two cuts in the genome and reconnects the incident extremities or telomeres. More formally, we can write as in [2]:

▶ **Definition 3.** *A* DCJ operation *transforms up to two the adjacencies $ab, cd \in A_{\mathbb{A}}$ or telomeres s, t of genome $\mathbb{A}$ in one of the following ways:*

- *$ab, cd \to ac, bd$ or $ab, cd \to ad, bc$*
- *$ab, s \to as, b$ or $ab, s \to bs, a$*
- *$ab \to a, b$*
- *$s, t \to st$*

To model markers being gained or lost, we introduce segmental insertions and deletions.

▶ **Definition 4.** *An* insertion *of length k transforms a genome $\mathbb{A}$ into $\mathbb{A}'$ by adding a chromosome segment $p = p_1, p_2, ..., p_{2k-1}p_{2k}$ to the genome. Note that this adds the markers $(p_1, p_2), ..., (p_{2k-1}, p_{2k}) \in M_{\mathbb{A}'}$. An insertion may additionally either add the adjacency $p_{2k}p_1 \in A_{\mathbb{A}'}$, apply the transformation $ab \to ap_1, p_{2k}b$ for an adjacency ab or the transformation $s \to p_1 s$ for a telomere s. A* deletion *of length k removes the chromosome segment $p = p_1, ..., p_{2k}$ and creates the adjacency ab if previously $ap_1, p_{2k}b \in A_{\mathbb{A}}$.*

We are now in a position to formulate the distance problem as finding a shortest transformation of DCJ and indel operations of one genome into the other.

▶ **Problem 5.** *Given two genomes, $\mathbb{A}, \mathbb{B}$ and a homology ($\equiv$), find a shortest sequence $s_1, ..., s_k$ of DCJ and indel-operations transforming $\mathbb{A}$ into a genome equal to $\mathbb{B}$. We call the length of k the DCJ-indel distance of $\mathbb{A}, \mathbb{B}$ under ($\equiv$) and write $d_{DCJ}^{id}(\mathbb{A}, \mathbb{B}, \equiv) = k$.*

The original DCJ-indel model by Braga et al. [6] only allowed indels on chromosome segments of singular markers to avoid scenarios that deleted and reinserted whole chromosomes. For a resolved homology $\overset{\star}{\equiv}$, we call $\overline{d_{DCJ}^{id}}(\mathbb{A}, \mathbb{B}, \overset{\star}{\equiv})$ the *restricted* DCJ-indel distance if we allow only indels of segments comprised solely of singular markers in scenarios in Problem 5.

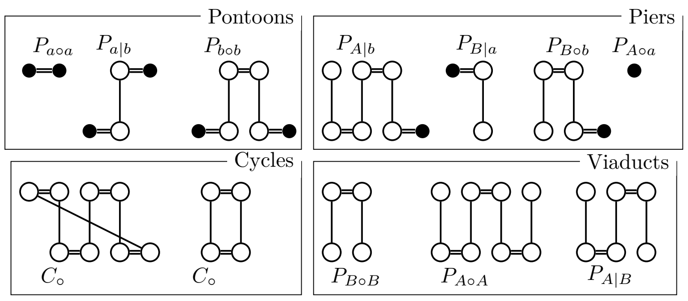

Figure 4 All different types of components in a simple MRD. Vertices of genome $\mathbb{A}$ are on the top, vertices of genome $\mathbb{B}$ are on the bottom of each component. Lava vertices are filled black.

For unresolved homologies, we can apply the same model by just finding a matching on the original homology. However, in order to not create a similar "free lunch" issue, we restrict ourselves to an established model, the *Maximum Matching model* [11]. We call a matching ($\overset{\pm}{\equiv}$) on a homology ($\equiv$) *maximal* if it has at most one singular marker for every family in ($\equiv$).

▶ **Problem 6.** *Given two genomes, $\mathbb{A}, \mathbb{B}$ and a homology ($\equiv$), find a maximal matching ($\overset{\pm}{\equiv}$) on ($\equiv$), such that $\overline{d^{id}_{DCJ}}(\mathbb{A}, \mathbb{B}, \overset{\pm}{\equiv})$ is minimized.*

3 A New DCJ-Indel Distance Formula

We note that the only maximal matching on a resolved homology ($\overset{\star}{\equiv}$) is ($\overset{\star}{\equiv}$) itself. Thus, for resolved homologies, in any scenario for Problem 6, we know deletions can only affect singular markers. Let us now regard the MRD of a pair of genomes $\mathbb{A}, \mathbb{B}$ for a resolved homology ($\overset{\star}{\equiv}$). Since each marker has at most one homologue, each vertex is connected to at most one extremity edge. Since adjacency edges form a matching on the vertices, again, the graph consists only of simple cycles and paths. All cycles are even and we write the set of cycles as $\mathbb{C}_\circ$. Paths can end either in a vertex without an extremity edge or adjacency edge. We name the vertices, in which a path ends in its *endpoints*. Vertices without extremity edges are special, because, as we established earlier, they are the extremities of the markers that will be part of indels during the sorting. We therefore name them *lava vertices*. The other type of vertex is a vertex without an adjacent adjacency edge, i.e. a *telomere*. Note that there is a special case wherein a lava vertex can also be a telomere. We can then identify different types of paths by their endpoints. We write a or b for a lava vertex and A or B for a telomere, depending on whether its part of genome $\mathbb{A}$ or $\mathbb{B}$. We then obtain a partition of paths into 10 different subsets, namely $\mathbb{P}_{A\circ A}, \mathbb{P}_{A|B}, \mathbb{P}_{B\circ B}, \mathbb{P}_{A\circ a}, \mathbb{P}_{A|b}, \mathbb{P}_{B|a}, \mathbb{P}_{B\circ b}, \mathbb{P}_{a\circ a}, \mathbb{P}_{a|b}, \mathbb{P}_{b\circ b}$. In order to be consistent with [3], we use $\circ$ and $|$ to distinguish even and odd paths respectively. Furthermore, we write $p_{x(*)y}$ as a shorthand for the cardinality of $\mathbb{P}_{x(*)y}$ and $P_{x(*)y}$ for a generic example of an element of $\mathbb{P}_{x(*)y}$.

Usually it is not necessary to think of all 10 different sets as separate entities, because they behave very similarly with respect to applied DCJ or indel operations. In textual form we therefore often use a coarser distinction, naming paths with two lava vertices as *pontoons*, paths with a telomere and a lava vertex as *piers* as well as paths with two telomeres as *viaducts*. An overview of this notation is given in Figure 4.

Another notation, we adopt from [3] is for a DCJ $ab, cd \to ac, bd$ affecting the adjacencies ab and cd in components K_{ab}, K_{cd} of the MRD respectively, we can instead view the DCJ as $K_{ab}, K_{cd} \to K_{ac}, K_{bd}$ transforming the components K_{ab}, K_{cd} into K_{ac}, K_{bd}. In combination

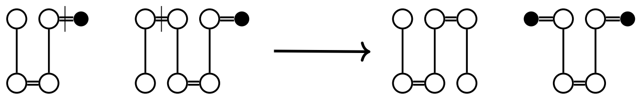

Figure 5 An example of a DCJ operation that can be written as $P_{A\circ a}, P_{B|a} \to P_{A|B}, P_{a\circ a}$.

with the generic member notation from above, we can write operations abstractly like so: $P_{A\circ a}, P_{B|a} \to P_{A|B}, P_{a\circ a}$. For reference, we have also shown this DCJ operation in Figure 5. Based on this notation and with the help of observations from [3], it is possible to derive a distance formula. We do so in detail in Supplement S.1. However, this formula is equivalent to that of Compeau and BWS as we will see in the following subsection. We thus only state it here.

▶ **Theorem 7.** *For two genomes $\mathbb{A}, \mathbb{B}$ and a resolved homology $(\overset{\star}{\equiv})$ for which both genomes contain no circular singletons, we have the distance formula*

$$\overline{d^{id}_{DCJ}}(\mathbb{A}, \mathbb{B}, \overset{\star}{\equiv}) = n - c_\circ + \left\lceil \frac{p_{a|b} + \max(p_{A\circ a}, p_{B|a}) + \max(p_{A|b}, p_{B\circ b}) - p_{A|B}}{2} \right\rceil$$

with n the number of matched markers*, $n = |\{(m,n) \in M_{\mathbb{A}} \times M_{\mathbb{B}} \mid m \overset{\star}{\equiv} n\}|$.*

Note that the constraint to genomes without circular singletons constitutes no serious restriction, as Compeau showed that circular singletons each require one indel operation and can thus be dealt with in pre-processing [8].

To more easily address individual terms in the formula, we use the followig shorthands,

$$\begin{aligned} F &:= n - c_\circ + \tilde{P} := n - c_\circ + \left\lceil \frac{\tilde{p}}{2} \right\rceil \\ &:= n - c_\circ + \left\lceil \frac{p_{a|b} + \max(p_{A\circ a}, p_{B|a}) + \max(p_{A|b}, p_{B\circ b}) - p_{A|B}}{2} \right\rceil . \end{aligned}$$

3.1 Relation of the BWS- and Compeau-Conceptualization

We now examine how the terms in our distance formula relate to both the Compeau- and BWS-conceptualizations of the DCJ-indel model. In doing that, we uncover the nature of the relation between these two views that have been perceived as entirely separate since their conception [8].

Braga et al. [6] and Compeau [8] use the adjacency and breakpoint graphs respectively. Both graphs are strongly related to the MRD. In fact, one obtains the adjacency graph by collapsing all adjacency edges of a simple MRD and the breakpoint graph by collapsing all its extremity edges. In order to avoid confusion, we will present their results here as if they had been formulated on a simple MRD. When consulting the original works in [6, 8], the reader should keep this in mind. Particularly in [8] the length of a path is determined by its adjacency edges instead of by its extremity edges as defined here. Therefore, parities of viaducts and pontoons are exactly opposite in [8] to the ones stated here.

We will compare the models by examining the chromosome segments that are deleted or inserted (see Definition 4), which we refer to as *indel groups*. We say an adjacency ab or its extremities a, b are *part of an indel group* p if there is $a'b' \equiv ab$ with $a'b' \in p$ or p starts and ends in a' and b'. In terms of indel groups, our view is closely related to the BWS-conceptualization, because both create the indel groups implicitly during sorting (see Supplement S.1). In terms of the graph, our conceptualization is more closely related to

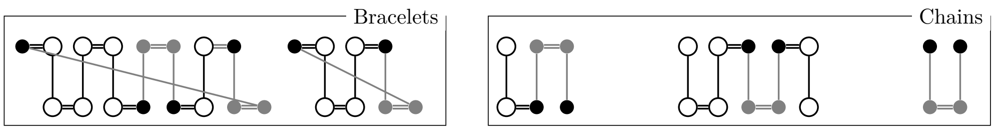

Figure 6 Components resulting from a completion as in [8]. Vertices and Edges added during completion are colored in grey.

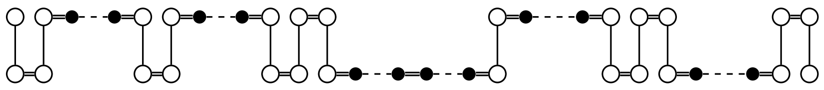

Figure 7 Path as found in [4] as another way of writing the paths in [6] by adding indel edges between lava vertices of the same gene. Indel edges here drawn in dashed. In this work, indel edges are omitted and the collection of components arising is called a *bridge*.

Compeau's because it essentially operates on the same type of components (lava vertices are called *open* in [8], piers are π- and γ-paths and pontoons are $\{\pi,\pi\}$-, $\{\pi,\gamma\}$- and $\{\gamma,\gamma\}$-paths). However, in [8], indels are not modeled as an explicit operation, but instead emulated by integrating or excising artificial circular chromosomes during sorting. Adding the correct chromosomes, the *completion*, is therefore the main problem solved in [8]. These additional chromosomes are then the explicitly constructed indel groups in the sorting. Because the homology of the markers needed for the completion is already known beforehand on a resolved homology, the task is to find the correct new adjacencies to add to the graph. Then, if an adjacency $a'b'$ is found in the completion, the extremities $a \equiv a', b \equiv b'$ of the originally singular markers will be part of the same indel group. Once the completion is constructed, there are no more lava vertices in the graph. Instead, former piers and pontoons are joined into new components, either *bracelets*, which are circular and consist of pontoons only, or *chains*, which consist of two piers and possibly pontoons. An example of a completion can be found in Figure 6.

In [6], lava vertices are avoided by viewing singular markers as part of adjacencies of matched markers, called *$\mathcal{G}$-adjacencies*. This is equivalent to connecting the head and tail vertex of a singular marker with a special type of edge, called *indel edge* as is done in [4]. We call a collection of components that can be traversed as one once indel edges are introduced a *crossing*. We distinguish between circular crossings called *ferries* and linear crossings called *bridges*.

▶ **Definition 8.** *A* pontoon bridge *$b_1,..,b_k$ for $k \geq 2$ is a string of components b_i, such that b_1, b_k are piers, $(b_i)_{i=2}^{k-1}$ are pontoons and there are singular markers $(m_i)_{i=1}^{k-1}$ with $m_i \neq m_j$ for $i \neq j$ whose extremities are contained as lava vertex in b_i, b_{i+1} for all m_i. A string of components is called a* bridge *if it is a pontoon bridge or consists of a single viaduct.*

▶ **Definition 9.** *A* pontoon ferry *$f_1,...,f_l$ for $l \geq 1$ is a string of pontoons f_i, such that here are singular markers $(m_i)_{i=1}^{l}$ with $m_i \neq m_j$ for $i \neq j$ whose extremities are contained as lava vertices in f_i, f_{i+1} for all m_i for $i < l$ and the extremities of m_l are contained in f_1 and f_l. A string of components is called a* ferry *if it is a pontoon ferry or consists of a single cycle.*

Ferries and bridges are cycles and paths in [6] respectively. An example of a bridge can be found in Figure 7. Crossings are first sorted separately in [6], so we start our comparison by doing the same. We thus aim to find *internal* operations that only involve components of the same crossing. During sorting, we want to make sure that the operations we apply are not only optimal in the context of the crossing, but in the graph as a whole. There are

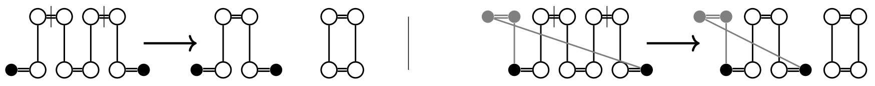

Figure 8 Safe DCJ operations accumulating markers separated by even pontoons (in [6] called a *run*) remain optimal in the safe bracelet joining the extremities of these markers.

certain operations that are guaranteed to be optimal because they reduce F in any MRD by 1, no matter which other components are found in the graph. We call such an operation *safe*. For example, extracting a cycle from any component is safe (as $\Delta c_\circ = 1$), whereas recombining two even piers, such as $P_{A\circ a}, P_{A\circ a} \to P_{A\circ A}, P_{a\circ a}$ is not safe, because it is only optimal under the premise that $p_{A\circ a} > p_{B|a}$. There are only 7 distinct types of safe DCJ operations. We list them in Table 1. We also note that as in [6], instead of sorting $\mathbb{A}$ to $\mathbb{B}$, we can sort both $\mathbb{A}$ and $\mathbb{B}$ to a common genome. By thinking this way, we can better exploit the symmetry of the situation.

Table 1 All safe types of DCJ operations. Each reduces the F by 1, no matter the number of other components in the graph. Above are all safe operations in a pure DCJ scenario. The operations below can also function as safe deletions if one of the resultants in brackets is removed. For reference: $F = n - c_\circ + \left\lceil (p_{a|b} + \max(p_{A\circ a}, p_{B|a}) + \max(p_{A|b}, p_{B\circ b}) - p_{A|B})/2 \right\rceil$.

Safe operation	$-\Delta c_\circ$	$\Delta p_{a\mid b}$	$\Delta \max(p_{A\circ a}, p_{B\mid a})$	$\Delta \max(p_{A\mid b}, p_{B\circ b})$	$-\Delta p_{A\mid B}$
$K \to K' + C_\circ$	-1	0	0	0	0
$P_{A\circ A} \to P_{A\mid B}, P_{A\mid B}$	0	0	0	0	-2
$P_{B\circ B} \to P_{A\mid B}, P_{A\mid B}$	0	0	0	0	-2
$P_{A\circ A}, P_{B\circ B} \to P_{A\mid B}, P_{A\mid B}$	0	0	0	0	-2
$P_{a\mid b}, P_{a\mid b} \to (P_{a\circ a})^*, (P_{b\circ b})^*$	0	-2	0	0	0
$P_{A\circ a}, P_{B\mid a} \to P_{A\mid B}, (P_{a\circ a})^*$	0	0	-1	0	-1
$P_{A\mid b}, P_{B\circ b} \to P_{A\mid B}, (P_{b\circ b})^*$	0	0	0	-1	-1

The most obvious safe operation is the extraction of an even cycle from another component. If one continues to extract even cycles from an even pontoon $p = x_1...x_k$, one arrives at the pontoon $p' = x_1x_k$, which consists of a single adjacency. The corresponding markers can then be dealt with with the same indel operation, meaning x_1, x_k are part of the same indel group. Braga et al. notice the same thing in [6]; they refer to markers that are only separated by even pontoons as a *run*, which they notice can be "accumulated" in this fashion. For an extensive example, see Figure S.2.1 in Supplement S.2, Steps (a), (b). In [8], genomes are not explicitly sorted, so there is no true equivalent to safe operations, but Compeau systematically finds chains and bracelets he can be sure are optimal in any breakpoint graph (Algorithm 9, Steps 1 to 3). We therefore call these chains and bracelets safe, too. In fact, the very first safe bracelet Compeau identifies, is a 1-bracelet consisting of a single even pontoon (Lemma 5 in [8]). If one creates this bracelet from the even pontoon $p = x_1...x_k$ the adjacency added for the completion is $x'_1x'_k$ with $x_1 \equiv x'_1$ and $x_k \equiv x'_k$. Thus, here too, x_1x_k are part of the same indel group. This way of constructing the indel groups is shown in Supplement S.2, Figure S.2.2 with Bracelets (a), (b). Notice also that the safe operations sorting the two adjacent lava vertices of an even pontoon together remain optimal in a bracelet like this (see Figure 8).

The next safe bracelet Compeau finds, is joining two odd pontoons together. He shows that it is safe by ruling out all other uses of two pontoons as at best co-optimal (Lemma 6, Proof of Thm 8 and Step 2 of Algorithm 9 in [8]). An example can be found as Bracelet (c) of Figure S.2.2. This again, corresponds to a safe operation, namely $P_{a|b}, P_{a|b} \to P_{a\circ a}, P_{b\circ b}$. In fact, all safe chains and bracelets of two components correspond directly to safe operations. We

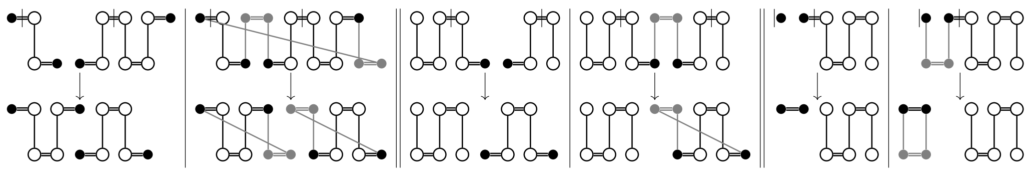

Figure 9 For all safe DCJ operations with two piers or pontoons as sources, there is a safe bracelet or chain in which the same operation is optimal and vice versa.

have visualized this fact in Figure 9. Note that the corresponding safe operation again remains optimal in the safe chain or bracelet. Because of this more direct correspondence between the Compeau-conceptualization and our formula, we focus more on the correspondence between the BWS-conceptualization and our formula in the following. Braga et al. identify the same operation by noticing that the number of runs can be reduced by 2 if one applies cuts in between between runs of $\mathbb{A}$ and $\mathbb{B}$ (see Proposition 3 in [6]). This is of course precisely a DCJ with two odd pontoons as sources in our model. Because the resultants of this operation are the two even pontoons $P_{a\circ a}, P_{b\circ b}$, these can in turn be reduced to single adjacencies by excising even cycles. Again, the implication for indel groups in all models is that for two odd pontoons $p_1 = a_1x_1, ..., x_kb_1, p_2 = a_2x_{k+1}, ..., x_lb_2$, the adjacency a_1a_2 can be part of the same indel group if b_1b_2 is part of the same indel group and vice versa. This equivalence is further illustrated by comparing the effects of Steps (c) and (d) of Figure S.2.1 to Bracelet (c) of Figure S.2.2 of Supplement S.2.

Dealing in this fashion with all pontoons of a crossing, we reduce all but possibly one odd pontoon to single adjacency edges, which can then be dealt with in a single indel operation. Because ferries must contain an even number of odd pontoons, they can be sorted entirely by safe operations in this way. To quantify the number of operations needed, Braga et al. define the *indel potential* $\lambda(X)$ of a crossing X as the number of indel operations obtained in a DCJ-optimal sorting [6]. Since it is possible to trade off indel and DCJ operations, this definition is not easily reflected in the other conceptualizations. However, as they show that sorting a crossing X separately needs $d^{id}_{DCJ}(X) = \mathrm{d}_{DCJ}(X) + \lambda(X)$ steps, we can also think of the indel potential as the overhead introduced by the singular markers if we sort the crossing separately. In [6], it is shown that $\lambda(X) = \left\lceil \frac{\Lambda(X)+1}{2} \right\rceil$ with $\Lambda(X)$ the number of runs for a crossing X. If a ferry contains at least two runs, we can find a bijection between runs and odd pontoons. Denoting $q(X)$ as the contribution to quantity q by crossing X. We can thus write $\Lambda(X) = p_{a|b}(X)$ for a ferry with at least two runs. Therefore, we find for a ferry X with at least two runs, their formula translates to ours,

$$\begin{aligned} n(X) - c(X) + \lambda(X) &= n(X) - c(X) + \left\lceil \frac{\Lambda(X)+1}{2} \right\rceil \\ &= n(X) - 1 + \frac{\Lambda(X)+2}{2} = n(X) + \frac{\Lambda(X)}{2} = n(X) + \left\lceil \frac{p_{a|b}(X)}{2} \right\rceil . \end{aligned}$$

Similarly, this equivalence can be shown if there is only 1 run in X. By the Compeau method, if there are d singular markers, d markers are added as part of completion chromosomes, so the number of markers after completion is $N = n + d$. Meanwhile, each $P_{a\circ a}$ and $P_{b\circ b}$ creates a bracelet. Each pair $P_{a|b}, P_{a|b}$ also forms a bracelet. Since $d = p_{a|b} + p_{a\circ a} + p_{b\circ b}$, we have

$$\begin{aligned} n(X) + \left\lceil \frac{p_{a|b}(X)}{2} \right\rceil &= n(X) + \frac{p_{a|b}(X)}{2} = n(X) - \frac{p_{a|b}(X)}{2} + p_{a|b}(X) \\ &= n(X) + d(X) - \frac{p_{a|b}(X)}{2} - p_{a\circ a}(X) - p_{b\circ b}(X) \\ &= N(X) - (p^{\pi,\pi}(X) + p^{\gamma,\gamma}(X) + \left\lfloor \frac{p^{\pi,\gamma}(X)}{2} \right\rfloor), \end{aligned}$$

which is precisely the Compeau formula if no piers or viaducts are involved. We see that our formula acts as a sort of missing link between the two other formulas here. Since ferries can be dealt with entirely with internal safe operations, this formula can even be generalized to the whole graph for circular genomes. In fact, this has been done in [4], yielding our formula for this specific case.

Using this way of examining the contribution of individual crossings, we were also able to re-calculate the indel potential with our formula for all 10 types of bridges in [6]. The results can be found in Table S.2.1 of Supplement S.2. Notably, when sorting a bridge independently, one can also first exhaust all safe operations. After this, only the piers and possibly a single odd pontoon might be "left over" (see also Figure S.2.1 after Step (d)). We call these components *unsaturated*. Since each safe operation also has a corresponding safe chain or bracelet, these are also the only components, which end up in unsafe chains if one restricts the completion to a single crossing (compare to Figure S.2.2). Since every other component can be dealt with safe operations, unsaturated components are the only ones that might have to be involved in what is called in [6] a *(path) recombination*, that is, a DCJ operation going beyond a single crossing. When studying recombinations, we can therefore abstract from any concrete bridge $p = p_1, ..., p_k$ with piers p_1, p_k and only write it as its unsaturated components, that is $p_1 p_k$ if p contains an even number of odd pontoons or as $p_1 P_{a|b} p_k$ otherwise. We call this the *reduced bridge.* Interestingly, Braga et al. make the same abstraction and identify the bridges by the genome of their telomeres and the genome of the first and last run. This direct correspondence is illustrated by comparing Columns 1 and 4 of Table S.2.1. In [6], another observation is that (reduced) bridges of the type $P_{A|b}, P_{B\circ b}$ or $P_{A\circ a}, P_{B|a}$ never need to appear as sources for any recombination. Using our conceptualization, we can confirm that because $P_{A|b}, P_{B\circ b} \to P_{A|B}, P_{b\circ b}$ and $P_{A\circ a}, P_{B|a} \to P_{A|B}, P_{a\circ a}$ are safe operations, these types of bridges can be sorted entirely by internal safe operations. It therefore makes sense to group them as in [6] with viaducts, the other type of bridge that can be sorted in this way.

All other bridges might need recombinations to be sorted optimally. If there is a safe operation between the components of two bridges, we know that this recombination must be optimal. In fact, if we only regard unsaturated components, we see that the only remaining safe operations are (i) $P_{A\circ a}, P_{B|a} \to P_{A|B}, P_{a\circ a}$, (ii) $P_{A|b}, P_{B\circ b} \to P_{A|B}, P_{b\circ b}$ and (iii) $P_{a|b}, P_{a|b} \to P_{a\circ a}, P_{b\circ b}$. We know (either by combinatorics or Table S.2.1) that each source of (i) and (ii) appears in 3 types of (reduced) bridges and thus there are $3 \times 3 = 9$ path recombinations facilitated by each of these two safe operations. For (iii), we have 4 types of (reduced) bridges containing $P_{a|b}$ and thus $\binom{4}{1} + \binom{4}{2} = 10$ path recombinations using this operation. Of course, these are not mutually exclusive, but since Operations (i) and (ii) involve the end of a bridge and one of its resultants is a viaduct, we can always choose to do one of these operations first, upon which all other possible safe operations on piers and pontoons will be in the same component and will not require any further recombinations. In [6] all of these recombinations are catalogued. We were able to confirm this by recreating their tables of recombinations with $\Delta d \leq 0$ as Table S.2.2 of Supplement S.2. It is easily checked that (i) and (ii) each occur 9 and (iii) occurs 10 times. The unsaturated components after the operation in these cases form precisely those bridges listed in [6] as the resultant(s). The precise difference for the distance as opposed to sorting the crossing separately can then be derived by comparing the term $\tilde{P}$ in our formula on the graphs containing each bridge separately and on a graph containing the union of the two bridges (see Table S.2.2 Columns 3, 6, 9, 10). In summary, we can see that in all but two cases, the DCJ chosen to recombine the bridges in [6] is safe and the resultants are exactly comprised of the unsaturated components after the operation.

The two exceptions are the recombinations of $P_{A\circ a}, P_{A\circ a}$ with $P_{B\circ b}, P_{B\circ b}$ and $P_{A|a}, P_{A|b}$ with $P_{B|a}, P_{B|a}$ (marked with $\star$ in the table). In these cases, there is no safe operation and therefore all piers remain unsaturated. The reason this recombination can still be done in some cases is that an unsafe operation like $P_{A\circ a}, P_{B\circ b} \to P_{A|B}, P_{a|b}$ in this specific case reduces F by one, but since there are equally optimal internal operations (i.e. $P_{A\circ a}, P_{A\circ a} \to P_{A\circ A}, P_{a\circ a}$) in this case, this recombination actually never has to be used. The only task remaining is then to find a sequence of recombinations that improve upon the distance. Braga et al. give this as their *recombination groups*. We have listed these groups in Table S.2.3 of Supplement S.2. The first observation is that by exhausting all safe DCJ operations in a recombination group, we are able to create the unsaturated components of what are called in [6] *reusable resultants*. In combination with our observations about pairwise recombinations, we thus know that all recombinations in the groups can be facilitated purely by safe DCJs. We also see that in many cases, after sorting a group, no further unsaturated components are present. In the other cases, Braga et al. make sure that all partners of the unsaturated components are "used up" in earlier recombinations of the table (see last column) such that the unsafe operations sorting the unsaturated components are still optimal.

4 Capping-free Generalization to Natural Genomes

In this section, we describe briefly how to generalize the distance formula presented as Theorem 7 into an ILP for which no capping of the MRD is required. For reference, the full ILP is given in Appendix A as Algorithm 1. The ILP works by determining a matching on the markers as described in Problem 6. The basic framework (Constraints `C.01` to `C.06`) is the same as for `ding` [4] and the ILP by Shao et al. [17]: Variable x is used to indicate whether or not an edge is part of the matching and z_v marks the vertex v with the lowest index $\mathtt{ix}(v)$ in its component. We also adopt the way circular singletons are dealt with in [4] as Constraint `C.18`. The only major change we make w.r.t. [4] in Constraints `C.01` to `C.06` is the addition of Constraint `C.02`, where we allow for different matching models than the maximum matching model by specifying an upper (U_f) and lower bound (L_f) for the number of markers to be matched per family f. Specifications for how to set these bounds to achieve the maximum matching model and other popular models can be found in Table 2 of Appendix A. Another minor change is that we have a variable $d_{g(u)}$ indicating whether a gene $g(u)$ of an extremity u is to be singular instead of indel edges as in [4].

To distinguish different types of paths, we use binary variables to track endpoints, namely $n_v^a, n_v^b, n_v^A, n_v^B$ as well as $m_v^a, m_v^b, m_v^A, m_v^B$ for each vertex v. A variable n_v^i is used to represent the sub-path starting with the adjacency edge at v ending in $i \in \{A, B, a, b\}$ while the m_v^i does the same, just for the sub-path starting with the extremity edge at v. We set these variables accordingly at the end of a path (see Constraints `C.07`, `C.08`). We also require only one of the variables be set per vertex (`C.09`). The variables are then required to be the same if their respective vertices are connected by the respective edge (see Constraint `C.10`), "passing" the label through the edge. Lastly, we require the m and n variables of a vertex to be equal unless that vertex is labeled by z_v, i.e. it has the smallest identifier in the component (`C.11`). If z_v is set, we then report the component type based on the adjacent sub-paths in r_v^{ij} if the sub-paths have the correct labels m_v^i and n_v^j (or m_v^j and n_v^i) set (Constraints `C.13`, `C.14`). We reserve the special case $m_v^i = 0 \forall i \wedge n_v^j = 0 \forall j$ for cycles and report cycles via r_v^c in these constraints. The remaining constraints deal with applying the maximization function (`C.15`, `C.16`) for $\tilde{p}$ and calculating $\tilde{P}$ as variable q (`C.17`).

As an additional optimization, we set m variables to 0 in those components of the MRD that have either no telomere or indels of a given type in Constraint `C.19`.

5 Evaluation of the ILP

We implemented the ILP described in the previous section and made it publicly available here: `https://gitlab.ub.uni-bielefeld.de/gi/ding-cf`. We refer to this implementation as `ding-cf` for the rest of this work.

In this section, we show results of applying the ILP to both simulated and real data and comparing its performance to the python3 version of `ding` [4], namely `dingII`, a similar ILP solution to the DCJ-indel distance problem for natural genomes. In contrast to `ding-cf`, `dingII` uses the capping technique.

We first test the ILPs on simulated data in Subsection 5.1 before demonstrating the practical usefulness of rearrangement analyses even on contig level resolved genomes by analysing 11 Drosophila genomes in Subsection 5.2.

5.1 Performance Evaluation on Simulated Data

We initially planned to use the simulation script that comes with `dingII`, but due to the script regularly encountering stack overflows on large genomes owing to its reliance on recursion, we instead re-implemented it in `C++`. This implementation is available at `https://gitlab.ub.uni-bielefeld.de/gi/ffs-dcj`.

The re-implementation has the same features as the original script with only two minor changes. Firstly, instead of the number of DCJ-operations to be performed being passed to the script with additional numbers of other operations (insertions, deletions and duplications) being randomly performed according to rate parameters, our simulation takes a fixed number of total operations and distributes them according to rates relative to a rate of 1 for DCJ operations. Secondly, our simulation is not yet able to simulate arbitrary trees, but instead only simulates the topology `(A,B);`, which was used in [4]. For more detail on the simulation, the interested reader is referred to the description of the original simulation script in [4].

In our experiments, we simulated two genomes from a common root for each sample. In all experiments, we set the length of the root genomes to 20,000 markers and performed 10,000 operations in total, with an insertion rate of 0.1 and an deletion rate of 0.2 unless specified otherwise. For reference, this amounts to 5882 DCJ operations in expectation for a duplication rate of 0.4 to compare to experiments run with the python script of `dingII`. The shape parameter for the Zipf distribution was set to 4 for indel lengths and to 6 for duplication lengths. In all experiments, we used `gurobi10.0` on a single thread on an AMD EPYC 7452 Processor to solve the ILPs, limiting its runtime to $1h$ ($3600s$). Both experiments were designed to test parameters to which ILPs like `ding` have been shown to be sensitive.

In our first experiment, we increased the duplication rate in steps of 0.1 from 0.1 to 1.1, generating 10 genome pairs from a root genome with 1 linear chromosome per step. We then created the ILPs for `dingII` and `ding-cf`. The number of ambiguous families ranged from 615 to 2760 (median 2708) in this experiment with the maximum family size per sample reaching up to 7 markers.

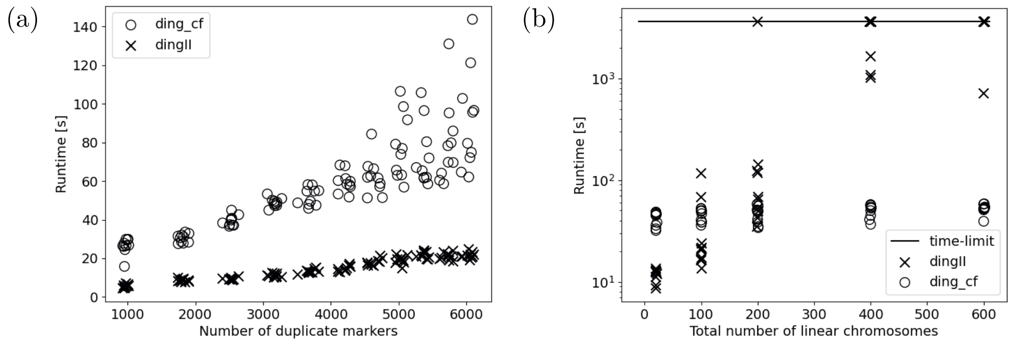

Figure 10 Runtimes for `dingII` and `ding-cf` for genomes simulated in 10,000 steps from a common root, in (a) increasing the duplication rate in steps of 0.1 from 0.1 to 1.1, in (b) increasing the number of linear chromosomes in the root genome progressively from 10 to 50 to 100, 200 and 300.

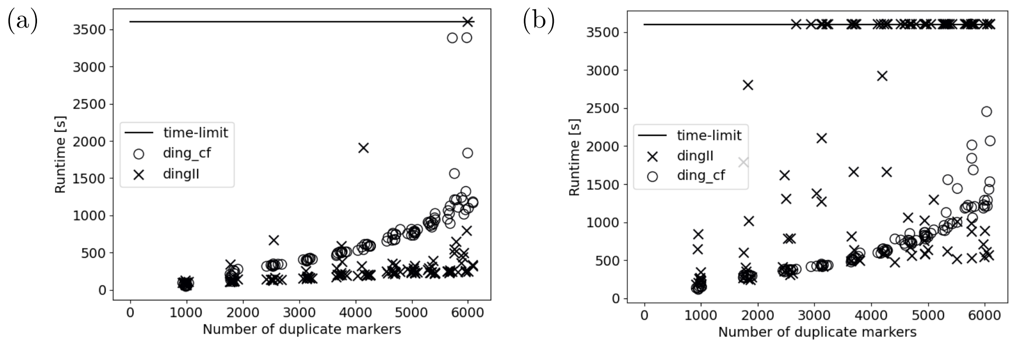

Figure 11 Runtimes for `dingII` and `ding-cf` for genomes simulated in 10,000 steps from a common root increasing the duplication rate in steps of 0.1 from 0.1 to 1.1 with (a) 100 total linear chromosomes and (b) 200 total linear chromosomes on average per sample pair.

We show the results in runtimes of `gurobi10.0` in Figure 10 (a). We see clearly that `dingII` has a performance advantage over `ding-cf` as long as the number of linear chromosomes is low. This is not surprising as `dingII` works in a very similar manner to `ding-cf`, but with fewer variables because it only needs to identify odd pontoons due to capping transforming other types of paths to pontoons and cycles. However, we see that the performance loss is not dramatic, staying well within a few minutes of solving time for this experiment. Nonetheless, the expected exponential increase in runtimes of `ding-cf` happens earlier than the one demonstrated for `ding` in [4]. We were able to further verify that on genomes with few linear chromosomes, `ding-cf` behaves similarly to `ding` for varying different parameters but having worse performance overall in Supplement S.3.

To test the actual use case for `ding-cf`, that is, high numbers of linear chromosomes, we increased the number of linear chromosomes in the root genome progressively from 10 to 50 to 100, 200 and 300 chromosomes with a fixed duplication rate of 0.4 and 10 samples per step. The runtimes are shown in Figure 10. We see that up to 100 linear chromosomes in the simulated pair of genomes, `dingII` on average outperforms `ding-cf`, but its runtime rises exponentially until the majority of the `dingII` ILPs are not solved within an hour of solving time. Meanwhile, the runtimes of `ding-cf` are stable throughout the experiments, staying below 100 seconds in each case.

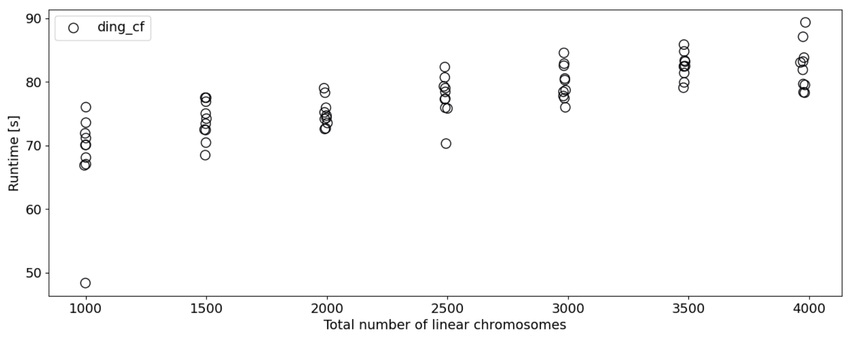

Figure 12 Runtimes for `ding-cf` for genomes simulated from a root with 500 to 2000 chromosomes in steps of 250.

In order to test the composite effect of the number of duplicates and the number of linear chromosomes on solving times, we repeated the first experiment (Figure 10 (a)) with 50 and 100 linear chromosomes at the root genome, resulting in total numbers of about 100 and 200 linear chromosomes for each pair. The results (shown in Figure 11) indicate that both ILPs react strongly to the combined effect for the first increase to 100 linear chromosomes with `dingII` still outperforming `ding-cf` on these samples whereas the second increase to 200 only has a minor effect on `ding-cf` while for `dingII` many of the pairs with high duplicate numbers become unsolvable within an hour.

To confirm that the number of linear chromosomes alone only plays a minor part in the runtime of `ding-cf`, we ran another experiment, this time keeping the duplication rate fixed at 0.4 and increasing the number of linear chromosomes in the root genome from 500 to 2000 in steps of 250 with 10 samples per step. The runtimes are given in Figure 12 and exhibit only a minor, linear increase. In fact, the increase is so slow that even for 2000 linear chromosomes at the root (c.a. 4000 linear chromosomes of the pair in total), the runtime is still below 100 seconds for all 10 samples.

5.2 Analysis of Drosophila Genomes

We obtained 11 assemblies of species in the *Drosophila* genus previously analyzed by Rubert and Braga [15]. We used `FFGC` to extract the longest transcript of each locus and ran `OrthoFinder` version 2.3.7 [10] to obtain orthologous groups. We then translated the genomes into unimog files using the orthogroups as families and translating linear contigs into linear chromosomes. We then filtered out any empty chromosomes. The genomes obtained in this fashion comprised 13,143 markers spread on 97 linear chromosomes on average. More detailed statistics about the genomes after this preprocessing step are listed in Table S.4.1 of Supplement S.4.

We then used `ding-cf` to calculate pairwise distances, running `gurobi10.0` on a single thread on an AMD EPYC 7452 Processor for 24 hours. Of the 55 resulting ILPs, we obtained an exact result for 9 and approximate results for 45, all of which deviated at less than 2% from the exact solution. Only one run, namely *D. melanogaster* vs *D. willistoni* did not yield any result within 24 hours. We therefore re-ran the solver on this ILP, this time using 15 threads and a time limit of 20 hours. In this run, an approximate solution with 0.48% gap was found. We give the distance data obtained in this manner in Table S.4.3 and detailed performance results in Table S.4.2 of Supplement S.4. Additionally we performed an experiment with the same parameters with the `dingII` ILP. Table S.4.2 shows that even though this dataset is not extremely fragmented, `ding-cf` outperforms `dingII` on the majority of samples.

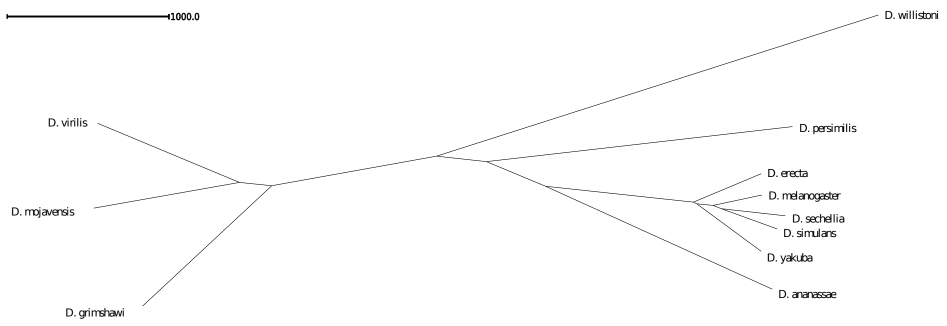

Figure 13 Neighbor joining tree inferred from the distances in Table S.4.3 using `SplitsTree4`. Edge lengths are drawn proportional to their weight. The absolute edge lengths can be found in Supplement S.4.

Phylogenetic Analysis. We proceeded to construct a phylogenetic tree via Neighbor Joining using `SplitsTree4` [13]. The tree, shown in Figure 13, is entirely consistent with the current state of knowledge about the Drosophila phylogeny. Additionally, the phylogenetic signal in the distance data is remarkably strong. To demonstrate this fact, we calculated the distance matrix for the path metric of the tree and compared it to the distances calculated by `ding-cf`. On average, the tree path metric deviates only by 0.53% per entry from the distances calculated by `ding-cf` with the largest relative difference being 2.2% for the distance of *D. melanogaster* and *D. simulans*. For reference, we give the full distance matrix of the path metric in Table S.4.4 of Supplement S.4. We were able to further confirm this strong correspondence between the tree and the distance data via a split decomposition with `SplitsTree4` in Supplement S.4.1 [1, 13]. Overall, judging from these experiments, `ding-cf` looks promising as a distance measure for phylogenetic analyses.

However, we want to draw the reader's attention to one possible pitfall of our method as a phylogenetic tool, namely that the fragmentation of the genome itself appears as a signal in the distance data. To emphasize this, let us pose a hypothetical extreme example: Consider a comparison between two assemblies $\mathbb{A}, \mathbb{B}$ with n markers each, with a matching between all markers of $\mathbb{A}$ and $\mathbb{B}$. Suppose $\mathbb{A}$ is fully assembled into one chromosome and $\mathbb{B}$ fragments into n contigs of one marker. No matter the actual structure of the underlying (true) genome of $\mathbb{B}$, the DCJ distance between the assemblies $\mathbb{A}$ and $\mathbb{B}$ is always $n - 1$. The size of this effect for practical levels of fragmentation needs to be investigated, particularly whether these problems could be exacerbated by biases in the assembly method used to arrive at the studied pair of genomes, such as might be the case for comparative assembly strategies.

Detecting Synteny. We extracted the matchings from the ILP solutions calculated by `gurobi` and plotted them with `Circos` [14]. We show the matching between *D. virilis* and *D. mojavensis* in Figure 14 as compared to just the marker matches identified by `OrthoFinder`. The plots for all other pairs can be found in Supplement S.4.2. We see that even though there are some big rearrangements, such as inversions and transpositions as indicated by the arcs as well as an abundance of duplicates, the calculated matching identifies large syntenic blocks, sometimes even matching the majority of markers of whole contigs to each other.

Moreover, for many of the smaller contigs all markers are matched to markers of exactly one large contig of the other species. Matchings like this could therefore possibly be used to aid in improving very fragmented assemblies, given a sufficiently closely related and resolved reference genome.

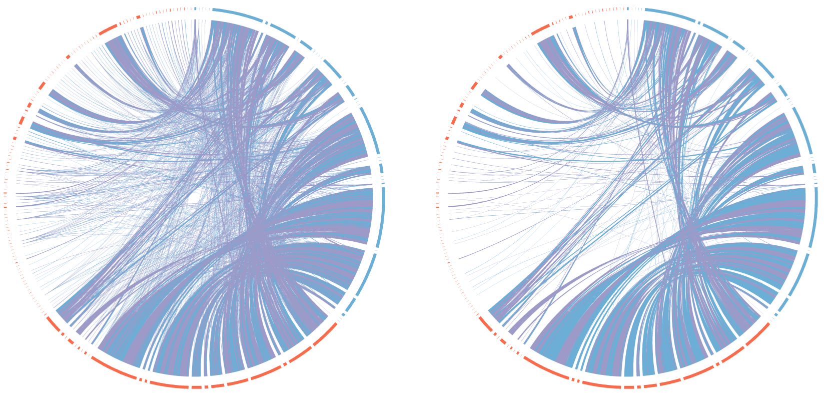

Figure 14 Circos plots for Contigs of *D. virilis* (red segments) and *D. mojavensis* (blue segments). Blue arcs show common markers with the same direction, purple arcs show common markers with different directions. On the left: before matching. On the right: after matching with `ding-cf`.

6 Conclusion

We presented a new, simpler distance formula for the DCJ-indel model. Using this distance formula, we were able to explain the previously unclear relationship between the BWS- and Compeau-conceptualizations of the DCJ-indel model. Furthermore, our formula is easily generalizeable to a performant ILP solution that enables the distance computation even for genomes fragmented into thousands of contigs. We have shown that a DCJ-indel analysis can be meaningful even with relatively fragmented genomes by applying the ILP to 11 *Drosophila* assemblies. From this we obtained a well resolved phylogeny with little noise in the distance data, indicating that our method could be well suited for distance based phylogenetic analyses provided the effect size of genome fragmentation in the particular use case can be bounded. We also showed that the ILP can be used to disambiguate orthologous and paralogous regions, which has potential use cases in orthology assignment and the finalization of fragmented assemblies.

Furthermore, we are confident that using this new formula, capping-free versions of other existing algorithms, such as for the family-free distance problem as in [16, 15] and parsimony problems as in [9] can be devised.

References

1 Hans-Jürgen Bandelt and Andreas W.M. Dress. Split decomposition: A new and useful approach to phylogenetic analysis of distance data. *Molecular Phylogenetics and Evolution*, 1(3):242–252, 1992. `doi:10.1016/1055-7903(92)90021-8`.

2 Anne Bergeron, Julia Mixtacki, and Jens Stoye. A unifying view of genome rearrangements. In Philipp Bücher and Bernard M. E. Moret, editors, *Algorithms in Bioinformatics*, pages 163–173, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

3 Leonard Bohnenkämper. The floor is lava - halving genomes with viaducts, piers and pontoons. In *Comparative Genomics*. Springer International Publishing, to appear.

4 Leonard Bohnenkämper, Marília D.V. Braga, Daniel Doerr, and Jens Stoye. Computing the rearrangement distance of natural genomes. *Journal of Computational Biology*, 28(4):410–431, 2021. PMID: 33393848. `doi:10.1089/cmb.2020.0434`.

5 Marília D. V. Braga, Eyla Willing, and Jens Stoye. Genomic distance with DCJ and indels. In Vincent Moulton and Mona Singh, editors, *Algorithms in Bioinformatics*, pages 90–101, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

6 Marília D.V. Braga, Eyla Willing, and Jens Stoye. Double cut and join with insertions and deletions. *Journal of Computational Biology*, 18(9):1167–1184, 2011. PMID: 21899423. doi:10.1089/cmb.2011.0118.

7 Phillip E. C. Compeau. A simplified view of DCJ-indel distance. In Ben Raphael and Jijun Tang, editors, *Algorithms in Bioinformatics*, pages 365–377, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

8 Phillip Ec Compeau. DCJ-indel sorting revisited. *Algorithms for molecular biology : AMB*, 8(1):6–6, March 2013. doi:10.1186/1748-7188-8-6.

9 Daniel Doerr and Cedric Chauve. Small parsimony for natural genomes in the DCJ-indel model. *Journal of Bioinformatics and Computational Biology*, 19(06):2140009, 2021. PMID: 34806948. doi:10.1142/S0219720021400096.

10 David M. Emms and Steven Kelly. Orthofinder: solving fundamental biases in whole genome comparisons dramatically improves orthogroup inference accuracy. *Genome Biology*, 16(1):157, August 2015. doi:10.1186/s13059-015-0721-2.

11 Guillaume Fertin, Anthony Labarre, Irena Rusu, Eric Tannier, and Stéphane Vialette. *Combinatorics of Genome Rearrangements*. Computational Molecular Biology. The MIT Press, 2009.

12 Sridhar Hannenhalli and Pavel A. Pevzner. Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. *J. ACM*, 46(1):1–27, January 1999. doi:10.1145/300515.300516.

13 Daniel H. Huson and David Bryant. Application of Phylogenetic Networks in Evolutionary Studies. *Molecular Biology and Evolution*, 23(2):254–267, October 2005. doi:10.1093/molbev/msj030.

14 Martin Krzywinski, Jacqueline Schein, Inanc Birol, Joseph Connors, Randy Gascoyne, Doug Horsman, Steven J Jones, and Marco A Marra. Circos: an information aesthetic for comparative genomics. *Genome research*, 19(9):1639–1645, 2009.

15 Diego P. Rubert and Marília D. V. Braga. Gene Orthology Inference via Large-Scale Rearrangements for Partially Assembled Genomes. In Christina Boucher and Sven Rahmann, editors, *22nd International Workshop on Algorithms in Bioinformatics (WABI 2022)*, volume 242 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:22, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.WABI.2022.24.

16 Diego P. Rubert, Daniel Doerr, and Marília D. V. Braga. The potential of family-free rearrangements towards gene orthology inference. *Journal of Bioinformatics and Computational Biology*, 19(06):2140014, 2021. PMID: 34775922. doi:10.1142/S021972002140014X.

17 Mingfu Shao, Yu Lin, and Bernard M.E. Moret. An exact algorithm to compute the double-cut-and-join distance for genomes with duplicate genes. *Journal of Computational Biology*, 22(5):425–435, 2015. PMID: 25517208. doi:10.1089/cmb.2014.0096.

18 Sophia Yancopoulos, Oliver Attie, and Richard Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346, June 2005. doi:10.1093/bioinformatics/bti535.

A Full capping-free ILP

Table 2 Settings for U_f, L_f in Algorithm 1 to enforce different matching models described in [11].

	Maximum Matching (Full)	Intermediate Matching	Exemplar Matching
L_f	$\lfloor \lvert f \rvert / 2 \rfloor$	$\lfloor \lvert f \rvert / 2 \rfloor$	1
U_f	$\lfloor \lvert f \rvert / 2 \rfloor$	1	1

Algorithm 1 ILP for the capping-free computation of the DCJ-indel distance of natural genomes.

Objective:

$$\texttt{Minimize } \frac{1}{2}\sum_{e\in E_\xi} x_e - \sum_{u\in V} r_u^c + q + \sum_{k\in K} s_k$$

Constraints:

(C.01)	$x_e = 1$	$\forall\ e \in E_\gamma^A \cup E_\gamma^B$
(C.02)	$\sum_{\substack{\{x^h,y^h\},x,y \\ \text{of family } f}} x_e \leq U_f$	Family f
	$\sum_{\substack{\{x^h,y^h\},x,y \\ \text{of family } f}} x_e \geq L_f$	Family f
(C.03)	$\sum_{\{u,v\}\in E_\xi} x_{\{u,v\}} + d_{g(u)} = 1$	$\forall\ u \in V$
(C.04)	$x_e = x_d$	$\forall\ e, d \in E_\xi$ such that e and d are siblings
(C.05)	$y_v \leq y_u + \texttt{ix}(v)(1 - x_{\{v,u\}})$	$\forall\ \{v,u\} \in E$,
(C.06)	$\texttt{ix}(v) \cdot z_v \leq y_v$	$\forall\ v \in V$
(C.07)	$n_u^A = 1$	$\forall u$ telomere of A
	$n_{u'}^B = 1$	$\forall u'$ telomere of B
(C.08)	$m_u^i \geq d_{g(u)}$	$\forall u \in V$ with genome $i \in \{a,b\}$
(C.09)	$\sum_{j\in Y} m_u^j \leq 1$	$\forall u \in V$
	$\sum_{j\in Y} n_u^j \leq 1$	$\forall u \in V$
(C.10)	$m_u^j \leq m_v^j + (1 - x_e)$	$\forall e = \{u,v\} \in E_\xi \forall j \in Y$
	$n_u^j \leq n_v^j + (1 - x_e)$	$\forall e = \{u,v\} \in E_\gamma \forall j \in Y$
(C.11)	$m_u^j \leq n_u^j + z_u$	
	$n_u^j \leq m_u^j + z_u$	$\forall u \in V \forall j \in Y$
(C.12)	$r_u^c + \sum_{ij\in W} r_u^{ij} = z_u$	$\forall u \in V$
(C.13)	$r_u^{ij} - m_u^k - n_u^k \leq 0$	$\forall u \in V, \forall ij \in W \forall k \in \{i,j\}$
	$8r_u^c \leq 8 - (\sum_i m_u^{i\in Y} + \sum_{j\in Y} n_u^j)$	$\forall u \in V$
(C.14)	$r_u^{ij} \geq m_u^i + n_u^j - 1$	$\forall u \in V \forall ij \in W$
	$r_u^{ij} \geq m_u^j + n_u^i - 1$	$\forall u \in V \forall ij \in W$
(C.15)	$r_{ABa} \geq \sum_{u\in V} r_u^{Aa}$	
	$r_{ABa} \geq \sum_{u\in V} r_u^{Ba}$	
(C.16)	$r_{ABb} \geq \sum_{u\in V} r_u^{Ab}$	
	$r_{ABb} \geq \sum_{u\in V} r_u^{Bb}$	
(C.17)	$2q \geq \sum_{v\in V} r_v^{ab} + r_{ABa} + r_{ABb} - \sum_{v\in V} r_v^{AB}$	
(C.18)	$\sum_{g\in k} d_g - \lvert k\rvert + 1 \leq s_k$	$\forall k \in K$
(C.19)	$m_u^i = 0$	$\forall u \in V$ if u is in a component without type i.

Domains:

(D.01)	$d_g \in \{0,1\}$	gene g	(D.06)	$m_u^j \in \{0,1\}$	$\forall u \in V \forall j \in Y$	
(D.02)	$x_e \in \{0,1\}$	$\forall\ e \in E$	(D.07)	$r_u^{ij} \in \{0,1\}$	$\forall u \in V, \forall ij \in W \cup \{c\}$	with $Y = \{A,a,B,b\}$
(D.03)	$0\ \leq y_v \leq \texttt{ix}(v)$	$\forall\ v \in V$	(D.08)	$r_{ABa}, r_{ABb}, q \in \mathbb{N}_0$		and $W = \{AB, Aa, Ab, Ba, Bb, ab\}$.
(D.04)	$z_v \in \{0,1\}$	$\forall\ v \in V$	(D.09)	$s_k \in \{0,1\}$	$\forall\ k \in K$	with K the circular chromosomes of $\mathbb{A}$ and $\mathbb{B}$.
(D.05)	$n_u^j \in \{0,1\}$	$\forall u \in V \forall j \in Y$	(D.10)	$y_v \in \{0,...,\texttt{ix}(v)\}$	$\forall v \in V$	with $\texttt{ix}(v)$ the index of vertex $v \in V$.