A Unifying View of Genome Rearrangements

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

[1] Comparative Genomics Laboratory, 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. Genome rearrangements have been modeled by a variety of operations such as inversions, translocations, fissions, fusions, transpositions and block interchanges. The *double cut and join* operation, introduced by Yancopoulos *et al.*, allows to model all the classical operations while simplifying the algorithms. In this paper we show a simple way to apply this operation to the most general type of genomes with a mixed collection of linear and circular chromosomes. We also describe a graph structure that allows simplifying the theory and distance computation considerably, as neither capping nor concatenation of the linear chromosomes are necessary.

1 Introduction

The problem of sorting multichromosomal genomes can be stated as: Given two genomes A and B, the goal is to find a shortest sequence of rearrangement operations that transforms A into B. The length of such a shortest sequence is called the *distance* between A and B. Clearly, the solutions depend on what kind of rearrangement operations are allowed.

Given their prevalence in eukaryotic genomes [1], the usual choices of operations include translocations, fusions, fissions and inversions. However, there are some indications that transpositions should also be included in the set of operations [2], but the lack of theoretical results showing how to include transpositions in the models led to algorithms that simulate transpositions as sequences of inversions.

In [3], the authors describe a general framework in which circular and linear chromosomes can coexist throughout evolving genomes. They model inversions, translocations, fissions, fusions, transpositions and block interchanges with a single operation, called the *double cut and join* operation. This general model accounts for the genomic evidence of the coexistence of both linear and circular chromosomes or plasmids in many genomes [4,5].

In this paper, we present a simplified formalization of genomes with coexisting circular and linear chromosomes, and a formal treatment of sorting such genomes by the double cut and join operation. We introduce a very simple data structure,

P. Bücher and B.M.E. Moret (Eds.): WABI 2006, LNBI 4175, pp. 163–173, 2006.
© Springer-Verlag Berlin Heidelberg 2006

the *adjacency graph*, that is symmetric with respect to the two genomes under study and is closely related to the visual picture of the genomes themselves. We also show how the algebraic simplicity of the double cut and join operation yields efficient sorting algorithms that can be tailored to optimize the use of certain types of operations.

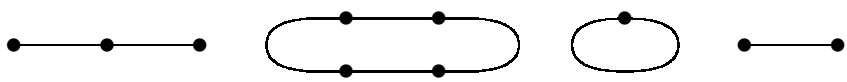
2 Notes on Graphs with Vertices of Degree One or Two

An essential ingredient in genome rearrangment studies are graphs where each vertex has degree one or two. Here we recall some of their properties.

Let G be a graph where each vertex has degree one or two. We call a vertex of degree one *external* and a vertex of degree two *internal*. An internal vertex connecting edges p and q is denoted by the unordered multiset $\{p, q\}$ and an external vertex incident to an edge p by the singleton set $\{p\}$.

It follows immediately from the definition of G that any connected component of G is either circular, consisting only of internal vertices, or it is linear, consisting of internal vertices bounded by two external vertices, one at each end. We denote circular components as *cycles* and linear components as *paths*. A cycle or path is *even* if it has an even number of edges, otherwise it is *odd*.

Example 1. The following graph has four vertices of degree one and six vertices of degree two. It has two cycles and two paths, one of which is even and one of which is odd.



Definition 1. *The* double cut and join *(DCJ) operation acts on two vertices u and v of a graph with vertices of degree one or two in one of the following three ways:*

(a) If both $u = \{p, q\}$ and $v = \{r, s\}$ are internal vertices, these are replaced by the two vertices $\{p, r\}$ and $\{s, q\}$ or by the two vertices $\{p, s\}$ and $\{q, r\}$.
(b) If $u = \{p, q\}$ is internal and $v = \{r\}$ is external, these are replaced by $\{p, r\}$ and $\{q\}$ or by $\{q, r\}$ and $\{p\}$.
(c) If both $u = \{q\}$ and $v = \{r\}$ are external, these are replaced by $\{q, r\}$.

In addition, as an inverse of case (c), a single internal vertex $\{q, r\}$ can be replaced by two external vertices $\{q\}$ and $\{r\}$.

Figure 1 illustrates the definition.

The DCJ operation, although defined locally on a pair of vertices, has global effects on the connected components of the graph. In order to describe these, we use a terminology essentially borrowed from biology.

First, consider Figure 2. If the two vertices are contained in two different paths and at least one of them is internal, then these paths exchange their ends, which

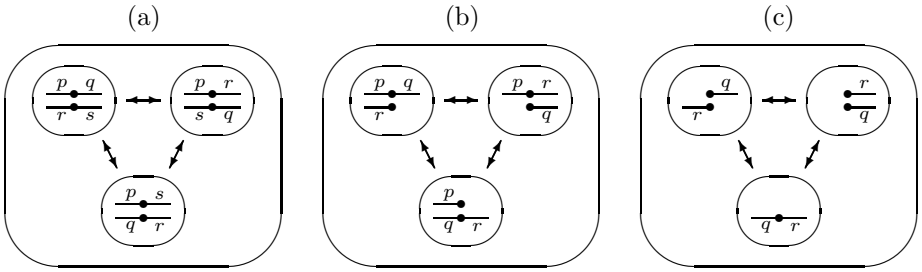


Fig. 1. Definition of the double cut and join operation. Note that the operations between the two top graphs of part (c) are the identity.

is called a *path translocation*. If both are external vertices of different paths, as in Figure 2 (c), then these paths are merged, called a *path fusion*. The inverse of a path fusion is a *path fission*.

The case shown in Figure 3, where both linear and circular components are mixed, is more intricate. If the DCJ operation acts on vertices contained in the same path and at least one of them is internal, then the intermediate part of the path is either reversed, called an *inversion*, or spliced out producing a new cycle, called an *excision*. The inverse operation of an excision is called an *integration*. If both are external vertices of the same path, as in Figure 3 (c), then a cycle is formed, called a *circularization*. Its opposite is a *linearization*.

If the vertices are contained in the same cycle, or in two different cycles, as shown in Figure 4, then we have either an *inversion*, a *cycle fusion* or a *cycle fission*.

The following lemma is an immediate consequence of the enumeration of all possible cases in Figures 2, 3 and 4:

Lemma 1. *The application of a single DCJ operation changes the number of circular or linear components by at most one.*

We will see in the next two sections how graphs with vertices of degree one or two appear in two natural ways when modeling genomes and genome rearrangements.

3 Genes, Chromosomes and Genomes

In this section we introduce our notation of genomes and how they are modeled as graphs with vertices of degree one or two.

A *gene* is an oriented sequence of DNA that starts with a *tail* and ends with a *head*. These are called the *extremities* of the gene. The tail of a gene a is denoted by a_t, and its head is denoted by a_h. In biology, the tail of a gene is often called its 3' end and the head its 5' end.

Two consecutive genes do not necessarily have the same orientation, since DNA is double stranded and the complementary strands are read by the transcription machinery in opposite direction. Thus an *adjacency* of two consecutive genes a and b, depending on their respective orientation, can be of four different types:

$$\{a_h, b_t\}, \{a_h, b_h\}, \{a_t, b_t\}, \{a_t, b_h\}.$$

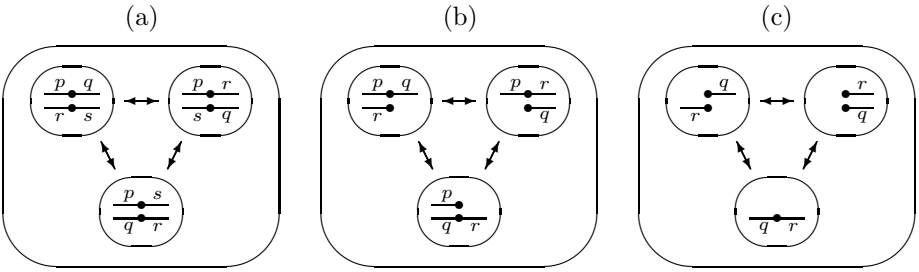


Fig. 2. The DCJ operation applied on one or two paths yields path translocations, fusions and fissions

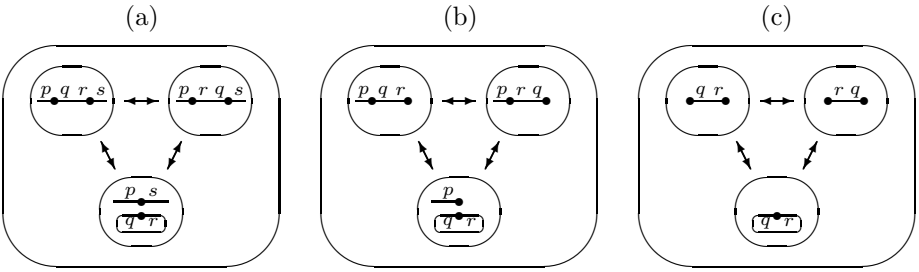


Fig. 3. The DCJ operation applied on a single path or a path and a cycle yields inversions, excisions, integrations, circularizations and linearizations

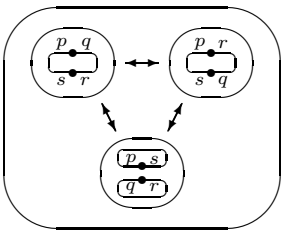


Fig. 4. The DCJ operation applied on a single cycle or on two cycles yields inversions, cycle fusions and fissions

An extremity that is not adjacent to any other gene is called a *telomere*, represented by a singleton set $\{a_h\}$ or $\{a_t\}$.

A *genome* is a set of adjacencies and telomeres such that the tail or the head of any gene appears in exactly one adjancency or telomere.

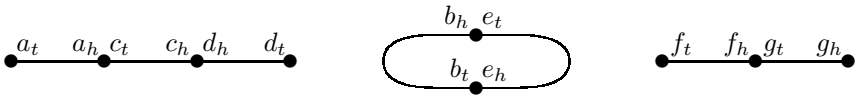
Given a genome, one reconstructs its *chromosomes* by representing the telomeres and adjacencies as vertices and then joining for each gene its tail and its head by an edge. Note that the *genome graph* obtained this way is a graph with vertices of degree one or two. The connected paths and cycles are chromosomes of the genome which are either linear or circular. Linear chromosomes are bounded by telomeres.

Chromosomes are often represented by lists of gene labels. These lists are obtained by choosing a telomere in a linear chromosome, or an arbitrary gene in a circular chromosome, and then enumerating the gene labels along the component, using positive signs to indicate genes that are read from tail to head and negative signs to indicate genes that are read from head to tail. For linear chromosomes, the enumeration stops at its other telomere, for circular chromosomes when the initial gene appears for the second time in the list. Positive signs may be omitted where convenient.

Example 2. Let

$$A = \{\{a_t\}, \{a_h, c_t\}, \{c_h, d_h\}, \{d_t\}, \{b_h, e_t\}, \{e_h, b_t\}, \{f_t\}, \{f_h, g_t\}, \{g_h\}\}$$

be a genome with seven genes $\{a, b, c, d, e, f, g\}$. The corresponding genome graph is the following:



One possible list representation of A is $\{(a, c, -d), (b, e, b), (f, g)\}$.

Since the chromosome graph is a graph with vertices of degree one or two, the double cut and join operation defined in Section 2 can be applied to these graphs. This operation is the same as defined, in different notation, by Yancopoulos *et al.* [3].

We can now formulate the problem that we consider:

The DCJ Sorting and Distance Problem. Given two genomes A and B defined on the same set of genes, find a shortest sequence of DCJ operations that transforms A into B. The length of such a sequence is called the *DCJ distance* between A and B, denoted by $d_{DCJ}(A, B)$.

Example 3. Consider the following two genomes that are defined over the set of genes $\{a, b, c, d, e, f, g\}$:

$$A = \{\{a_t\}, \{a_h, c_t\}, \{c_h, d_h\}, \{d_t\}, \{b_h, e_t\}, \{e_h, b_t\}, \{f_t\}, \{f_h, g_t\}, \{g_h\}\}$$
$$B = \{\{a_h, b_t\}, \{b_h, a_t\}, \{c_t\}, \{c_h, d_t\}, \{d_h\}, \{e_t\}, \{e_h\}, \{f_h, g_t\}, \{g_h, f_t\}\}$$

Sorting A into B can, for example, be done in the following five steps, where the affected gene extremities are underlined:

$$\begin{aligned}
A = {} & \{\{a_t\}, \{a_h, \underline{c_t}\}, \{c_h, d_h\}, \{d_t\}, \{b_h, e_t\}, \{e_h, \underline{b_t}\}, \{f_t\}, \{f_h, g_t\}, \{g_h\}\} \\
& \{\{\underline{a_t}\}, \{a_h, b_t\}, \{c_h, d_h\}, \{d_t\}, \{b_h, \underline{e_t}\}, \{e_h, c_t\}, \{f_t\}, \{f_h, g_t\}, \{g_h\}\} \\
& \{\{e_t\}, \{a_h, b_t\}, \{c_h, \underline{d_h}\}, \{\underline{d_t}\}, \{b_h, a_t\}, \{e_h, c_t\}, \{f_t\}, \{f_h, g_t\}, \{g_h\}\} \\
& \{\{e_t\}, \{a_h, b_t\}, \{c_h, d_t\}, \{d_h\}, \{b_h, a_t\}, \{\underline{e_h}, \underline{c_t}\}, \{f_t\}, \{f_h, g_t\}, \{g_h\}\} \\
& \{\{e_t\}, \{a_h, b_t\}, \{c_h, d_t\}, \{d_h\}, \{b_h, a_t\}, \{e_h\}, \{c_t\}, \{\underline{f_t}\}, \{f_h, g_t\}, \{\underline{g_h}\}\} \\
B = {} & \{\{a_h, b_t\}, \{b_h, a_t\}, \{c_t\}, \{c_h, d_t\}, \{d_h\}, \{e_t\}, \{e_h\}, \{f_h, g_t\}, \{g_h, f_t\}\}
\end{aligned}$$

The DCJ distance between A and B is $d_{DCJ}(A, B) = 5$.

4 The Adjacency Graph

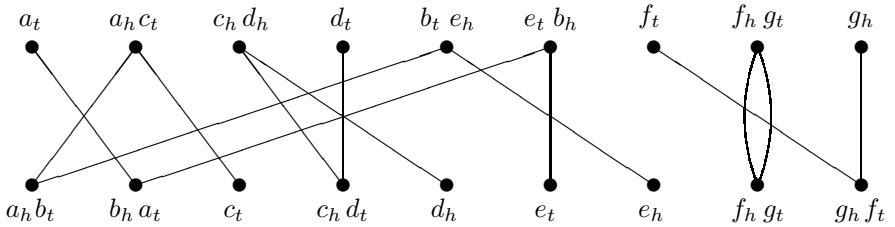
In order to solve the DCJ Distance Problem stated above, another graph of the type discussed in Section 2 proves to be useful, this time defined on the pair of genomes A and B.

Definition 2. *The* adjacency graph $AG(A, B)$ *is a graph whose set of vertices are the adjacencies and telomeres of A and B. For each $u \in A$ and $v \in B$ there are $|u \cap v|$ edges between u and v.*

Example 4. The adjacency graph of our two genomes

$$A = \{\{a_t\}, \{a_h, c_t\}, \{c_h, d_h\}, \{d_t\}, \{b_h, e_t\}, \{e_h, b_t\}, \{f_t\}, \{f_h, g_t\}, \{g_h\}\}$$
$$B = \{\{a_h, b_t\}, \{b_h, a_t\}, \{c_t\}, \{c_h, d_t\}, \{d_h\}, \{e_t\}, \{e_h\}, \{f_h, g_t\}, \{g_h, f_t\}\}$$

is the following:



Obviously, every vertex in the adjacency graph has degree one or two, therefore it is a union of cycles and paths. Since the graph is bipartite, all cycles have even length.

The adjacency graph can easily be constructed as shown in Algorithm 1. Let N be the number of genes in genomes A and B, respectively. Then Algorithm 1 takes $O(N)$ time and uses $O(N)$ space if the genomes are stored in a data structure where, for each gene extremity, one has constant time access to the adjacency or telomere that it is contained in. For example, this can be a table with two rows of length at most $2N$ storing the adjacencies and telomeres of the genome, and another table with two rows of length N storing for each gene in which columns of the first table to find its head and its tail. For genome

Algorithm 1 (Construction of the adjacency graph)

```
create a vertex for each adjacency and each telomere in genomes A and B
for each adjacency {p, q} in genome A do
    create an edge connecting {p, q} and the vertex of genome B that contains p
    create an edge connecting {p, q} and the vertex of genome B that contains q
end for
for each telomere {p} of genome A do
    create an edge connecting {p} and the vertex of genome B that contains p
end for
```

Table 1. Table storing the adjacencies and telomeres of genome A. Adjacencies have two entries, telomeres just one.

	1	2	3	4	5	6	7	8	9
first	a_t	a_h	c_h	d_t	b_h	e_h	f_t	f_h	g_h
second	–	c_t	d_h	–	e_t	b_t	–	g_t	–

Table 2. Table storing for each gene in A the location of its head and its tail in Table 1

	a	b	c	d	e	f	g
head	2	5	3	3	6	8	9
tail	1	6	2	4	5	7	8

$$A = \{\{a_t\}, \{a_h, c_t\}, \{c_h, d_h\}, \{d_t\}, \{b_h, e_t\}, \{e_h, b_t\}, \{f_t\}, \{f_h, g_t\}, \{g_h\}\}$$

from the previous example, the two tables are shown in the following.

5 Sorting by DCJ Operations

As we will see in this section, the adjacency graph allows a simple characterization of many of the properties of sorting by DCJ operations.

Lemma 2. *Let A and B be two genomes defined on the same set of N genes, then we have*

$$A = B \quad \textit{if and only if} \quad N = C + I/2$$

where C is the number of cycles and I the number of odd paths in $AG(A, B)$.

Proof. Let a be the number of adjacencies and t the number of telomeres in $A = B$, then $N = a + t/2$. The adjacency graph $AG(A, B)$ has $C = a$ cycles and $I = t$ odd paths, hence $N = a + t/2 = C + I/2$.

To show that $N = C + I/2$ implies $A = B$, assume an adjacency graph $G = AG(A, B)$ such that $N = C + I/2$. Let a be the number of adjacencies and t the number of telomeres in A, then $N = a + t/2$. Each cycle in G contains at least one adjacency of A, thus $C \leq a$. Each odd path in G contains exactly one telomere of A, thus $I \leq t$. From $C + I/2 = N = a + t/2$ it follows that $C = a$ and $I = t$. Thus all cycles have length two and all odd paths have length one, which is only possible if the genomes are equal. □

When a DCJ operation is applied to genome A, it acts on the adjacencies and telomeres of genome A. The same DCJ operation acts *also* on the adjacency graph since the adjacencies and telomeres of genome A are vertices of this graph. Since the adjacency graph is a union of paths and cycles, all the tools and terminology of Section 2 can be used.

In Lemma 1, we showed that the number of circular and linear components can change by at most one when a DCJ operation is applied to a graph that is a union of paths and cycles. In the case of adjacency graphs we have also constraints on the possible changes in the number of odd paths:

Lemma 3. *The application of a single DCJ operation changes the number of odd paths in the adjacency graph by –2, 0, or 2.*

Algorithm 2 (Greedy sorting by DCJ)

```
for each adjacency {p, q} in genome B do
    let u be the element of genome A that contains p
    let v be the element of genome A that contains q
    if u ≠ v then
        replace u and v in A by {p, q} and (u \ {p}) ∪ (v \ {q})
    end if
end for
for each telomere {p} in genome B do
    let u be the element of genome A that contains p
    if u is an adjacency then
        replace u in A by {p} and (u \ {p})
    end if
end for
```

Proof. Consider operations that are path translocations, fusions or fissions (Figure 2). Two odd paths can be either transformed into two odd paths, or into one or two paths of even length. Path(s) of even length(s) can be either transformed into path(s) of even length, or into two paths of odd length. One even and one odd path are always transformed into one even and one odd path. Finally, splitting one odd path always yields an even and an odd path.

Inversions, excisions, integrations, circularizations and linearizations (Figure 3) do not change the number of odd paths since all cycles have even length. No paths are involved in the DCJ operations of Figure 4. □

Lemma 3 allows to derive the following lower bound for the DCJ distance:

Lemma 4. *Let A and B be two genomes defined on the same set of N genes, then we have*

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

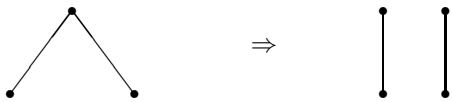
where C is the number of cycles and I the number of odd paths in $AG(A, B)$.

Proof. Since none of the cases of the DCJ operation modifies the number of cycles and odd paths simultaneously, this follows immediately from Lemmas 1, 2 and 3. □

The adjacency graph is also very useful when one wants to find an optimal sequence of sorting operations.

Observe that any pair of edges in the adjacency graph that connect two different vertices of genome A with an adjacency $\{p, q\}$ in genome B can be transformed by a single DCJ operation into a cycle of length two, plus the remaining structure, reduced by the two edges. This operation always increases $C + I/2$ by one since C is increased by one and we have already seen that no DCJ operation can simultaneously change C and I.

Now assume that all adjacencies of genome B are contained in cycles of length two. There might still be pairs of telomeres of B that form an adjacency in A. These adjacencies can be split into two telomeres, thus creating two odd paths of length one each, increasing I by two.



Pseudocode for this greedy sorting procedure is given in Algorithm 2. Note that the adjacency graph does not need to be constructed explicitly if the genomes are stored in the way sketched at the end of Section 4. Interestingly, the algorithm is optimal:

Theorem 1. *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 by Algorithm 2.

Proof. Lemma 4 together with the fact that Algorithm 2 increments in each iteration either C by one or I by two prove the distance formula.

The linear time complexity follows from the fact that our genome representation allows to find and perform each sorting operation in constant time and the DCJ distance is never larger than N. □

Remark 1. It is worth mentioning that our distance formula is equivalent to the result $d_{DCJ} = b - c$ given by Yancopulos *et al.* [3], where b is the number of breakpoints and c is the number of cycles of the breakpoint graph after appropriate *capping* of the linear chromosomes.

To see this, let l_A and l_B be the number of linear chromosomes in genomes A and B, respectively. Then the total number of breakpoints, as defined in [3], is $b = N + l_B + aa = N + l_A + bb$ where aa is the number of even paths that start and end in genome A and bb is the number of even paths that start and end in genome B. The number of cycles is $c = C + I + E$ where C is the number of cycles, I the number of odd paths and E the number of even paths in the adjacency graph $AG(A, B)$ as defined in this paper. Obviously $E = aa + bb$. Moreover, each linear chromosome is associated to two path ends, thus the number of linear chromosomes equals the number of paths, $l_A + l_B = I + E$. Together this implies that $2b = 2N + 2E + I$, giving $b - c = N - C - I/2$.

6 Conclusion

We have shown that, with a suitable representation, it is possible to model all rearrangement operations on the most general genome structure that mixes both circular and linear chromosomes.

The basic tools for this representation are graphs that are unions of paths and cycles. Surprisingly, this type of graph can be used for representing genomes, for computing the DCJ distance, and for suggesting rearrangement scenarios. This variety of uses suggests many interesting problems.

The first one is to investigate formal properties of graphs that are unions of paths and cycles, with respect to the DCJ operation. For example, the cyclic organization of these operations is a striking feature of Figures 2, 3 and 4 and offers new ways to classify rearrangement operations. These graphs also give a firm starting point to explore difficult rearrangement problems that involve either gene duplications [6] or missing information about the actual order of genes in a genome [7].

Last, but not the least, adding constraints on the type of allowed operations often yields equations of the form

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

where t represents the additional cost of not resorting to DCJ operations. For example, the Hannenhalli-Pevzner distance, that allows only translocations and inversions on linear chromosomes [8], can be recast as avoiding all DCJ operations that create a circular chromosome in either genome A or B. These operations live only on Figure 2 and the upper half of Figure 3.

Another kind of restriction has recently been studied in [9], where operations are fusions and fissions between circular unsigned chromosomes, and block interchanges within a circular unsigned chromosome. The authors assign equal weight to the three operations, even if a block interchange requires two DCJ operations, and propose an $O(N^2)$ time algorithm to sort these circular genomes. Their algorithm first applies fusions to both source and target genome, until they have two genomes whose chromosomes have equal gene content. These fusions can be identified in linear time by a search of the adjacency graph. They then sort the resulting genomes by block interchanges using an $O(N^2)$ time algorithm described in [10]. This can be done in the same time complexity, but with elementary means, using a modification of our Algorithm 2 where every intermediate chromosome created by a fission is immediately re-absorbed in the next step, such that only block interchanges are performed. The modification is to search, in the newly created circular chromosomes, a pair of genes that are adjacent in the target genome, but on different chromosomes in the source genome.

References

1. Sankoff, D., Mazowita, M.: Stability of rearrangement measures in the comparison of genome sequences. In Miyano, S., Mesirov, J., Kasif, S., Istrail, S., Pevzner, P., Waterman, M., eds.: Proceedings of RECOMB 2005. Volume 3500 of LNBI., Springer Verlag (2005) 603–614
2. Bergeron, A., Stoye, J.: On the similarity of sets of permutations and its applications to genome comparison. In Warnow, T., Zhu, B., eds.: Proceedings of COCOON 2003. Volume 2697 of LNCS., Springer Verlag (2003) 68–79

3. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics **21** (2005) 3340–3346
4. Casjens, S., Palmer, N., van Vugt, R., Huang, W.M., Stevenson, B., Rosa, P., Lathigra, R., Sutton, G., Peterson, J., Dodson, R.J., Haft, D., Hickey, E., Gwinn, M., White, O., Fraser, C.M.: A bacterial genome in flux: The twelve linear and nine circular extrachromosomal DNAs in an infectious isolate of the Lyme disease spirochete *Borrelia burgdorferi*. Mol. Microbiol. **35** (2000) 490–516
5. Volff, J.N., Altenbuchner, J.: A new beginning with new ends: Linearisation of circular chromosomes during bacterial evolution. FEMS Microbiol. Lett. **186** (2000) 143–150
6. Zheng, C., Lenert, A., Sankoff, D.: Reversal distance for partially ordered genomes. Bioinformatics **21** (2005) i502–i508 (Proceedings of ISMB 2005).
7. El-Mabrouk, N.: Genome rearrangements with gene families. In Gascuel, O., ed.: Mathematics of Evolution and Phylogeny. Oxford University Press (2005) 291–320
8. Hannenhalli, S., Pevzner, P.A.: Transforming men into mice (polynomial algorithm for genomic distance problem). In: Proceedings of FOCS 1995, IEEE Press (1995) 581–592
9. Lu, L., Huang, Y.L., Wang, T.C., Chiu, H.T.: Analysis of circular genome rearrangement by fusions, fissions and block-interchanges. BMC Bioinformatics **7** (2006)
10. Lin, Y.C., Lu, C.L., Chang, H.Y., Tang, C.Y.: An efficient algorithm for sorting by block-interchanges and its application to the evolution of vibrio species. J. Comp. Biol. **12** (2005) 102–112