Genome Aliquoting Revisited

Robert Warren and David Sankoff

University of Ottawa, Ottawa, ON, Canada

Abstract. We prove that the genome aliquoting problem, the problem of finding a recent polyploid ancestor of a genome, with breakpoint distance can be solved in polynomial time. We propose an aliquoting algorithm that is a 2-approximation for the genome aliquoting problem with double cut and join distance, improving upon the previous best solution to this problem, Feijão and Meidanis' 4-approximation algorithm.

Comparing two genomes with duplicated genes is difficult. None of the distances used to compare genomes today (breakpoint distance, reversal distance, double cut and join distance, etc...) handle duplicated genes. However, in the special case where all genes are duplicated the same number of times, there has been some success.

Informally, the *genome aliquoting problem* is the problem of finding a genome with one copy of every gene given a genome with exactly p copies of every gene such that the distance between the given and resulting genomes is minimized according to some distance metric. Thus, the genome aliquoting problem eliminates the duplicate genes allowing a genome to be compared with other genomes using an existing algorithm. Solving this problem will allow genomes which have undergone a recent polyploidization event, common in plants, to be compared.

There have been a number of solutions to the genome aliquoting problem where the genome has exactly two copies of every gene. This restricted version of the problem is called the *genome halving problem* and was first introduced in [3]. It was solved for reversal and translocation distance in [3,1] and for double cut and join distance in [10,6].

The *genome aliquoting problem* was introduced in [9] along with a sketch of a heuristic algorithm for the problem under double cut and join distance. [4] provided an exact solution under single cut or join distance which is also a 4-approximation algorithm under double cut and join distance. In this paper, we provide an exact polynomial-time algorithm under breakpoint distance which is also a 2-approximation algorithm under double cut and join distance. Since our algorithm is similar to that presented in [9] it also bounds that heuristic as a 2-approximation for double cut and join distance.

1 Duplicated Genomes

The fundamental elements that we study are *genes*. We represent each gene as a pair of *extremities* such that a gene x is represented by its *head* $\overrightarrow{x}$, which

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 1–12, 2010.

corresponds to the $3'$ end of the gene, and its *tail* $\overset{\succ}{x}$, which corresponds to the $5'$ end of the gene. A *genome* is represented by a multiset of 2-multisets and 1-sets of extremities, called *adjacencies* and *telomeres* respectively, such that, for each gene, the genome contains exactly the same number of heads as tails. The functions $\mathcal{G}(G)$ and $\mathcal{E}(G)$ return the set of all genes or extremities respectively of a genome G. The collection of copies of the same gene are called a *gene family* with the *size* being the number of copies in the genome (*i.e.* the number of heads, or, alternatively, the number of tails, that appear in a genome).

Genomes with one or more gene families with size greater than one are challenging to manipulate. Not only must there be the same number of heads and tails but to understand the layout of the genome we must know for duplicated genes which head corresponds to which tail, otherwise the layout of the genome is not unique. Similarly, to compute the distance of two genomes we must know which copy in one genome matches which copy in another genome. Genomes where this information is known are called *ordered genomes*. In such genomes we distinguish members of the same gene family by a subscript such that each head corresponds to the tail with the same subscript, *e.g.* $\overrightarrow{a_1}$ corresponds to $\overset{\succ}{a_1}$. Similarly, if we know the ordering between two genomes then each extremity in one genome must correspond to the extremity with the same subscript in the other genome, *e.g.* $\overrightarrow{a_1}$ in one genome must correspond to $\overrightarrow{a_1}$ in the other genome. By default we assume most genomes are ordered with themselves but the challenge of comparing genomes with duplicated genes comes from finding an ordering between two genomes as different orderings change the distance. Thus, to compare genomes with duplicated genes we must find an ordering between them that minimizes their distance.

For most situations, ordered genomes are all that is needed. However, ordered genomes can be difficult to use, *e.g.* $\{\overrightarrow{a_1}, \overrightarrow{b_1}\} \cap \{\overrightarrow{a_2}, \overrightarrow{b_2}\} = \emptyset$ and yet, frequently, it is desirable to perceive them as equal. Since these problems tend to occur frequently in our work we introduce the concept of an *unordered genome* where no effort is made to distinguish the genes. Let $\tilde{G}$ be the unordered counterpart of an ordered genome G. Similarly, if α is an element of G then $\tilde{\alpha}$ is an element of $\tilde{G}$ and if S is a subset of G then $\tilde{S}$ is a subset of $\tilde{G}$. Unordered extremities are distinguished from ordered extremities by the absence or presence respectively of a subscript. Transforming an ordered genome to an unordered genome is trivial but the reverse can be very difficult and is problem specific.

A concept that helps solve the genome aliquoting problem is :

Definition 1. *We define* perfection *between two elements of a genome to be that the two elements are the same or are disjoint. For adjacencies we also have the special requirement that they are not of the form $\{x, x\}$ for some extremity x. Since there are two types of elements, adjacencies and telomeres, two elements α and β can be in one of the following three configurations:*

- *let α and β be a pair of adjacencies. They are* perfect *if and only if they are disjoint or the same and neither are of form $\{x, x\}$ for some extremity x;*
- *let α and β be a pair of telomeres. Since all telomeres are either the same or disjoint all pairs of telomeres are* perfect*;*

– *let α be an adjacency and β be a telomere, since they cannot be the same they must be disjoint in order to be* perfect. *Also, α must not be of the form $\{x, x\}$ for some extremity x;*

A genome, or any set of adjacencies and telomeres, is perfect *if and only if all pairs of elements are perfect.*

For example, given a genome $G = \{\{\overrightarrow{a_1}\}, \{\overrightarrow{a_1}, \overrightarrow{b_1}\}, \{\overrightarrow{b_1}, \overrightarrow{d_1}\}, \{\overrightarrow{d_1}, \overrightarrow{e_1}\}, \{\overrightarrow{e_1}, \overrightarrow{b_2}\}, \{\overrightarrow{b_2}\}, \{\overrightarrow{a_2}\}, \{\overrightarrow{a_2}, \overrightarrow{c_1}\}, \{\overrightarrow{c_1}, \overrightarrow{c_2}\}, \{\overrightarrow{c_2}, \overrightarrow{d_2}\}, \{\overrightarrow{d_2}, \overrightarrow{e_2}\}, \{\overrightarrow{e_2}\}\}$ its corresponding unordered genome is $\tilde{G} = \{\{\overrightarrow{a}\}, \{\overrightarrow{a}, \overrightarrow{b}\}, \{\overrightarrow{b}, \overrightarrow{d}\}, \{\overrightarrow{d}, \overrightarrow{e}\}, \{\overrightarrow{e}, \overrightarrow{b}\}, \{\overrightarrow{b}\}, \{\overrightarrow{a}\}, \{\overrightarrow{a}, \overrightarrow{c}\}, \{\overrightarrow{c}, \overrightarrow{c}\}, \{\overrightarrow{c}, \overrightarrow{d}\}, \{\overrightarrow{d}, \overrightarrow{e}\}, \{\overrightarrow{e}\}\}$. In $\tilde{G}$, the adjacencies $\{\overrightarrow{a}, \overrightarrow{b}\}$ and $\{\overrightarrow{b}, \overrightarrow{d}\}$ are perfect because they are disjoint but $\{\overrightarrow{a}, \overrightarrow{b}\}$ and $\{\overrightarrow{a}, \overrightarrow{c}\}$ are not perfect because they are neither disjoint nor equal (they have $\overrightarrow{a}$ in common but do not share the other extremity). The adjacency $\{\overrightarrow{c}, \overrightarrow{c}\}$ can never be perfect because it is of the form $\{x, x\}$ for some extremity x. The telomere $\{\overrightarrow{a}\}$ is perfect when compared with both $\{\overrightarrow{a}\}$ and $\{\overrightarrow{e}\}$ since it is equal to the former and disjoint with the latter. $\{\overrightarrow{e}\}$ is not perfect when compared with $\{\overrightarrow{e}, \overrightarrow{b}\}$ because they are not disjoint.

In biology, a *polyploid* is a genome with multiple copies of the same chromosome. A perfect unordered genome where all gene families are of the same size shares this property and, hence, is called a *polyploid*. Similarly, an ordered genome G whose corresponding unordered genome $\tilde{G}$ is a polyploid is a *polyploid*.

Definition 2. *Given an ordered genome G with all gene families of size p, the* genome aliquoting problem *is to find a polyploid H with all gene families of size p ordered in relation to G such that the distance between G and H is minimal.*

2 Breakpoint Distance

A *breakpoint* (*BP*) is a difference in adjacencies or telomeres between two genomes, *e.g.* in $G = \{\{\overrightarrow{a_1}\}, \{\overrightarrow{a_1}, \overrightarrow{b_1}\}, \{\overrightarrow{b_1}, \overrightarrow{d_1}\}, \{\overrightarrow{d_1}, \overrightarrow{e_1}\}, \{\overrightarrow{e_1}, \overrightarrow{b_2}\}, \{\overrightarrow{b_2}\}, \{\overrightarrow{a_2}\}, \{\overrightarrow{a_2}, \overrightarrow{c_1}\}, \{\overrightarrow{c_1}, \overrightarrow{c_2}\}, \{\overrightarrow{c_2}, \overrightarrow{d_2}\}, \{\overrightarrow{d_2}, \overrightarrow{e_2}\}, \{\overrightarrow{e_2}\}\}$ and $H = \{\{\overrightarrow{a_1}\}, \{\overrightarrow{a_1}, \overrightarrow{b_1}\}, \{\overrightarrow{b_1}, \overrightarrow{c_1}\}, \{\overrightarrow{c_1}, \overrightarrow{d_1}\}, \{\overrightarrow{d_1}, \overrightarrow{e_1}\}, \{\overrightarrow{e_1}, \overrightarrow{a_2}\}, \{\overrightarrow{a_2}, \overrightarrow{b_2}\}, \{\overrightarrow{b_2}, \overrightarrow{c_2}\}, \{\overrightarrow{c_2}, \overrightarrow{d_2}\}, \{\overrightarrow{d_2}, \overrightarrow{e_2}\}, \{\overrightarrow{e_2}\}\}, \{\overrightarrow{b_1}, \overrightarrow{d_1}\}$ is a breakpoint in G since there is no equivalent element in H but $\{\overrightarrow{a_1}\}$ is not a breakpoint because it is in both genomes. Extending this notion to entire genomes we arrive at the following definition:

Definition 3. *The* breakpoint distance*, introduced in [7], between two genomes G and H is the number of breakpoints between G and H.*

The breakpoint distance is easy to calculate. From [8], given two genomes G and H, let $\mathcal{A}(G, H)$ be the set of adjacencies shared between G and H and let $\mathcal{T}(G, H)$ be the set of telomeres then the *breakpoint distance* between G and H can be calculated using the following equation:

$$\mathcal{D}_{BP}(G, H) = |\mathcal{G}(G)| - |\mathcal{A}(G, H)| - \frac{|\mathcal{T}(G, H)|}{2} \tag{1}$$

For example, given two duplicated genomes $G = \{\{\overrightarrow{a_1}\}, \{\overrightarrow{a_1}, \overrightarrow{b_1}\}, \underline{\{\overrightarrow{b_1}, \overrightarrow{d_1}\}}, \underline{\{\overrightarrow{d_1}, \overrightarrow{e_1}\}}, \underline{\{\overrightarrow{e_1}, \overrightarrow{b_2}\}}, \underline{\{\overrightarrow{b_2}\}}, \underline{\{\overrightarrow{a_2}\}}, \underline{\{\overrightarrow{a_2}, \overrightarrow{c_1}\}}, \underline{\{\overrightarrow{c_1}, \overrightarrow{c_2}\}}, \underline{\{\overrightarrow{c_2}, \overrightarrow{d_2}\}}, \underline{\{\overrightarrow{d_2}, \overrightarrow{e_2}\}}, \underline{\{\overrightarrow{e_2}\}}\}$ and $H = \{\{\overrightarrow{a_1}\}, \{\overrightarrow{a_1}, \overrightarrow{b_1}\}, \underline{\{\overrightarrow{b_1}, \overrightarrow{c_1}\}}, \underline{\{\overrightarrow{c_1}, \overrightarrow{d_1}\}}, \underline{\{\overrightarrow{d_1}, \overrightarrow{e_1}\}}, \underline{\{\overrightarrow{e_1}, \overrightarrow{a_2}\}}, \underline{\{\overrightarrow{a_2}, \overrightarrow{b_2}\}}, \underline{\{\overrightarrow{b_2}, \overrightarrow{c_2}\}}, \underline{\{\overrightarrow{c_2}, \overrightarrow{d_2}\}}, \underline{\{\overrightarrow{d_2}, \overrightarrow{e_2}\}}, \underline{\{\overrightarrow{e_2}\}}\}$ the breakpoint are underlined; since $\mathcal{A}(G, H) = 1$ and $\mathcal{T}(G, H) = 1$ and $\overline{\mathcal{G}(G)} = 8$, $\mathcal{D}_{BP}(G, H) = 6.5$.

3 Quasi Maximum Perfect and Covering Sets

Since polyploids are perfect genomes where every gene family has the same size, our strategy to aliquote a genome G is to find a perfect subset of G and then transform it into a perfect genome. Because manipulating G directly is challenging, we will manipulate its unordered counterpart $\tilde{G}$ instead.

There are many possible perfect subsets of $\tilde{G}$ and we need to choose the one which will minimize the distance. From Equation 1 we can see that the distance is minimized by maximizing the number of adjacencies and telomeres present in both genomes. Thus, we define the *weight* of a perfect set P in relation to a genome $\tilde{G}$ to be:

$$\mathcal{W}(P) = \mathcal{A}(\tilde{G}, P) + \frac{\mathcal{T}(\tilde{G}, P)}{2} \tag{2}$$

Define a *maximum perfect set* P as a perfect subset of $\tilde{G}$ with maximum weight.

While it is impossible to compute the distance using an unordered genome, it is easy to see that, once transformed into a genome and reordered, a maximum perfect subset of $\tilde{G}$ will minimize the distance. However, not all maximum perfect sets can be transformed into an ordered genome that can be compared with G. In addition to being ordered, to be compared two genomes must have the same set of extremities and it is possible to construct a maximum perfect set that is missing extremities from $\tilde{G}$. Thus, we introduce the notion of a *covering set*. A set C covers an unordered genome $\tilde{G}$ if and only if it contains at least one copy of every extremity in $\tilde{G}$.

Unfortunately, we cannot always find a maximum perfect and covering subset of $\tilde{G}$, for some unordered genomes no such subset exists. To solve the problem we remove the restriction that it must be a subset. However, without the restriction that the perfect set has to be a subset of $\tilde{G}$, the idea of a maximum perfect set does not make any sense; we could add an infinite number of adjacencies and telomeres to the set if they do not have to come from the genome. Thus, we introduce a *quasi maximum perfect set*: one with a maximum perfect set as a subset but with additional elements not from $\tilde{G}$. All maximum perfect sets are also quasi maximum perfect sets, although the reverse is not true.

It is easy to transform a quasi maximum perfect and covering set with respect to a genome $\tilde{G}$ into a polyploid that, if ordered, could be compared with G: we simply increase the number of copies of each element. Algorithm 1 does exactly this, although we omit the proof of its correctness due to space constraints.

Algorithm 1. REPLICATE

Input: P quasi maximum perfect and covering set with respect to a genome $\tilde{G}$ and p, the size of the gene families in $\tilde{G}$.
Output: A polyploid $\tilde{H}$ with all gene families of size p.
1 **foreach** $\alpha \in P$ **do**
2 **if** the multiplicity of α is not p **then**
3 add α to (or remove α from) $\tilde{H}$ until the multiplicity is p
4 **end**
5 **end**
6 **return** P

4 Reorder Genes

How a genome is reordered can affect the distance. However, with a quasi maximum perfect and covering set with respect to $\tilde{G}$ as a base, it is easy to reorder the resulting polyploid such that its distance to G is minimal. Algorithm 2 reorders the genome and the following theorem proves its correctness:

Theorem 1. *Given a genome G with all gene families of size p, let P be a quasi maximum perfect and covering set with respect to $\tilde{G}$ and let $\tilde{H}$=*REPLICATE*(P, p), then $\mathcal{D}_{BP}(G, H)$ where $H =$* REORDER*$(\tilde{H}, G)$ is minimal.*

Proof. $\mathcal{D}_{BP}(G, H)$ is minimal if there does not exist an other genome H' such that $\mathcal{D}_{BP}(G, H') < \mathcal{D}_{BP}(G, H)$. Assume towards contradiction that such an H' does exist.

Let E be the set of elements that are identical between H and G and let E' be the set of elements that are identical between H' and G. We observe that because H and H' are polyploids $\tilde{E}$ and $\tilde{E}'$ must be a perfect.

From Equation 1 only adjacencies and telomeres that are the same in both genomes reduce the distance. Since $\mathcal{D}_{BP}(G, H') < \mathcal{D}_{BP}(G, H)$, $\mathcal{W}(\tilde{E}') > \mathcal{W}(\tilde{E})$.

A quasi maximum perfect set must have a maximum perfect set as a subset; let $M \subseteq P$ be such a maximum perfect set of $\tilde{G}$. Since E is a subset of H it follows that $\tilde{E}$ must be a subset of $\tilde{H}$. Thus, M and $\tilde{E}$ are subsets of $\tilde{H}$ and, in fact, we will prove that $M = \tilde{E}$ by assuming towards contradiction that $M \neq \tilde{E}$.

$M \neq \tilde{E}$ if and only if the multiplicity of all elements in M is not equal to the multiplicity of all elements in $\tilde{E}$. Let α be an element that has different multiplicities in both M and $\tilde{E}$. Since M is a maximum perfect set of $\tilde{G}$ all of its elements can have a multiplicity of at most p and, because M is a maximum, the multiplicity of α in M is equal to the multiplicity of α in $\tilde{G}$. Since REPLICATE(P, p) creates $\tilde{H}$ by setting all the elements of P, which includes all of the elements of M, to multiplicity p, the multiplicity of α in $\tilde{H}$ is greater than or equal to that of M. It follows that the multiplicity of α in M is the multiplicity of α in $\tilde{G} \cap \tilde{H}$. From Lines Lines 2 – 6 in Algorithm 2 REORDER$(\tilde{H}, G)$ we can conclude that $\tilde{G} \cap \tilde{H} = \widetilde{(G \cap H)}$. By definition, $E = G \cap H$, hence, $\widetilde{(G \cap H)} = \tilde{E}$. Therefore, the multiplicities of α in M and in $\tilde{E}$ are equal; a contradiction. Thus, $M = \tilde{E}$.

Algorithm 2. REORDER

Input: A polyploid $\tilde{H}$ with all gene families of size p and a genome G with all gene families of size p where $\mathcal{E}(\tilde{G}) = \mathcal{E}(\tilde{H})$.
Output: A polyploid H with all gene families of size p such that $\mathcal{D}_{BP}(G, H)$ is minimal.

```
H ← ∅
foreach element α ∈ G do
    if α̃ ∈ H̃ then
        add α to H
    end
end
E ← E(G) \ E(H)
foreach element α̃ ∈ H̃ do
    if α̃ is an adjacency then
        let {x, y} = α̃ where x, y are extremities
        while there exists an i such that x_i ∈ E and a j such that y_j ∈ E do
            add {x_i, y_j} to H
            remove x_i from E
            remove y_j from E
        end
    else
        let {x} = α̃ where x is an extremity
        while there exists an i such that x_i ∈ E do
            add {x_i} to H
            remove x_i from E
        end
    end
end
return H
```

$\mathcal{W}(\tilde{E}') > \mathcal{W}(M)$ follows from the facts that $M = \tilde{E}$ and $\mathcal{W}(\tilde{E}') > \mathcal{W}(\tilde{E})$. But M is a maximum perfect set; a contradiction. Hence, H must be minimal. □

5 Fully Modified Clique Graphs

We have reduced the problem of aliquoting a genome to that of finding a maximum perfect and covering set. To find the maximum perfect and covering set, we construct a graph from the genome that highlights its important information.

Definition 4. *A* fully modified weighted clique graph *of a genome $\tilde{G}$, denoted $\mathcal{CG}(\tilde{G})$, is defined as follows:*

- *For each gene family x in $\tilde{G}$, $\mathcal{CG}(\tilde{G})$ has four vertices, one labeled $\vec{x}$, one labeled $\overleftarrow{x}$ and two without labels, called* null vertices*, of which one is connected to the vertex labelled with $\vec{x}$ and the other to the vertex labeled with $\overleftarrow{x}$;*

- *For each adjacency* $\{x, y\}$ *in* $\tilde{G}$ *where* $x \neq y$*, there is an edge between the vertices labeled* x *and* y *with weight equal to the multiplicity of* $\{x, y\}$ *in* $\tilde{G}$*;*
- *For each telomere* $\{x\}$ *in* $\tilde{G}$*, the edge between the vertex labeled* x *and its adjacent null vertex has weight equal to half of the multiplicity of* $\{x\}$ *in* $\tilde{G}$*;*
- *All other edges in* $\mathcal{CG}(\tilde{G})$ *have weights of* 0*.*

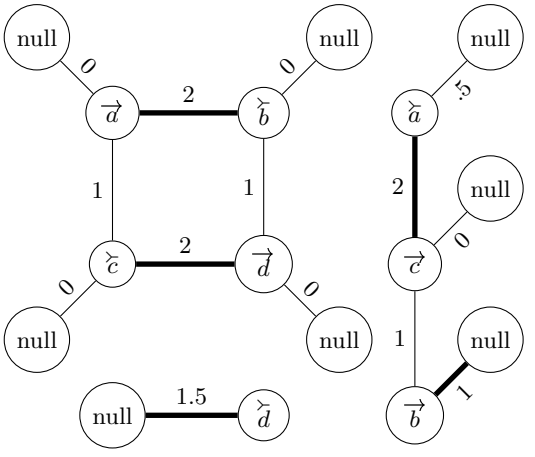

Fig. 1. An example of a fully modified weighted clique graph. The five bold edges represent all the edges that belong to the maximum weight matching of this graph.

For example, the modified weighted clique graph of the genome $\tilde{G} = \{\{\overleftarrow{a}\}, \{\overrightarrow{a}, \overleftarrow{b}\}, \{\overrightarrow{b}, \overrightarrow{c}\}, \{\overleftarrow{c}, \overrightarrow{a}\}, \{\overleftarrow{a}, \overrightarrow{c}\}, \{\overleftarrow{c}, \overrightarrow{d}\}, \{\overleftarrow{d}\}, \{\overleftarrow{d}\}, \{\overrightarrow{d}, \overleftarrow{c}\}, \{\overrightarrow{c}, \overleftarrow{a}\}, \{\overrightarrow{a}, \overleftarrow{b}\}, \{\overrightarrow{b}\}, \{\overrightarrow{b}\}, \{\overleftarrow{b}, \overrightarrow{d}\}, \{\overleftarrow{d}\}\}$ is depicted in Figure 1.

In the fully modified weighted clique graph, edges correspond to adjacencies or telomeres. Thus, the problem is to choose edges from the fully modified weighted clique graph such that the corresponding adjacencies and telomeres form a maximum perfect and covering set. From Definition 4 it is clear that the adjacencies or telomeres corresponding to any two edges in the fully modified weighted clique graph are perfect if and only if the edges share no vertex in common or correspond to the same edge (since there are no loops, there are no edges that correspond to adjacencies of the form $\{x, x\}$ for some extremity x). Finding a set of edges that share no vertices in common is the famous *maximum matching problem*, although, since our edges are weighted, in this case we are actually more interested in the *maximum weight matching problem* [5].

Given a matching, Algorithm 3 constructs its corresponding perfect set. However, it is the quality of the matching that determines the quality of the perfect set. The fully modified weighted clique graph was defined such that the weight of the maximum weight matching is equal to the weight of the perfect set defined by Algorithm 3. Thus, given a maximum weight matching, Algorithm 3 will construct a maximum perfect set:

Algorithm 3. PERFECTSET

Input: A genome $\tilde{G}$ and a matching M.
Output: A perfect set P.

```
P ← ∅
foreach edge e ∈ M do
    let e = {u, v} for vertices u and v
    if v is a null vertex then
        add the telomere that corresponds to u to P
    else if u is a null vertex then
        add the telomere that corresponds to v to P
    else
        add the adjacencies that corresponds to u and v to P
    end
end
return P
```

Lemma 1. *Given a genome $\tilde{G}$, let M be a matching of $\mathcal{CG}(\tilde{G})$ and let $P =$ PERFECTSET$(\tilde{G}, M)$. If M is a maximum weighted matching then P must be a quasi maximum perfect set.*

Proof. Assume towards contradiction that M is a maximum weight matching but P is not a quasi maximum perfect set then there must exist a perfect set P' such that $\mathcal{W}(P) < \mathcal{W}(P')$. Let M' be the matchings that correspond to P', since the weight of the perfect set and its corresponding matching are the same, $\mathcal{W}(M) < \mathcal{W}(M')$. But M is a maximum weight matching, a contradiction. □

While in many cases the maximum weight matching will correspond to a quasi maximum perfect set that also covers the genome, simply finding a maximum weight matching of the fully modified weighted clique graph does not guarantee that the perfect set will be covering. Fortunately, any matching can be easily modified by Algorithm 4 so that this is the case:

Algorithm 4. EXTENDEDMAXIMUMWEIGHTMATCHING

Input: A fully modified weighted clique graph $\mathcal{CG}(\tilde{G})$ of a genome $\tilde{G}$.
Output: A maximum weight matching M of $\mathcal{CG}(\tilde{G})$ where every non-null vertex in $\mathcal{CG}(\tilde{G})$ is incident with an edge in M.

```
M ← MaximumWeightMatching(CG(G̃))
foreach non-null vertex v in CG(G̃) do
    if v is not incident with an edge in M then
        let u be the null vertex adjacent to v
        add {u, v} to M
    end
end
return M
```

Lemma 2. *Given a genome* $\tilde{G}$, $M = \textsc{ExtendedMaximumWeightMatching}(\tilde{G})$ *is a maximum weight matching such that* $P = \textsc{PerfectSet}(\tilde{G}, M)$ *covers* $\tilde{G}$.

Proof. Since every non-null vertex corresponds to an extremity in a genome, the perfect set will only be covering if every non-null vertex is incident with an edge in the matching from which the perfect set was created.

Algorithm 4 first finds any maximum weight matching M'. If every non-null vertex is incident with an edge in M' then the algorithm does not modify the matching and the result is a maximum weight matching whose corresponding perfect set covers $\tilde{G}$.

Each non-null vertex has a null vertex to which only it is adjacent. For non-null vertices not incident in an edge in the maximum weight matching, Algorithm 4 adds the edge between this vertex and its null counterpart to the matching. Clearly, the resulting perfect set will cover $\tilde{G}$. Thus, we must ensure that the resulting set of edges is still a matching of maximum weight.

The resulting set of edges must be a matching since the non-null vertex is not incident with any edges in the matching and the null vertex is only adjacent with the non-null vertex. It must be a maximum weight matching since all weights in a fully modified weighted clique graph are non-negative.

Therefore, $M = \textsc{ExtendedMaximumWeightMatching}(\tilde{G})$ is a maximum weight matching such that $P = \textsc{PerfectSet}(\tilde{G}, M)$ covers $\tilde{G}$. □

From Lemma 1 and Lemma 2 we can conclude the following theorem:

Theorem 2. *Let* $M = \textsc{ExtendedMaximumWeightMatching}(\tilde{G})$ *and* $P = \textsc{PerfectSet}(\tilde{G}, M)$, *where* $\tilde{G}$ *is an unordered genome.* P *is a quasi maximum perfect and covering set of* $\tilde{G}$.

6 Implementation

Algorithm 5 is the complete breakpoint aliquoting algorithm bringing together all the algorithms discussed in the previous sections. It follows from Theorem 2 and Theorem 1 that Algorithm 5 produces the optimal breakpoint aliquoting.

Most of the algorithms discussed in this paper run in linear time, however, the best known algorithm for computing the maximum weight matching runs in

Algorithm 5. BreakpointAliquoting

Input: A genome G with all gene families of size p.
Output: A polyploid H with all gene families of size p.

$M \leftarrow \textsc{ExtendedMaximumWeightMatching}(\mathcal{CG}(\tilde{G}))$
$P \leftarrow \textsc{PerfectSet}(\tilde{G}, M)$
$\tilde{H} \leftarrow \textsc{Replicate}(P, p)$
$H \leftarrow \textsc{Reorder}(\tilde{H}, G)$
return H

$O(ev \log v)$ time[5] where e is the number of edges and v is the number of vertices. If we have a genome with n gene families of size p, the fully modified weighted clique graph will have $4n$ vertices and $(2p-2)n$ edges. Thus, our algorithm runs in $O(pn^2 \log n)$ time.

7 Double Cut and Join Distance

Double cut and join distance and breakpoint distance are closely related. BP distance counts the number of different elements between two genomes. For the most part this is the same as DCJ distance; it generally takes one DCJ operation to correct one difference between two genomes. However, sometimes a DCJ operation corrects two differences instead of one and the DCJ distance must take this into account; this is the only difference between the two distances and it occurs infrequently with most data sets encountered in practice.

For a detailed explanation of DCJ distance we refer the reader to [2]. However, to understand this section we must briefly explain how DCJ distance is computed. An *adjacency graph* is a bipartite graph with two sets of vertices labeled with the elements of the two genomes that are to be compared respectively. There is an edge connecting two vertices for each extremity in common between their corresponding adjacencies. Recall that to be compared using a distance algorithm like DCJ, both genomes must have the same genes, hence the same extremities, and, in the case of genomes with duplicated genes, must be ordered.

From the adjacency graph, the DCJ distance can be computed:

$$\mathcal{D}_{DCJ}(G,H) = |\mathcal{G}(G)| - c - \frac{i}{2} \tag{3}$$

where c is the number of cycles and i is the number of paths with an odd number of edges in the adjacency graph.

The following theorem gives the relation between the DCJ and BP distances:

Theorem 3. *Given two genomes G and H:*

$$\mathcal{D}_{DCJ}(G,H) \leq \mathcal{D}_{BP}(G,H) \leq 2 \cdot \mathcal{D}_{DCJ}(G,H) \tag{4}$$

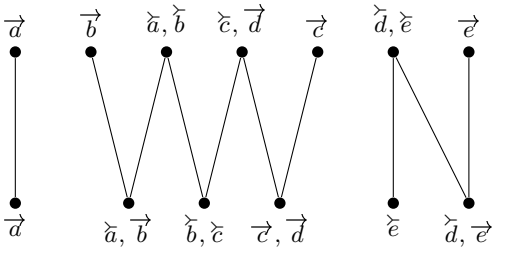

Fig. 2. An example of an adjacency graph

Proof. First, for $\mathcal{D}_{DCJ}(G,H) \leq \mathcal{D}_{BP}(G,H)$, observe that every adjacency shared between the two genomes causes a cycle of size 2 in the adjacency graph. It follows that $|\mathcal{A}(G,H)|$ is equal to the number of cycles of size 2 in the adjacency graph and, hence, less than or equal to the number of cycles in the adjacency graph. By similar reasoning, $|\mathcal{T}(G,H)|$ is equal to the number of paths of size 1 in the adjacency graph and, hence, less than or equal to the number of paths in the adjacency graph. It follows that $|\mathcal{A}(G,H)| + \frac{|\mathcal{T}(G,H)|}{2} \leq c + \frac{i}{2}$ where c is the number of cycles and i the number of odd paths in the adjacency graph. Since this factor decreases the distance, it follows that $\mathcal{D}_{DCJ}(G,H) \leq \mathcal{D}_{BP}(G,H)$.

Second, for $\mathcal{D}_{BP}(G,H) \leq 2 \cdot \mathcal{D}_{DCJ}(G,H)$, we will assume the most extreme case, where $|\mathcal{A}(G,H)| + \frac{|\mathcal{T}(G,H)|}{2} = 0$ but the number of cycles and paths in the adjacency graph is otherwise maximal. This produces the worst possible BP distance, $|\mathcal{G}(G,H)|$, but it is otherwise a maximal DCJ distance.

To determine the maximum number of cycles and paths in the adjacency graph, observe that every edge in the adjacency graph corresponds to one extremity shared between the genomes. Since $|\mathcal{A}(G,H)| + \frac{|\mathcal{T}(G,H)|}{2} = 0$ it follows that there are no cycles of size 2 and odd paths of size 1, so the smallest cycles and odd paths are of size 4 (since all cycles are even) and 3 respectively. Thus, the maximum number of cycles and odd paths is $\frac{|\mathcal{G}(G,H)|}{2}$. Thus, in the worst case scenario, BP distance is equal to twice the DCJ distance. □

Because BP distance can be equal to the DCJ distance, it is possible that the genome that results from Algorithm 5 could represent optimal aliquoting for both BP and DCJ. Because $\mathcal{D}_{BP}(G,H) \leq 2 \cdot \mathcal{D}_{DCJ}(G,H)$,the distance between the original genome and its optimal BP aliquoting is no more than twice the distance between the original genome and its optimal DCJ distance. Thus,

Theorem 4. *Algorithm 5 is a 2-approximation for the genome aliquoting problem using DCJ distance.*

8 Conclusion

Since the input to our algorithm is of size pn where p is the size of the gene families and n is the number of gene families, our algorithm runs in sub-cubic time. Thus, there exists a polynomial time algorithm that solves the genome aliquoting problem for BP distance. However, our algorithm is not without limitations.

The output of our algorithm consists of a mixture of linear and circular chromosomes with the only restriction being that adjacencies of the form $\{x, x\}$ for some extremity x are not permitted. In many cases it is desirable to restrict the output to something more similar to the input. For example, if the input consists of linear chromosomes then the output should consist of linear chromosomes. We believe it is possible to modify the output of our algorithm to provide the desired output in most cases without changing the distance. Some of the genome halving algorithms do precisely this [3,1]. On the other hand, sometimes circular chromosomes with adjacencies of the form $\{x, x\}$ are desirable in the output,

in which case it might be possible, in some cases, to get a better result than produced by our algorithm.

It remains an open problem whether or not a polynomial time algorithm for the genome aliquoting problem with DCJ distance exists. However, as we have shown a good approximation algorithm does exist. We remain optimistic that a polynomial time algorithm for DCJ distance can be found in the future.

References

1. Alekseyev, M.A., Pevzner, P.A.: Whole genome duplications and contracted breakpoint graphs. SIAM Journal on Computing 36(6), 1748–1763 (2007)
2. Bergeron, A., Mixtacki, J., Stoye, J.: A unifying view of genome rearrangements. In: Bücher, P., Moret, B.M.E. (eds.) WABI 2006. LNCS (LNBI), vol. 4175, pp. 163–173. Springer, Heidelberg (2006)
3. El-Mabrouk, N., Sankoff, D.: The reconstruction of doubled genomes. SIAM Journal on Computing 32, 754–792 (2003)
4. Feijão, P., Meidanis, J.: SCJ: a novel rearrangement operation for which sorting, genome median and genome halving problems are easy. In: Salzberg, S.L., Warnow, T. (eds.) Algorithms in Bioinformatics. LNCS, vol. 5724, pp. 85–96. Springer, Heidelberg (2009)
5. Lovász, L., Plummer, M.D.: Matching Theory. AMS Chelsea Publishing, Providence (2009)
6. Mixtacki, J.: Genome halving under DCJ revisited. In: Hu, X., Wang, J. (eds.) COCOON 2008. LNCS, vol. 5092, pp. 276–286. Springer, Heidelberg (2008)
7. Sankoff, D., Blanchette, M.: The median problem for breakpoints in comparative genomics. In: Jiang, T., Lee, D.T. (eds.) COCOON 1997. LNCS, vol. 1276, pp. 251–264. Springer, Heidelberg (1997)
8. Tannier, E., Zheng, C., Sankoff, D.: Multichromosomal median and halving problems under different genomic distances. Bioinformatics 10, 120 (2009)
9. Warren, R., Sankoff, D.: Genome aliquoting with double cut and join. BMC Bioinformatics 10(1), S2 (2009)
10. Warren, R., Sankoff, D.: Genome halving with double cut and join. Journal of Bioinformatics and Computational Biology 7(2), 357–371 (2009)