BIOINFORMATICS

Vol. 1 no. 1 2002
Pages 1–12

Breakpoint Medians and Breakpoint Phylogenies: A Fixed-Parameter Approach

Jens Gramm and Rolf Niedermeier

Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, D-72076 Tübingen, Fed. Rep. of Germany

ABSTRACT

With breakpoint distance, the genome rearrangement field delivered one of the currently most popular measures in phylogenetic studies for related species. Here, BREAKPOINT MEDIAN, which is NP-complete already for three given species (whose genomes are represented as signed orderings), is the core basic problem. For the important special case of three species, approximation (ratio $7/6$) and exact heuristic algorithms were developed. Here, we provide an exact, fixed-parameter algorithm with provable performance bounds. For instance, a breakpoint median for three signed orderings over n elements that causes at most d breakpoints can be computed in time $O((2.15)^d \cdot n)$. We show the algorithm's practical usefulness through experimental studies. In particular, we demonstrate that a simple implementation of our algorithm combined with a new tree construction heuristic allows for a new approach to breakpoint phylogeny, yielding evolutionary trees that are competitive in comparison with known results developed in a recent series of papers that use clever algorithm engineering methods.

INTRODUCTION

Many computational biology problems are NP-hard, causing research on approximation results for them. An alternative, increasingly successful line of research tries to solve NP-hard biological problems *exactly*, often using heuristic approaches based on branch-and-bound or enumerative methods, see, e.g., (Moret *et al.*, 2001b; Sankoff and Blanchette, 1998). Clearly, this takes into account exponential time worst-case complexity. That is why it is important to try to identify (hopefully) "small problem parameters" (such as number of species considered, the Hamming distance between two related sequences, the number of errors allowed, etc.) and to restrict the obviously inherent combinatorial explosion of the underlying problem to these parameters.[†] Successful examples for these so-called *fixed-parameter* algorithms (which are exponential only with respect to the parameter) have recently been obtained in numerous settings such as motif search (Blanchette *et al.*, 2002), gene duplication problems (Hallett and Lagergren, 2000), syntenic distance computation (Liben-Nowell, 2001), and others.

In this work, we present a fresh, fixed-parameter approach to the BREAKPOINT MEDIAN problem (Sankoff and Blanchette, 1998) together with a well-known application in the recently intensively studied field of breakpoint phylogenies (Cosner *et al.*, 2000a,b; Moret *et al.*, 2002a,b, 2001a,b). BREAKPOINT MEDIAN is defined as follows:

Input: Signed orderings $\pi_1, \pi_2, \ldots, \pi_k$ on n elements and a positive integer d.

Question: Is there a signed ordering π such that $\sum_{i=1}^{k} d_{bp}(\pi_i, \pi) \leq d$?

Herein, $d_{bp}(\pi_i, \pi)$ denotes the *breakpoint distance* between orderings π_i and π, see Preliminaries for definitions. From a broader perspective, BREAKPOINT MEDIAN fits into the larger field of consensus analysis problems, which occur in many computational biology settings. Subject to different types of input strings (here we have orderings) and different kinds of distance measures (here we have breakpoint distance), various complexity results and algorithms were published in this context during the last years, e.g., (Caprara, 2001; Fellows *et al.*, 2002; Higuera and Casacuberta, 2000; Li *et al.*, 2002; Siepel and Moret, 2001). Not surprisingly, in general BREAKPOINT MEDIAN is NP-complete, and remains so even in the case of only three input orderings (Bryant, 1998; Pe'er and Shamir, 1998). In the case of three input orderings, Pe'er and Shamir (2000) developed a polynomial-time algorithm with approximation ratio $7/6$. Keeping an eye on its application in the phylogenetic context, however, note that Moret *et al.* (2001b) emphasize that "because suboptimal solutions can yield very different evolutionary reconstructions, exact solutions are strongly preferred over approximate solutions." Hence, exact algorithms are of concern.

For the case of three input orderings, Sankoff and Blanchette (1998) presented a mathematically unanalyzed greedy heuristic for BREAKPOINT MEDIAN using branch-and-bound. Using a simple observation due to Bryant (1998), one can easily deduce that w.l.o.g. it can

[†] This is known as the parameterized complexity approach, see (Alber *et al.*, 2001; Downey and Fellows, 1999; Fellows, 2001) for details.

be assumed that $d \geq n$—otherwise, there is a simple polynomial-time preprocessing dealing with the case $d < n$. This implies that Sankoff and Blanchette's algorithm, which is based on a search tree method, already delivers "fixed-parameter tractability with respect to parameter d" for the problem. By way of contrast, we also employ a search tree method which deviates from Sankoff and Blanchette's approach in that it employs a significantly different branching strategy. Thus, we can present an algorithm solving BREAKPOINT MEDIAN in time $O((2.15)^d \cdot kn)$, which is practical (as demonstrated by our experiments) when d is not too large, a reasonable assumption in applications. Notably, with increasing k, the base of our exponential base becomes smaller and smaller. Observe that, because BREAKPOINT MEDIAN is already NP-complete for $k = 3$, in a way the parameterization with d is "enforced"—the problem is *fixed-parameter intractable* with respect to parameter k unless P = NP. Besides experimental investigations for our BREAKPOINT MEDIAN algorithm itself, we also use it to propose a new approach to breakpoint phylogeny, applying it to chloroplast gene order data in Campanulaceae (Cosner *et al.*, 2000a). This complements celebrated "fixed-parameter heuristics" (Cosner *et al.*, 2000a,b; Fellows, 2001) for computing breakpoint phylogenies. For our new approach, it is important that we can find optimal breakpoint medians also for $k > 3$. Roughly speaking, our approach allows us to avoid to search through the huge space of candidate phylogenetic trees (implicitly done in all other methods), but we, in a sense, can create a tree by a natural, iterative heuristic. Its validity is substantiated by experiments on real data, where we achieve results that are competitive in comparison with the previous ones developed in a series of papers using clever algorithm engineering (Sankoff and Blanchette, 1998; Cosner *et al.*, 2000a,b; Moret *et al.*, 2002a, 2001b). Our results are promising and make us confident that our approach to solving BREAKPOINT MEDIAN might be useful in future applications.

PRELIMINARIES

We start with introducing orderings as they are used to model genome rearrangements. Given a set $G = \{1, \ldots, n\}$, an *ordering* π on G is a $1:1$ function $\pi : G \to G$. We require that every ordering is extended by two special elements s, marking the start, and t, marking the end, and write ordering π as $\langle\, s\ \pi(1)\ \pi(2)\ \ldots\ \pi(n)\ t\,\rangle$. We write G_s for $G \cup \{s\}$ (G_t and $G_{s,t}$, analogously). An ordering π is *signed* iff every $\pi(x)$, $x \in G$, is equipped with a sign $\{+, -\}$, denoting the "orientation" of the element, such that $\pi(x)$ can be, for $y \in G$, a "positive" element $+y$ (or, for sake of brevity, only y), having left-to-right orientation, or a "negative" element $-y$, having right-to-left orientation. Note that a signed ordering contains either y or $-y$, but not both at the same time. The special elements s and t are always unsigned. We write $G^{\pm}$ for the set $\{-1, 1, -2, 2, \ldots, -n, n\}$ and $G_s^{\pm}$ for $G^{\pm} \cup \{s\}$ ($G_t^{\pm}$ and $G_{s,t}^{\pm}$ analogously).

Example. The signed ordering

$$\pi = <\ s\ +1\ -3\ -2\ +4\ t\ >$$

is the ordering where $\pi(1) = 1$, $\pi(2) = -3$, $\pi(3) = -2$, and $\pi(4) = 4$. □

We use $succ_\pi(x)$, for a signed ordering π and $x \in G_s$, to denote the *successor* $y \in G_{s,t}^{\pm}$ of element x in π, which is defined w.r.t. x's direction: For an element $x \in G$ occurring positively in π, the successor is the element following x. An $x \in G$ occurring negatively, however, has "reverse" orientation; hence, from x's point of view, its successor is the "reverse version" of the element preceding x. For instance, in ordering π as given above, the successor of element 1 is -3. Element 2, however, occurs negatively in π and the element following 2 w.r.t. this orientation is 3. Formally, for $x \in G^{\pm}$, we define $succ_\pi(x) := y$ if we can find $l \in G$ such that one of the following two conditions applies:

1. $\pi(l) = x$ and $\pi(l+1) = y$, or
2. $\pi(l) = -x$ and $\pi(l-1) = -y$.

Note that this definition also includes $x < 0$. For the special cases that $x = s$ or that $y \in \{s, t\}$, we define $succ_\pi(s) := y$ if $\pi(1) = y$; for $x \in G^{\pm}$, $succ_\pi(x) := t$ if $\pi(n) = x$, and $succ_\pi(x) := s$ if $\pi(1) = -x$. The predecessor $pred_\pi(y)$ for $y \in G_t^{\pm}$ is defined analogously. For $x \in G^{\pm}$, this definition satisfies $succ_\pi(x) = -pred_\pi(-x)$.

Example (continued). For π as defined before, we give the tables of successors and predecessors; we only list positive elements since $succ_\pi(x) = -pred_\pi(-x)$. In the last column, we indicate which of the two possibilities in the above definition of $succ_\pi$ applies (since $pred_\pi(x) = -succ_\pi(-x)$, we can also give the corresponding cases for the predecessor table).

element x	$succ_\pi(x)$	
s	1	–
1	-3	(1)
2	3	(2)
3	-1	(2)
4	t	–

element x	$pred_\pi(x)$	
1	s	–
2	-4	(1)
3	2	(1)
4	-2	(2)
t	4	–

For instance, $succ(2) = 3$ since (case (2) applies) we find $l = 3$ with $\pi(l) = -2$ and $\pi(l-1) = -3$; $pred(2)$ is -4 since $pred(2) = -succ(-2)$, and (for $succ(-2)$ case (1) applies) we find $l = 3$ with $\pi(l) = -2$ and $\pi(l+1) = 4$. □

The set $succ^*_\pi(x)$ of "reachable" elements for a signed ordering π and $x \in G^\pm$ contains $y \in G^\pm$ iff there are $x_1, \ldots, x_r \in G^\pm$ for which the following conditions apply:

1. $x = x_1$,
2. for $l = 1, \ldots, r-1$: $succ_\pi(x_l) = x_{l+1}$, and
3. $y = x_r$.

Analogously, we define $pred^*_\pi(x)$.

Example (continued). Considering the ordering given above, we obtain:

x	$succ^*_\pi(x)$
s	$\{1, -3, -2, 4, t\}$
1	$\{-3, -2, 4, t\}$
2	$\{3, -1, s\}$
3	$\{-1, s\}$
4	$\{t\}$

x	$pred^*_\pi(x)$
1	$\{s\}$
2	$\{-4, t\}$
3	$\{2, -4, t\}$
4	$\{-2, -3, 1, s\}$
t	$\{4, -2, -3, 1, s\}$

□

Given two signed orderings π_1 and π_2, both over G, we call a pair (x, y), $x \in G^\pm_s$ and $y \in G^\pm_t$, a *breakpoint* of π_1 w.r.t. π_2, if

1. $x = s$ or $\pi_1(l) = x$ for some $l \in G$, and
2. $succ_{\pi_1}(x) = y$ and $succ_{\pi_2}(x) \neq y$.

Using the notion of breakpoints, we define the *breakpoint distance* d_{bp} between two signed orderings as follows:

$$d_{bp}(\pi_1, \pi_2) = |\{\, (x, y) \mid x, y \in G^\pm_{s,t}, \\ x, y \text{ is breakpoint of } \pi_1 \text{ w.r.t. } \pi_2 \,\}|$$

Due to symmetry, $d_{bp}(\pi_1, \pi_2) = d_{bp}(\pi_2, \pi_1)$.

Example. Given orderings

$$\pi_1 = < s\ +1\ -3\ -2\ +4\ t >$$

and

$$\pi_2 = < s\ +1\ +2\ +3\ +4\ t >,$$

the breakpoint distance is $d_{bp}(\pi_1, \pi_2) = 2$: One breakpoint is $(1, -3)$, since $\pi_1(1) = 1$, $succ_{\pi_1}(1) = -3$, and $succ_{\pi_2}(1) = 2 \neq -3$; the other breakpoint is $(-2, 4)$, since $\pi_1(3) = -2$, $succ_{\pi_1}(-2) = 4$, and $succ_{\pi_2}(-2) = -1 \neq 4$. □

Now, we introduce the central problem of this paper. As input, it takes signed orderings $\pi_1, \pi_2, \ldots, \pi_k$. The input orderings do not necessarily contain the additional symbols s and t which are necessary for a correct definition of breakpoint distance; these symbols are added by the algorithm in the beginning.

BREAKPOINT MEDIAN

Input: Signed orderings $\pi_1, \pi_2, \ldots, \pi_k$ on n elements and a positive integer d.

Question: Is there a signed ordering π such that $\sum_{i=1}^k d_{bp}(\pi_i, \pi) \leq d$?

Preprocessing the input instance. The following intuitive lemma from (Bryant, 1998), gives us a way to simplify a given input instance by preprocessing:

Lemma 1. *Given signed orderings $\pi_1, \pi_2, \ldots, \pi_k$, all on a set G of n elements, and elements $x, y \in G^\pm_{s,t}$, which are adjacent in $\pi_1, \pi_2, \ldots, \pi_k$, i.e., $succ_{\pi_r}(x) = y$ for all $r = 1, \ldots, k$. Then x and y are also adjacent in an optimal breakpoint median π, i.e., $succ_\pi(x) = y$.* □

Using Lemma 1, we can preprocess the instance by "contracting" elements adjacent in all input sequences. From a parameterized complexity point of view, this preprocessing can be interpreted as a kind of so-called "reduction to problem kernel," where the original instance consisting of k orderings of n elements each is reduced to a new instance consisting of k orderings of at most d elements each (still, of course, all orderings have the same number of elements). Therefore, we can assume that in the given set $\Pi = \{\pi_1, \pi_2, \ldots, \pi_k\}$, for every element x, there are at least two orderings in which x has different successors and there are at least two orderings in which x has different predecessors.

Surprisingly, an optimal breakpoint median can have adjacencies that are not present in any of the input orderings (Bryant, 1998).

Lemma 2. *Given signed orderings $\pi_1, \pi_2, \ldots, \pi_k$, all on a set G of n elements, and an optimal breakpoint median π. Then there can be elements $x, y \in G^\pm_{s,t}$ with $succ_\pi(x) = y$ and $succ_{\pi_r}(x) \neq y$ for all $r = 1, \ldots, k$.* □

Proof. We consider the following example taken from Bryant (1998).

Given three orderings

$$\pi_1 = < s\ +5\ +6\ +7\ +4\ +1\ +2\ +3\ t >,$$
$$\pi_2 = < s\ +1\ +5\ +6\ +4\ +2\ +3\ +7\ t >,$$
$$\pi_3 = < s\ +1\ +2\ +5\ +4\ +3\ +6\ +7\ t >,$$

then

$$\pi = < s\ +1\ +2\ +3\ +4\ +5\ +6\ +7\ t >$$

is an optimal breakpoint median, although neither the adjacency $succ_\pi(3) = 4$ nor the adjacency $succ_\pi(4) = 5$ is present in any of the three given orderings. The reason is that π realizes all adjacencies that are present in at least two input orderings. This outweighs the disadvantage of setting $succ_\pi(3) = 4$ and $succ_\pi(4) = 5$ since 4 has different successors in π_1, π_2, and π_3 and the same is true for the predecessors of 4. □

Lemma 2 implies that, when searching an optimal breakpoint median, it is not sufficient to only consider adjacencies present in the input orderings.

A FIXED-PARAMETER ALGORITHM FOR BREAKPOINT MEDIAN

In this section, we present a fixed-parameter algorithm that solves BREAKPOINT MEDIAN in time $O((2.15)^d \cdot kn)$.

Notation

Our algorithm starts its search for a median ordering π with the set of unconnected elements G, i.e., no element is assigned a successor or a predecessor. For an element $x \in G$ for which we have not yet chosen a successor (or predecessor), we write $succ_\pi(x) = \emptyset$ (or $pred_\pi(x) = \emptyset$). We introduce a *link* between elements x and y, $x \in G_s^\pm$, $y \in G_t^\pm$, with $succ_\pi(x) = \emptyset$ and $pred_\pi(y) = \emptyset$, when we set $succ_\pi(x) := y$. The algorithm searches a median by introducing link by link into π. As long as there are elements $x \in G_{s,t}^\pm$ with $succ_\pi(x) = \emptyset$ or $pred_\pi(x) = \emptyset$, we call the ordering π *partial*. Otherwise, we call it *complete*. A predicate complete(π) is used to test whether π is complete. A (partial) ordering obtained from a partial ordering π by setting $succ_\pi(x) := y$ will be referred to by $\pi[succ(x) = y]$ which also implies that $pred_\pi(y) = x$. Analogously, we use $\pi[pred(x) = y]$.

In its search for the median, the algorithm prefers links that are also present in the input orderings. However, due to Lemma 2, we also have to consider links that are not present in the input orderings. In this case, the algorithm may defer the determination of a successor (or predecessor); this will be indicated by setting $succ_\pi(x) := \perp$ (or $pred_\pi(x) := \perp$). If $succ_\pi(x) = \perp$ or $pred_\pi(x) = \perp$, then x is referred to as an *isolated* element. Note that we call the ordering complete if we have chosen successor and predecessor values—including $\perp$—for all elements.

For sake of brevity, we introduce the predicate invalid(π) that tests whether a given π contradicts the notion of a partial signed ordering as described here: invalid(π) is true if one of the following three conditions holds:

- there is some $x \in G$ with $x \in succ_\pi^*(x)$,
- $t \in succ_\pi^*(s)$, but $|succ_\pi^*(s)| \leq |G| + 1$, or
- there are $x \in G_s^\pm$ and $i \in \{1, \ldots, k\}$ with $succ_\pi(x) = \perp$, $succ_{\pi_i}(x) = y$, and $pred_\pi(y) = \perp$.

Given a set of signed orderings Π, all on set G, and $x \in G_{s,t}^\pm$, we define $succ(\Pi, x)$ as the set of elements y for which $succ_\pi(x) = y$ for $\pi \in \Pi$. Analogously, we define $pred(\Pi, x)$. Further, we write $\#(\Pi, succ(x) = y)$ to denote the number of orderings $\pi \in \Pi$ in which $succ_\pi(x) = y$; $\#(\Pi, pred(x) = y)$ is defined analogously.

The Algorithm

The algorithm takes as input signed orderings $\pi_1, \ldots, \pi_k$ and a positive integer d, and reports, if existent, a median π for which $\sum_{i=1}^k d_{bp}(\pi_i, \pi) \leq d$. In the following, firstly, we specify how to compute a successor and predecessor table for such a π (which can contain $\perp$ entries); secondly, we describe how to obtain π from this table. The recursive algorithm builds a search tree to construct π from initially unconnected elements; in one node of the search tree, it selects an element $x \in G_{s,t}^\pm$ with $succ_\pi(x) = \emptyset$ (or $pred_\pi(x) = \emptyset$). It decides on a set of possible successor (or predecessor) values and recursively considers these values by branching into one subcase for each successor (or predecessor) value in the set. In this search, we keep track of the number of induced breakpoints: The algorithm is started with a parameter d denoting the allowed number of breakpoints. It maintains a counter Δd denoting the difference between d and the number of breakpoints that are already induced by the (partial) ordering π as it is constructed up to this point. When branching into one subcase while introducing a new link, Δd is decreased by the number of breakpoints caused by this new link. The recursion stops as soon as we introduced more breakpoints than were allowed, i.e., if $\Delta d < 0$. A solution is found when we complete the ordering with a non-negative Δd parameter.

As possible successors for an element x, we consider the successors of x in the input orderings. Due to Lemma 2, we also have to allow successors which do not occur in any of the input orderings; such a link causes k breakpoints. We handle this in an additional "isolation" subcase by deferring the choice of a successor and setting $succ_\pi(x) := \perp$ in π's current successor table, thus marking x as "isolated." In the end, when π is not partial any more, we will simply link arbitrary elements, which are both isolated, in this way (we will later show that we can, in linear time, determine such links without making π an invalid ordering). Since both elements are marked as isolated, we can assume that we introduce a link that causes k breakpoints and this introduced link corresponds to two isolation subcases. Therefore, to take these k breakpoints into account, we will decrease Δd by $k/2$ in each isolation subcase.

The following algorithm is initially called with BM(π, d) for the trivial partial ordering π containing no successor or predecessor entry for any element and the breakpoint parameter d from the BREAKPOINT MEDIAN formulation. If a value for d is not known, it could be determined, e.g., using binary search.

Algorithm BM$(\pi, \Delta d)$
Global input variables: Set Π of k signed orderings $\pi_1, \ldots, \pi_k$ over $G_{s,t}$ and a positive integer d.
Local input variables: A partial ordering π and an integer Δd denoting the breakpoint counter.
Output: A complete ordering, if existent, that can be obtained by completing π in a way such that at most Δd

new breakpoints are introduced.

(Case 1) *Recursion ends.*
If $(\Delta d < 0)$, **then** return;
/* π causes more breakpoints than allowed. */
If invalid(π), **then** return;
/* π is not an valid ordering */
If complete(π), **then** report π; /* solution found */

(Case 2) *Recursion continues.*
Choose $x \in \{1, \ldots, n\}$
with $succ_\pi(x) = \emptyset$ or $pred_\pi(x) = \emptyset$;
If $(succ_\pi(x) = \emptyset)$, **then**
/* Try to link x with its successors */
/* in the input orderings: */
For every $y \in succ(\Pi, x)$ **do**
If $(pred_\pi(y) = \emptyset)$, **then**
newd $:= \Delta d - (k - \#(\Pi, succ(x) = y))$;
Call BM$(\pi[succ(x) = y]$, newd$)$;
Endif
Endfor
/* Try to isolate x: */
Call BM$(\pi[succ(x) = \perp], \Delta d - k/2)$;
Else If $(pred_\pi(x) = \emptyset)$, **then**
/* Try to link x with its predecessors */
/* in the input orderings: */
For every $y \in pred(\Pi, x)$ **do**
If $(succ_\pi(y) = \emptyset)$, **then**
newd $:= \Delta d - (k - \#(\Pi, pred(x) = y))$;
Call BM$(\pi[pred(x) = y]$, newd$)$;
Endif
Endfor
/* Try to isolate x: */
Call BM$(\pi[pred(x) = \perp], \Delta d - k/2)$;
Endif

Obtaining an Ordering from the Successor and Predecessor Table

The recursive procedure given above specifies how to compute a successor and predecessor table of a proposed solution π; this table may contain $\perp$ entries to indicate isolated elements. In the following, we show that we can easily output an actual ordering using the information stored in the table.

To this end, observe that, given a BREAKPOINT MEDIAN instance and the table for a complete ordering, the number of $\perp$ entries is even: The reason is that the table has an even number of entries (since an element $x \in G$ has a successor and a predecessor entry, s has only a successor entry, and t has only a predecessor entry) and every new link between two elements adds exactly two entries.

Given a set of signed orderings Π and a signed ordering π, all over $G^{\pm}_{s,t}$, a sequence of elements $x_1, \ldots, x_r \in G^{\pm}_{s,t}$ is called *isolated block* b of π if

1. $succ_\pi(x_l) = x_{l+1}$ for $l = 1, \ldots, r-1$,
2. $pred_\pi(x_1) \notin pred(\Pi, x_1)$ or $x_1 = s$, and
3. $succ_\pi(x_r) \notin succ(\Pi, x_r)$ or $x_r = t$.

Elements x_1 and x_r are called *endpoints* of the isolated block and they are referred to as $e_1(b)$ and $e_2(b)$. An isolated block with $x_1 = s$ is referred to as b_s; an isolated block with $x_r = t$ is referred to as b_t. Given an isolated block b consisting of the sequence $x_1, \ldots, x_r \in G^{\pm}_{s,t}$, we use $-b$ to denote the "reversal" of b, i.e., the sequence $-x_r, -x_{r-1} \ldots, -x_1$, which is also an isolated block. If π consists of isolated blocks b_s, b_1, $b_2, \ldots, b_q$, b_t, we use $(b_s, b_1, b_2, \ldots, b_q, b_t)$ to denote the ordering π' that is obtained from π by setting $succ_{\pi'}(e_2(b_s)) := e_1(b_1)$, $succ_{\pi'}(e_2(b_1)) := e_1(b_2)$ etc. In the following lemma, we spell out that isolated blocks can be permuted and reversed without affecting the induced number of breakpoints.

Lemma 3. *Given a* BREAKPOINT MEDIAN *instance and an ordering* $\pi = (b_s, b_1, b_2, \ldots, b_q, b_t)$ *with isolated blocks* b_s, b_1, $b_2, \ldots, b_q$, b_t. *Then,* π *causes exactly as many breakpoints as* $(b_s, b_1, b_2, \ldots, b_{i-1}, b_{i+1}, b_i, \ldots, b_q, b_t)$ *for* $i = 1, ..., q-1$ *and* $(b_s, b_1, b_2, \ldots, b_{i-1}, -b_i, b_{i+1}, \ldots, b_q, b_t)$ *for* $i = 1, ..., q$.

Proof. (Sketch) Every link between two isolated blocks induces exactly k breakpoints, no matter in which order or orientation the isolated blocks are arranged (except of the fact that b_s is required to be the first and b_t the last of the isolated blocks). □

Lemma 3 shows how to obtain a solution from the information supplied by algorithm BM. To report the median ordering, we output the isolated blocks, beginning with b_s, linking the remaining isolated blocks arbitrarily, while ending with b_t.

Correctness of Algorithm BM

Lemma 4. *(1) A solution* π *supplied by Algorithm BM satisfies* $\sum_{i=1}^{k} d_H(\pi, \pi_i) \leq d$ *and*
(2) if such a solution exists, then Algorithm BM finds one.

Proof. The first statement is assured since Algorithm BM maintains a breakpoint distance counter Δd that is initialized by d, and it is decreased appropriately when two elements are linked or an element is made isolated. In the following, we explain that this "bookkeeping" of the value of Δd is correct. In a branching which links two elements, Δd is decreased exactly by the number of breakpoints that are caused by this new link. More subtle is a branching in which we turn an element x into an isolated element by setting, e.g., $succ_\pi(x) = \perp$. When reporting the solution, we will link x with another isolated element y with $pred_\pi(y) = \perp$ (which exists since the number of isolated elements in π must be even). On the one

hand, the algorithm accounts $k/2$ breakpoints when setting $succ_\pi(x) := \perp$ and another $k/2$ breakpoints when setting $pred_\pi(y) = \perp$. On the other hand, with k input orderings this link causes exactly k breakpoints. Since a solution is reported only if $\Delta d \geq 0$, this shows that a solution supplied by the algorithm causes at most d breakpoints.

Regarding the second statement, consider a BREAKPOINT MEDIAN instance and a solution ordering π. We will show that the algorithm finds either π or a solution that induces at most as many breakpoints as π. Consider elements $x, y \in G^{\pm}_{s,t}$ which are linked in π, e.g., $succ_\pi(x) = y$, and which are linked in the same way in at least one of the input orderings, i.e., $succ_{\pi_i}(x) = y$ for $i \in \{1, \ldots, k\}$. Then, our algorithm explores linking x and y either when considering element x (trying all successors occurring in one of the input orderings) or element y (trying all predecessors occurring in one of the input orderings). Next, consider elements $x, y \in G^{\pm}_{s,t}$ which are linked in π, e.g., $succ_\pi(x) = y$, and which are not linked in the same way in any of the input orderings, i.e., $succ_{\pi_i}(x) \neq y$ for all $i = 1, \ldots, k$. Algorithm BM covers this situation when setting $succ_\pi(x) := \perp$ and $pred_\pi(y) := \perp$. Although it is possible that the algorithm may, when reporting the solution, arrange the isolated blocks in a way such that the successor of x is an isolated element different from y, the obtained solution induces, by Lemma 3, at least as many breakpoints as π. Summarizing, for *every* solution π of the given BREAKPOINT MEDIAN instance, our algorithm either finds π or a solution that can be obtained from π by a rearrangement of its isolated blocks. □

Background on Branching Numbers and Branching Vectors

The running time of our algorithm is determined by the size of the search tree, which depends on the degree(s) of the search tree nodes and the height of the tree. Consider a node in our search tree branching into r subcases; branching into subcase i, $1 \leq i \leq r$, let d_i denote the value by which we decrease the value of the breakpoint counter Δd. We stop the recursion when $\Delta d \leq 0$. Then, we describe the branching in this node by a *branching vector* $(d_1, d_2, \ldots, d_r)$. Assuming that all nodes in the tree have the same branching, we determine the number of leaves of this search tree by the recurrence relation

$$a_{\Delta d} = a_{\Delta d - d_1} + a_{\Delta d - d_2} + \ldots + a_{\Delta d - d_r} \qquad (1)$$

with $a_i = 1$ for $i \leq 0$. Observe that (up to constant factors) the number of leaves in our setting is a valid estimate for the search tree size because we are only interested in trees where each inner node has at least two children. One obtains an upper bound on the search tree size by computing an upper bound for the recursion corresponding to a worst-case branching that can occur in the search tree.

In the following, we omit further details on analyzing these recurrences which can be found, e.g., in (Kullmann, 1999). One can upperbound $a_{\Delta d}$ in recursion (1) by $O(\alpha^{\Delta d})$ where $\alpha := 1/\beta$ and β is the positive real root of the characteristic polynomial

$$p(z) = 1 - z^{d_1} - z^{d_2} - \ldots - z^{d_r}; \qquad (2)$$

note that $p(z)$ is decreasing for $z \geq 0$ and has a single positive real root β with $0 < \beta < 1$. The value α is called the *branching number* of the corresponding recurrence.

Running Time for $k = 3$ Orderings

The following estimation of Algorithm BM's running time is based on the analysis of its search tree size. By Lemma 1 and the subsequent comment given there, we conclude that every recursive call made in Algorithm BM reduces breakpoint counter Δd by at least one. Given an element x with $succ_\pi(x) = \emptyset$ (or $pred_\pi(x) = \emptyset$, resp.), there are, in fact, two possible situations: Either (1) x has the same successor y_1 in two of the input orderings, and successor $y_2 \neq y_1$ in the third input ordering, or (2) x has pairwisely different successors y_1, y_2, and y_3 in the three input orderings. In (1), we decrease Δd by 1 when we set $succ_\pi(x) := y_1$, we decrease Δd by 2 when we set $succ_\pi(x) := y_2$, and we decrease Δd by $3/2$ when we set $succ_\pi(x) := \perp$. This yields branching vector $(1, 2, 3/2)$ and the branching corresponds to branching number 2.15. In (2), we decrease Δd by 2 when we set $succ_\pi(x) := y_i$ for all $i = 1, 2, 3$, and we decrease Δd by $3/2$ when we set $succ_\pi(x) := \perp$. This yields branching vector $(2, 2, 2, 3/2)$ and the branching corresponds to branching number 2.12. This gives an upper bound of $O((2.15)^{\Delta d})$ on the search tree size. In every search tree node, we can, in linear time, test whether the ordering is invalid or completed, and select the branching subcases. This gives the following result.

Proposition 1. BREAKPOINT MEDIAN *for* $k = 3$*, i.e., three signed input orderings, can be solved in time* $O((2.15)^d \cdot n)$.

Note that the branch-and-bound technique based on lower bounds as introduced by Sankoff and Blanchette (1998) can also be used in the framework of our algorithm to further improve its performance in practice.

We emphasize that a key distinguishing point between the algorithm of Sankoff and Blanchette and the above one seems to be that above there is a special treatment of the case of isolated elements and isolated blocks, which is not considered as such by Blanchette and Sankoff. This is advantageous in some cases (namely, when the lower

bound used in the branch-and-bound algorithm of Sankoff and Blanchette has little effect).

Running Time for More Than Three Orderings

Studying BREAKPOINT MEDIAN with $k > 3$, we observe, with growing k, an increasing number of possible branching situations in Algorithm BM. For instance, consider $k = 4$. If we choose an element x for branching which has successor y_1 in three of the four input orderings and which has successor y_2 in the remaining input ordering, then we decrease Δd by 1 when we set $succ_\pi(x) := y_1$, we decrease Δd by 3 when we set $succ_\pi(x) := y_2$, and we decrease Δd by $4/2$ when we set $succ_\pi(x) :=\perp$; this branching corresponds to branching vector $(1, 3, 2)$. The other branching possibilities of Algorithm BM for $k = 4$ are characterized by branching vectors $(3, 3, 3, 3, 2)$, $(3, 3, 2, 2)$, and $(2, 2, 2)$. The branching vectors correspond to characteristic polynomials $p(z)$, e.g., the branching vector $(3, 1, 2)$ corresponds to $p(z) := 1 - z^3 - z - z^2$. The following lemma shows how to characterize the branchings of Algorithm BM by their polynomials. The branchings of BM that are possible for fixed k are referred to as *k-branchings*.

Lemma 5. *A k-branching corresponds to a characteristic polynomial of the form*

$$p(z) := 1 - \sum_{i=1}^{k-1} a_i z^{k-i} - z^{k/2},$$

where $a_1, \ldots, a_{k-1}$ are non-negative integers with $\sum_{i=1}^{k-1} a_i \cdot i = k$.

Proof. Consider a k-branching that selects an element x and branches on the possible successors of x. It creates a subcase for every distinct successor of x that occurs in one of the k given orderings; additionally it creates the "isolation" subcase by setting $succ(x) :=\perp$. In general, we can have, for integers $1 \leq i \leq k-1$ and $0 \leq a_i \leq k$, a_i many pairwisely distinct successors y such that $succ(x) = y$ in exactly i input orderings. Obviously, with k input orderings, we have $\sum_{i=1}^{k-1} a_i \cdot i = k$. The branch in which we choose to set $succ_\pi(x) := y$ in the solution π causes $k-i$ many breakpoints when $succ(x) = y$ in i input orderings. Therefore, we decrease Δd by $k - i$ in this branch. We have a_i branches of this kind. Summarizing over all branches, the k-branching is characterized by the characteristic polynomial $p(z) := 1 - \sum_{i=1}^{k-1} a_i z^{k-i} - z^{k/2}$ with $\sum_{i=1}^{k-1} a_i \cdot i = k$. □

In the remainder of this section, we show that the worst-case branching number of Algorithm BM becomes better with increasing k. To this end, we, firstly, show that the branching $(k-1, 1, k/2)$, characterized by the polynomial

$$p_k(z) := 1 - z^{k-1} - z - z^{k/2},$$

is the worst-case branching among all k-branchings (this yields Proposition 2). We refer to the branching number of $(k - 1, 1, k/2)$ by c_k. Secondly, we prove that c_k decreases with increasing k (this yields Proposition 3).

In order to show Proposition 2, which is the harder part when compared to Proposition 3, we need the following two technical lemmas. Lemma 6 easily follows by function analysis using a computer algebra system such as Maple or Mathematica.

Lemma 6. *For all integers $k \geq 3$ and $p_k(z) = 1-z^{k-1}-z-z^{k/2}$, it holds that*

$$p_k(\sqrt[k-2]{\tfrac{1}{k-1}}) < 0$$ □

Lemma 7 describes the worst-case k-branching among those k-branchings having the same number of subcases.

Lemma 7. *A k-branching with i subcases, $i \in \{2, \ldots, k-1\}$ has the same or a better branching number than the particular k-branching with i subcases characterized by the polynomial $1 - (i-1)z^{k-1} - z^{i-1} - z^{k/2}$.*

Proof. (Sketch) We can transform the branching vector for the branching characterized by polynomial

$$1 - (i-1)z^{k-1} - z^{i-1} - z^{k/2},$$

i.e., branching vector

$$(k-1, \ldots, k-1, i-1, k/2),$$

into the branching vector for an arbitrary k-branching with i subcases by a succession of steps of the following form: Each step takes two entries from the branching vector, which have values v_1 and v_2, resp., with $v_1 > v_2$, and decreases v_1 by one and increases v_2 by one. Lemma 8.5 in (Kullmann, 1999) shows that each of these steps improves the corresponding branching number or leaves it unchanged. □

For example, in the case of $k = 3$, we can conclude by Lemma 7 that the branching number for the branching vector $(2, 2, 2)$ has no higher branching number as the branching number for the branching vector $(3, 1, 2)$; the reason is that we can transform $(3, 1, 2)$ to $(2, 2, 2)$ by decreasing the value of the first entry by one while increasing the value of the second entry by one.

Proposition 2. *For every integer $k \geq 3$, the branching number of any k-branching of Algorithm BM is upper-bounded by c_k.*

Proof. By Lemma 5, k-branchings are characterized by polynomials

$$p(z) := 1 - \sum_{i=1}^{k-1} a_i z^{k-i} - z^{k/2},$$

where $a_1, \ldots, a_{k-1}$ are non-negative integers with $\sum_{i=1}^{k-1} a_i \cdot i = k$. One particular k-branching is the one characterized by the polynomial

$$p_k(z) := 1 - z^{k-1} - z - z^{k/2}.$$

The branching number c_k is $1/\beta$ for the positive real root β of polynomial p_k; let q denote another polynomial of those polynomials described by p and let $c = 1/\gamma$ for the positive real root γ of q (note that the polynomials described by p have a single positive real root which is between 0 and 1).

We show that $\beta \leq \gamma$ which implies that $c_k \geq c$. We assume that $\beta > \gamma$ and derive a contradiction. Setting $p_k(\beta) = q(\gamma)$, we can conclude that

$$\begin{aligned}\beta^{k-1} + \beta + \beta^{k/2} &= \sum_{i=1}^{k-1} a_i \gamma^{k-i} + \gamma^{k/2} \\ &< \sum_{i=1}^{k-1} a_i \beta^{k-i} + \beta^{k/2}.\end{aligned}$$

This can be simplified to

$$\beta^{k-1} + \beta < \sum_{i=1}^{k-1} a_i \beta^{k-i}. \tag{3}$$

We distinguish two cases.
Case 1 applies if

$$\sum_{i=1}^{k-1} a_i \beta^{k-i} \leq k\beta^{k-1}.$$

Together with inequality 3, this implies that $\beta^{k-1} + \beta \leq k\beta^{k-1}$ which can be simplified to $\beta \leq (k-1)\beta^{k-1}$ and further to $1/(k-1) \leq \beta^{k-2}$. On the one hand, this means that

$$\beta \geq \sqrt[k-2]{\frac{1}{k-1}}.$$

On the other hand, we derive by Lemma 6 that

$$\beta < \sqrt[k-2]{\frac{1}{k-1}},$$

a contradiction.
Case 2 applies if

$$\sum_{i=1}^{k-1} a_i \beta^{k-i} > k\beta^{k-1}. \tag{4}$$

By Lemma 7, the branching number of the branching characterized by polynomial q is upperbounded by the branching number of the branching characterized by polynomial $1 - jz^{k-1} - z^j - z^{k/2}$, for an appropriately chosen $j \in \{1, \ldots, k-2\}$; more precisely, j is chosen such that $j+1$ denotes the number of subcases in the branching characterized by q. We conclude (omitting some details)

$$\sum_{i=1}^{k-1} a_i \gamma^{k-i} + \gamma^{k/2} \leq j\gamma^{k-1} + \gamma^j + \gamma^{k/2}. \tag{5}$$

Firstly, using inequality (5) and setting $p_k(\beta) = q(\gamma)$ yields

$$\beta^{k-1} + \beta + \beta^{k/2} \leq j\gamma^{k-1} + \gamma^j + \gamma^{k/2}$$

and, since we assume that $\beta > \gamma$, also

$$\beta^{k-1} + \beta + \beta^{k/2} \leq j\beta^{k-1} + \beta^j + \beta^{k/2}.$$

This simplifies to $\beta^{k-1} + \beta < j\beta^{k-1} + \beta^j$ and further to

$$1 < (j-1)\beta^{k-2} + \beta^{j-1}. \tag{6}$$

Secondly, by inequality (5), we derive that $j\beta^{k-1} + \beta^j \geq \sum_{i=1}^{k-1} a_i \beta^{k-i}$. Together with inequality (4), this yields

$$j\beta^{k-1} + \beta^j > k\beta^{k-1},$$

which simplifies to $\beta^j > (k-j)\beta^{k-1}$ and further to $1/(k-j) > \beta^{k-j-1}$. We conclude that

$$\beta < \sqrt[k-j-1]{\frac{1}{k-j}}.$$

Substituting this into (6), we obtain $1 < f(k,j)$ for

$$f(k,j) := (j-1)\left(\frac{1}{k-j}\right)^{\frac{k-2}{k-j-1}} + \left(\frac{1}{k-j}\right)^{\frac{j-1}{k-j-1}}$$

and $j = 1, \ldots, k-2$. Function analysis, however, yields $f(k,j) \leq 1$ for $j \geq 1$, a contradiction. □

Proposition 3. *For every integer $k \geq 3$, $c_k > c_{k+1}$.*

Proof. Let β_k be the smallest positive zero of polynomial

$$p_k(z) = 1 - z^{k-1} - z - z^{k/2}.$$

Then, $c_k = 1/\beta_k$. Analogously, let β_{k+1} be the smallest positive zero of polynomial

$$p_{k+1}(z) = 1 - z^k - z - z^{(k+1)/2},$$

such that $c_{k+1} = 1/\beta_{k+1}$. Since $p_k(\beta_k) = 0$ and $p_{k+1}(\beta_{k+1}) = 0$, it follows that

$$\beta_k^{k-1} + \beta_k + \beta_k^{k/2} = \beta_{k+1}^k + \beta_{k+1} + \beta_{k+1}^{(k+1)/2}.$$

Assuming $\beta_k > \beta_{k+1}$, we derive a contradiction. With $0 < \beta_{k+1} < \beta_k < 1$, we have also

$$\beta_{k+1}^k < \beta_k^{k-1} \text{ and } \beta_{k+1}^{(k+1)/2} < \beta_k^{k/2}.$$

This yields

$$\beta_{k+1}^k + \beta_{k+1} + \beta_{k+1}^{(k+1)/2} < \beta_k^{k-1} + \beta_k + \beta_k^{k/2},$$

a contradiction. □

Theorem 1. BREAKPOINT MEDIAN *for* $k \geq 3$ *can be solved in time* $O((2.15)^d \cdot kn)$.

Proof. By Proposition 2, c_k is the worst-case branching number among all k-branchings of Algorithm BM. By Proposition 3, $c_3 < c_k$ for $k > 3$. By Proposition 1, $c_3 = 2.15$, which concludes the proof. □

Normalized distance parameter.

On the one hand, as indicated before, the exponential base c_k (corresponding to the branching vector $(k-1, 1, k/2)$) in the running time of Algorithm BM becomes better for increasing k. For instance, we have $c_3 = 2.15$, $c_4 = 1.84$, $c_5 = 1.68$, $c_{20} = 1.21$, $c_{50} = 1.11$, and $c_{100} = 1.06$, which tends to 1 when k goes to infinity. For gene orderings considered so far, however, values of k between 3 and 50 are the current state of the art. On the other hand, since BREAKPOINT MEDIAN sums up the distance over all k input orderings, with increasing k the distance parameter d should necessarily also increase. Hence, it would be natural to consider some kind of "normalized" parameter d', which does not increase with increasing k. The most natural question would be whether BREAKPOINT MEDIAN is fixed-parameter tractable with respect to parameter $d' := d/k$. So far, we could not prove this and it remains an issue of future theoretical work. But, at least, concrete calculations for values of k in the range up to $k \approx 50$ demonstrate that we still can obtain exact algorithms with reasonable exponential terms. Furthermore, for "lighter" normalized parameterization such as $d' := d/\sqrt{k}$, the algorithm clearly exhibits fixed-parameter behavior. Notably, in our practical applications Algorithm BM always provided sufficient efficiency. One thing to additionally take into account here is that our estimates for the search tree sizes always are worst-case; in practice, our algorithm turned out to be much faster than could be expected from the theoretical (worst-case) running time analysis.

BREAKPOINT MEDIAN EXPERIMENTS

We did experiments on synthetic BREAKPOINT MEDIAN instances. The results are shown in Figure 1. The implementation was done using GNU C++ version 3.0.4, and the running time was measured on a Sun Blade 100 machine with Sparc 2e processor (500 MHz) and 512 MB main memory under Solaris 5.8. For given values of n, k, and d, we produced a dataset containing k orderings over n elements such that there exists a median having a total breakpoint distance of at most d as follows. We started with k pairwisely equal orderings. Then, we randomly decided between reversals or transpositions, and performed this operation on a randomly selected segment in a randomly selected ordering. We maintain a counter for the potential breakpoints: for a reversal we counted two, for a transposition we counted three breakpoints. In this way, we continued to rearrange the data as long as the breakpoint counter was smaller than d. In general, especially with large numbers of generated breakpoints, we had less breakpoints than we counted, since not every operation really introduces *new* breakpoints; for "reasonable" values of n, k, and d, e.g., $d \leq (1/4)kn$, however, the number of breakpoints was close to d. For each set of values for n, k, and d, we measured the average performance on 25 different input instances. In Figure 1, one can see that even instances with a comparatively high number of input orderings and correspondingly high values of induced breakpoints can still be solved efficiently.

k	d	running time in sec	k	d	running time in sec
3	30	0.10	40	400	0.25
	60	0.10		800	0.38
	90	0.19		1200	0.51
	120	0.30		1600	0.63
10	100	0.13	100	1000	0.60
	200	0.14		2000	1.09
	300	0.17		3000	1.90
	400	0.19		4000	2.57

Fig. 1. Experimental results: Running times on synthetic data obtained for k orderings on $n = 100$ elements, for several ratios of breakpoint score d to the instance size nk, namely $d = 0.1nk$, $d = 0.2nk$, $d = 0.3nk$, and $d = 0.4nk$.

APPLICATION TO PHYLOGENY RECONSTRUCTION

An application of BREAKPOINT MEDIAN is given in the reconstruction of breakpoint phylogenies, i.e., the problem of finding the most parsimonious phylogenetic tree w.r.t. breakpoint distance. In this section, we outline a new heuristic strategy computing the breakpoint phylogeny for a set of gene order data which uses the BREAKPOINT MEDIAN algorithm as a central subprocedure. In compari-

son with previous heuristics (Moret *et al.*, 2002a, 2001a; Sankoff and Blanchette, 1998), our approach does not exhaustively explore the whole search space consisting of all binary trees with k leaves, but resolves the grouping of taxa, level by level, from a (hypothetical) root down to the leaves of the phylogenetic tree.

A Heuristic Computing Breakpoint Phylogenies

Given gene orderings $\Pi = \{\pi_1, \ldots, \pi_k\}$ for a set of k taxa, the algorithm starts by computing a root node, called *virtual root* of the tree (only necessary for the construction) and, then, the algorithm recursively divides the set of taxa into two subsets, associating new nodes with these subsets; the new nodes become child nodes of the virtual root and roots for the subtrees corresponding to the subsets. The recursion ends when the subsets have size exactly one.

To label the virtual root node, our heuristic computes the breakpoint median π_r for the given set of gene orderings. To obtain a bipartition of the set of taxa, we consider all 2^{k-1} distinct bipartitions of Π into non-empty sets Π_1 and Π_2. We compute the optimal breakpoint medians π_1 for $\Pi_1 \cup \{\pi_r\}$, inducing a score of d_1 breakpoints, and π_2 for $\Pi_2 \cup \{\pi_r\}$, inducing a score of d_2 breakpoints. Among all these bipartitions, we choose the ones with a minimum total number of induced breakpoints, i.e., the ones for which $d_1 + d_2$ is minimum. The breakpoint medians π_1 and π_2 corresponding to such an optimal bipartition are chosen to label the two child nodes of the node labeled π_r.[‡] We choose π_1 in this way (π_2 is analogous), such that π_1 is not only a good median w.r.t. the orderings in Π_1, but also takes into account the information on the orderings in Π_2 which is reflected in π_r. Now, if Π_1 (Π_2 is completely analogous) consists of two elements only, we create two child nodes of the π_1 node, each child labeled with one element from Π_1. If Π_1 contains more than two elements, we process this set recursively, taking the π_1 node as the virtual root and Π_1 as the set of gene orderings, again considering all bipartitions of Π_1.

In comparison to previous approaches that search through the whole space of all $\prod_{j=3}^{k}(2j-5)$ possible trees over k taxa (Moret *et al.*, 2002a, 2001a; Sankoff and Blanchette, 1998), the search space of our heuristic is determined by $k2^{k-1}$ considered bipartitions.

The Campanulaceae Dataset

Using our heuristic, we analyzed the dataset introduced by Cosner *et al.* (2000a,b), which contains signed gene order information for 13 chloroplast genomes of the plant family Campanulaceae. These data are referred to in a considerable number of papers (e.g., (Caprara, 2001; Moret *et al.*, 2001a,b)) and, therefore, seem to be an appropriate challenge dataset.

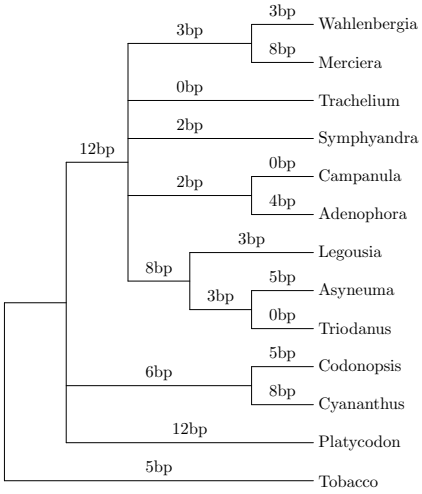

Fig. 2. Tree with breakpoint score 89 that is found by our heuristic method for the Campanulaceae dataset (outgroup taxon tobacco).

In these data, every gene occurs exactly once in all orderings. Within 1 min 45 sec, we processed the dataset and the best tree we found caused 89 breakpoints, i.e., we found no tree causing less breakpoints. The topology of this tree is given in Figure 2; the displayed tree is not binary, since we contracted branches of score zero, i.e., whose endpoints are labeled by the same orderings. Due to the contracted inner branches the shown tree corresponds to 216 different binary topologies for each of which we can give a labeling of the inner nodes that yields the breakpoint score 89. These tree topologies are exactly the 216 different tree topologies that are also found by Moret *et al.* (2001b). They used a (heuristically determined) constraint tree topology in order to explore the space of all binary tree topologies that are a refinement of this constraint tree. This way, they selected the 216 topologies from 10,395 considered binary tree topologies.

Our result is preferable to a tree causing 96 breakpoints that is, according to (Cosner *et al.*, 2000a), found by the "BPAnalysis" program (Sankoff and Blanchette, 1998) within a day, and it is competitive with the tree causing 89 breakpoints reported by Cosner *et al.* (2000a,b). Recently, Moret *et al.* (2001b) also mentioned that they have found trees causing 84 breakpoints. However, using the "GRAPPA" software (Moret *et al.*, 2001a) we could not reproduce the results but only found the 216 mentioned topologies causing 89 breakpoints.

To find the optimal trees by searching the whole space

[‡]We optionally allow to investigate all optimal medians that are found for $\Pi_1 \cup \{\pi_r\}$ and for $\Pi_2 \cup \{\pi_r\}$ and to run the described recursion for each combination of optimal medians separately. In the Campanulaceae dataset, however, these medians turned out to be unique in most cases.

of all phylogenetic trees over 13 taxa, even when using the highly optimized "GRAPPA" software which includes additional bounding techniques, Moret *et al.* (2002b) still need "a few hours on a single workstation". Only when using a well resolved constraint tree (such that the solution is required to be a refinement of the constraint tree), Moret *et al.* (2001b) reduced the size of the search space significantly.

CONCLUSION

We gave an efficient fixed-parameter algorithm for BREAKPOINT MEDIAN. The parameterization is with respect to distance parameter d. By experimental results, we demonstrated that this allows for a new methodological approach to breakpoint phylogeny.

It remains open whether the closely related BREAKPOINT CENTER, where, by way of contrast to BREAKPOINT MEDIAN, not the sum of distances but the *maximum* distance shall be minimized, is also fixed-parameter tractable with respect to d.

In the multiple sequence alignment context, there also arises a median problem, but then with edit distance instead of breakpoint distance. It is interesting to ask whether this median problem is fixed-parameter tractable w.r.t. the corresponding distance parameter—an approach similar to the one employed here seems possible. Note, however, that trying to confine the combinatorial explosion to the number k of input species again seems fruitless, since it can be deduced from results of Bodlaender *et al.* (1995) and Higuera and Casacuberta (2000) that the problem is $W[t]$-hard for all t (see (Downey and Fellows, 1999) for a definition). This, roughly speaking, shows fixed-parameter intractability w.r.t. parameter k unless a very unlikely collapse in computational complexity theory occurs.

As to BREAKPOINT MEDIAN, future theoretical research might deal with the mentioned normalization effects for d. Also, it would be desirable to extend the algorithm to the case in which not all orderings are over the same set of elements or when elements occur more than once in one ordering; these cases apply when genomes have a different set of genes or contain duplicated genes. Further experiments could address the application of the presented breakpoint phylogeny heuristic to new biological datasets or to synthetic datasets. Also, w.r.t. the desired application to phylogeny reconstruction, it might be useful to extend the considerations and experiments to weighted variants of BREAKPOINT MEDIAN. Finally, it would be desirable to identify further applications of BREAKPOINT MEDIAN besides the breakpoint phylogeny application presented here.

Acknowledgements: We thank Falk Hüffner for implementing the BREAKPOINT MEDIAN algorithm, and Robert Jansen, Li-San Wang, and Tandy Warnow for making the Campanulaceae data available to us. We thank the anonymous referees of *ECCB 2002* for valuable hints improving the presentation of our work.

Jens Gramm was supported by the Deutsche Forschungsgemeinschaft (DFG), research project "OPAL" (optimal solutions for hard problems in computational biology), NI 369/2-1.

REFERENCES

J. Alber, J. Gramm, and R. Niedermeier. Faster exact solutions for hard problems: a parameterized point of view. *Discrete Mathematics*, 229(1-3):3–27, 2001.

M. Blanchette, B. Schwikowski, and M. Tompa. Algorithms for phylogenetic footprinting. *Journal of Computational Biology*, 9(2):211–224, 2002.

H. L. Bodlaender, R. G. Downey, M. R. Fellows, M. T. Hallett, and H. T. Wareham. Parameterized complexity analysis in computational biology. *Computer Applications in the Biosciences*, 11: 49–57, 1995.

D. Bryant. The complexity of the Breakpoint Median problem. Technical report CRM-2579. Centre de recherches mathématiques, Université de Montréal. 1998.

A. Caprara. On the practical solution of the Reversal Median problem. In *Proc. of the 1st WABI*, number 2149 in LNCS, pages 238–251, 2001. Springer.

M. E. Cosner, R. K. Jansen, B. M. E. Moret, L. A. Raubeson, L. S. Wang, T. Warnow, and S. Wyman. An empirical comparison of phylogenetic methods on chloroplast gene order data in Campanulaceae. In D. Sankoff and J. Nadeau (eds.): *Comparative Genomics*, pages 99–121. Kluwer, 2000.

M. E. Cosner, R. K. Jansen, B. M. E. Moret, L. A. Raubeson, L. S. Wang, T. Warnow, and S. Wyman. A new fast heuristic for computing the breakpoint phylogeny and experimental phylogenetic analyses of real and synthetic data. In *Proc. of the 8th ISMB*, pages 104–115, 2000. AAAI Press.

R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer. 1999.

M. R. Fellows. Parameterized complexity: the main ideas and some research frontiers. In *Proc. of the 12th ISAAC*, number 2223 in LNCS, pages 441–453, 2001. Springer.

M. R. Fellows, J. Gramm, and R. Niedermeier. Parameterized intractability of Closest Substring. In *Proc. of the 19th STACS*, number 2285 in LNCS, pages 262–273, 2002. Springer.

M. T. Hallett and J. Lagergren. New algorithms for the duplication-loss model. In *Proc. of the 4th ACM RECOMB*, pages 138–146, 2000. ACM Press.

C. de la Higuera and F. Casacuberta. The topology of strings: two NP-complete problems. *Theoretical Computer Science*, 230:39–48, 2000.

O. Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223: 1–72, 1999.

M. Li, B. Ma, and L. Wang. On the Closest String and Substring Problems. *Journal of the ACM*, 49(2):157–171, 2002.

D. Liben-Nowell. Gossip is synteny: incomplete gossip and the syntenic distance between genomes. In *Proc. of 12th ACM-SIAM SODA*, pages 177–185, 2001. ACM Press. To appear in *Journal*

of Algorithms.

B. M. E. Moret, D. A. Bader, and T. Warnow. High-performance algorithm engineering for computational phylogenetics. *Journal of Supercomputing*, 22:99–111, 2002.

B. M. E. Moret, J. Tang, L. Wang, T. Warnow. Steps toward accurate reconstruction of phylogenies from gene-order data. To appear in *Journal of Computer and System Sciences*. 2002.

B. M. E. Moret, S. K. Wyman, D. A. Bader, T. Warnow, and M. Yan. A new implementation and detailed study of breakpoint analysis In *Proc. of the 6th Pacific Symposium on Bioinformatics*, pages 583–594. 2001.

B. M. E. Moret, L. Wang, T. Warnow, and S. K. Wyman. New approaches to phylogeny reconstruction from gene order data. *Bioinformatics* 17:S165–S173, 2001.

I. Pe'er and R. Shamir. The median problems for breakpoints are NP-complete. Electronic Colloquium on Computational Complexity, Technical Report 98-071, Trier, 1998.

I. Pe'er and R. Shamir. Approximation algorithms for the Permutations Median Problem in the Breakpoint Model. In D. Sankoff and J. Nadeau (eds.): *Comparative Genomics*, pages 225–241. Kluwer, 2000.

D. Sankoff and M. Blanchette. Multiple genome rearrangement and breakpoint phylogeny. *Journal of Computational Biology*, 5:555–570. 1998.

A. C. Siepel and B. M. E. Moret. Finding an optimal inversion median: experimental results. In *Proc. of the 1st WABI*, number 2149 in LNCS, pages 189–204, 2001. Springer.