Genome Rearrangement and Planning

Esra Erdem
Institute of Information Systems
Vienna University of Technology, Vienna, Austria
esra@kr.tuwien.ac.at

Elisabeth Tillier
Ontario Cancer Institute
620 University Avenue, Toronto, Canada
e.tillier@utoronto.ca

Abstract

The genome rearrangement problem is to find the most economical explanation for observed differences between the gene orders of two genomes. Such an explanation is provided in terms of events that change the order of genes in a genome. We present a new approach to the genome rearrangement problem, according to which this problem is viewed as the problem of planning rearrangement events that transform one genome to the other. This method differs from the existing ones in that we can put restrictions on the number of events, specify the cost of events with functions, possibly based on the length of the gene fragment involved, and add constraints controlling search. With this approach, we have described genome rearrangements in the action description language ADL, and studied the evolution of *Metazoan* mitochondrial genomes and the evolution of *Campanulaceae* chloroplast genomes using the planner TLPLAN. We have observed that the phylogenies reconstructed using this approach conform with the most widely accepted ones.

Introduction

In biology, evolutionary trees (or phylogenies) can be reconstructed from the comparison of genomes of species (Sankoff & Blanchette 1998). An approach to quantifying the evolution of genomes from a common ancestor is to determine the number of rearrangement events, such as transpositions, inversions, or transversions, that change the order of genes in a genome. The fewer the number of such events, the closer the genomes in the phylogeny. The genome rearrangement problem is the problem of finding the minimum number of such successive events between two genomes, and it is conjectured to be NP-hard. This paper studies the genome rearrangement problem in the context of planning.

In a planning problem, we want to find a plan—a sequence of actions that leads to the given goal. Classical planning is NP-hard for plans of polynomially-bounded length (Bylander 1994). This result holds also in the presence of actions with conditional effects (Erol, Nau, & Subrahmanian 1995), and temporal goals (Baral, Kreinovich, & Trejo 2001).

We consider genome rearrangement as a planning problem: given two genomes and a positive integer k, find a sequence of at most k events that transforms one genome to the other. We describe the planning domain and the problem in ADL (Pednault 1989). With some heuristics expressed in temporal logics, we use TLPLAN[1] (Bacchus & Kabanza 2000) to compute solutions.

Most of the existing systems, like GRAPPA (Moret *et al.* 2001), consider inversions. Like DERANGE 2 (Blanchette, Kunisawa, & Sankoff 1996), our approach can handle transpositions and transversions as well, possibly with some weights. Moreover, we can specify costs of actions with complex functions, possibly depending on the length of the gene sequence that goes under transformation, or the number of breakpoints. Also we can put constraints on the number of transpositions, inversions, or transversions. The flexibility of describing weight or cost functions, and adding control information is important in understanding the frequency of different events and testing evolutionary hypotheses.

With this planning approach, based on the number of events, we have generated a distance matrix for mitochondrial genomes of *Metazoa* (animals with a nervous system, and muscles) and for chloroplast genomes of *Campanulaceae* (flowering plants) and used a distance matrix program, like FITCH and NEIGHBOR (available with PHYLIP (Felsenstein 2004)), to construct phylogenies. We have observed that the phylogenies constructed this way conform with the most widely accepted ones.

We have also experimented with 100 randomly generated genome rearrangement problems. For 51 problems, the solutions computed by TLPLAN include less number of events compared to the ones computed by DERANGE 2;[2] for 33 problems, it is the other way around.

Problem Description

The *genome* of a single-chromosome organism can be represented by circular configurations of numbers $1, \ldots, n$, with a sign $+$ or $-$ assigned to each of them. For instance, Figure 1(a) shows a genome for $n = 5$. Numbers $\pm 1, \ldots, \pm n$ will be called *labels*. Intuitively, a label corresponds to a gene, and its sign corresponds to the orientation of the gene.

[1] http://www.cs.toronto.edu/˜fbacchus/tlplan.html .
[2] http://www.mcb.mcgill.ca/˜blanchem/software.html .

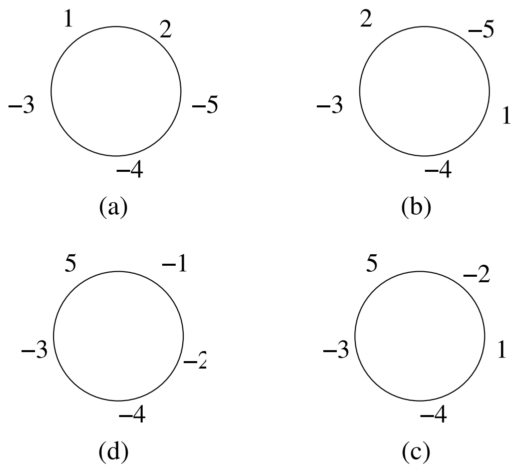

Figure 1: (a) A genome; (b) a transposition of (a); (c) an inversion of (b); (d) a transversion of (c).

By $(l_1, \ldots, l_n)$ we denote the genome formed by the labels $l_1, \ldots, l_n$ ordered clockwise. For instance, each of the expressions $(1, 2, -5, -4, -3), (2, -5, -4, -3, 1), \ldots$ denotes the genome in Figure 1(a).

Evolution of genomes from a common ancestor is studied in terms of events, such as inversions, transpositions, and transversions.

About genomes g, g' we say that g' is a *transposition* of g (or *can be obtained from g by a transposition*) if, for some labels $l_1, \ldots, l_n$ and numbers k, m ($0 < k, m \leq n$),

$$g = (l_1, \ldots, l_n),$$
$$g' = (l_k, \ldots, l_m, l_1, \ldots, l_{k-1}, l_{m+1}, \ldots, l_n).$$

For instance, the genome in Figure 1(b) is a transposition of the genome in Figure 1(a). Given two genomes g and g', the problem of finding the smallest number of successive transpositions by which g' can be obtained from g is conjectured to be in NP (Bafna & Pevzner 1998).

Similarly, about genomes g, g' we say that g' is an *inversion* of g (or *can be obtained from g by an inversion*) if, for some labels $l_1, \ldots, l_n$ and a number m ($0 < m \leq n$),

$$g = (l_1, \ldots, l_n),$$
$$g' = (-l_{m-1}, \ldots, -l_1, l_{m+1}, \ldots, l_n).$$

For instance, the genome in Figure 1(c) is an inversion of the genome in Figure 1(b). Given two genomes g and g', the problem of finding the smallest number of successive inversions by which g' can be obtained from g is in P (Hannenhalli & Pevzner 1995).

About genomes g, g' we say that g' is an *transversion* (or *inverted transposition*) of g (or *can be obtained from g by a transversion*) if, for some labels $l_1, \ldots, l_n$ and numbers k, m ($0 < k, m \leq n$),

$$g = (l_1, \ldots, l_n),$$
$$g' = (-l_m, \ldots, -l_k, l_1, \ldots, l_{k-1}, l_{m+1}, \ldots, l_n).$$

For instance, the genome in Figure 1(d) is a transversion of the genome in Figure 1(c).

The *edit distance* between genomes g and g' is the smallest number k such that g' can be obtained from g by k successive inversions, transpositions, and transversions. The problem of finding the minimum edit distance between two genomes is conjectured to be in NP. We will call this problem the "genome rearrangement problem."

We consider the decision problem corresponding to the genome rearrangement problem: given two genomes g and g', and a positive integer k, decide whether g' can be obtained from g by at most k successive events.

In a planning problem, we are given an initial state and a goal, and we want to find a plan—a sequence of actions that leads to the goal from the initial state. Therefore, we view the genome g as the initial state, and the genome g' as the goal state, and try to find a sequence of at most k actions, i.e., transpositions, inversions, and transversions, that would lead to the goal from the initial state.

Solutions as Plans

We represent a genome by specifying the clockwise order of labels in that genome. For that we introduce a fluent $clockwise(L, L1)$ expressing that label $L1$ comes after label L in clockwise direction.

We introduce three actions to describe transpositions, inversions, and transversions:

- $transpose(L1, L2, L)$ ("the gene sequence starting with the gene $L1$ and ending at the gene $L2$ is inserted after gene L"),
- $invert(L, L1)$ ("the action of inverting gene sequence starting with the gene labeled L and ending with the gene labeled $L1$"), and
- $transvert(L1, L2, L)$ ("the gene sequence starting with the gene $L1$ and ending at the gene $L2$ is first inverted and then inserted after gene L").

For instance, suppose that we are given two genomes, $(1, 2, -5, -4, -3)$ and $(1, 5, -4, -3, -2)$, and $k = 3$. In the corresponding planning problem, the initial state is described by one of these two genomes, say the former:

```
(define (initial0)
       (clockwise 1 2)
       (clockwise 2 -5)
       (clockwise -5 -4)
       (clockwise -4 -3)
       (clockwise -3 1))
```

and the goal state is described by the other one:

```
(define (goal0)
       (clockwise 1 5)
       (clockwise 5 -4)
       (clockwise -4 -3)
       (clockwise -3 -2)
       (clockwise -2 1))
```

in the language of TLPLAN, which is basically first-order logic written in a lisp syntax. The maximum plan length k is set to 3 by the fact:

```
(set-initial-facts (= k 3)) .
```

With the description of this planning problem

```
(set-goal (goal0))
(set-initial-world (initial0))
```

TLPLAN computes the following 3-step plan

```
(transvert -5 -5 1)
(transpose -4 -3 5)
(invert 2 2)
```

according to which the genome $(1, 2, -5, -4, -3)$ can be transformed to $(1, 5, -4, -3, -2)$ as follows: first -5 is inverted and then inserted after 1, next the sequence $-4, -3$ is inserted after 5, and finally 2 is inverted. Here, by default, the cost of each action is 1, and depth-best-first search strategy is applied. When the search criterion is set to best-first, TLPLAN computes the following 2-step plan

```
(transvert -5 -5 1)
(transvert 2 2 -3) .
```

When the cost of a transversion is set to 2, it computes the following 2-step plan

```
(transvert 2 2 -3)
(invert -5 -5) .
```

In such a planning problem, we can put additional constraints as discussed in the following sections.

Planning Domain Description

We describe each genome rearrangement event, i.e., transposition, inversion, and transversion, as an ADL-style operator in the language of TLPLAN. For instance, a transposition is described as follows:

```
(def-adl-operator (transpose ?x ?y ?z)
  ; preconditions
  (pre (?x) (label ?x)
       (?y) (label ?y)
       (?z) (label ?z)
       (cantranspose ?x ?y ?z))

  ; insertion of ?x ?y after ?z
  ; in (?x1,?x..?y,?y1..?z,?z1,...)
  ; is (?x1,?y1..?z,?x..?y,?z1,...)
  (exists (?x1) (clockwise ?x1 ?x)
       (?y1) (clockwise ?y ?y1)
       (?z1) (clockwise ?z ?z1)
       (and (add (clockwise ?x1 ?y1)
                 (clockwise ?z ?x)
                 (clockwise ?y ?z1))
            (del (clockwise ?x1 ?x)
                 (clockwise ?y ?y1)
                 (clockwise ?z ?z1))))) .
```

The first line above describes the name of the ADL-style operator. The next four lines following the comment preceded by semi-colon describe the preconditions of a transposition: all `?x`, `?y`, and `?z` are labels, and the sequence starting with `?x` and ending with `?y` can be inserted after `?z`. Finally, the add list and the delete list are described, when the sequence `?x..?y` is inserted after `?z` in the genome `(?z,?z1,...,?x1,?x,...,?y,?y1,...)`. Here `(cantranspose ?x ?y ?z)` is defined as follows:

```
(def-defined-predicate
  (cantranspose ?x ?y ?z)
  (and
    ; the length of the plan constructed
    ; so far is less than k
    (< (plan-length) (k))

    (not (= ?x ?z))
    (not (= ?y ?z))

    ; ?z is not followed by ?x
    (not (clockwise ?z ?x))

    ; ?z is not between ?x and ?y
    (notbetween ?z ?x ?y))) .
```

An inversion is described similarly by an ADL-style operator:

```
(def-adl-operator (invert ?x ?y)
  ; preconditions
  (pre (?x) (label ?x)
       (?y) (label ?y)
       (caninvert ?x ?y))

  ; change the sign of ?x
  (del (label ?x))
  (add (label (* -1 ?x)))

  ; inversion of ?x..?y in
  ; (?x1,?x..?z1,?z2..?y,?y1,...) is
  ; (?x1,-?y..?-z2,-?z1..-?x,?y1,...)

  ; first invert every sequence ?z1 ?z2
  ; in ?x..?y
  (implies
    (not (= ?x ?y))
    (forall
      (?z1 ?z2) (clockwise ?z1 ?z2)
      (implies
        (and (in ?z1 ?x ?y)
             (in ?z2 ?x ?y))
        (and (del (label ?z2))
             (add (label (* -1 ?z2)))
             (del (clockwise ?z1 ?z2))
             (add (clockwise
                       (* -1 ?z2)
                       (* -1 ?z1)))))))

  ; then change the neighbors of ?x1?
  ; and ?y1
  (exists
    (?x1) (clockwise ?x1 ?x)
    (?y1) (clockwise ?y ?y1)
    (and
       (add (clockwise ?x1 (* -1 ?y ))
            (clockwise (* -1 ?x ) ?y1))
       (del (clockwise ?x1 ?x)
            (clockwise ?y ?y1))))) .
```

Similarly, we describe a transversion as an ADL-style operator.

Useful Heuristics

For a more efficient computation of "good" plans, we use some heuristics that reduce the search space, and that control the search.

Some heuristics are expressed as a part of the preconditions of actions. For instance, here are some heuristics we consider for transpositions:

- One can insert a sequence of labels after label `?z` if the label following `?z` is not "good"—it is different from the one following `?z` in the goal state:

```
(not (goodafter ?z))
```

- One can insert `?x..?y` after a label, if the sequence `?x..?y` is "good"—it is a subsequence of the circular ordering described by the goal state:

```
(goodsequence ?x ?y)
```

  and if `?x..?y` is not a subsequence of a larger good sequence:

```
(not (goodafter ?y))
(not (goodbefore ?x))
```

Similarly, we extend the preconditions of an inversion and transversions. With these heuristics, for instance, the number of world states searched to solve the planning problem corresponding to the genome rearrangement problem for *Campanula*'s cpDNA (chloroplast DNA) and *Tobacco*'s cpDNA is reduced by a factor of 140; the computation time improves by a factor of 3.

Some heuristics are expressed in the first order temporal logic described in (Bacchus & Kabanza 2000). Here are some of these heuristics:

- If the position of a label in the current state is good relative to the goal state, then it is not allowed to be moved to another location in the next state:

```
(define (control1)
 (always
   (forall
     (?x) (label ?x)
     (implies (goodposition ?x)
       (next (goodposition ?x))))))
```

- If a gene sequence can get into its goal position with one transposition, then we move it:

```
(define (control2)
 (always
   (implies
     (exists
       (?x) (label ?x) (?y) (label ?y)
       (canmovetofinal ?x ?y))
     (exists
       (?x) (label ?x) (?y) (label ?y)
       (and
         (canmovetofinal ?x ?y)
         (next (and
           (goodposition ?x)
           (goodposition ?y)
           (goodsequence ?x ?y))))))))
```

- We say that there is a *breakpoint* between two genomes if one of the genomes includes the pair l, l' and the other genome includes neither the pair l, l' nor the pair $-l', -l$. For instance, there are 3 breakpoints between $(1, 2, 3, 4, 5)$ and $(1, 2, -5, -4, 3)$. We ensure that the number of breakpoints decrease at each time stamp by the formula:

```
(define (control3)
 (always
   (exists (?d)
      (= ?d (breakpoints))
      (next (< (breakpoints) ?d)))))
```

  where `(breakpoints)` is the number of breakpoints in the current state relative to the goal.

After expressing such temporal constraints, we conjoin them as the search control

```
(set-tl-control
  (and (control1)
       (control2)
       (control3)))) .
```

For instance, two thirds of the world states searched to solve the planning problem that correspond to the genome rearrangement problem for *Drosophila yakuba*'s mitochondrial DNA (mtDNA) and Human's mtDNA are pruned with these heuristics; the computation time improves by a factor of 3.

The breakpoint heuristic (`control3`) is used in many existing systems (e.g., GRAPPA) to approximate the true evolution; `control1` is complementary to this heuristic. The control strategy described in `control2` is a "trigger" control like `bbw-control1` of (Bacchus & Kabanza 2000).

Other Genome Rearrangement Problems

We can solve variations of the genome rearrangement problem by adding some constraints.

- Cost of events can be specified as part of their definitions. For instance, we can add

```
(cost 2)
```

  in the definition of `transpose` to express that the cost of a transposition is 2. Alternatively, we can describe the cost of `(transpose ?x ?y ?z)` by the length of the gene sequence involved:

```
(cost (length ?x ?y)) .
```

- In addition to the constraint on the plan length, we can put a constraint on the cost of the plan. For that, we can add

```
(< (plan-cost) (c))
```

  as a precondition of operators; here `c` is the given maximum cost.

- We can also put constraints on the number of transpositions, inversions, and transversions. For that, first we add new function fluents `nt`, `ni`, and `nti` respectively, which are initially set to `0`, and then incremented by 1 at each occurrence of the corresponding operator. For instance, `nt` is incremented by 1 by including in the definition of `transpose` the following line:

  ```
  (add (= (nt) (+ 1 (nt))))
  ```

  Then we need to include in the definition of `cantranspose` the constraint

  ```
  (< (nt) (mt))
  ```

  expressing that the total number `nt` of transpositions so far is less than the given maximum number `mt` of transpositions.

- Another constraint can be put on the length of sequences that go under transformations. For instance, to ensure that in `(transpose ?x ?y ?z)` the length of `?x..?y` is less than 5, we can add to the preconditions of `transpose` the following:

  ```
  (< (length ?x ?y) 5) .
  ```

Experimental Results

As in GRAPPA, before we start searching for a sequence of events that transforms a genome to another, we "condense" the given genomes by identifying the common subsequences and replacing them by some new identifiers. For instance, consider the genomes

$$g = (1, 2, 3, 4, 5)$$
$$g' = (1, 2, -5, -4, 3).$$

The sequence $1, 2$ is a subsequence of both g and g', so we can replace it by some identifier, say a. Similarly, the sequence $4, 5$ is a subsequence of g and its inversion is a subsequence of g', so we can replace $4, 5$ in g by some identifier, say b, and its inversion in g' by $-b$. Thus we obtain $(a, 3, b)$ and $(a, -b, 3)$. After relabeling these two circular sequences by integers, we get two condensed genomes

$$h = (1, 2, 3)$$
$$h' = (1, -3, 2).$$

By this way, the problem of transforming g to g' is reduced to the smaller problem of transforming h to h'.

We keep track of which label of the condensed genomes stands for which gene sequence so that, after computing a plan, we can "expand" these labels with respect to the original gene sequences. For instance, one way to obtain h' from h is by the event $transvert(3, 3, 1)$, which stands for $transvert(4, 5, 2)$ in the original setting.

To be able to define costs of events with respect to their length, we define a new function $originalLength$ which describes the length of a sequence in the original setting. For instance, $originalLength(1, 2)$ in h is 3, $originalLength(1, 3)$ in h is 5. Then, for instance, we can express the cost of `(transpose ?x ?y ?z)` by

```
(cost (originalLength ?x ?y))
```

in the definition of `transpose`. Note that, in this case, ensuring that the original length of `?x..?y` is less than 5 may prevent finding solutions since each label `?x` and `?y` may stand for a gene sequence of length greater than 5. On the other hand, we can define the cost of `transpose` as

```
(cost
  (ceil (/ (originalLength ?x ?y) 5)))
```

to express that each transposition of a condensed genome stands for some number of transpositions of length at most 5 in the original setting. Here `(ceil n)` returns the nearest integer greater than or equal to `n`.

We have experimented with two sets of data: one consisting of *Metazoan* mtDNAs as in (Blanchette, Kunisawa, & Sankoff 1999) and the other consisting of *Campanulaceae* cpDNAs as in (Cosner *et al.* 2000).

For instance, suppose that the initial state describes the mitochondrial genome of Human:

$$(1, -32, 17, 2, 23, 12, 3, 20, 6, 30, 7, 8, 21, 31, 24, 9, -10, -18, 11, 33, -28, 19, 14, 34, 13, 25, 4, 22, -29, 26, 5, 35, -15, -27, -16, -36)$$

and the goal state describes the mitochondrial genome of *Drosophila yakuba* (an insect):

$$(1, 25, 2, 23, 17, 12, 3, 20, 6, 15, 30, 27, 31, 18, -19, -9, -21, -8, -7, 33, -28, 10, 11, 32, -4, -24, -13, -34, -14, 22, -29, 26, 5, 35, -16, -36).$$

Suppose that the costs `cti`, `ct`, `ci` of a transversion, a transposition, an inversion are 1, 2, 3 respectively, and the maximum numbers `mti`, `mt`, `mi` of transversions, transpositions, inversions are all 30. Then, with the maximum plan length `k` set to 35, and the maximum plan cost `c` set to 70, TLPLAN computes the following 14-step plan

```
(transpose 17 17 23)
(transvert -15 -15 6)
(transvert -18 -18 31)
(transvert -27 -27 30)
(transvert -32 -32 11)
(transpose 25 25 1)
(transvert -10 -10 -28)
(transvert 19 19 18)
(transvert 4 4 32)
(transvert 24 24 -4)
(transvert 14 13 -24)
(transpose 33 10 9)
(transvert 7 21 9)
(invert 9 9)
```

in less than 2 seconds with a depth-best-first search strategy. Here TLPLAN examines 36 world states, and 21 of them are pruned by temporal search control.

For each pair of *Metazoan* mitochondrial genomes, we have computed a small number of events using TLPLAN, and constructed a distance matrix. After that, we have used the distance matrix programs FITCH and NEIGHBOR (available with PHYLIP[3] (Felsenstein 2004)) to construct phylogenies. For instance, with the settings above, according

[3] http://evolution.genetics.washington.edu/phylip.html .

to the phylogeny constructed by the program NEIGHBOR, chordates (e.g., Human) and echinoderms (e.g., sea star) are grouped together, and molluscs (e.g., snail) and annelids (e.g., segmented worms) are grouped together; these results conform with the most widely accepted view of metazoan systematics (Metazoa Systematics Page).

Similarly, for the the chloroplast genomes of *Campanulaceae*, after constructing a distance matrix based on the results found by TLPLAN, we have computed a phylogeny with NEIGHBOR. The groupings of chloroplast genomes in this phylogeny are identical to the ones in the consensus tree presented in Figure 4 of (Cosner *et al.* 2000).

We have also experimented with 100 random genome rearrangement problems generated for genomes with 120 genes (similar to the chloroplast genomes above), that involve 3 transpositions, 3 inversions, and 3 transversions. We computed solutions using TLPLAN for `k`=18..21, `c`=`mt`=`mi`=`mti`=20, `ci`=`ct`=`cti`=1, with equal weights of events. The only other system that can handle all three kinds of events above is DERANGE 2, so we also computed solutions using it. For 51 problems, the solutions computed by TLPLAN are more parsimonious (include fewer events) and are closer to the true solution. DERANGE 2 performs better in 33 of the problems and equally in 16.

We have tried to solve a simpler version of the genome rearrangement problem (with only transpositions of size 1 and 2) with a satisfiability planning approach (Kautz & Selman 1992) (using the planner SATPLAN[4]), with a heuristic-search planning approach (Bonet & Geffner 2001) (using the planner FF[5]), and with an answer set planning approach (Lifschitz 1999) (using the answer set solver SMODELS[6]). We have observed that TLPLAN performs better in terms of computation time and space.

Conclusion

We have proposed to solve genome rearrangement problems as planning problems, possibly using some heuristics guiding the search. In our experiments with *Metazoan* mitochondrial genomes and *Campanulaceae* chloroplast genomes, we observed that although we do not put tight constraints on the number of events, useful heuristics guide TLPLAN to compute "good" plans, and thus the phylogenies constructed using these plans conform with the most widely accepted ones. In our experiments with random genome rearrangement problems, we observed that as these constraints get tighter, TLPLAN computes more accurate solutions. TLPLAN was used mainly because it allows us to formalize temporal search control as well as atemporal heuristics, and to put constraints on the number of actions as well as their costs; the results above support this choice.

Acknowledgments

Thanks to Fahiem Bacchus for helpful suggestions on the use of TLPLAN, to Jijun Tang for his help in using GRAPPA to condense the given genomes, and to Li-San Wang for providing us his program to generate random genome rearrangement problems.

[4]`http://www.cs.washington.edu/homes/kautz/satplan/` .

[5]`http://www.mpi-sb.mpg.de/~hoffmann/2002.html` .

[6]`http://www.tcs.hut.fi/Software/smodels/` .

References

Bacchus, F., and Kabanza, F. 2000. Using temporal logic to express search control knowledge for planning. *Artificial Intelligence* 116(1–2):123–191.

Bafna, V., and Pevzner, P. 1998. Sorting by transpositions. *SIAM Journal of Discrete Mathematics* 11:224–240.

Baral, C.; Kreinovich, V.; and Trejo, R. 2001. Computational complexity of planning with temporal goals. In *Proc. of IJCAI*, 509–514.

Blanchette, M.; Kunisawa, T.; and Sankoff, D. 1996. Parametric genome rearrangement. *Gene-Combis* 172:11–17.

Blanchette, M.; Kunisawa, T.; and Sankoff, D. 1999. Gene order breakpoint evidence in animal mitochondrial phylogeny. *Journal of Molecular Evolution* 49:193–203.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1–2).

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69(161–204).

Cosner, M.; Jansen, R.; Moret, B.; Raubeson, L.; Wang, L.; Warnow, T.; and Wyman, S. 2000. An empirical comparison of phylogenetic methods on chloroplast gene order data in Campanulaceae. In Sankoff, D., and Nadeau, J., eds., *Comparative Genomics*. Kluwer Acad. Pub. 99–122.

Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1995. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence* 76:75–88.

Felsenstein, J. 2004. PHYLIP (phylogeny inference package) version 3.6. Distributed by the author.

Hannenhalli, S., and Pevzner, P. 1995. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations with reversals). In *Proc. of STOC*, 178–189.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *Proc. of ECAI*, 359–363.

Lifschitz, V. 1999. Action languages, answer sets and planning. In *The Logic Programming Paradigm: a 25-Year Perspective*. Springer Verlag. 357–373.

Metazoa Systematics Page, The University of California Museum of Paleontology (`http://www.ucmp.berkeley.edu/help/index/metazoa.html`).

Moret, B.; Wyman, S.; D.Bader; Warnow, T.; and Yan, M. 2001. A new implementation and detailed study of breakpoint analysis. In *Proc. of PSB*, 583–594.

Pednault, E. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. of KR*, 324–332.

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