Computing Genome Edit Distances using Domain-Independent Planning

P@trik Haslum
Australian National University
`firstname.lastname@anu.edu.au`

Abstract

The use of planning for computing genome edit distances was suggested by Erdem and Tillier in 2005, but to date there has been no study of how well domain-independent planners are able to solve this problem. This paper reports on experiments with several PDDL formulations of the problem, using several state-of-the-art planners. The main observations are, first, that the problem formulation that is easiest for planners to deal with is not the obvious one, and, second, that plan quality – in particular consistent and assured plan quality – remains the biggest challenge.

Introduction

For several decades now, the comparative study of biological sequence data (i.e., DNA or protein amino acid sequences) has played an increaing role in determining the evolutionary history of life on Earth (e.g., Page and Holmes 1998). While early studies examined differences in the nucleotide sequences of individual genes, more recent work has also examined differences in the arrangement of genes across a whole genome (Sankoff et al. 1992; Boore and Brown 1998; Snel, Huynen, and Dutilh 2005).

A computational problem that arises in this context is the calculation of edit distances between sequences: that is, the number of "mutation events" required to transform one sequence into another. The edit operations considered depends on the type of sequence. For DNA sequences, they are typically insertion, deletion or substitution of individual nucleotides. Minimising this edit distance is the famous sequence alignment problem. For genome rearrangement, the edit operations usually assumed are inversions and transpositions (and combined transposition and inversion, called transversions) of subsequences within the genome, plus insertion/duplication and deletion if needed to account for differences in gene content.

Erdem and Tillier (2005) suggested that the calculation of genome edit distance under these operations can be considered as a planning problem: The arrangement of genes is the state, and each edit operation is an action that modifies it. These actions may furthermore be assigned different costs, to account for different relative frequencies with which they are assumed to occur in the organisms studied. They developed a solution to the problem based on TLPlan, which was later refined by Uras and Erdem (2010).

However, the system developed by Erdem and colleagues is a plain depth-first state-space search guided by entirely domain-specific heuristics. Although the problem is formulated in a planning language, the solver makes essentially no use of this. Consequently, there has been no comprehensive study of how well state-of-the-art domain-independent planners are able to solve the genome edit distance problem.[1]

In this paper, I discuss alternative encodings of the genome edit distance problem in PDDL, and evaluate the ability of some current domain-independent planners to solve the resulting formulations. My aim is not to show that domain-independent planners are better than Erdem et al.'s system (quite unlikely, given the extent to which it is tailored to the domain), but to find out exactly how well – or badly – such planners perform on this problem, and what shortcommings of these planners may need to be addressed to improve their usefulness.

Phylogenetic Reconstruction and Genome Edit Distance

A phylogeny (evolutionary tree) is a tree where the leaf nodes represent living taxa (species or species groups) and interior nodes the ancestral organisms from which they have evolved. Typically, these ancestors are long extinct. The problem of phylogenetic reconstruction is to find the best, i.e., most plausible, such tree, based on observed characters of the living taxa. In a genome-based phylogenetic analysis, these characters are the content and arrangement of genes in the organisms genomes. What constitutes a "most plausible" phylogeny depends on what criteria are assumed. A common criterion is parismony, i.e., preferring trees with the smallest total amount of change. Criteria such as maximum likelihood can also be used, but depend on additional assumptions about the evolutionary process.

Finding a maximum parsimony phylogeny based on genome rearrangement events is computationally hard. Distance-based methods simplify the problem by divding it into two steps: The first is to compute a matrix of pair-wise distances between the taxa, and the second to construct a tree with minimum total distance. Optimal tree construction

[1] Erdem and Tillier report that they formulated a highly simplified version of the problem in PDDL, and had little success solving it using HSP and SATPLAN.

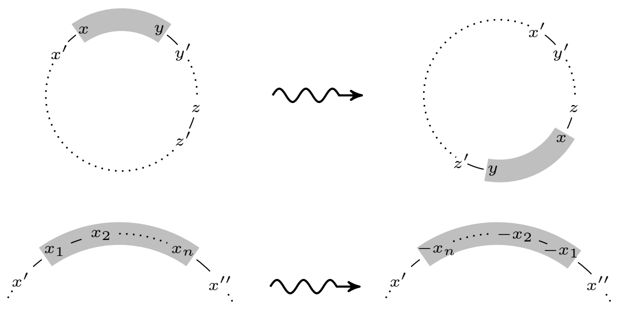


Figure 1: Edit operations on a circular genome. Above: Transposition of the x–y segment to after z. Below: Inversion of the x_1–x_n segment.

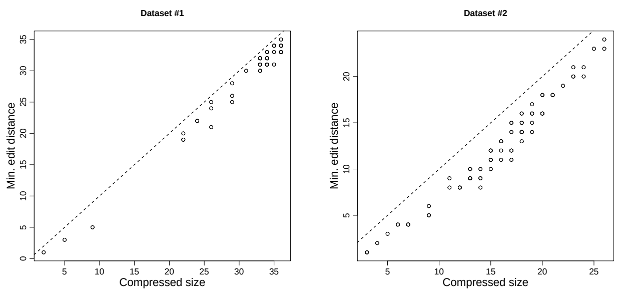


Figure 2: Comparison between compressed genome size and (weighted) edit distance. Note that this is the smallest known edit distance, and does not necessarily reflect the true minimum.

is still hard, but there are widely used and seemingly good approximate methods (e.g., Saitou and Nei 1987). Computing the entires in the distance matrix in this setting gives rise to the genome edit distance problem. Note that the distance-based tree construction is an approximation: even if distances correspond to optimal rearrangement sequences, a minimum distance tree is not necessarily a most parsimonious tree.

Genome Edit Distance

A genome is a linear or circular sequence of genes. In addition to their order, each gene has an orientation ("normal" or "inverted", relative to the, arbitrarily chosen, direction of the sequence). The definition of an edit distance measure requires defining the edit operations, and their relative weights. For comparison of genomes with equal gene content, i.e., which differ only in the order and orientation of their genes, the usual edit operations are inversion and transposition, and the combination of both, called transversion. Transposition moves a segment of the genome (which may consist of a single gene) to a different location, while inversion reverses a segment and reverses the orientation of each gene within it. The two operations are schematically illustrated in figure 1. Transversion simultaneously inverts a segment and moves it to a new location. For comparing genomes with unequal content, operations such as insertion, duplication and deletion of genes must also be used. The inversion-only distance can be computed in polynomial time (Hannenhalli and Pevzner 1995) but no polynomial time algorithm is known for the larger set of edit operations. The problem is conjectured to be NP-hard.

That edit operations are not equally frequent is reflected by giving them different weight in the distance calculation. The relative frequency of apparent occurrence of transposition and inversion is not known precisely, and varies between different species groups. Blanchette et al. (1996) suggest a relative weight of 2–2.5 for transpositions and transversions to 1 for inversions. This weight range is where the number of operations in an approximately minimum weight transformation between one pair of mitochondrial genomes (human and a *Drosophila*) diverges from the number between random sequences.

Although differences that can be ascribed to inversions and transpositions can be observed in the genomes of related organisms, the mechanisms that cause these changes are not well understood. Some studies have suggested that the apparent transpositions and inversions are in fact the result of duplications followed by gene loss (Lavrov, Boore, and Brown 2002). Thus, there are reasons to experiment with different sets of edit operations, and different weighting schemes. The generality of domain-independent planning can offer an advantage in that.

Problem Simplification

Uras and Erdem (2010) describe two simplifications that they apply to problem instances before attempting to solve them. The first is only relevant when comparing genomes with unequal content, and is to remove from both genomes being compared any gene that appears only in one of them, and adding the number of such deletions to the edit cost. (Each such gene must obviously be inserted/deleted, which can be done at the beginning/end of the plan.)

The second is to "compress" both genomes by replacing common substrings with (new) atomic symbols, since there is no reason to apply any edit operation that breaks up such a substring. Note that common substrings may appear inverted in one of the genomes. Compression is done pairwise, per problem, so the substrings need not be common to all genomes in the data set. This simplification turns out to be very important for performance, since it can drastically reduce the size of the problem. Genomes in the second data set (cloroplast genomes of 13 plants from the *Campanulaceae* family) contain 105 genes each, but after compression no pair has more than 26 elements (genes or substrings). Size after compression also turns out to be a fairly good predictor of the edit distance: figure 2 shows a comparison. (This was also noted by Nadeau & Taylor 1984.) In data set #1 (mitochondrial genomes of 11 animal species), which generally has a much higher degree of rearrangement between genome pairs, the effect of compression is much less dramatic.

Formulation in PDDL

In Erdem and Tillier's formulation, the arrangement of genes is described by a binary predicate, (cw ?x ?y), with the meaning that ?y is the next gene from ?x in clockwise order. I will call this the "relational" encoding. The alternative is a "positional" encoding, i.e., to specify the position of each gene w.r.t. a fixed frame of reference, for example by a predicate (at ?x ?p) meaning gene ?x is at position ?p. The orientation of each individual gene is additionally specified by one of the predicates (normal ?x) and (inverted ?x) being true. Both encodings have their advantages and disadvantages.

As noted above, correct and precise relative weights for transpositions and inversions are not known. What is important when formulating edit distance computation as a planning problem is allowing for the possibility of specifying different weights, and generating plans that minimise the weighted distance.

Formulations Based on the Relational Encoding

It is easy to think of each edit operation as an action, but these actions are not so easy to formulate in PDDL, because they have complex preconditions and/or effects. Consider, for example, inversion: the segment to be inverted can be defined by the two genes at its ends, but the operation will have an effect also on every gene between them. The precondition of transposition must enforce that the new location (gene z in figure 1) does not lie within the segment that is moved (i.e., not between x and y). Since the relational encoding specifies only the "neighbours" in the genome, this "betweeness" is a transitive closure property. There are (at least) three ways to formulate the operations:

1. Use PDDL2.2's derived predicates and axioms to define the between and not-between properties, and use quantified conditional effects to encode the effect of inversion. This is, essentially, the formulation used in Erdem et al.'s TLPlan-based system. Note that the recursive derived predicates used in this formulation cannot be expressed in ADL, which allows only first-order pre- and effect conditions.[2]
2. Break each operation up into a sequence of actions, each of which affects only a fixed-size genome part (e.g., only two neighbouring genes). This requires additional predicates to control the sequencing of these actions so that they actually correspond to complete and correct edit operations.
3. Use a separate action for each size of segment operated on, with a matching number of arguments (for example, an action (invert-3 ?x-pre ?x ?y ?z ?z-post) for inverting the segment ?x–?y–?z of length 3).

The third option is used in Erdem & Tillier's simplified PDDL formulation, which permits only operations on segments of limited size. For the general problem, it becomes infeasible because of the very large number of action parameters, which make grounding impossible. (Grounding, which is used by nearly all modern domain-independent planners, is a significant obstacle for some other formulations as well, as discussed below.)

Single-Step Formulation In the single-step formulation, each edit operation is performed by one action, using derived predicates and quantified conditional effects to specify actions' preconditions and effects. Formulating axioms for between is straightforward. For example,

```
(:derived (between ?x ?y ?z) (= ?z ?x))
(:derived (between ?x ?y ?z) (= ?z ?y))
(:derived (between ?x ?y ?z)
          (exists (?w) (and (cw ?x ?w) (not (= ?y ?w))
                            (between ?w ?y ?z)))))
```

i.e., ?z is between ?x and ?y, inclusive, iff ?z equals either ?x or ?y, or the next gene ?w clockwise from ?x is not equal to ?y and ?z is between ?w and ?y. (A similar definition can be written for not-between, so it is not necessary to use negation over the between predicate.) The effect of reversing the cw relation in the segment between ?x and ?y (an effect of the inversion operation) can then be written as:

```
(forall (?v ?w)
  (when (and (between ?x ?y ?v) (between ?x ?y ?w)
             (cw ?v ?w))
        (and (not (cw ?v ?w)) (cw ?w ?v))))
```

Most modern domain-independent planners work (internally) on a grounded representation, and this is a major obstacle to using this formulation. The transposition operation involves six distinct genes whose neighbour relations will change, and thus the transpose action has six arguments. Apart from a few disequalities (e.g., $z \neq x'$; cf. figure 1), all possible instantiations lead to potentially applicable actions, which makes the number of ground actions huge. No problem with genomes containing more than 7 elements could be grounded. Note that the issue here is not the (in-)efficiency of the grounding process, but the size of the grounded problem. Thus, advanced grounding techniques, such as combined grounding and relaxed reachability analysis (Helmert 2009), will not help.

However, since there are at most binary predicates, the effect of an operation can be divided into a series of steps each of which can be performed by an action with only two parameters (e.g., for transposition: break x'–x; break y–y'; connect x'–y'; etc). Part of the inversion operation must still be written using quantified conditional effects, but there are only a quadratic number of instances, each with an at most quadratic number of conditional effects after grounding the quantifiers. In this formulation, all problems (up to size 26) can be grounded effectively.

Multi-Step Formulation The use of derived predicates and conditional effects can be avoided by "simulating" their effects through sequences of actions.[3] This can be done in several ways: the following is just one alternative.

The edit operations can all be viewed as first cutting out a segment of the genome and then re-inserting it somewhere else (transposition), or inserting it reversed in the same place

[2]Because TLPlan, which uses no domain-independent reasoning, uses action descriptions only to generate successor states in a "black box" maner, it has a very expressive input language which includes, for example, procedurally defined (recursive) functions.

[3]This idea is also used in compilations that remove these features (cf. Nebel 2000, and Thiebaux, Hoffmann, and Nebel 2003). Note that these compilations do not preserve plan length, which is why zero-cost actions become necessary to model the relative weight of edit operations in this formulation.

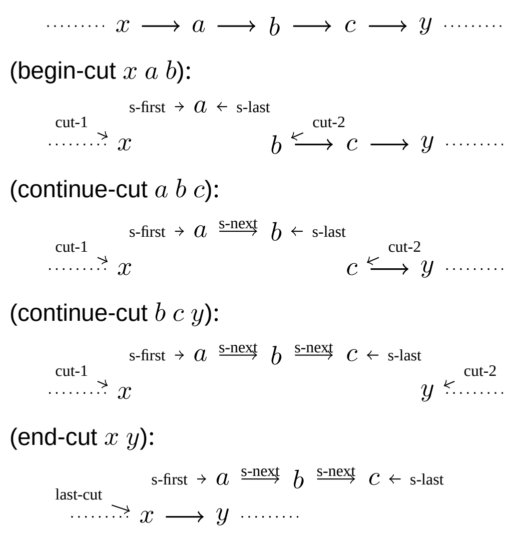


Figure 3: Cutting segment a–b–c. The unlabelled arrows represent the cw relation.

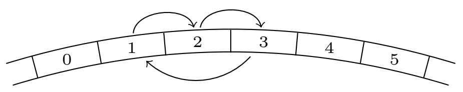
(inversion) or elsewhere (transversion). Cutting, inserting and inserting-in-reverse a segment can all be done in a gene-by-gene fashion. Figure 3 illustrates how a segment of length 3 is cut out. Sequencing of the step actions is controlled by auxiliary predicates: for example, the precondition of action (continue-cut ?x ?y ?z) requires that (s-last ?x), (cut-2 ?y) and (cw ?y ?z) hold. At the end of the operation, the end points and sequence of the cut segment are identified by these auxiliary predicates, so that this information can be used to control insertion. Also marked is the point in the genome where the cut was made: this permits to distinguish inversion, which is a reversed insertion at that point, from transversion, and thus to assign them different weights.

Expressed in a "natural" way, this formulation has actions with three parameters, which is still enough to make grounding large problems difficult. Through further splitting of the steps it can be brought down to two parameters, allowing all problems (up to size 26) to be grounded effectively.

In this formulation, the number of actions in a single edit operation varies with the length of the transposed or inverted segment (this is in contrast to the split version of the single-step formulation described above). Thus, to accurately encode relative weights (not dependent on segment length) it is necessary to assign some actions zero cost.

Formulations Based on the Positional Encoding

In the positional encoding, all complex relations are over positions. For example, transposing the segment at positions 1–2 to after the gene at position 3 (assuming the segment inbetween slides counter-clockwise) results in a permutation moving the gene at position 1 to position 2, the gene at 2 to 3, and the gene at 3 to position 1:



This permutation is the same regardless of which genes occupy the affected positions. Thus, for a given genome size, the permutations caused by edit operations can be computed in advance and provided to the planner through static predicates. For example, the effects of transposing the segment ?x–?y to ?z can be written

```
(forall (?g - gene ?v ?w - pos)
  (when (and (transpose-shift ?x ?y ?z ?v ?w) (at ?g ?v))
        (and (not (at ?g ?v)) (at ?g ?w))))
```

where (transpose-shift ?x ?y ?z ?v ?w) is the static predicate that specifies the permutation, i.e., that the transposition moves the gene at position ?v to position ?w. In the above example, we would have (transpose-shift p1 p2 p3 p1 p2), (transpose-shift p1 p2 p3 p2 p3), and (transpose-shift p1 p2 p3 p3 p1). (The predicate is false for all positions not involved in the move.) This allows a single-action-per-edit-operation domain to be written without derived predicates, and using actions with no more than three parameters. Grounding this formulation is still challenging, but here the size of the grounded problem is moderate (although the number of ground actions is roughly cubic, most do not affect a large part of the genome and so have a relatively modest number of conditional effects). Thus more efficient grounding techniques may make it practical.

It is also worth noting that the formulation could be made more compact by making use of the recent addition of "object fluents" to PDDL (Helmert, Do, and Refanidis 2008), which model multi-valued state variables. For example, the effect of the transposition above could be written

```
(forall (?v ?w - pos)
  (when (transpose-shift ?x ?y ?z ?v ?w)
        (assign (gene-at ?w) (gene-at ?v))))
```

i.e., without quantifying over the possible content of each reassigned position. This would reduce the number of conditional effects to linear. However, there is, to my knowledge, no planner that natively supports effects of this kind. (Fast Downward, and planners derived from it, such as LAMA, internally use a format based on grounded multi-valued state variables, but allow only constants on the left-hand side of assignements.)

The positional encoding has another significant drawback when applied to circular genomes, in that it introduces an arbitrary fixed reference point. Thus, the fact that two circular arrangements may be equal but placed differently w.r.t. this reference point must be taken into account. This can be done by introducing a "rotate" action, which shifts the whole genome relative to the reference point without changing the arrangement. Applications of this action do not count towards the edit distance, i.e., it must have zero cost.

Genome Data Sets

Experiments were done on two data sets.[4]. The same data sets were used by Erdem and Tillier (2005).

Data set #1 comprises the mitochondrial genomes of 11 species of the animal kingdom. They are a diverse collection, with one to three exemplars from each of six major

[4]Obtained from `http://grimm.ucsd.edu/MGR/examples.html`

Formulation		Max Arity	# Grnd	# Sol
		LAMA		
Relational				
Single-step	(R1)	6	18 (7)	18 (7)
Single-step, partly split	(R1′)	4	34 (12)	28 (11)
Single-step, fully split	(R1″)	2	156 (26)	38 (15)
Multi-step	(Rm)	3	156 (26)	156 (26)
Multi-step, split	(Rm')	2	156 (26)	156 (26)
Positional				
Single-step	(P1)	3	122 (19)	40 (15)
		Mp		
Relational				
Multi-step	(Rm)	3	156 (26)	36 (14)
Multi-step, split	(Rm')	2	156 (26)	45 (14)
Positional				
Single-step	(P1)	3	34 (12)	25 (11)

Table 1: Comparison of problem formulations on data set #2. “Max Arity” is the largest number of action parameters. “# Grnd” is the number of instances (out of 156) that could be grounded (and translated to SAS+) within 2Gb memory. In parenthesis, the largest problem size that could be grounded. “# Sol” is the number of instances solved, within a 30 minute time and 2Gb memory limit. In parenthesis, the largest problem size that was solved. Mp was run with a higher memory limit, 3Gb, but a 30 minute time limit for grounding and solving combined. However, the time for grounding and preprocessing is negligeable; when grounding fails it is by exhausting memory.

groups: chordates, echinoderms, arthropods, mollusks, annelids and nematodes. (Each of these groups is believed to be monophyletic.) Blanchette et al. (1999) used this data to investigate gene order evidence for different theories about the evolutionary relationship between these groups. Most mitochondiral genomes have 37 genes, without duplicates. However, one gene is missing from all nematodes, so only the arrangement of the remaining 36 genes is compared. Genomes of species from different groups generally show a high degree of rearrangement, and thus compress poorly, while within some groups the genomes are very similar, resulting in small problems after compression. Thus, in this set there is an uneven spread of problem sizes (cf. figure 2).

Data set #2 comprises the chloroplast genomes of 13 plants from the *Campanulaceae* family. It was used by Cosner et al. (2000) to compare different phylogenetic analysis methods. The genomes contain 105 different genes, some duplicated. Cosner et al. removed duplicate genes, so that each genome has a length of 105. There are large segments common to all genomes in the set, and therefore compression of common substrings reduces the size of problems significantly, to an even spread between 3 and 26.

Experiments

The experiments aim to determine which of the different formulations planners find easiest to deal with, and generally evaluate the ability of some domain-independent planners to compute genome edit distances. Note that this means ability to generate solutions of consistent quality. The purpose of computing edit distances is to *compare* them (i.e, organism A is assumed to be more closely related to organism B than to C if the edit distance between the genomes of A and B is smaller than the distance between A and C.) This does not mean it is necessary to find optimal plans: as long as the relative differences between the distances computed for different pairs is the same as between the minimal edit distances between the same pairs, any information contained in the true edit distances is not lost. If, however, the cost of the plans produced by a planner can differ significantly and unpredictably from the optimal cost, to the extent that this “random” difference overshadows any relation between distances between different pairs, the results are of no use.

Comparison of Formulations Table 1 summarises the comparison of problem formulations. This was done on data set #2, because it shows a fairly even spread of problem sizes. To the extent of my knowledge, the only planner capable of handling all formulations, and trying to minimise plan cost, is LAMA (Richter and Westphal 2010). Thus, the difficulty of solving each formulation is estimated by how many, and how large, problems this planner solves. As mentioned, grounding is major obstactle for problem formulations that use actions with a large number of parameters. It is, however, clearly not the only source of difficulty: in all non-STRIPS formulations, except relational single-step, some problems that could be grounded were not solved. To get some idea of how much these results are specific to LAMA, I also ran the Mp planner (a heuristically enhanced SAT-based planner; cf. Rintanen 2011). Mp ignores plan quality, and thus is not really suited to calculating the weighted edit distance, but its ability to find any solution can still be used as a measure of problem difficulty. As it does not support derived predicates, Mp could only be tested on three formulations, but on these it mostly agrees with LAMA.

Comparison of Edit Distances Figure 4 shows a comparison of the edit distances computed by different planners, as well the inversion-only distance computed by a domain-specific system, GRIMM.[5] The planners are LAMA, Mp and several variants of greedy best-first search with the FF heuristic (Fast Downward implementation). The first two variants use a cost-sensitive version of the heuristic, i.e., one that estimates the true cost of the relaxed plan, and the standard unit-cost version, which estimates the size of the relaxed plan, in a standard, single GBFS search. The third (“cGBFS/FF”) uses the unit-cost heuristic, but continues to search after the first solution has been found, for a plan of lower real cost. In the final variant (“GBFS/FF+RR”), some decisions that are normally made arbitrarily (order of successors and the choice of minimum-cost supporting action in the heuristic) are randomised, and the search repeated as many times as possible within the time limit, keeping the best plan found.

The smallest known distance is found by combining the results of all planners, exploiting the symmetry of the edit

[5] `http://grimm.ucsd.edu/GRIMM/`

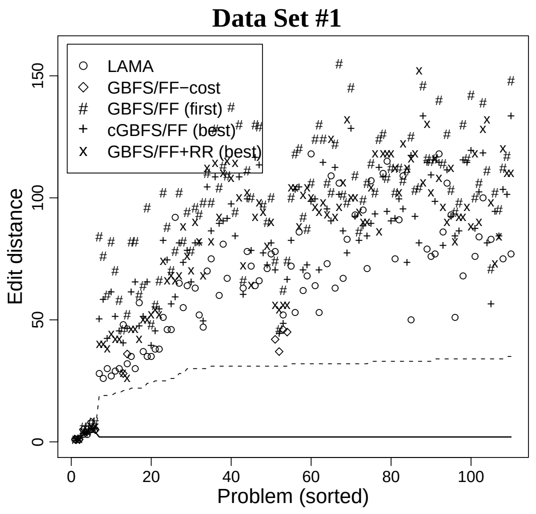


Figure 4: Comparison of edit distances found by different planners on the relational multi-step split formulation (Rm'). Problems are sorted by the minimum (known) distance (shown as a dashed line); this includes the inversion-only distance obtained from GRIMM. The solid line shows the highest lower bound proven by A*/LM-Cut. For data set #2, the lower graph shows a close up of the region below distance 50, for better vertical resolution. This graph also includes the distances computed by Erdem & Tillier's (2005) system ($\cdots$). Note, however, that those were computed with transversion weighted at 1, instead of 2; the corresponding distance under the weighting used here may be anywhere in $[1, 2]$ times that shown in the graph.

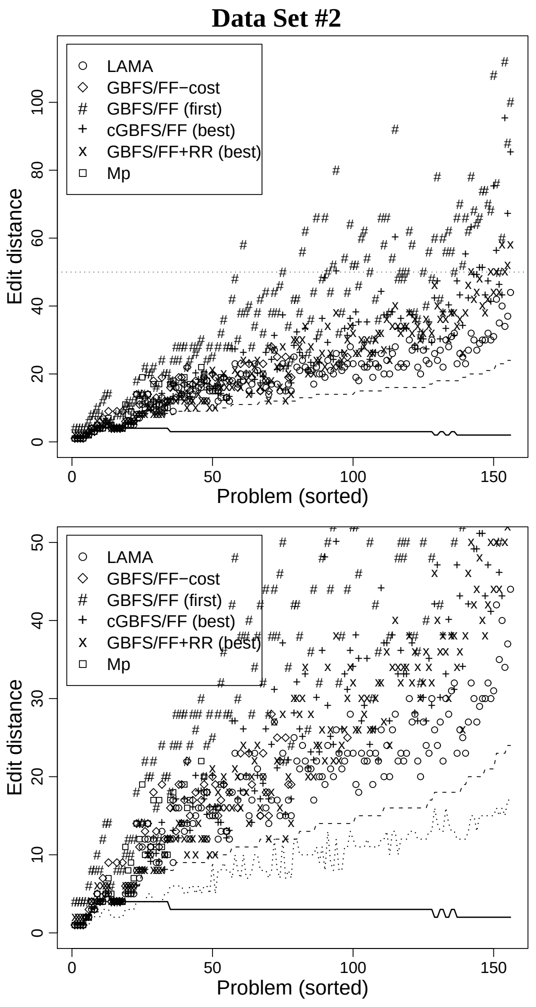


Figure 4 (continued).

distance (the minimum cost of transforming genome g into g' is the same as of transforming g' into g; both problems are solved), and the inversion-only distance computed by GRIMM. (Since a weight of 1 for inversions in used in the planning formulation, the unweighted inversion-only distance is a valid upper bound on optimal plan cost.) For most problems, the latter is clearly smaller, but there are a few instances in which a planner finds a lower-cost plan. For data set #2 the comparison also includes the distances computed by Erdem & Tillier's TLPlan-based system. These distances, however, were computed with transversion weighted at 1, and thus are not directly comparable. (Note that this distance is sometimes below the lower bound!) The corresponding distance under the weighting used here may be anywhere in $[1, 2]$ times that shown in the graph. For data set #1, the weights used by Erdem & Tillier differ too much from those used here to allow for any meaningful comparison. Finally, the highest f-value proven by A* search using the LM-Cut heuristic is provided as a lower bound.

Three important observations can be made: First, the cost-ignorant planners exhibit a very large spread in plan costs, relative to the smallest known. This makes them unusable for computing edit distances, since the noise that is created by the planners is clearly enough to drown out any signal present in the true minimum distances. Second, using the cost-sensitive FF heuristic results in plans of much better quality, but is much less effective; it solves only 30% of instances overall (whereas with the unit-cost FF heuristic the planner solves all but one problem). LAMA, and the two iterated variants of GBFS with the unit-cost FF heuristic, achieve more consistent quality, while still solving most instances (90% for LAMA). Even so, the difference between the lowest cost plan found by any of these planners and the smallest known edit distance is of the same magnitude as the edit distance itself, which is clearly too much. Finally, there is an enourmous gap between the smallest known edit distance and the lower bound. This is a significant problem, since it means there is no way to tell how close to the true minimum these distances are. In other words, we cannot know for sure if the smallest known edit distances equal the true minimum, or if they are just noise.

Conclusions

Computing minimum, or at least consistent, genome edit edistances is clearly a challenging problem for current domain-independent planners. The "obvious" formulation (relational single-step) of the problem requires advanced features of PDDL (e.g., conditional effects with derived predicates in effect conditions), which are not dealt with effectively even by the very few planners that support them. Furthermore, all but very small instances this formulation cannot be grounded, which rules out all modern domain-independent planners. However, these problems can be overcome by formulating the problem in a way that, while perhaps appearing "unnatural", is better suited to the planners. This formulation needs only STRIPS with action costs.

Finding plans of consistent (high) quality, while scaling up to large problem instances, remains a challenge. It is interesting to note that the time taken by GBFS with the unit-cost FF heuristic to solve even the largest problems is no more than around 15 minutes. How to make use of that efficiency to find high-quality plans is an important question for future planning research, not only for its application to the genome edit distance problem. (The simple "randomise and repeat" scheme tried here is not a good enough answer.) Most important, however, is the lack of a sufficiently strong lower bound. As long as the gap between the lower bound and the smallest known distance is as large as the distance itself, we cannot be confident that edit distances computed by planners reflect the truth.

Finally, with these results in hand, we may ask: would biologists, interested in computing genome edit distances, want to use domain-independent planners? The promise of domain-independent planning in this application is its generality, providing a means to explore different sets of edit operations without the need to develop new, customised heuristics or special-purpose algorithms for each set. However, there are clearly hurdles to be overcome before this promise can be realised. Formulating the problem such that current planners can effectively solve it requires insight into the workings of the planners, and may even be more difficult for a non-specialist than developing a problem-specific search heuristic. Better methods of finding high-quality plans, and of finding lower bounds to provide assurance of the quality of those plans, are also needed.

Acknowledgements I thank Esra Erdem for kindly providing details about her formalisation of the genome rearrangement problem, and data on experiments with it. This work was supported by ARC project DP0985532.

References

Blanchette, M.; Kunisawa, T.; and Sankoff, D. 1996. Parametric genome rearrangement. *Gene* 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.

Boore, J., and Brown, W. 1998. Big trees from little genomes: mitochondrial gene order as a phylogenetic tool. *Current Opinion in Genetics & Development* 8:668–674.

Cosner, M.; Jansen, R.; Moret, B.; Raubeson, L.; Wang, L.-S.; Warnow, T.; and Wyman, S. 2000. An empirical comparison of phylogenetic methods on chloroplast gene order data in campanulaceae. In *Comparative Genomics*, 99–121. Kluwer Acad. Publ., Montreal, Canada.

Erdem, E., and Tillier, E. 2005. Genome rearrangement and planning. In *Proc. 20th National Conference on AI (AAAI'05)*.

Hannenhalli, S., and Pevzner, P. 1995. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In *Proc. 27th ACM-SIAM Symposium on the Theory of Computing (STOC'95)*, 178–189.

Helmert, M.; Do, M.; and Refanidis, I. 2008. Changes in PDDL 3.1. `http://ipc.informatik.uni-freiburg.de/PddlExtension`.

Helmert, M. 2009. Concise finite-domain representation for PDDL planning tasks. *Artificial Intelligence* 173:503–535.

Lavrov, D.; Boore, J.; and Brown, W. 2002. Complete mtDNA sequences of two millipedes suggest a new model for mitochondrial gene rearrangements: Duplication and nonrandom loss. *Molecular Biology and Evolution* 19(2):163–169.

Nadeau, J., and Taylor, B. 1984. Lengths of chromosomal segments conserved since divergence of man and mouse. *Proc. National Academy of Sciences USA* 81:814–818.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *Journal of AI Research* 12:271–315.

Page, R., and Holmes, E. 1998. *Molecular Evolution: A Phylogenetic Approach.* Blackwell Science, Oxford.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of AI Research* 39:127–177.

Rintanen, J. 2011. Heuristics for planning with sat and expressive action definitions. In *Proc. 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*.

Saitou, N., and Nei, N. 1987. The neighbour-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution* 4:406–425.

Sankoff, D.; Leduc, G.; Antoine, N.; Paquin, B.; Lang, B.; and Cedergren, R. 1992. Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome. *Proc. National Academy of Sciences USA* 89:6575–6579.

Snel, B.; Huynen, M.; and Dutilh, B. 2005. Genome trees and the nature of genome evolution. *Annual Review of Microbiology* 59:191–209.

Thiebaux, S.; Hoffmann, J.; and Nebel, B. 2003. In defense of PDDL axioms. In *Proc. 18th International Conference on Artificial Intelligence (IJCAI'03)*, 961–968.

Uras, T., and Erdem, E. 2010. Genome rearrangement and planning: Revisited. In *Proc. 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, 250–253.