

# Lower Bounds for Maximum Parsimony with Gene Order Data

Abraham Bachrach, Kevin Chen, Chris Harrelson, Radu Mihaescu,  
Satish Rao, and Apurva Shah

Department of Computer Science  
UC Berkeley

**Abstract.** In this paper, we study lower bound techniques for branch-and-bound algorithms for maximum parsimony, with a focus on gene order data. We give a simple  $O(n^3)$  time dynamic programming algorithm for computing the maximum circular ordering lower bound. The well-known gene order phylogeny program, GRAPPA, currently implements the brute-force exponential time algorithm and the *Swap-as-you-go* heuristic. Our experiments show a significant improvement over both these methods in practice. Next, we show that the linear programming-based lower bound of Tang and Moret can be greatly simplified, allowing us to solve the LP very efficiently. Finally, we formalize the problem of computing the circular ordering lower bound, when the tree topologies are generated bottom-up, as a *Path-Constrained Travelling Salesman Problem*, and give a 3-approximation algorithm for it. This is a special case of the more general *Precedence-Constrained Travelling Salesman Problem* and has not previously been studied, to the best of our knowledge.

## 1 Introduction

Currently, the most accurate methods for phylogenetic reconstruction from gene order data are based on branch-and-bound search for the most parsimonious tree under various distance measures. These include GRAPPA[1], BP-Analysis [2], and the closely-related MGR [3]. Since scoring a particular tree topology is typically a hard computational problem for gene order data, good lower bounds are essential to make these algorithms practical.

There has been a good deal of recent work on designing good lower bounds for various distance measures. These techniques are divided between those specially designed for specific distance measures [4–6] and those that hold for arbitrary metrics [7–9]. Our lower bounds fall into the latter category. Lower bounds that hold for arbitrary metrics are particularly appealing in the context of gene order phylogeny, because an important direction for the field is to extend current methods to use more realistic metrics than the breakpoint or inversion distances currently used. There is a growing body of algorithmic work on various distance measures, including transpositions, chromosome fusions/fissions, insertions/deletions and various combinations of these (see [10] for a comprehensive

survey) and our lower bounds apply to all of them. One notable exception to this is the tandem duplication and random loss model [11], which is well-suited to animal mitochondrial genomes, but is asymmetric and therefore not amenable to the lower bounds presented in this paper.

In this paper, we give efficient implementations of two lower bounds. The first is a simple dynamic programming algorithm to compute the maximum circular ordering lower bound in  $O(n^3)$  time and  $O(n^2)$  space. Since the exact running time of the algorithm depends on the choice of root, we also provide an algorithm to compute the optimal root for a given un-rooted tree topology in  $O(n^2)$  time. Next, we greatly simplify the LP-based lower bound of [9] and show how to implement it in  $O^*(n^3)$  time in the worst case and  $O^*(n^{2.5})$  time amortized over all binary trees. Finally, we study the problem of lower bounding the tree score when the only a partial topology has been constructed so far and rephrase this as a *Path-Constrained Travelling Salesman Problem*. This is a special case of the *Precedence-Constrained Travelling Salesman Problem*[12], in which we are given a partial order graph on a subset of the cities and asked to return a min cost tour that respects the partial ordering. Our version of the problem is simply the case where the partial order graph is a directed path. To our knowledge, this problem has not been previously studied and we give a fast algorithm that computes a 3-approximation for it. The solution can then be transformed into a lower bound by dividing the score by 3. We have implemented our dynamic programming lower bound and show that it gives better results on the benchmark Campanulaceae data set.

## 2 The Circular Ordering Lower Bound

Given a rooted binary tree in which one of each pair of children is designated the left child and the other the right child, consider the left-to-right ordering of the leaves,  $\pi$ , induced by a depth-first search of the tree. For a given metric  $d(\cdot)$  on pairs of leaves, say the inversion distance, we define the circular ordering lower bound to be  $C(\pi) = \sum_{i=1}^n d(\pi(i), \pi(i+1))$ , where we define  $\pi(n+1) = \pi(1)$  for notational convenience. By repeatedly invoking the triangle inequality, it is easy to see that  $\frac{C(\pi)}{2}$  is a lower bound on the cost of the tree, and the bound is tight if the distance  $d(\cdot)$  is the shortest path metric induced by the tree.

The same tree topology can induce more than one leaf ordering  $\pi$  by swapping left child and right child at internal nodes of the tree, and some leaf orderings may produce a higher lower bound than others. A brute-force exponential time algorithm for computing the maximum circular ordering is to enumerate all possible combinations of swaps at internal nodes. This method is implemented in GRAPPA, along with a heuristic, called the *Swap-as-you-go* heuristic [7], in which a DFS traversal of the tree is performed, and a swap performed at each internal node when it is visited, inducing a total of  $O(n)$  permutations  $\pi$ .

In this section, we show that, in fact, the maximum circular ordering for a given tree can be computed in  $O(n^3)$  by a straightforward dynamic programming algorithm. We first note that the root of the tree does not affect the parsimony

score or the maximum circular ordering lower bound. Since the branch-and-bound algorithm searches over un-rooted topologies, for the presentation of the algorithm we will assume an arbitrarily chosen root. On the other hand, the exact running time of our algorithm will depend on the position of the root and we will consider the problem of optimal root placement after giving the description of the algorithm.

At each internal node,  $v$ , let  $S_v$  be the set of leaves in the subtree rooted at  $v$ . The dynamic programming algorithm constructs a table  $M_v$  which contains, for each pair of leaves  $A, B \in S_v$ , the maximum score attainable by a *linear* ordering of the vertices in  $S_v$  that begins with  $A$  and ends with  $B$ , if one exists. Note that such an ordering exists if and only if  $A$  and  $B$  are leaves in subtrees rooted at different children of  $v$ .

Let the children of  $v$  be  $l$  and  $r$ . We inductively assume that the tables  $M(r)$  and  $M(l)$  have already been constructed. Let  $ll$  and  $lr$  be  $l$ 's children and  $rl$  and  $rr$  be  $r$ 's children, and let us assume that the subtrees rooted at  $ll$ ,  $lr$ ,  $rl$  and  $rr$  have  $a, b, c$  and  $d$  leaves respectively. Intuitively, we could construct  $M(v)$  by considering all possible quartets of leaves  $A \in S_{ll}, B \in S_{lr}, C \in S_{rl}$  and  $D \in S_{rr}$ . We will then perform  $O(abcd)$  operations at node  $v$ , which would lead to a running time of  $O(n^4)$  for the whole tree.

We can do better than this naive implementation in the following way. Let  $A \in S_{ll}$  and  $C \in S_{rl}$ . Let

$$\delta(A, C) = \max_{B \in S_{lr}} [M_l(A, B) + d(B, C)]. \quad (1)$$

So  $\delta(A, C)$  is the highest score attainable by a linear ordering of the leaves in  $S_l$  which also takes a final step to  $C$ . Now for  $D \in S_{rr}$  we obtain

$$M_v(A, D) = \max_{C \in S_{rl}} [\delta(A, C) + M_r(C, D)]. \quad (2)$$

The maximum circular ordering lower bound is given at the root by the expression

$$\max_{A \in S_l, D \in S_r} [M_v(A, D) + d(A, D)]$$

## 2.1 Analyzing the running time

To analyze the running time of the algorithm, assume inductively that the time to process an  $n$ -node tree is  $O(n^3)$ . Keeping the notation as before, at a given node  $v$ , by induction, it takes  $O((a+b)^3)$  time to compute the table  $M_l$  and  $O((c+d)^3)$  time to compute  $M_r$ . To complete the computation Equation 1 for every pair  $A \in S_{ll}, C \in S_{rl}$  (there are  $ac$  of them), we need  $O(b)$  time and to complete the computation in Equation 2 for all  $ad$  pairs  $A \in S_{ll}, D \in S_{rr}$  we need  $O(c)$  time per pair. The running time to compute the entries  $M_v(A, D)$  with  $A \in S_{ll}$  and  $D \in S_{rr}$  is therefore  $O(abc + acd)$ . Of course we need to do this for all possible choices of subtrees of  $l$  and  $r$ , therefore the total running time at node  $v$  will be  $O(abc + acd + abd + bcd)$ . The obvious inequality

$$(a + b + c + d)^3 \geq (a + b)^3 + (c + d)^3 + abc + acd + abd + bcd$$

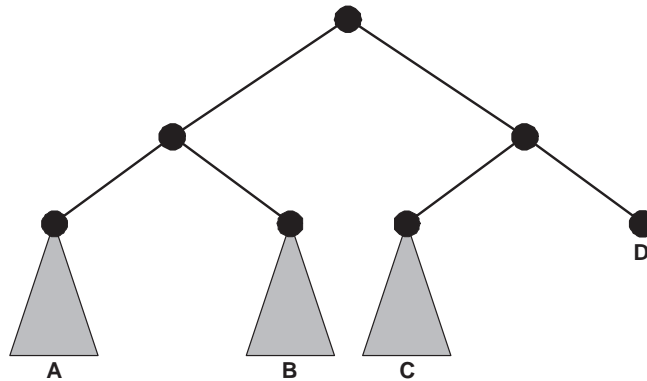
inductively gives a total running time of  $O(n^3)$  for the entire tree on  $n$  leaves, irrespective of the root placement.

Observe that at each step we are keeping one table of size  $O(ab)$  for a node with  $a$  leaves under its left child and  $b$  leaves under its right child. An argument identical to the one above proves that the space needed for building a tree on  $n$  leaves is at most  $O(n^2)$ . This proves the following theorem:

**Theorem 1.** *There exists an  $O(n^3)$  time,  $O(n^2)$  space algorithm for computing the maximum circular ordering lower bound.*

The worst case time and space are achieved by a balanced binary tree. Conversely, the best case for the algorithm is achieved when the tree is a caterpillar. In this case, running time is only  $O(n^2)$ .

Up to now, we have placed the root arbitrarily. However, observe that the choice of the root can affect the running time of the algorithm. For example, consider a rooted tree consisting of a root node with four grandchildren, where three of the four subtrees rooted at the grandchildren are caterpillars with  $n$  leaves (Figure 1) and the fourth one is a singleton leaf.



**Fig. 1.** Each of the shaded triangles  $A, B, C$  represent a caterpillar on  $O(n)$  leaves, while  $D$  is the singleton tree.

The running time for this rooting is  $\Omega(n^3)$ . However, if we root the tree somewhere inside  $C$ , then the running time becomes  $O(n^2)$ .

The optimal choice of root can be found easily in  $O(n^2)$  time simply by considering each of the possible root placements in turn, and traversing the tree in DFS order, starting at each root, to compute the running time that the algorithm would need if that particular rooting were chosen. Since the running time of the algorithm is  $\Omega(n^2)$ , this does not increase the asymptotic running time.

It is easily seen that for a balanced binary tree, one can never do better than  $\Omega(n^3)$  regardless of the rooting, however it is not clear what is the average running time of the algorithm for an optimal rooting across the set of all trees. We leave as an open problem the question of computing the optimal running time in the average case.

### 3 An LP-based Lower Bound

A recent paper by Tang and Moret gives an alternative lower bound for general metrics based on linear programming [9]. In their LP, there is a variable  $x_e$  for each tree edge,  $e$ , which corresponds to the length of the edge, and a variable  $y_{i,j}$  for each pair of non-leaf vertices  $i, j$  at distance 2 in the tree (in terms of edges), corresponding to the distance between these two vertices. If  $i$  and  $j$  are leaves, we define  $y_{i,j} = d(i, j)$ .

The object is to minimize the sum of the  $x_e$  subject to triangle inequality constraints and *perfect median* constraints. There is one perfect median constraint for each internal vertex  $v$ . Let  $i, j, k$  be the three neighbors of this vertex. The sum of the edge lengths of the three edges adjacent to  $v$  is constrained to be equal to  $\frac{y_{i,j} + y_{j,k} + y_{k,i}}{2}$ , which essentially says that the circular ordering lower bound for the local subtree around each internal vertex is tight - hence the term *perfect median*. The motivation for this constraint is an empirical observation that in many practical situations, many of the perfect median constraints are indeed satisfied. In addition to these constraints, there are also two triangle inequality constraints for each pair of leaves  $i, j$ , in which two paths made up of sums of  $x$  and  $y$  variables between the leaves are constrained to be at least the measured distance between the two leaves,  $d(i, j)$  (the choice of the two paths is not deterministically given in the paper).

In this section, we show that this LP is equivalent to the following very simple LP, which has appeared previously in the literature (e.g. [13]). We do away completely with the  $y$  variables and keep only the  $x$  variables. Let  $L$  be the set of leaves of the tree,  $path(i, j)$  be the (unique) shortest path from  $i$  to  $j$  in the tree, and  $d(i, j)$  be the measured distance between  $i$  and  $j$  (e.g.  $d(i, j)$  could be the inversion distance between  $i$  and  $j$ ). The LP, which we call the *triangle inequality LP* is:

$$\begin{aligned} & \min \sum_{e \in E} x_e \\ & \text{subject to} \\ & \forall i, j \in L \quad \sum_{e \in path(i, j)} x_e \geq d_{ij} \\ & x_e \geq 0 \end{aligned}$$

**Theorem 2.** *The LP of Tang and Moret is equivalent to the triangle inequality LP.*

*Proof.* To show equivalence between the two LP's, we need to prove that all constraints in the LP of Tang and Moret that involve  $y$  variables can *always* be satisfied by a solution that satisfies all the constraints that involve only  $x$  variables (i.e. the triangle inequality LP). We consider two cases separately: first, at interior nodes with no neighbors that are leaves, and second, at interior nodes with exactly two neighbors that are leaves. For trees with more than three leaves, there are no other cases to consider.

- Case 1: Given a solution to the triangle inequality LP, at an interior node  $v$  with neighbors  $i, j, k$ , none of which are leaves, we can just set  $y_{ij}$  variable to be the sum of  $x_{\{i,v\}}$  and  $x_{\{j,v\}}$ , and likewise for  $y_{jk}$  and  $y_{ki}$ . This satisfies both the perfect median constraint for this interior node and any triangle inequality constraints involving  $y_{ij}, y_{jk}$  or  $y_{ki}$ .
- Case 2: At an interior node  $v$  with adjacent leaves  $i, j$  and a third adjacent node  $k$ , this may not be possible in general because  $d(i, j)$  could be smaller than the sum of  $x_{\{i,v\}}$  and  $x_{\{j,v\}}$ . But to compensate, we can simply increase the other two  $y$  variables by the appropriate amount in order to satisfy the perfect median constraint, and this will still satisfy the relevant triangle inequality constraints. Note that this would in principle violate local triangle inequality constraints between  $i$  and  $k$  and between  $j$  and  $k$ , but these constraints are not included in the LP of Tang and Moret. If they were included, then the LP would not be satisfiable by the most parsimonious tree, so its use as a lower bound would be in question.

□

Given our simpler LP, we now observe that it has the form of a pure covering LP, for which many efficient approximation schemes exist. In particular, we can apply the algorithm of [14], which solves the LP to within a factor of  $1 + \epsilon$  in time  $O^*(\frac{1}{\epsilon^2}M)$  where  $M$  is the number of non-zero entries in the constraint matrix and the  $O^*(\cdot)$  notation hides polylogarithmic factors. We can always divide the solution by  $1 + \epsilon$  to get a lower bound on the tree score.

$M$  is the sum of the path lengths between all pairs of leaves in the tree and ranges from  $n^2 \log(n)$  for a complete binary tree to  $n^3$  for a caterpillar. In general,  $M$  can be bounded by the product of the number of constraints,  $n^2$ , and the height of the tree. Using a classical result of Flajolet and Odlyzko, which states that the height of a random binary tree is  $O(\sqrt{n})$  [15], this shows that the lower bound takes  $O^*(n^{2.5})$  time per tree, amortized over all binary trees.

Also, note that while the LP-based algorithm is fastest on balanced trees, the dynamic programming algorithm is fastest on unbalanced trees. Hence, we can optimize the running time (slightly) by first computing the running time for the two algorithms and running the lower of the two. The running time computation can be performed using the optimal root placement algorithm, in  $O(n^2)$  time so it does not affect the asymptotic running time.

## 4 The Path-Constrained Travelling Salesman Problem

The current implementation of GRAPPA generates each possible full tree topology and computes the circular ordering lower bound. This strategy can be improved by generating trees bottom-up instead, inserting leaves one at a time into a partial topology, and computing the circular ordering lower bound on the partial topology. It is easy to see that a lower bound on a partial topology is a valid lower bound on any extension of it into a full tree. In this way, entire subtrees of the branch-and-bound recursion tree can be pruned at an earlier stage.

When such a bottom-up strategy is adopted, the problem of computing a circular ordering that is consistent with the partial tree can be rephrased as a *Precedence-Constrained Travelling Salesman Problem*. Here, we are asked to produce a min-cost tour of a set of cities that respect a set of precedence constraints in the form of a directed acyclic graph (DAG). This problem has been studied in [12], in which hardness results were given for the special cases of metrics induced by a line or hypercube, suggesting that the problem is hard in many practical instances.

For our application, we are interested in specializing not the metric, but the constraint graph. In contrast to [12], our special case is easy to approximate. In general, the problem of computing a maximum circular ordering lower bound can be seen as a version of TSP. Given a partial topology, a maximum linear ordering on this topology induces a constraint graph in which the DAG is a directed path. Clearly, Path-Constrained TSP is NP-hard since it contains TSP as a special case. However, we are able to prove the following result:

**Theorem 3.** *There exists a 3-approximation for the path-constrained travelling salesman problem.*

*Proof.* We will call the edges and vertices of the constraint graph *constraint edges* and *constraint vertices* respectively. First, take the complete graph on the  $n$  vertices with the distance  $d(i, j)$  on the edge from  $i$  to  $j$  and form a new graph by adding an auxiliary vertex  $x$  with 0-weight edges connected to each of the constraint vertices. Find a minimum spanning tree in this new graph. The optimal solution will contain the 0-weight edges, plus a tree growing from each of the constraint vertices. Discarding the 0-weight edges and the auxiliary vertex  $x$ , we have a forest where each tree contains exactly one of the constraint vertices. We produce a travelling salesman tour of each tree by the usual “Twice-around-the-spanning-tree” technique. The sum of the tree costs is a lower bound on  $OPT$  because the optimal solution must have a path between consecutive constraint vertices and removing the last edge on each of these paths produces a forest in which each tree contains exactly one of the constraint vertices. Therefore the sum of the costs of these tours is at most twice  $OPT$ . The final travelling salesman tour of the whole graph combines the individual tours with the (undirected) constraint edges, with short-cutting if necessary. The sum of all of the path edges is a lower bound on  $OPT$  by the triangle inequality so the final solution is a 3-approximation to the optimal path-constrained TSP.  $\square$

The approximation ratio implies a lower bound for the branch-and-bound algorithm by taking the cost of the solution and dividing by 3. We also note that the well-known Christofides technique of adding a min cost matching on vertices of odd degree does not seem to be easily applicable in our problem.

## 5 Experiments

We ran our maximum circular-ordering dynamic programming algorithm on a benchmark set of 12 chloroplast genomes from the Campanulaceae family (see Table 5) on a workstation with 1Gb of memory. The lower bound strategy implemented in GRAPPA is to try the default circular ordering first, then the *Swap-as-you-go* heuristic, stopping at the first time a lower bound that exceeds the current best score is found. Our implementation adds our dynamic programming algorithm for computing the max circular ordering lower bound as an additional bound if the first two bounds are not good enough. Our results show that we get a significant improvement in both number of trees scored and in total running time.

These experiments used the naive  $O(n^4)$  implementation of the dynamic programming so the running time would be even faster for the  $O(n^3)$  implementation. In practice, it may be possible to improve the running time slightly by sharing partial computations between different trees, since many subtrees are shared between trees.

	Camp 10		Camp 12	
	Trees scored	Time (min)	Trees scored	Time (min)
GRAPPA	6,391	3:40	80,551	165
Our algorithm	4,277	2:37	42,124	102

**Fig. 2.** Camp 12 is the full data set of 12 chloroplast genomes and Camp 10 is a subset of 10 of those genomes. Both data sets are distributed with the GRAPPA source code.

## 6 Discussion

It has recently come to our attention that the  $O(n^3)$  result for the maximum circular ordering problem (Section 2) was previously achieved in two completely different contexts [16, 17].

We also remark that the problem we consider in this paper is related to the classic *Distance Wagner Method*, in which the problem is to compute the shortest tree that dominates a given distance matrix [18]. Our problem differs in that we are also given the tree topology as an input.

## 7 Acknowledgements

We thank Bernard Moret for helpful discussions and Jijun Tang for sharing his code with us. We also thank the three anonymous reviewers for pointing us to references [16–18, 13]. This work was supported by NSF grant EF 03-31494. Radu Mihaescu was also supported by the Fannie and John Hertz Foundation.

## References

1. B. Moret, S. Wyman, D. Bader, T. Warnow, and M. Yan. A new implementation and detailed study of breakpoint analysis. In *PSMB*, 2001.
2. D. Sankoff and M. Blanchette. Multiple genome rearrangement and breakpoint phylogeny. *J. Comput. Biol.*, 5(3):555–570, 1998.
3. G. Bourque and P. Pevzner. Genome-scale evolution: Reconstructing gene orders in the ancestral species. *Genome Res.*, 12(1):26–36, 2002.
4. D. Bryant. A lower bound for the breakpoint phylogeny problem. *Journal of Discrete Algorithms*, 2:229–255, 2004.
5. P. Purdom, P. Bradford, K. Tamura, and S. Kumar. Single column discrepancy and dynamic max-mini optimizations for quickly finding the most parsimonious evolutionary trees. *Bioinformatics*, 16(2):140–151, 2000.
6. B. Holland, K. Huber, D. Penny, and V. Moulton. The minmax squeeze: Guaranteeing a minimal tree for population data. *Mol. Biol. and Evol.*, 22(2):235–242, 2005.
7. J. Tang. A study of bounding methods for reconstructing phylogenies from gene-order data. PhD Thesis, 2003.
8. J. Tang, B. Moret, L. Cui, and C. dePamphilis. Phylogenetic reconstruction from arbitrary gene-order data. In *BIBE*, 2004.
9. J. Tang and B. Moret. Linear programming for phylogenetic reconstruction based on gene rearrangements. In *CPM*, 2005.
10. B. Moret, J. Tang, and T. Warnow. Reconstructing phylogenies from gene-content and gene-order data. In O Gascuel, editor, *Mathematics of Evolution and Phylogeny*. Oxford Univ. Press, 2004.
11. K. Chaudhuri, K. Chen, R. Mihaescu, and S. Rao. On the tandem duplication-random loss model of genome rearrangement. In review.
12. M. Charikar, R. Motwani, P. Raghavan, and C. Silverstein. Constrained TSP and low power computing. In *WADS*, 1997.
13. G. Lancia and R. Ravi. GESTALT: Genomic steiner alignments. In *CPM*, 1999.
14. N. Young. Sequential and parallel algorithms for mixed packing and covering. In *FOCS*, 2001.
15. P. Flajolet and A. M. Odlyzko. The average height of binary trees and other simple trees. *J. Computer System Sci.*, 25:171–213, 1982.
16. Z. Bar-Joseph, E. D. Demaine, D. K. Gifford, A. M. Hamel, T. S. Jaakkola, et al. K-ary clustering with optimal leaf ordering for gene expression data. *Bioinformatics*, 19(9):1070–1078, 2003.
17. R. E. Burkard, V. G. Deineko, and G. J. Woeginger. The travelling salesman and the pq-tree. *Mathematics of Operations Research*, 24:262–272, 1999.
18. M. Farach, S. Kannan, and T. Warnow. A robust model for finding optimal evolutionary trees. In *STOC*, 1993.