[40] M.S. Springer, G.C. Cleven, O. Madsen, W.W. de Jong, V.G. Waddell, H.M. Amrine, and M.J. Stanhope. Endemic african mammals shake the phylogenetic tree. *Nature*, 388(6637):61–64, 1997.

[41] M. Steel. The complexity of reconstructing trees from qualitative characters and subtress. *Journal of Classification*, 9(1):91–116, 1992.

[42] David E. Stewart and Zbigniew Leyk. *Meschach Library*. Australian National University, Canberra, 1994. `http://www.netlib.org/c/meschach`.

[43] Korbinian Strimmer and Arndt von Haeseler. Quartet puzzling: A quartet maximum-likelihood method for reconstructing tree topologies. *Molecular Biology and Evolution*, 13(7):964, 1996. Software available at `ftp://ftp.ebi.ac.uk/pub/software/unix/puzzle/`.

[44] J.D. Thompson, D.G. Higgins, and T.J. Gibson. CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalty and weight matrix choice. *Nucleic Acids Research*, 22:4673–4780, 1994.

[45] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM review*, 38(1):49, 1996.

[46] Tandy Warnow. Some combinatorial optimization problems in phylogenetics. unpublished.

[47] M.S. Waterman, T.F. Smith, M. Singh, and W.A. Beyer. Additive evolutionary trees. *Journal of Theoretical Biology*, 64:199–213, 1977.

[25] D. Graur. Towards a molecular resolution of the ordinal phylogeny of the eutherian mammals. *FEBS letters*, 325(1/2):152, 28 June 1993.

[26] Dan Graur. Mammalian phylogeny: using every available molecule. In *Proceedings of the Training Course in Molecular Evolution*, pages 6–8, Bari, Italy, 1996. CIHEAM.

[27] Dan Graur, Laurent Duret, and Manolo Gouy. Phylogenetic position of the order lagomorpha (rabbits, hares and allies). *Nature*, 379(6563):333, 1996.

[28] Dan Graur, Manolo Gouy, and Laurent Duret. Evolutionary affinities of the order perissodactyla and the phylogenetic status of the superordinal taxa ungulata and altungulata. *Molecular Phylogenetics and Evolution*, 7(2):195, April 1997.

[29] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, Berlin, 1987.

[30] Dan Gusfield. The steiner tree problem in phylogeny. Technical Report 332TH, Yale university, New Haven, September 1984.

[31] J.A Hartigan. Minimum mutation fits to a given tree. *Biometrics*, 29:53–65, 1973.

[32] M. Kimura. *The Neutral Theory of Molecular Evolution*. Cambridge University Press, Cambridge, 1983.

[33] A. Krettek, A. Gullberg, and U. Arnason. Sequence analysis of the complete mitochondrial DNA molecule of the hedgehog, erinaceus europaeus, and the phylogenetic position of lipotyphla. *Journal of Molecular Evolution*, 41(6):952, 1995.

[34] N. Linial, A. Magen, and M. Saks. Low distortion euclidean embeddings of trees. In *STOC*, 1998. to appear.

[35] Ron Ophir, 1997. Personal communication.

[36] Dan Pelleg. *Software for Constructing Phylogenies from Quartets*. Computer Science department, Technion, April 1998. `http://www.cs.technion.ac.il/Labs/cbl/publications.html`.

[37] C. Phillips and T.J. Warnow. The asymmetric median tree — a new model for building consensus trees. *Discrete Applied Mathematics*, 71(1–3):311, 1996.

[38] N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4, 1987.

[39] R.R. Sokal and P.H.A. Sneath. *Principles of numerical taxonomy*. A Series of books in biology. W.H. Freeman, San Francisco, 1963.

[10] W.H.E Day and D. Sankoff. Computational complexity of inferring phylogenies by compatibility. *Systematic Zoology*, 35:224–229, 1986.

[11] L. Duret, D. Mouchiroud, and M. Gouy. Hovergen: a database of homologous vertebrate genes. *Nucleic Acids Research*, 22(1):2360, 1994.

[12] Peter Erdös, Mike Steel, László A. Székely, and Tandy Warnow. Constructing big trees from short sequences. In Pierpaolo Degano, Robert Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Automata, Languages and Programming, 24th International Colloquium*, volume 1256 of *Lecture Notes in Computer Science*, pages 827–837, Bologna, Italy, 7–11 July 1997. Springer-Verlag.

[13] M. Farach, S. Kannan, and T. Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, 13(1/2):155–179, January 1995.

[14] J. Felsenstein. The number of evolutionary trees. *Systematic Zoology*, 27:27–33, 1978.

[15] J. Felsenstein. Confidence limits on phylogenies: An approach using the bootstrap. *Evolution*, 39:783–791, 1985.

[16] J. Felsenstein. PHYLIP - phylogenetic inference package, (version 3.2). *Cladistics*, 5:164–166, 1989.

[17] W. M. Fitch. A non-sequential method for constructing trees and hierarchical classifications. *Journal of Molecular Evolution*, 18(1):30–37, 1981.

[18] W.M. Fitch. Towards defining the course of evolution: minimum change for a specified tree topology. *Systematic Zoology*, 20:406–416, 1978.

[19] W.M. Fitch and E. Margolaish. Construction of phylogenetic trees. *Science*, 155:279–284, 1967.

[20] L.R. Foulds and R.L Graham. The steiner problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, 3:43–49, 1982.

[21] K. Fujisawa, M. Kojima, and K. Nakata. *SDPA user's manual.* Tokyo Institute of Technology, Tokyo, 1996. `ftp://ftp.is.titech.ac.jp/pub/OpRes/software/SDPA`.

[22] Olivier Gascuel. BIONJ: An improved version of the NJ algorithm based on a simple model of sequence data. *Molecular Biology and Evolution*, 14(7):685, 1997.

[23] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the Association for Computing Machinery*, 42(6):1115–1145, November 1995.

[24] J. Gower and G. Ross. minimum spanning tress and single linkage cluster analysis. *Applied Statistics*, 18:54–64, 1969.

Bibliography

[1] Richa Agarwala, Vineet Bafna, Martin Farach, Babu Narayanan, Mike Paterson, and Mikkel Thorup. On the approximability of numerical taxonomy (fitting distances by tree metrics). In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 365–372, New York/Philadelphia, 28–30 January 1996. ACM/SIAM.

[2] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995.

[3] H.-J. Bandelt and A. Dress. Reconstructing the shape of a tree from observed dissimilarity data. *Advances in Applied Mathematics*, 7:309–343, 1986.

[4] V. Berry and O. Gascuel. Inferring evolutionary trees with strong combinatorial evidence. Research Report CS-RR-341, Department of Computer Science, University of Warwick, Coventry, UK, March 1998. `http://www.dcs.warwick.ac.uk/pub/reports/rr/341.html`.

[5] Hans L. Bodlaender, Michael R. Fellows, and Tandy Warnow. Two strikes against perfect phylogeny. In Werner Kuich, editor, *Automata, Languages and Programming, 19th International Colloquium*, volume 623 of *Lecture Notes in Computer Science*, pages 273–283, Vienna, Austria, 13–17 July 1992. Springer-Verlag.

[6] J. Bourgain. On lipschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 52:46–52, 1985.

[7] Buneman. The recovery of trees from measures of dissimilarity. In F.R. Hodson, D.G. Kendall, and P. Tautu, editors, *Anglo-Romanian Conference on Mathematics in the Archaeological and Historical Sciences*, pages 387–395, Mamaia, Romania, 1971. Edinburgh University Press.

[8] Jaime Cohen and Martin Farach. Numerical taxonomy on data: Experimental results. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 410–417, New Orleans, Louisiana, 5–7 January 1997.

[9] W.H.E Day. Computationally difficult parsimony problems in phylogenetic systematics. *Journal of Theoretical Biology*, 103:429–438, 1983.

nature is out of the scope of this thesis.

From the algorithmic point of view, this work raises a number of interesting open problems. It would be nice to prove any performance guarantee for the "geometric" reconstruction method. In fact, we are not aware of any efficient algorithm for reconstructing trees from weighted quartets whose output is guaranteed to be above one third of the maximum. (A tree chosen at random is expected to satisfy one third of the quartets). For realistic datasets, any decrease of the exponent's base for an "exact" algorithm will be significant. (The dynamic programming algorithm takes about 5 days for datasets with $n = 20$ on a \$30K machine in 1997 prices.)

The fact that the score of the optimal tree is only 73.4% seems to indicate that the input is not very reliable. However, even with this limited quality data, both the geometric and the optimal trees make biological sense. We note, in particular, the position of Insectivora as an outgroup to the other eutherian taxa [33], and the monophyly of the extended superordinal taxon Paenungulata, which includes Edentata [40]. We are currently compiling an updated dataset of mammalian protein sequences, with which we hope to resolve the 100-year-old problem of mammalian ordinal phylogeny "not vaguely and generally, but with all possible precision in place, weight, and measure" (Sir William Herschel, 1738–1822).

Chapter 6

Discussion

We believe that the methods presented in this paper are significant from a biological and evolutionary point of view. They allow us to combine partial phylogenetic trees into a tree containing all the taxa of interest. This enables the use of most available sequences, unlike "traditional" reconstruction methods. Another advantage is that our algorithms do not require the full set of $\binom{n}{4}$ quartets. This has practical implications since the experimental results suggest that ignoring the lower-confidence quartets may improve the generated tree.

The databases of sequences are updated periodically with the accumulation of new sequence data. Typically such updates still leave many sequences unknown for many taxa, but they will tend to produce more reliable quartet information. Our methods should thus enable a periodical reassessment of phylogenetic theories.

Obviously the biological components of this study will have to be refined and updated in the future. Most importantly, the performance of different tree making methods and confidence measures will have to be assessed against real data and simulation results. Another point of interest will be the assignment of confidence values to the generated tree edges. Our algorithms give no indication that some edges in the generated tree are not evident from the data. These may in fact be furcations which are arguable from a biological point of view. A direction for further research and experimentation would be to find an assignment of support values, given the data and the produced trees.

Our simulated phylogenies raise a different issue. Often the optimal tree generated from the simulated quartets is not the original tree. In other words, once errors are inserted into the data-set, it is possible that the true tree is not the one which receives maximal support. The errors may accumulate to evidence in favor of some other tree. Of course such phenomenon is problematic if one wishes to construct phylogenies based on available data. However, we feel that the models of confidence and error assumed may be too "harsh" to model biological reality. For example, the confidence evaluation function may not produce an accurate confidence value C_e. The ratio of tree path lengths to one another is probably not a good enough estimate of the quartet satisfaction. This issue should be settled by using a more refined error model, which simulates *sequences* according to some known evolutionary model. A simulation of this

Tree size	$\texttt{score}_{Q_e}(G)$		$\texttt{score}_{Q_r}(T)$	
	`transform`	`uniform`	`transform`	`uniform`
16	96.5 %	90.5 %	100 %	71.5 %
24	94.0 %	88.6 %	100 %	69.7 %
32	91.7 %	86.9 %	100 %	68.2 %

Table 5.4: Effect of error model on quality of trees

m	$\texttt{score}_{Q_e}(G(m))$	$\texttt{score}_{Q_r}(G(m))$	$\texttt{score}_{Q_e}(P(m))$	$\texttt{score}_{Q_r}(P(m))$
0	90.2 %	71.9 %	99.8 %	78.3 %
0.20	90.9 %	80.4 %	99.9 %	88.8 %
0.30	94.5 %	93.2 %	99.9 %	98.8 %
0.40	96.1 %	98.9 %	97.4 %	100.0 %
0.50	95.5 %	100.0 %	90.5 %	100.0 %
0.60	94.6 %	100.0 %	85.6 %	100.0 %

Table 5.5: Effect of disregarding low-confidence quartets. Results for the `2phase` model with $n = 15$. m is the threshold value. Q_r denotes the set of quartets after the insertion of errors and the deletion of quartets with low confidence level.

scores of trees generated under the `transform` model to those generated by the `uniform` model, which contains errors. Again, the numbers are in Table 5.4.

For the `2phase` model we marked by m the threshold value for input quartets (quartets with confidence lower than m were not considered by the algorithm). Here we find that the higher m is, the better is the generated tree. This effect can be predicted directly from the model, which assigns high confidence values to correct topologies. See Table 5.5. $\texttt{score}_{Q_e}(G(m))$ and $\texttt{score}_{Q_e}(P(m))$ are computed with respect to *all* $\binom{n}{4}$ Q_e quartets. This means it is possible to have $\texttt{score}_{Q_e}(G(m)) > \texttt{score}_{Q_e}(P(m))$, e.g., for $m = 0.6$). $\texttt{score}_{Q_r}(G(m))$ and $\texttt{score}_{Q_r}(P(m))$ are computed with respect to the same set of quartets (all those with confidence $\geq m$). Therefore, for all m, $\texttt{score}_{Q_r}(P(m)) \geq \texttt{score}_{Q_r}(G(m))$.

Tree size	Model	$\texttt{score}_{Q_e}(G)$		$\texttt{score}_{Q_e}(P)$		$\frac{\texttt{score}_{Q_e}(G)}{\texttt{score}_{Q_e}(P)}$	
16	simple	96.3	%	100.0	%	96.3	%
24	simple	93.4	%	100.0	%	93.4	%
32	simple	91.4	%	100.0	%	91.4	%
10	2phase	94.0	%	98.5	%	95.4	%
12	2phase	92.3	%	99.4	%	92.9	%
13	2phase	93.3	%	99.7	%	93.6	%
15	2phase	90.2	%	99.8	%	90.4	%
16	2phase	89.1	%	99.8	%	89.3	%
16	transform	96.5	%	100.0	%	96.5	%
24	transform	94.0	%	100.0	%	94.0	%
32	transform	91.7	%	100.0	%	91.7	%
10	uniform	92.6	%	94.9	%	97.6	%
12	uniform	92.6	%	97.4	%	95.1	%
13	uniform	91.1	%	98.4	%	92.6	%
15	uniform	91.1	%	98.6	%	92.4	%
16	uniform	90.5	%	98.6	%	91.8	%
24	uniform	88.6	%				
32	uniform	86.9	%				

Table 5.3: Summary of simulation results

The full set of experiments may be found and downloaded at [36]. Here we go over the main results. Tests were conducted for tree sizes $n = 10$, 12, 13, 15, 16, 24 and 32. Optimal trees were obtained only for trees not larger than 16 leaves. For each model and each tree at least 100 runs were made. For the `2phase` model, experiments were made that consider just those quartets in Q_r above some confidence value m. About 40 runs were done for each of the following values of m: 0, 0.2, 0.3, 0.4, 0.5, 0.6.

Table 5.3 details the results for the various error models and tree sizes. As can be seen, in all error models and all tree sizes the score of the geometric tree $\texttt{score}_{Q_e}(G)$ was above 85%. For trees with less than 24 leaves $\texttt{score}_{Q_e}(G)$ was always above 89%. The score of the generated tree as fraction of the score of the optimal tree $\texttt{score}_{Q_e}(G)/\texttt{score}_{Q_e}(P)$ was always above 90%.

One general observation is that the bigger the true tree, the lower the quality of the generated tree. One possible reason is that the relative quality of the input decreases. This can be deduced from Table 5.4, by looking at the score of the real tree, with respect to the erroneous quartets. Note that it is much lower than the score of the geometric tree. We deduce that our optimization criteria is not well-suited for highly erroneous input, in the respect that it does not target the algorithm towards the optimal tree.

The insertion of errors into the reported quartets does indeed worsen the quality of the generated tree. This can be seen by comparing the average

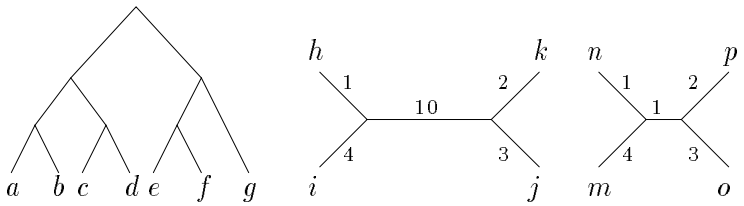


Figure 5.2: Example of the calculation of confidence level. Edge weights equal 1 unless noted otherwise. For $ab|cd$, $C_e = 0.5$. For $ef|ga$, $C_e = 0.22$. For $ef|ab$ and $hi|jk$, $C_e = 0.67$. For $mn|op$, $C_e = 0.17$.

which topology is reported. First, the confidence values C_e are transformed to the $[1/3, 1]$ range as before. We denote these new values by C'_e. Now, the true topology is reported with probability C'_e, and the other two have equal probability $(1 - C'_e)/2$. The reported confidence level is drawn uniformly from the range $[0, t]$ where t is the probability associated with the specific topology (either C'_e or $(1 - C'_e)/2$. Both the reported topology and the reported confidence value are selected using a single uniform selection of a number in the range $[0, 1]$. Under this model, a quartet with high confidence value will be reported with the true topology with high probability, although the confidence level reported for it may be very low. The next model guarantees that correct topologies will not receive low confidence values. This may correspond to the real-life situation where definite quartets are supported by much data, and so the reported confidence value is high, while quartets with low confidence level may have an incorrect topology which has a marginal advantage over the scores of the other possibilities. This model is called `2phase`, because the decision over the reported confidence is performed only after the decision regarding the reported topology. Each topology is selected with probability proportional to its assumed confidence value as in the `uniform` model. If an incorrect topology is selected, then a uniformly distributed value in the range $[0, (1 - C'_e)/2]$ is taken as before (we recall this value is never greater than 1/3). Otherwise, C_r is taken uniformly from the range $[0.5C'_e, C'_e]$. This choice guarantees that the reported confidence for correct topologies is not too low.

We introduce some notations. Let T be the true tree generated by the random process, and G the tree generated by the geometric algorithm. By P we denote the optimal tree, given the same input as given to the geometric algorithm. Q_e is the set of $\binom{n}{4}$ quartets calculated from a tree T with n leaves, while Q_r is the set of quartets after biased by the error model. Given a tree X, denote by $\texttt{score}_Q(X)$ the total weight of quartets in a set Q satisfied by X. In these terms, the following figures were obtained: $\texttt{score}_{Q_e}(G)$, $\texttt{score}_{Q_r}(G)$, $\texttt{score}_{Q_e}(P)$, $\texttt{score}_{Q_r}(P)$, $\texttt{score}_{Q_r}(T)$, $\texttt{score}_{Q_e}(T)$ [2]. We will generally refer to the average of these values taken over a number of experiments.

[2]Not necessarily each datum was computed in each experiment.

Graur's algorithm uses a different approach and applies a neighbor-joining algorithm on a non-metric distance matrix, defined by pairwise agglomeration within quartets [35].

5.2 Simulations

Real-life data has its merits, but the fact remains that the true underlying phylogeny is unknown. For this reason, we have conducted experiments where a true tree is known, and scoring of various algorithms is compared to it. The course of one such experiment starts by choosing a tree at random, and extraction of its induced quartets and their weights. Next, the geometric algorithm is run on the quartets, and the quality of the output is assessed. Some 2000 experiments of this kind were conducted. Most of these included an additional stage, where the optimal tree was found using the dynamic-programming algorithm. This optimal tree was used both to determine the quality of the input and to evaluate the performance of the geometric algorithm against a global maximum.

Each run started with the creation of a random tree. The random process begins with a single node (root), and adds one leaf in each iteration. This is done by randomly selecting a leaf and splitting it into two new leaves. Then, all external edges are "stretched" by a length taken randomly with exponential distribution with mean $1/k$, where k is the number of leaves generated so far. The tree-generation software was taken from [8].

Given a tree T, a confidence level for each quartet $ab|cd$ was computed as follows. Denote by $d_T(x, y)$ the tree distance between leaves x and y. The evaluated confidence level C_e is defined by:

$$C_e \stackrel{\text{def}}{=} 1 - \frac{d_T(a,b) + d_T(c,d)}{\min\left\{d_T(a,c) + d_T(b,d), d_T(a,d) + d_T(b,c)\right\}}$$

This measure is proportional to the ratio of length of the "central" path in the quartet to the sum of the two leaf-to-leaf paths including it, and normalized to the range $[0, 1]$. The idea is that if a quartet has a long central path, as compared to the pairwise paths of the paired taxa, then we are more confident of its topology. For an example see Figure 5.2.

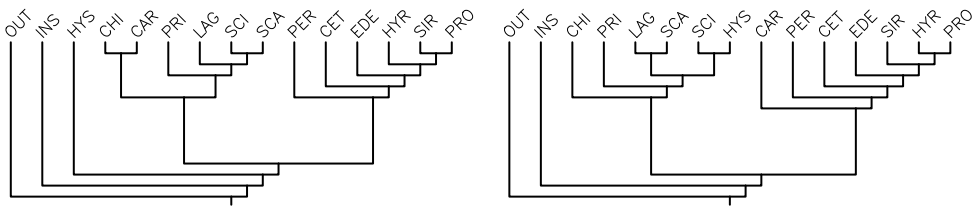
Given an evaluated confidence C_e for a quartet $ab|cd$, the value C_r reported to the algorithm for this quartet was calculated according to the specific error model. In some error models, the topology reported to the algorithm could also deviate from the correct one according to some probabilistic criterion. In the simplest model there is no error: the reported value C_r always equals C_e and the true topology is reported. We call this model `simple`. The `transform` model also reports the topology as-is, but the confidence level is transformed to the range $[1/3, 1]$ by the following calculation: $C_r \leftarrow (1 + 2C_e)/3$. This corresponds to cases where the reported topology is chosen to be the one with maximal score among the three possible topologies. The scores are normalized so their sum is 1, so the best one cannot have score smaller than $1/3$. The `uniform` error model uses a uniformly distributed random selection to decide

Tree	Score		Satisfied
	with Compensation	without Compensation	Quartets
Gregory (1910)	56.1%	47.8%	44.8%
Simpson (1945)	55.2%	49.2%	46.2%
Mckenna (1975)	54.0%	46.9%	44.0%
Novacek (1982)	55.6%	43.1%	39.6%
Shoshani (1986)	61.1%	bifurcating	57.7%
Novacek *et. al.* (1988)	53.0%	36.2%	33.3%
Novacek (1992)	58.1%	48.7%	45.3%
Graur (1997)	71.0%	bifurcating	66.7%
Puzzling	71.6%	bifurcating	68.1%
Geometric	72.1%	bifurcating	68.7%
Optimal	73.4%	bifurcating	70.2%

Table 5.1: Comparison of published trees. Trees with year numbers are from [25]. "Puzzling" was generated by the algorithm in [43], and our quartet data.

n	time
15	7 min.
16	28 min.
17	109 min.
18	7 hr.
19	29 hr.
20	128 hr.

Table 5.2: Run times for the "exact" algorithm with all $\binom{n}{4}$ quartets in the input. For $n = 15, 16, 20$ the times are measured on a \$30K machine, in 1997 prices (Sun Ultra-4 at 300 MHz). For other values of n they are estimated.



geometric tree optimal tree

Figure 5.1: Phylogenetic trees obtained by the geometric and optimal methods

topologies and associated weights was the input to our algorithms.

The version that produced the best geometric tree used the following parameters: Only quartets with confidence level 90% or higher were included in the objective function. There were 452 such quartets, which means that about two thirds of the input quartets were ignored.[1] In addition, we imposed the "distance constraint" $\langle v_i, v_j \rangle \leq 0.75$ for each pair of points. The execution took less than 5 seconds on a Sun Ultra-4 machine. The resulting tree has score 72.1%.

The dynamic-programming algorithm took into account all 1365 weighted quartets. The execution took some 7 minutes on a Sun Ultra-4 machine (see Table 5.2). The resulting *optimal* tree is unique and has score 73.4%. The two trees are shown in Figure 5.1.

We computed the score (with respect to the same four-taxon input) of other phylogenetic trees that were published in the literature (references in [25]). Table 5.1 lists these scores. It should be noted that most of the earlier trees are multifurcating. For these trees, the contribution of each unresolved quartet is computed either as 1/3 of its confidence ("with compensation") or as 0 ("without compensation"). As seen, our algorithms produce the best trees with respect to the quartet satisfaction criterion on our data-set. We remark the quartet data was not available during the construction of most of the trees (therefore their optimization criteria were different). The exceptions, beside our algorithms, are the quartet-puzzling tree and Graur's tree from 1997. The quartet puzzling implementation was obtained from [43] and modified to construct a tree from our (unweighted) quartet data. After 1000 puzzling trees are constructed, a consensus tree is computed using the CONSENSE program in [16]. This output consensus tree is what we call the "puzzling" tree. The whole process takes about 10 seconds on the same Sun machine. As seen, the performance of this method falls just a little short of our geometric algorithm.

[1] To ensure that no taxon is severly underrepresented in this reduced quartet set, we counted the number of times that each taxon appears in the input. There are $4 \times 452 = 1808$ appearances, and under a uniform representation distribution one would expect each taxon to appear in 1808/15 of them, or about 120 times. Our calculations show that the least frequent taxon (INS) appears in 82 of the input quartets (4.5%), while the most frequent taxon (PRO) appears in 157 quartets (8.7%).

Chapter 5

Computational Results

In this chapter we describe the results of executing our algorithms on real and simulated data. We implemented both algorithms, as well as supporting software. The geometric algorithm was implemented using the semidefinite package SDPA [21]. The dynamic programming algorithm was coded in `C++`. Additional software included Perl scripts for conducting large-scale experiments over the parameter space and collection of the results and the related statistics. Other programs include those which assign scores to trees or build instances of SDP. Since SDP packages available today [21, 45] are hard to interact with, "envelope" software was written to smooth the interface. In particular, one module allows to formulate a semidefinite program in a human readable — and editable – way, in the spirit of LP packages in common use. All in all, some 20,000 lines of code were written. The entire software, data, and test-cases are available on the net [36].

5.1 Real Data

We briefly describe how the four-taxon trees, which constitute the actual input to our algorithms, were obtained. The following 14 mammalian taxa were used: CAR = Carnivora (carnivores), CET = Cetartiodactyla (even-hoofed ungulates and cetaceans), CHI = Chiroptera (bats), EDE = Edentata (sloths and armadillos), HYR = Hyracoidea (hyraxes), HYS = Hystricognathi (guinea pigs and porcupines), INS = Insectivora (hedgehogs and shrews), LAG = Lagomorpha (rabbits and hares), PER = Perissodactyla (horses), PRI = Primates (humans, apes and monkeys), PRO = Proboscidea (elephants), SCA = Scadentia (tree shrews), SCI = Sciurognathi (rats, mice and squirrels), and SIR = Sirenia (sea cows and dugongs). As an outgroup (OUT) to the above eutherian mammalian taxa we used chicken or marsupial.

These 15 taxa were subject to the following process: Protein sequences have been collected from the HOVERGEN [11] database. Homologous sequences were aligned using the CLUSTAL W program [44]. Genetic distances were computed as in [32]. Four-taxon phylogenetic trees were inferred by using maximum parsimony [16]. Reliability of the internal branch in each 4-taxon tree was assessed by 1000 bootstrap replicates [15]. This list of $\binom{15}{4} = 1365$ quartet

then decide whether a 1^+ or a 2^+ will be in each such place, and finally choosing which of the three non $+$ symbols will be written in any of the remaining $n-4$ places. This counts up to:

$$\binom{n}{4} 2^4 \cdot 3^{n-4} = \left(\frac{2}{3}\right)^4 \binom{n}{4} 3^n$$

To see why this equals A, fix ℓ to be the number of places in the vector in which there are 1's or 2's (of either kind). We now count the number of legitimate vectors. First, choose the ℓ places out of possible n that will host 1's or 2's. Now choose whether a 1 or a 2 will be in each of those places. Finally, select 4 of which to be marked with a $+$ sign. Summing over all possible values of ℓ, we obtain A.

The proof for B follows the same general guidelines. Consider the alphabet of symbols $\{0, 1, 2, 0^+, 1^+, 2^+\}$. Again, construct an n-long vector from these symbols. Again, 4 symbols are allowed to carry the $+$ mark, but the 0^+ symbol must appear exactly once. We suggest this straightforward solution. Select the places with $+$ signs in them. Select one of three possible non $+$ symbols for each of the remaining $n-4$ places. Choose which one of the four $+$ places will contain the 0^+ symbol. Finally, decide whether a 1^+ or a 2^+ goes in each of the other three places. This counts up to:

$$\binom{n}{4} 3^{n-4} \binom{4}{1} 2^3 = \frac{2^5}{3^4} \binom{n}{4} 3^n$$

The alternative counting is as follows. Once again fix ℓ to be the number of places in the vector in which there are 1's or 2's (of either kind). Choose the ℓ places out of possible n that will host 1's or 2's. Now choose whether a 1 or a 2 will be in each of those places. Select which three of these ℓ places will be marked by $+$. Finally, select which of the $n-\ell$ zeroes will carry the $+$ sign. Summing over all possible values of ℓ, we have B. $\square$

In the general case, the given input quartets are kept in a list which is sequentially scanned for each partition $\langle S_1, S_2 \rangle$. But when all possible $k = \binom{n}{4}$ input quartets are present, the performance can be improved further yet. Consider a partition of a set S. Instead of checking satisfiability of every one of the $\binom{n}{4}$ quartets, we can inhibit the test for those quartets which are known *a priory* to be unresolved by S. In particular, it suffices to inspect only those quartets which have three or more elements in S. To implement this, we prepare (in a preprocessing stage) a four-dimensional array of size n^4, where each dimension is indexed by taxa. The entries in the array are the input quartets and their weights. Given S, construct a list of the taxa in S. To scan all quartets which contain 4 taxa from S, enumerate all 4-tuples of elements in the list. For each 4-tuple the access to the appropriate quartet in the array takes constant time, and the elementary step takes constant time as well. For the quartets having exactly 3 elements from S use a similar enumeration. Once a relevant quartet is accessed, we process it (with respect to $\langle S_1, S_2 \rangle$) as before.

To analyze the amount of saving, we first notice that if we consider a random set $S \subset \{1, \ldots, n\}$ (each element chosen independently with probability 1/2) and a random quartet q, the probability that S contains three or more elements from q is 5/16. But while our dynamic programming algorithm does go over all sets S, it does not weigh them uniformly: Larger sets are weighted more heavily (to be precise, a subset of size ℓ has probability $2^\ell/3^n$). Since larger sets will contain 3 or more elements from a given quartet more frequently, the overall saving effect will be less dramatic. However, it is still meaningful. We argue that we achieve a factor 11/27 saving, compared to going over all quartets for each S. For a set S with ℓ taxa, the number of quartets having all 4 taxa in S is $\binom{\ell}{4}$. The number of quartets having 3 taxa in S is $\binom{\ell}{3}(n-\ell)$. So overall, the number of quartets that will be inspected at all stages of the dynamic programming algorithm equals

$$\sum_{\ell=0}^{n} \binom{n}{\ell} 2^\ell \left[\binom{\ell}{4} + \binom{\ell}{3}(n-\ell) \right]$$

The next lemma compares this to the number of quartets inspected using the "all quartets" approach. Before we proceed we remark that the incorporation of this improvement into the algorithm improved the running time by about 30%.

Lemma 4

$$\sum_{\ell=0}^{n} \binom{n}{\ell} 2^\ell \left[\binom{\ell}{4} + \binom{\ell}{3}(n-\ell) \right] = \frac{16}{27} \binom{n}{4} 3^n$$

Proof: Let $A = \sum_{\ell=0}^{n} \binom{n}{\ell} 2^\ell \binom{n}{4}$ and let $B = \sum_{\ell=0}^{n} \binom{n}{\ell} 2^\ell \binom{\ell}{3}(n-\ell)$. We first show that $A = \frac{16}{81}\binom{n}{4}$. Later we prove $B = \frac{32}{81}\binom{n}{4}$.

We will use a combinatorial argument to calculate A. Consider an alphabet of 5 symbols: $\{0, 1, 2, 1^+, 2^+\}$. Count the number of vectors of length n which consist of these symbols, with the added constraint that exactly 4 symbols carry the + sign. A straightforward solution is to choose 4 places for the + symbols,

characteristic vectors (4-tuples over $\{1, 2, N\}$). By setting a convention that the first and second elements in the vector represent the elements of the first pair in the quartet, a single table may be used for all input quartets.

Checking satisfiability of a given quartet q as described takes 5 memory accesses (4 to construct the characteristic vector and 1 for the table lookup). In case the quartet is satisfied, an additional memory access is required to retrieve its weight, and one floating point addition. So for k input quartets, the value $\texttt{sum}(\text{TOP}_Q(T(S_1, S_2)))$ can be computed by performing up to $6k$ memory accesses and k floating point additions. Viewing the 6 memory accesses and the 1 addition as one basic step, this means that finding $\texttt{opt_score}_Q(S)$ for S of size i and Q of size k takes $2^i \cdot k$ basic steps. This computation is held for every set $S \subset \{1, \ldots, n\}$, and therefore the total number of steps in computing the optimal tree is

$$\sum_{i=0}^{n} \binom{n}{i} 2^i k = k3^n \ .$$

For 15 taxa this is a 5×10^5 speedup factor, compared to the naive, exhaustive-search approach.

In terms of memory, for each subset of $\{1, \ldots, n\}$ we need an entry with its optimal score and optimal partition. This means $\Theta(2^n)$ memory requirements.

4.4.1 Ramifications

We now turn to describe some improvements of the algorithm. Even though none of these improvements has a significant effect on the asymptotic behavior of the algorithm, they all make constant factor improvements to the actual running time. These seemingly minor improvements of reducing the running time by constant factors make the difference between the processing of real-life datasets in reasonable time and resorting to toy problems.

The first observation is that each partition $\langle S_1, S_2 \rangle$ of S is processed twice; the second time is when S_1 and S_2 exchange roles. Therefore, the result will not change if we only scan those partitions where S_1 precedes S_2 in lexicographic order. This gives a factor 2 saving.

A common practice in phylogenetic studies is to include in the dataset one "external" taxon, which is known not to be affiliated with any other taxon in the data. This taxon, labeled "outgroup", is later used to root the resulting tree by placing the new root at the edge that connects the outgroup with the rest of the tree[2]. When we know in advance which taxon is the outgroup, we consider at the top-level only the partition that places the outgroup in one subtree, and every other taxon in the other. This means that an optimal tree should be found just for the set of $n - 1$ "internal" taxa. Since the dependence of the run time on n, the number of taxa is proportional to 3^n, this reduction of n by 1 yields a factor 3 saving.

[2]In fact, the resulting tree may be rooted on any of the possible $2n - 3$ edges without affecting quartet satisfaction. Out of these trees, $n - 1$ correspond to the "outgroup" model where the root is adjacent to a leaf.

4.3 The Code

Let $\{1\},\ldots,\{n\},\{1,2\},\ldots,\{n-1,n\},\ldots,\{1,\ldots,n\}$ be an ordering of all 2^n-1 non-empty subsets of $\{1,\ldots,n\}$, in which every set appears after all its subsets. For each non-empty subset S, we compute the optimal score of a phylogenetic tree for S. In addition we store for each S a partition $\langle S_1,S_2\rangle$ that induces the score $\texttt{opt_score}_Q(S)$. In the backtracking phase, this partition is used to reconstruct an optimal phylogenetic tree.

Optimal score computation

Go over the sets $S\subseteq\{1,\ldots,n\}$ in order,

- If $|S|\leq 2$, then $\texttt{opt_score}_Q(S)=0$.
- Compute the score of each S-partition, $\langle S_1,S_2\rangle$, which equals

$$\texttt{sum}(\mathrm{TOP}_Q(T(S_1,S_2)))+\texttt{opt_score}_Q(S_1)+\texttt{opt_score}_Q(S_2)\ .$$

- Let $\texttt{opt_score}_Q(S)$ be a maximal score, and let $\texttt{opt_tree}_Q(S)$ be a partition for which the maximal score is obtained.

The optimal score for the whole set is stored in the entry $\texttt{opt_score}_Q(\{1,\ldots,n\})$.

Tree Reconstruction

We construct a phylogenetic tree T with an optimal score as follows:

- The root of T is determined by an optimal partition $\langle S_1,S_2\rangle$ of $S=\{1,\ldots,n\}$.
- Recursively build a phylogenetic tree for S_1 and S_2.

4.4 Algorithm Complexity

The elementary step in the algorithm is checking whether a quartet q is satisfied by a given partition of S, $\langle S_1,S_2\rangle$. We notice that each of the four taxa in q satisfies exactly one of the following three properties with respect to the partition $\langle S_1,S_2\rangle$:

1. The taxon is in S_1.
2. The taxon is in S_2.
3. The taxon is in neither.

We can therefore identify with q a characteristic vector which describes the presence of its taxa in S_1 and S_2. The vector `112N`, for example, means the two elements of first pair are in S_1, while one of the elements of the second pair is in S_2, and the other does not belong to the set of taxa currently in discussion. By corollary 3, we conclude that the root of $T(S_1,S_2)$ is $\texttt{qlca}(q)$ and that it satisfies q. All we need, then, is to construct at the preprocessing stage, a lookup table of size 3^4. The entries in this table are all the possible

set of quartets in Q that have v as their `qlca` and are satisfied by T. The following equality describes a partition of $\mathrm{SAT}_Q(T)$ to three disjoint subsets

$$\mathrm{SAT}_Q(T(v)) = \mathrm{TOP}_Q(T(v)) \cup \mathrm{SAT}_Q(T(v_\ell)) \cup \mathrm{SAT}_Q(T(v_r)) \ . \qquad (4.1)$$

(The equality follows from Lemma 2 and Observation 1.) For a set $A \subseteq Q$ of quartets, let $\mathtt{sum}(A) \stackrel{\mathrm{def}}{=} \sum_{q \in A} C_q$ denote the sum of their weights. The *score* of the subtree $T(v)$ (with respect to Q) is defined as

$$\mathtt{score}_Q(T(v)) \stackrel{\mathrm{def}}{=} \mathtt{sum}\left(\mathrm{SAT}_Q(T(v))\right) \ .$$

By Equation (4.1)

$$\mathtt{score}_Q(T(v)) = \mathtt{sum}(\mathrm{TOP}_Q(T(v))) + \mathtt{score}_Q(T(v_\ell)) + \mathtt{score}_Q(T(v_r)) \ . \quad (4.2)$$

Let S be a set of three or more taxa. Denote by $\mathtt{opt_score}_Q(S)$ the maximum score with respect to Q among all trees that have S as their set of leaves [1]. We denote by $\mathtt{opt_tree}_Q(S)$ a tree which attains the maximum score. For every proper partition of S into two subsets S_1 and S_2, let $T(S_1, S_2)$ denote a tree whose left subtree equals $\mathtt{opt_tree}_Q(S_1)$ and its right subtree equals $\mathtt{opt_tree}_Q(S_2)$. By equation (4.2), we have

$$\mathtt{score}_Q(T(S_1, S_2)) = \mathtt{sum}(\mathrm{TOP}_Q(T(S_1, S_2))) + \mathtt{opt_score}_Q(S_1) + \mathtt{opt_score}_Q(S_2) \ .$$

This implies that

$$\mathtt{opt_score}_Q(S) = \qquad (4.3)$$

$$\max_{S_1 \cup S_2 = S} \left(\mathtt{sum}(\mathrm{TOP}_Q(T(S_1, S_2))) + \mathtt{opt_score}_Q(S_1) + \mathtt{opt_score}_Q(S_2) \right) \ .$$

Let $\langle S_1, S_2 \rangle$ be a partition of S which attains the maximum, then $\mathtt{opt_tree}_Q(S)$ is defined as $T(S_1, S_2)$.

Equation (4.3) yields a recursive algorithm to compute the optimal tree (with respect to the given list of weighted quartets, Q): Given Q and S, go over all partitions $\{S_1, S_2\}$ of S, and choose a partition which maximizes

$$\mathtt{sum}(\mathrm{TOP}_Q(T(S_1, S_2))) + \mathtt{opt_score}_Q(S_1) + \mathtt{opt_score}_Q(S_2) \ .$$

This partition defines which taxa belong to the left subtree, and which to the right one. Apply the procedure recursively until each subtree has size smaller than 3. An optimal tree may be constructed by means of backtracking the partitioning steps.

The drawback of this recursive algorithm is that the score of each set S is recomputed whenever S is encountered as a subset in a partition. (Thus if S is of size i, its score will be computed 2^{n-i} times.) In order to avoid this wasteful repetition, we now employ the dynamic programming paradigm. We make a record of computed $\mathtt{opt_score}_Q(S)$ values, so that we will not have to recompute them. To do this, we scan the subsets $S \subseteq \{1, 2, \ldots, n\}$ by increasing size of S. This guarantees that in the computation for a set S, all subsets of S are already scanned over.

[1] By the definition, trees with one or two leaves do not contain any `qlca`, so for sets S of size 1 or 2 we define $\mathtt{opt_score}_Q(S) = 0$.

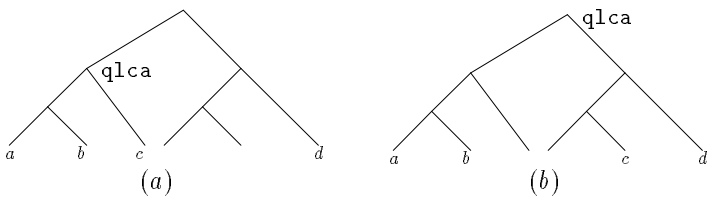


Figure 4.1: Two possible arrangements of a `qlca` for quartets over $\{a, b, c, d\}$.

Observation 1 *Every quartet q has a unique* $\texttt{qlca}(q)$.

Proof: Assume, towards a contradiction, that the quartet q has two distinct $\texttt{qlca}(\cdot)$ in T, p_1 and p_2. By definition 2 the two points cannot have an ancestor-descendant relationship. Therefore, the two must belong to two separate subtrees. Definition 3 then implies that each $T(p_i)$ contains at least 3 different taxa from q, so q must have at least 6 taxa — a contradiction. □

Lemma 2 *Given a tree T and a quartet q, the subtree rooted at* $\texttt{qlca}(q)$ *determines whether q is satisfied in the tree T.*

Proof: Let $q = ab|cd$ and $v = \texttt{qlca}(q)$. We look at v's children v_ℓ, v_r and the subtrees rooted at them $T(v_\ell)$, $T(v_r)$. At least one of these subtrees contains exactly two taxa e, f from $\{a, b, c, d\}$. The quartet q is satisfied by T iff the pair $\{e, f\}$ is either $\{a, b\}$ or $\{c, d\}$. □

Corollary 3 *Given a quartet $q = ab|cd$ and a tree T, let $v = \texttt{qlca}(q)$. Then T satisfies q if and only if at least one of the following holds:*

1. $\{a, b\} \subseteq L(T(s))$.
2. $\{c, d\} \subseteq L(T(s))$.

Where s is either v's left child ($s = v_\ell$) or v's right child ($s = v_r$).

We remark that the node in T where q's satisfaction is determined can in fact "be pushed" even further down, to every u which is on a path between an `lca` v of two taxa from u and $\texttt{qlca}(q)$ above v. There could be either one such node u, if we are in case (a) of Figure 4.1, or more nodes in case (b) of Figure 4.1. We postpone the "recognition event" to $\texttt{qlca}(q)$ because this choice facilitates the use of dynamic programming.

4.2 The Algorithm

Let Q be a fixed set of input quartets. Let T be a rooted tree, and v a node in T. We denote by $\mathrm{SAT}_Q(T(v))$ the set of quartets $q \in Q$ such that q is satisfied by T, and $\texttt{qlca}(q)$ is a node in $T(v)$. Let $\mathrm{TOP}_Q(T(v)) \subset \mathrm{SAT}_Q(T(v))$ be the

Chapter 4

The "Exact" Algorithm

Since the underlying problem is NP-hard we can not hope to give a polynomial time algorithm to solve it optimally. However, in this chapter we show how an optimal phylogenetic tree can be found, using a dynamic programming algorithm, with a modest exponential time. The method is applicable to instances with modest size (say $n \leq 21$). In particular, we have used it to find the optimal phylogenetic tree for the 15 taxa input set. We emphasize that even for 15 taxa, it is not feasible to exhaustively check all quartets for every possible bifurcating tree. (The number of unrooted trees for $n = 15$ is approximately $\approx 8 \cdot 10^{12}$, the number of quartets is approximately $1,300$, so the number of steps would be more than 10^{16}).

4.1 Definitions

The following discussion deals with *rooted* bifurcating trees. For a node v, its left and right children will be denoted by v_ℓ and v_r, respectively. Given a rooted tree T and a node v in it we denote by $T(v)$ the subtree of T rooted at v. We denote by $L(T)$ the set of leaves (*i.e.*, taxa) of the tree T. For a pair of nodes u, v the *least common ancestor* of u and v, $\texttt{lca}(u, v)$ is defined as an ancestor p of both u and v such that no node in $T(p)$ other than p is an ancestor of both u and v (see Figure 4.1).

Definition 2 *Given a quartet $q = ab|cd$ and a tree T, the* quartet *least common ancestor of q, $\texttt{qlca}(q)$ is defined as a node p that is the* `lca` *of two or more pairs of elements from $\{a, b, c, d\}$, and no node in $T(p)$ except p is the* `lca` *of two or more pairs of elements from $\{a, b, c, d\}$.*

For the implementation of our algorithm and the proof of its correctness, we will use an equivalent definition.

Definition 3 *Given a quartet $q = ab|cd$ and a tree T, the* `qlca` *of q is a node p such that*

1. *$|L(T(p)) \cap \{a, b, c, d\}| \geq 3$.*
2. *For any child s of p, $|L(T(s)) \cap \{a, b, c, d\}| \leq 2$.*

score of the tree. For example, we force the points to maintain some small pairwise distance by adding the $\binom{n}{2}$ constraints

$$\langle v_i, v_j \rangle \quad \leq \quad 1 - \varepsilon \qquad (1 \leq i < j \leq n)$$

for some positive ε (say $\varepsilon = 0.25$). This "represses" the SDP's tendency to find embeddings where points are very close to each other. This tendency may seem harmless, but it has a negative effect on the tree building method, which we describe in Section 3.3.

We have also experimented with strict constraints of other kinds. For example, requiring that the geometric distances between pairs of points in a quartet will obey the four-point condition [46] or approximate it. The failure of many of these formulations is widely attributed to the SDP process (i.e., infeasible program). As it turns out, the performance of the SDP degrades as the number of constraints to be taken into account increases.

3.3 Geometrical Clustering

Having solved the SDP problem, one seeks a tree that reflects the geometric data. For instance, if two points reside in the same geometric vicinity, they should have a common ancestor which is not too much high up the resulting tree. To this end, we employ a simple neighbor joining clustering heuristic (see [38]). The program is initialized with n clusters, each containing a single point. An invariant kept by the algorithm is that each cluster has a point in $\mathcal{R}^n$ associated with it. This is the case at initialization, and remains true as new clusters are formed. At each step of the algorithm the number of clusters is decreased by one, by removing two clusters and adding one. This defines a tree, as the newly-added node represents the father of the two deleted nodes in the output tree. The selection of the clusters to be removed is done by calculating the pairwise distances between clusters, and selecting the pair having the shortest Euclidean distance. The point associated with the new cluster is the center of mass of the points of the removed clusters (the "mass" of a point being the number of taxa it represents). When the number of clusters reaches one, the tree is completed. The resulting tree is rooted, but we disregard the rooting, since the input is inherently unrooted. To root the tree we use external information – the identity of the outgroup taxon. We root the tree by forcing the outgroup to be an immediate descendent of the root. We remark that this heuristic is a special case of the general neighbor joining approach. The main difference is that since clusters are associated with points in Euclidean n space, the center of mass of the nearest neighbor pair is a natural choice for the new cluster. This choice again induces Euclidean distances between the new cluster and the "old" ones.

matrix V as n vectors $v_1, \ldots, v_n$ in $\mathcal{R}^n$. Now, the variables x_{ij} of the matrix X simply correspond to $\langle v_i, v_i \rangle$, the inner product of v_i and v_j. Thus a linear constraint on the x_{ij}'s is simply a linear constraint on the inner products of the vectors v_i's. The objective function and the constraints are simply linear combinations of the inner products. Hence we can use SDP to solve any problem of the form:

> Find n vectors $v_1, \ldots, v_n \in \mathcal{R}^n$ so as to maximize the quantity $\sum_{i,j} c_{ij} \langle v_i, v_j \rangle$, subject to the constraints $\sum_{i,j} a_{ij}^{(k)} \langle v_i, v_j \rangle \leq b^{(k)}$.

In what follows we will use the last interpretation of SDP to find an embedding of n points which represent the n taxa.

3.2 Geometric Embedding of Taxa

We will generally denote the embedding of taxon i by the point v_i in $\mathcal{R}^n$. Given a list of k quartets $a_j b_j | c_j d_j$ and confidence values C_j $(j = 1, \ldots, k)$, we solve the following semidefinite program:

maximize:

$$\begin{array}{ll} & \sum_{1 \leq j \leq k} C_j (\langle a_j, b_j \rangle + \langle c_j, d_j \rangle) \\ -0.5 & \sum_{1 \leq j \leq k} C_j (\langle a_j, c_j \rangle + \langle a_j, d_j \rangle + \langle b_j, c_j \rangle + \langle b_j, d_j \rangle) \end{array}$$

such that

$$\langle v_i, v_i \rangle \quad = \quad 1 \qquad (1 \leq i \leq n)$$

This formulation embodies many local constraints, such as "taxa a and b should be close together", or "taxa a and d should be far apart", into a single expression. This approach is similar to the one used in [23] to approximate MAX CUT. Consider the quartet $ab|cd$ with confidence level C. Its maximum contribution to the objective function occurs when a and b are placed at the same point, while both c and d are placed at the antipodean point on the unit sphere. In this case $\langle a, b \rangle = \langle c, d \rangle = 1$, while $\langle a, c \rangle = \langle a, d \rangle = \langle b, c \rangle = \langle b, d \rangle = -1$. The overall contribution of the quartet will be $4C$ with this embedding. The worst embedding places a and c together, and b and d at the antipodean point. This will contribute $-4C$ to the objective function. The semidefinite program will therefore look for a global embedding which maximizes "good" quartet placements and avoids "bad" ones.

We have experimented with this approach, and it turned out that small variants of it are helpful in improving the final result (the produced tree). One such variant is ignoring quartets with low confidence in the objective function. Only quartets with confidence level above a certain threshold (90% for example) are included. One possible explanation why this may prove helpful is that low confidence quartets probably carry most of the inconsistencies, and their inclusion may lower the value of the objective function. We have also discovered that by imposing additional constraints we get (small) improvements in the

Chapter 3

The Geometric Algorithm

Our geometric algorithm assigns each taxa a point in $\mathcal{R}^n$. This is done by applying the semidefinite programming method, to be described shortly. The embedding is built using "hints" from the input set. For example, if the quartet $ab|cd$ is in the list, we try to place a and b close to each other, but a and d far apart. Once the points are embedded in $\mathcal{R}^n$, a clustering heuristic converts the embedding into a tree.

3.1 Semidefinite Programming

We briefly explain the paradigm of SDP — semidefinite programming (see [29] for a thorough explanation). We describe how SDP can be augmented by the incomplete Cholesky decomposition method to find embeddings of points in $\mathcal{R}^n$, subject to certain constraints.

Definition 1: For positive integers m and n, a semidefinite program is defined over a collection of n^2 real variables $\{x_{ij}\}_{i=1,j=1}^{n,n}$. The input consists of a set of mn^2 real numbers $\{a_{ij}^{(k)}\}_{i=1,j=1,k=1}^{n,n,m}$, a vector of m real numbers $\{b^{(k)}\}_{k=1}^{m}$ and a vector of n^2 real numbers $\{c_{ij}\}_{i=1,j=1}^{n,n}$. The objective is to find $\{x_{ij}\}_{i=1,j=1}^{n,n}$ so as to

$$\begin{array}{rc} \text{maximize} & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\ \text{subject to} & \\ \forall k \in \{1,\ldots,m\} & \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^{(k)}x_{ij} \leq b^{(k)}. \\ \text{and} & \text{The matrix } X = \{x_{ij}\} \text{ is symmetric} \\ & \text{and positive semidefinite.} \end{array}$$

A symmetric matrix X is positive semidefinite when there exists a real matrix V such that $V^T \cdot V = X$. Given a positive semidefinite matrix X, such V can be found in polynomial time using incomplete Cholesky decomposition. The semidefinite program itself is solvable in time polynomial in the size of its input [29].

The matrix V can be used to interpret the solution obtained by the semidefinite programming problem geometrically. Interpret the columns of the $n \times n$

Williamson [23]. Software packages for SDP and for the associated incomplete Cholesky decomposition are publicly available [45, 21, 42].

size four. The next stage combines the multiple topologies (called *quartets*) into a single phylogeny. Given a complete set of correct quartets, the true tree can be constructed in polynomial time. However, doing so when the individual topologies may contain errors is NP-hard [41]. Published quartet methods include the so-called "naive" method, [46, p. 25], the Buneman tree [7, 4], split decomposition [46], the short quartet method [12], neighbor-joining variants [26, 27, 28], and quartet puzzling [43]. See also [3, 17]. In general, these heuristics deal with unweighted quartets (i.e., each quartet has the same significance). Some also require as input the full set of $\binom{n}{4}$ quartets. In principle, the quartet puzzling algorithm can be extended to the unweighted general case, but no experiments in this direction have been done.

2.4 Evaluation

The strength of a phylogeny reconstruction method may be measured in several ways. A method is called *consistent* if it converges to the true tree as the length of the input sequences tends to infinity. Because arbitrary long sequences are not available in real life, an important measure of a consistent methods is its *convergence rate*. A method is said to be *robust* if it can handle violations of the underlying model. Distance-based methods are called *combinatorially consistent* if they recover the true tree whenever the input matrix is additive, and *continuous* if small changes in the input incur small changes of the output. For these and other definitions see [46].

Given a "true" evolutionary tree and a phylogeny which tries to approximate it, one can evaluate the quality of the phylogeny either topologically or numerically. The topological test compares the sets of partitions induced by tree edges in both trees. In contrast, a numerical approach examines the similarity between the two distance matrices under some metric. It is important to note that the "true" tree is not available to us for real-life instances of evolutionary problems. Therefore a comparison is usually made either to simulated trees, or between the outputs of two different methods.

2.5 Semidefinite Programming

Semidefinite programming is an optimization problem. The program variables are the elements of a real $n \times n$ symmetric matrix, with the additional demand that the matrix be symmetric and positive semidefinite (see Section 3.1). The objective function, as well as the program constraints are linear combinations of the variables. In the past two decades SDP has been used to solve and analyze many graph and combinatorial problems [2]. Being solvable in polynomial time, algorithms developed using it are of practical importance. One popular algorithmic approach is to view the columns of the variable matrix as points in $\mathcal{R}^n$. Under this representation, SDP embeds n points in n-space while maximizing a linear combination of the inner products of the associated vectors; the constraints are also linear combinations of the inner products. Most notable in the current context is the 0.87-approximation to MAX-CUT by Goemans and

methods. The general maximum parsimony problem is NP-hard [9, 20, 30], but finding the internal labeling for a given topology can be done efficiently [18, 31]. This leads to algorithms which scan many different topologies and output the most parsimonious one.

Another approach tries to construct a tree such that for each character, the node-sets that correspond to any character state form a connected subgraph. Maximizing the number of characters for which this is true is called the *maximum compatibility* problem. This problem, too, is NP-hard [10, 37, 46]. The point of intersection between maximum parsimony and maximum compatibility is called the *perfect phylogeny* problem. It decides whether all of the characters are compatible. Perfect phylogeny was shown to be NP-hard by equivalence to the *triangulating colored graphs* problem [5, 41]. Although hard in the general case, polynomial-time algorithms for perfect phylogeny exist in cases where some of the input parameters (e.g., the number of states) are fixed [46].

2.2 Distance-Based methods

A distance based phylogeny method would typically take as input a symmetric zero-diagonal $|S| \times |S|$ matrix. Here, too, the goal is to produce a phylogeny whose induced metric represents the input data in the best way possible. If the input distance matrix M is realizable by a tree and its induced path lengths, then M is said to be *additive*. The special case where all leaves have the same distance from the root is called an *ultrametric*. Ultrametric trees correspond to the biological theory that substitutional events in different species occur at the same rate (this is the "molecular clock" assumption, nowadays popularly discredited).

Given an additive metric, constructing the tree is easy [47]. The problem is that real-life input is erroneous. Some of the errors are inherent in the assumed model of evolution. Therefore, we seek phylogenies with induced metric which approximates the "best possible" tree metric under some criterion. Again, most of the problems in this domain are NP-hard.

One popular heuristic approach is *agglomerative clustering*. Here, at each iteration two nodes are joined into one father node. Examples include the popular *Neighbor Joining* method [19], the *Fitch-Margoliash* method [19], and BIONJ [22].

As it turns out, approximating ultrametrics under the L_∞ criterion is doable in polynomial time [24, 39, 13, 46]. Consequently, there are algorithms that use an approximated ultrametric to produce an approximation of the additive metric [1, 8].

2.3 Quartet Methods

The fact that small phylogenies are easier to infer than large ones leads to another approach. The key idea is to consider small subsets of taxa, one at a time, and infer the phylogenies for these subsets. All methods use subsets of

Chapter 2

Prior Work

Phylogeny reconstruction is an old problem; it has been investigated for over 150 years. Consequently, the related literature is vast. The algorithmic challenges that follow the fast developments in the bimolecular field have recently drawn the attention of computer scientists as well. In this chapter we merely quote the major computer-theoretic research issues related to phylogeny reconstruction, and give selected references.

In general, phylogeny methods are divided into *character-based* and *distance-based* methods. We will consider each in turn. An orthogonal classification identifies the *quartet method* as one possible algorithmic approach. Being in the general framework of this work, the existing quartet methods will also be listed. We also browse through the different techniques to evaluate the quality of a phylogeny. Another important aspect of this work is the usage of the *semidefinite programming* tool. We bring some related pointers in this regard as well.

2.1 Character-Based methods

A character-based method considers qualitative characters of the input taxa. Any such character is a partition of the input set according to the value each taxon takes. Each equivalence class defined thus is called a *character state*. For example, a DNA sequence is composed of the nucleotides A, C, T and G. Sequences of different species are aligned in a *multiple sequence alignment* process which inserts gaps into the sequences so as to maximize the resemblance of the sequences to each other when laid out side by side. Thus, each position of the aligned sequence (called *site*) is a character with five states (A, C, T, G, or the gap symbol -). Input for such a method will typically be a $|S| \times |C|$ matrix, where S is the taxa set and C is the character set. Each entry denotes a state which a particular taxon exhibits for a given character.

Given sequences for each taxon, a natural approach is to build a tree with the internal nodes, as well as the external ones, labeled by the sequences; the labels at the leaves are given as input, and the internal labels are computed. The optimization criterion is the sum of Hamming distances between neighboring sequences. Methods that try to minimize it are called *maximum parsimony*

technique that we call *geometric clustering*. This technique is similar to the traditional neighbor joining [38], but differs in the phase where the metric is updated. At each iteration, we join the pair of points with minimum Euclidean distance, and take their center of mass as the common ancestor. This defines the new metric naturally. We have implemented this algorithm on input of 15 taxa (14 mammalian orders and an outgroup taxon) that was derived from the HOVERGEN database [11]. The resulting tree satisfies 72.1% of the weighted quartets. This is an improvement over all previously published trees (see Table 5.1).

Even for taxa sets of modest size the number of trees is too large to enable an exhaustive search for the best one (for $n = 15$ it is just below 8×10^{12}). We have developed an exponential time, exact algorithm, which finds an optimal tree using the dynamic-programming paradigm. Its running time is $O(k \cdot 3^n)$, where $k \leq \binom{n}{4}$ is the number of quartets in the input. This is a fairly modest exponential growth, and it enabled us to implement and run the algorithm on the same 15 taxa input set. The optimal tree satisfies 73.4% of the weighted quartets. It is interesting that the geometric algorithm, which is essentially a heuristic with no performance guarantees, came so close to the optimal result.

The remainder of this work is organized as following: Chapter 2 surveys some of the past research done in the areas of phylogeny and semidefinite programming. Chapter 3 explains the geometric algorithm, while Chapter 4 explains the exact algorithm. Chapter 5 describes the computational results — both on actual input and in simulations. Chapter 6 contains some concluding remarks.

so their two adjacent nodes become connected. This process is repeated until no internal nodes of degree two are left. It is easy to see that there are four possible induced topologies for the quadruple — three quartets and the star topology (the latter can be induced only by a multifurcating tree). Given a specific quartet and the induced four taxa subtree, we say that the quartet is *unresolved* if the tree induces the star topology on the four taxa. Otherwise, the quartet is either *satisfied* (if the topology induced by the tree is the same as the quartet's topology), or else *violated*.

Given a tree T and a set of quarters Q, we would like to know how well does T represent Q. To do this, we find the subset of quartets $S \subset Q$ that are satisfied by T, and the subset of quartets $U \subset Q$ that are unresolved by T. We now define the *score* of the tree as follows:

$$\sum_{s \in S} C_s + \frac{1}{3} \sum_{u \in U} C_u \ .$$

That is, we add the confidence weights of the satisfied quartets, plus one third of the weights of the unresolved quartets. This latter term was chosen because there are three possible pairings for every quadruple. Therefore this term equals the expected increase to the tree score that will result from a random bifurcation of the tree (performed at nodes with more than two descendents). In a different variant of our method the score is zeroed, so unresolved quartets do not contribute to the score. We remark that in the history of phylogeny many multifurcating trees were published. However, modern algorithms, including ours, attempt to produce a bifurcating tree.

An upper bound on the score of any tree is $\sum_{q \in Q} C_q$. This upper bound can be achieved only if there exists a tree that satisfies all quartets. We usually state the score of a tree as a fraction of the upper bound, although it is possible that no tree of score 100% exists.

We are now ready to formulate the problem precisely. Given a set of quartets Q and associated confidence scores, find a tree T with maximal score[2]. This problem is NP–hard even if all the confidence weights have values 0 or 1 [41]. Exhaustive search methods seem infeasible even for modest values of n, because the number of unrooted bifurcating trees [14] with n leaves is

$$\frac{(2n-5)!}{2^{n-3}(n-3)!} \ .$$

Our first algorithm has the following geometric intuition: We embed the n taxa as n points in $\mathcal{R}^n$, while trying to preserve distance relations that are implied by the input quartets. To find such embedding[3], we employ semidefinite programming. Specifically, the points are embedded on the n-dimensional unit sphere. To construct a tree from the embedding, we employ an iterative

[2]One may consider a similar problem where for every 4-tuple of taxa, all three quartet topologies are given in the input (with different weights). Small modifications of both the geometric and the exact algorithm will solve this variant as well.

[3]A related notion of low distortion Euclidean embeddings for weighted trees has been studied in [6, 34].

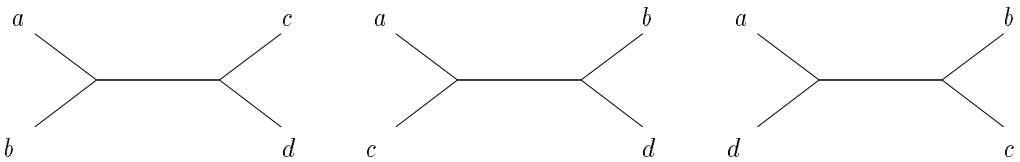


Figure 1.1: The three possible quartets for the species set $\{a, b, c, d\}$

topology is chosen. After determining quartet topologies, these quartets should be merged into a single, coherent, phylogeny.

Combinatorially, the justification for the quartet approach is that if all $\binom{n}{4}$ quartets are given with the correct topology, then the underlying tree is uniquely determined, and can be efficiently reconstructed. Of course, in reality there are errors, so a tree consistent with all given quartets may not exist[1]. Hence the goal is to construct a binary tree with n leaves such that the total weight of the satisfied quartets is maximized (an NP hard problem). Our contribution in this research is in suggesting new algorithmic approaches to solve this optimization problem. We develop two algorithms: a "geometric" algorithm and an "exact" algorithm.

More precisely, the problem is defined over a set of n taxa, numbered $1, \ldots, n$. A *quartet* is a quadruple of taxa, with an associated topology — a partition of the four taxa into two pairs of taxa (*e.g.* $\{a, b\}$ and $\{c, d\}$). This subdivision expresses the most likely topology induced by the underlying n-taxa phylogeny. Such a quartet is denoted $ab|cd$. Figure 1.1 shows the three possible topologies over a given set of 4 species. The input to the problem consists of a set of $k \leq \binom{n}{4}$ such quartets. We denote the associated taxa for the j-th quartet by $a_j b_j | c_j d_j$. No two quartets share the same set of four taxa. Each input quartet is accompanied by a positive weight, denoted by C_j, which represents the confidence in the quartet topology.

The output is an unrooted tree with n leaves, which correspond to the input taxa. Finding the root is considered a different problem altogether and modern phylogeny methods produce unrooted trees. The common workaround is to use an *outgroup*, namely an external taxon, not belonging to the same family of taxa. Using the outgroup taxon, the tree can be (uniquely) rooted such that the outgroup is an immediate descendent of the root. Once the tree is rooted, the branching of its nodes is viewed as a description of the evolution of the n taxa. If all internal nodes of a tree have out-degree 2, the tree is called *bifurcating*. Otherwise, the tree is said to be *multifurcating*.

Given a tree T and a quadruple $\{a, b, c, d\}$, we can compute the quartet topology induced by T, using the following procedure. First, all leaves but a, b, c and d are deleted from the tree. Edges adjacent to deleted leaves are also removed. Next, internal nodes with degree two are contracted and deleted,

[1] Determining if a given subset of quartets is satisfiable is an NP complete problem [41].

Chapter 1

Introduction

Given a set of taxa, typically biological species, the aim of phylogeny reconstruction is to build a tree which models the course of evolution and specification of the set over time. The leaves of the trees are labeled with the given (i.e. extant) taxa, and the internal nodes denote extinct, or hypothesized, taxa. The root of the tree is a common ancestor of every taxon.

Molecular data are the basis for current phylogenetic analysis of biological evolution. These usually are in the form of homologous proteins or DNA sequences of different species. Ideally, we would have each sequence available for every species. In reality, this idealized scenario hardly occurs. A small number of species (human for one) focus research attention and have much sequence data available for them. This is also true on the sequence axis. A small number of sequences (e.g., hemoglobin) are available for many species. However, for the most part, the data is sporadic. Given a protein, probably just a small fraction of the species in question will have it sequenced. This affects phylogeny reconstruction, because current phylogeny methods require all aligned sequences as their input. The trade-off a biologist faces is between the number of taxa and the amount of molecular data used in a study. If one restricts oneself only to sequences that are common to all the taxa under study, one may end up ignoring the vast majority of data. If, on the other hand, one increases the amount of molecular data used in a study, one ends up seriously restricting taxonomic sampling (i.e. study fewer taxa).

An approach that tries to utilize most available data while avoiding the problem of taxonomic sampling is the *quartet approach*, suggested in [26] and used in [27, 28, 12, 43]. The method requires that common sequences be available just for 4-taxon subsets of the original taxa set. This increases the amount of data in use, since different sequences may be used for different subsets. The topology of each 4-taxon tree (quartet) is resolved and augmented with a measure of reliability or confidence (e.g., bootstrap value, likelihood) associated with the internal branch of the tree. This confidence value reflects two factors: how solid is the plurality of sequences supporting the conclusion (i.e., the strength of the phylogenetic signal), and what is the size of the sequence population used to build the tree. For each group of 4 taxa, there are three possible unrooted phylogenetic trees. In this type of study, only the most supported

Notations and Abbreviations

$d_T(x, y)$ — Tree distance between leaves x, y.
$\langle x, y \rangle$ — Inner product of vectors x and y: $\sum_{i=1}^{n} x_i \cdot y_i$.
C_e — Evaluated confidence level .
C_r — Confidence level as reported to program.
$\texttt{score}_Q(T)$ — Total weight of quartets in Q satisfied by tree T.
T — A rooted tree.
$T(v)$ — Subtree of T rooted at node v.
$L(T)$ — The leaves of T.
$\texttt{lca}(u, v)$ — Least common ancestor of nodes u and v.
v_l — Left child of node v.
v_r — Right child of node v.
$ab|cd$ — A quartet separating taxa a, b from c, d.
q — A quartet.
$\texttt{qlca}(q)$ — Least common ancestor of any two pairs of elements of quartet.

Abstract

In this work we present two new approaches for constructing phylogenetic trees. The input is a list of weighted quartets over n taxa. Each quartet is a subtree on four taxa, and its weight represents a confidence level for the specific topology. The goal is to construct a binary tree with n leaves such that the total weight of the satisfied quartets is maximized (an NP hard problem).

The first approach we present is based on geometric ideas. Using semidefinite programming, we embed the n points on the n-dimensional unit sphere, while maximizing an objective function. This function depends on Euclidean distances between the four points, and reflects the quartet topology. Given the embedding, we construct a binary tree by performing *geometric clustering*. This process is similar to the traditional neighbor joining, with the difference that the update phase retains geometric meaning: When two neighbors are joined together, their common ancestor is taken to be the center of mass of the original points.

The geometric algorithm runs in $poly(n)$ time, but there are no guarantees on the quality of its output. In contrast, our second algorithm is based on dynamic programming, and it is guaranteed to find the optimal tree (with respect to the given quartets). Its running time is a modest exponential, so it can be implemented for modest values of n.

We have implemented both algorithms, and ran them on real data for $n = 15$ taxa (14 mammalian orders and an outgroup taxon). The two resulting trees improve previously published trees and seem to be of biological relevance. On this dataset, the geometric algorithm produced a tree whose score is 98.2% of the optimal value on this input set (72.1% vs. 73.4%). In addition, we have carried out experiments to test the algorithm's ability to reconstruct simulated trees. The score of such a reconstructed tree is over 85% on the average. Both these facts give rise to the hope that the geometric approach will prove viable even for larger cases where the exponential, dynamic programming approach is no longer feasible.

List of Figures

1.1 The three possible quartets for a given species set 4

4.1 Two possible arrangements of a `qlca` 15

5.1 Phylogenetic trees obtained by the geometric and optimal methods 22

5.2 Example of the calculation of confidence level 25

List of Tables

5.1 Comparison of published trees 23
5.2 Run times for the "exact" algorithm 23
5.3 Summary of simulation results 26
5.4 Effect of error model on quality of trees 27
5.5 Effect of disregarding low-confidence quartets 27

Contents

Abstract 1

Notations and Abbreviations 2

1 Introduction 3

2 Prior Work 7
- 2.1 Character-Based methods . . . 7
- 2.2 Distance-Based methods . . . 8
- 2.3 Quartet Methods . . . 8
- 2.4 Evaluation . . . 9
- 2.5 Semidefinite Programming . . . 9

3 The Geometric Algorithm 11
- 3.1 Semidefinite Programming . . . 11
- 3.2 Geometric Embedding of Taxa . . . 12
- 3.3 Geometrical Clustering . . . 13

4 The "Exact" Algorithm 14
- 4.1 Definitions . . . 14
- 4.2 The Algorithm . . . 15
- 4.3 The Code . . . 17
- 4.4 Algorithm Complexity . . . 17
  - 4.4.1 Ramifications . . . 18

5 Computational Results 21
- 5.1 Real Data . . . 21
- 5.2 Simulations . . . 24

6 Discussion 28

This research was carried out in the Faculty of Computer Science
under the supervision of Prof. Benny Chor.

To my grandmother Ahuva and my grandfather Eliezer.

My deepest thanks go to Benny Chor. Benny has the rare ability to provide invaluable guidance in the most friendly atmosphere possible. For this reason our meetings were always so pleasant and productive. I wish to thank Dan Graur for sharing his vast biological knowledge and helping us computer freaks generate something useful "for once". In gratitude to Amir Ben-Dor, a genuine researcher and a good friend.

The generous financial help of the Technion is gratefully acknowledged.

Algorithms for Constructing Phylogenies from Quartets

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

Dan Pelleg

Submitted to the Senate of the Technion — Israel Institute of Technology
Nissan, 5758 Haifa April 1998

Algorithms for Constructing Phylogenies from Quartets

Dan Pelleg