Computational Aspects of Inferring Evolutionary History

Jesper Jansson

June, 1999

Department of Computer Science

Lund University
Lund Institute of Technology
Lund, Sweden

Abstract

This Licentiate thesis consists of an introduction and three articles related to computational biology. The main theme is combinatorial problems involving evolutionary trees.

The first problem considered, the maximum homeomorphic agreement subtree problem (MHT), amounts to comparing different evolutionary trees for a fixed set of species and identifying a subtree common to all of them. The next problem, the maximum inferred consensus tree problem (MICT), suggests a topological method for tree construction; a few variants of MICT are also studied. Finally, in the Hamming center problem (HCP), we want to find an unbiased representation for a set of strings of equal length. HCP is a special case of a problem in which all internal nodes of a given evolutionary tree have to be labeled.

In the first article included in the thesis, we present some new results on the inapproximability of MHT and the computational complexity of MICT. The second article deals with a variant of MICT in which a left-right ordering is imposed upon the species. The third article contains a randomized approximation algorithm for HCP.

Preface

This is a thesis for a Licentiate degree (a Swedish degree halfway between M.S. and Ph.D.) in computer science. It consists of an introduction and three articles related to computational biology.

The articles presented here have been or will be published in:

- L. Gąsieniec, J. Jansson, A. Lingas, and A. Östlin. On the Complexity of Constructing Evolutionary Trees. To appear in the *Journal of Combinatorial Optimization*'s forthcoming special issue on Computational Molecular Biology, 1999.
- L. Gąsieniec, J. Jansson, A. Lingas, and A. Östlin. Inferring Ordered Trees from Local Constraints. *Proceedings of Computing: the Fourth Australasian Theory Symposium* (CATS'98), Perth, Springer-Verlag Singapore Pte. Ltd., 1998, pp. 67–76.
- L. Gąsieniec, J. Jansson, and A. Lingas. Efficient Approximation Algorithms for the Hamming Center Problem. Technical Report LU-CS-TR:99-211, Department of Computer Science, Lund University, 1999.

The first article is a slightly improved version of:

- L. Gąsieniec, J. Jansson, A. Lingas, and A. Östlin. On the Complexity of Computing Evolutionary Trees. *Proceedings of the Third Annual International Computing and Combinatorics Conference* (COCOON'97), Shanghai, LNCS 1276, Springer-Verlag Berlin Heidelberg, 1997, pp. 134–145.

A preliminary version of the third article was published as a short form paper in:

- L. Gąsieniec, J. Jansson, and A. Lingas. Efficient Approximation Algorithms for the Hamming Center Problem. *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA'99), Baltimore, 1999, pp. S905–S906.

Acknowledgments

I would like to thank Rolf Karlsson who introduced me to algorithm theory for convincing me to pursue my graduate studies in theoretical computer science, Andrzej Lingas for invaluable advice and guidance and for sharing his broad experience, Leszek Gąsieniec for providing challenging problems to work on, and Anna Östlin for successful collaboration.

I would also like to express my gratitude to everybody else at the Department of Computer Science at Lund University for inspiration and for being around. For interesting lunch time discussions, I thank Mikael Hammar, Anna Torstensson, Jonas Månsson, and Björn Brodén. Finally, I thank my parents and other relatives, all my friends, and especially Veena Andersson for encouragement and moral support.

Contents

PART I **1**

Introduction 1

Computational biology 1

Evolutionary trees 2

Thesis outline 4

The maximum homeomorphic agreement subtree problem (MHT) . . . 5

The maximum inferred consensus tree problem (MICT) 6

The maximum inferred local consensus tree problem (MILCT) 7

The ordered local consensus tree problem (OLCT) 7

The maximum ordered local consensus tree problem (MOLCT) 7

The Hamming center problem (HCP) 8

New results 9

References 10

PART II **13**

On the Complexity of Constructing Evolutionary Trees 13

PART III **31**

Inferring Ordered Trees from Local Constraints 31

PART IV **43**

Efficient Approximation Algorithms for the Hamming Center Problem . . . 43

Introduction

Computational biology

Computational biology is a relatively new, interdisciplinary field of science which combines techniques from mathematics, statistics, and computer science to help solve problems in modern molecular biology. Computers are used to collect, store, and process information obtained from biological experiments; however, since many of the problems are of a combinatorial nature or involve vast amounts of data, brute force calculations alone do not suffice. The need for efficient algorithms has become increasingly evident, and consequently, algorithm theory is now an essential component of computational biology.

Strings are central to molecular biology. The primary structure of a protein is a one-dimensional chain of amino acid residues, and therefore expressible as a string over a finite alphabet. Similarly, the linear sequence of bases in a DNA molecule can be represented by a string over the alphabet {A,C,G,T}. Thus, computational problems in molecular biology often reduce to string manipulation problems that may be solved by methods originally developed in other areas of computer science. For example, many problems related to comparing or aligning molecular sequences admit efficient solutions based on dynamic programming, a fundamental algorithmic technique [6, 10, 12]. Another example is a data structure known as the suffix tree, which has been successfully applied to various string problems such as recognizing repetitive structures like periods, palindromes, and other regularities, finding the longest common substring of two strings, inexact pattern matching, primer selection, and discovering DNA contamination [6].

Computational biology has also been influenced by graph theory. In general, combinatorial relationships between objects are naturally described by graphs (represent the objects as vertices in a graph, and connect the vertices according to how the objects are related), which means that certain types of problems can be succinctly formulated in terms of graphs. Historically, graph theory has contributed to a wide range of disciplines; electrical engineering, telecommunications, chemistry, sociology, operations research, and abstract algebra are but a few! More specifically, in computational biology, interval graphs have been used to identify overlaps of gene fragments and to construct restriction maps that show the locations of certain short specific sequences called restriction sites along DNA [10, 12]. However, the most important application of graph theory to computational biology is probably as a tool to model evolution.

Why would anybody want to model evolution in the first place? There are two good reasons. First of all, before the mechanisms of evolution can be explained, its effects must be examined and well understood. By inferring the evolutionary history of different sets of species, scientists can deduce in which order and at what rate genetic changes have taken place, which yields clues about the causes of evolution. Secondly, classifications of species based on their evolutionary history are indicative and meaningful, and help scientists organize and exchange information.

Evolutionary trees

An *evolutionary tree* (or *phylogenetic tree*, or *phylogeny*) is an unordered tree whose leaves are labeled by distinct species. Internal nodes represent common ancestors, and ideally, the topological structure of the tree reflects the species' evolutionary relationships. Sometimes weights are assigned to the edges to illustrate evolutionary distance, i.e., estimates of the time taken for ancestral species to evolve into other species. In some settings, the data does not uniquely determine a root, which leads to unrooted (as opposed to rooted) trees. See Figure 1 for an example of a rooted, unweighted evolutionary tree.

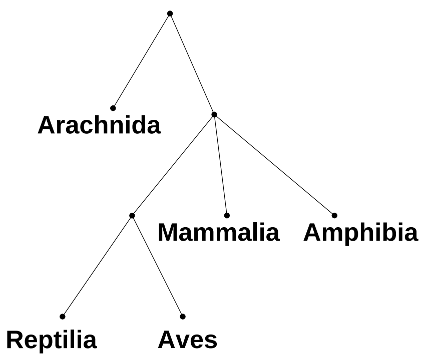

Figure 1: A rooted, unweighted evolutionary tree for five species.

Note that the "species" do not necessarily have to be distinct biological species; they might be entire populations or categories of species, or other entities subject to evolution such as proteins, nucleic acids, or even languages. In fact, one very famous paper about weighted evolutionary trees was published in a volume on archaeology [1]. As an additional example, a team of researchers from universities in the United States and Great Britain is working on a project aimed at removing discrepancies between 58 surviving manuscripts of Chaucer's *The Canterbury Tales*. Assuming that various mutation processes took place every time a manuscript was copied, they attempt to trace variations in the texts in order to generate a version as close to Chaucer's original intentions as possible by using evolutionary trees [2].

To describe evolutionary processes with graph structures is an old idea. Long before the present-day theories of molecular biology, Charles Darwin, Edward Blyth, Ernst Haeckel, and other 19th century natural scientists compared the development of different species originating from a common ancestor to a branching tree. Initially, biologists built evolutionary trees from data based on observable morphological features and intuition; later, biochemical data and results from molecular sequence comparisons became available as well. The assumptions behind what distinguishes good optimization criteria vary according to the particular application and the type of data[1], which explains why so many kinds of evolutionary trees and methods for reconstructing evolutionary history have been invented. No single method appears to work well all the time, so different methods are still common.

Existing techniques for inferring evolutionary trees can be divided into:

Character-state methods: Describe each species by a vector of character states and look for a tree that clusters species with a lot in common.

Distance-based methods: Compute the evolutionary distance between every pair of species, then try to build an edge-weighted tree which complies with these values.

Maximum likelihood methods: View evolution as a stochastic process and use statistical methods to find the most probable tree.

Topological methods: Combine knowledge about local topologies for small sets of species into a larger tree.

For a comprehensive description of these different approaches, please refer to [7] and [9].

Many of the evolutionary tree construction algorithms have been implemented in PHYLIP, a popular package of computer programs that is available for free on the Internet [5]. [11] and [13] contain links to PHYLIP and numerous other software packages for phylogenetic analysis.

[1] For instance, the parsimony principle (which says that one should attempt to minimize the number of state changes induced by the constructed tree) is often used when treating morphological data but may be somewhat less appropriate for DNA sequence data. This is because DNA sequence positions do not evolve independently and because changes during evolution in DNA are much more frequent than changes in morphological characters, implying that back substitutions should be expected.

Thesis outline

In the articles comprising this thesis, we study some abstracted combinatorial problems involving evolutionary trees. The number of different evolutionary trees is enormous even for small sets of species[2]. Therefore, it is not feasible to solve tree optimization problems by exhaustive search. Instead, other methods should be employed if possible. Generally, it is essential to fully understand a problem's complexity to know under what circumstances efficient algorithms are possible; if the problem can be proved to be computationally difficult, one then knows that it is wiser to concentrate on approximation algorithms or to look at special cases of the problem than to spend time trying to find a non-existent fast, exact algorithm.

The following problems are considered:

- The maximum homeomorphic agreement subtree problem (MHT)
- The maximum inferred consensus tree problem (MICT)
- The maximum inferred local consensus tree problem (MILCT)
- The ordered local consensus tree problem (OLCT)
- The maximum ordered local consensus tree problem (MOLCT)
- The Hamming center problem (HCP)

MHT, MICT, and MILCT are treated in the first article, "On the Complexity of Constructing Evolutionary Trees"; OLCT and MOLCT in the second, "Inferring Ordered Trees from Local Constraints"; and HCP in the third, "Efficient Approximation Algorithms for the Hamming Center Problem".

Listed below are formal descriptions of all the problems, along with some comments. A short summary of our results concludes this section.

[2]There are $\prod_{i=2}^{n}(2i-3) = (2n-3)!!$ different unordered, rooted, binary trees leaf-labeled by a set of n species, and even more unordered, rooted trees with arbitrary degrees. Note that $(2n-3)!! > (2n-4)!! = \frac{1}{4} \cdot 2^n (n-2)!$, which is superexponential in n.

The maximum homeomorphic agreement subtree problem (MHT)

Input: k rooted trees $T_1, T_2, ..., T_k$ for a set A of n species.

Output: A largest subset B of A such that the restrictions of $T_1, T_2, ..., T_k$ to B are topologically identical (the *restriction* of T_i to B is the tree obtained from T_i by deleting all leaves labeling species that are not in B and eliminating internal nodes with only one child).

Figure 2 shows an example. Obviously, MHT is hard to compute by hand if k and n are large.

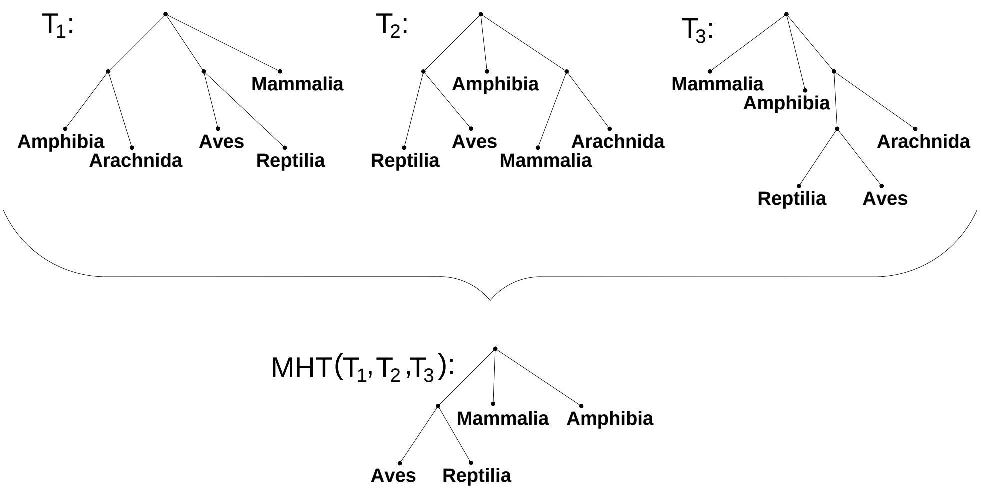

Figure 2: Three trees and their maximum homeomorphic agreement subtree.

The main motivation for studying MHT is the following: Assume somebody has already come up with a number of evolutionary trees for a fixed set of species by applying different tree construction methods or clustering criteria to the data (as mentioned above, there is no definitive standard method), or by conducting repeated experiments. Furthermore, these trees do not agree because of distortions due to assumptions inherent to the methods used or because of measurement errors in the data. We would then like to find a subtree contained in every one of the trees with as many leaves labeled by species as possible since such a subtree more likely represents genuine evolutionary structure in the data. In this way, we would get an indication of which ancestral relationships can be regarded as resolved and which species need to be subjected to further experiments.

Maximum homeomorphic agreement subtrees can estimate a classification's stability to small changes in the data. Algorithms for MHT can also be used to prepare and improve bilingual context dictionaries for automated language translation systems, as described in [4]. Here, the goal is to align two given parse trees corresponding to the same sentence in two languages.

The maximum inferred consensus tree problem (MICT)

Input: A set of topological constraints of the form $\{i,j\} < \{k,l\}$, where i,j,k,l are leaves, and $\{i,j\} < \{k,l\}$ means that the lowest common ancestor of i and j is a proper descendant of the lowest common ancestor of k and l.

Output: A rooted tree that is consistent with as many constraints as possible.

MICT does not require any absolute evolutionary distances to be calculated. Hence, this method of phylogenetic reconstruction might be suitable when the overall branching structure is of primary interest and one has access to the outcomes of experiments that settle the relative topologies of a few species at a time[3].

A constraint involving three species only is called a *3-leaf constraint.* Observe that the constraint $\{i,j\} < \{i,k\}$ uniquely determines the relative topology for species i, j, and k (see Figure 3), and that this relationship could just as well have been expressed as $\{i,j\} < \{j,k\}$. Thus, $\{i,j\} < \{i,k\}$ is often written as $(\{i,j\},k)$ for convenience.

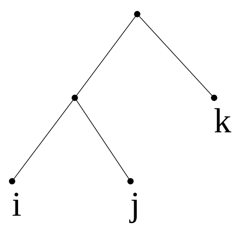

Figure 3: The 3-leaf constraint $(\{i,j\},k)$ induces a rooted binary tree.

[3]For example, Sibley-Ahlquist experiments can decide which two species in a set of three that are the most similar [8].

A special case of MICT occurs if all the given constraints are 3-leaf constraints:

The maximum inferred local consensus tree problem (MILCT)

Input: A set of 3-leaf constraints.

Output: A rooted tree that is consistent with as many constraints as possible.

MILCT is similar to *the maximum quartet consistency problem* (MQC) in which the input is a set of unrooted, labeled 4-leaf trees. MQC constitutes the combinatorial phase of a topological method for inferring unrooted evolutionary trees called *the quartet method*. For references to the heuristics for MQC that have been proposed, accompanied by some recent results, see [7]. MILCT can be regarded as the restriction of MQC to instances where all quartets share one species (preferably a so-called *outgroup*) which corresponds to the root; therefore, even though MQC has been shown to be computationally hard, MILCT might be easier.

The ordered local consensus tree problem (OLCT)

Input: A set of 3-leaf constraints and an explicit left-right ordering of the species.

Output: A rooted tree that is consistent with all of the constraints and conforms to the given ordering, if such a tree exists.

Ordered evolutionary trees arise when one wants to produce a visual representation of the information contained in an ordinary evolutionary tree that also complies with a certain left-right ordering of the species. The way that the species are ordered can be used to present comparative data such as relative size or age, so ordered trees provide a way to convey additional information readably.

Even if an evolutionary tree consistent with all the 3-leaf constraints exists for a given instance of OLCT, it does not automatically follow that an ordered tree for the desired ordering does. The ordering can contradict the constraints.

The maximum ordered local consensus tree problem (MOLCT)

Input: A set of 3-leaf constraints and an explicit left-right ordering of the species.

Output: A rooted, ordered tree that is consistent with as many constraints as possible.

MOLCT is the optimization version of OLCT.

The Hamming center problem (HCP)

Input: A set S of k binary strings of length n.

Output: A binary string β of length n that minimizes the maximum Hamming distance between β and any of the strings in S.

The *Hamming distance* between two binary strings of the same length is equal to the total number of positions in which the strings differ. The string β is called a *Hamming center* of S.

Although HCP as defined above only deals with strings over $\{0, 1\}$, it can be generalized to cover strings over alphabets of any size.

When classifying sequences, consensus representatives are useful. The around 100000 different proteins in humans can be divided into 1000 (or less) protein families, which makes it easier for researchers to understand their structures and biological functions [6]. A lot of information about a newly discovered protein may be deduced by establishing which family it belongs to. During identification, it is more efficient to try to align the new protein to representatives for various families than to individual family members. Conversely, given a set S of related sequences, one way to find other similar sequences is by computing a representative for S and then using the representative to probe a genome database. The representative should resemble every sequence in S, and must be chosen carefully. For example, the sequence r that minimizes the sum of all pairwise distances between r and elements in S is biased towards sequences that occur frequently, but using a center as representative will avoid this problem[4].

In a problem termed *the phylogenetic alignment problem*, an evolutionary tree T leaf-labeled by a set L of strings is available, and the object is to determine a labeling of T's internal nodes such that small edge distances are induced. (Next, one can derive a multiple alignment that is consistent with the fully labeled tree, and then remove all strings corresponding to internal nodes to obtain a good multiple alignment for L.) HCP is the special case of the phylogenetic alignment problem in which T is a star graph with k leaves, each leaf is labeled by a string from S, the distance function used is the Hamming distance, and the optimization criterion is that the maximum distance induced by any edge should be minimized.

HCP has also been studied in coding theory. In this context, S corresponds to a binary code, i.e., a set of code words. The significance of HCP is that given

[4]Depending on the application, the difference between strings is sometimes measured in terms of edit distance, which also takes insertions and deletions into account, rather than Hamming distance, which just considers substitutions.

a binary code, the value of an important parameter called *the covering radius* can be calculated immediately if a Hamming center is known. Several uses of the covering radius in information theory are mentioned in [3].

New results

In the first article included in this thesis, we show that MHT cannot be approximated within a factor of N^{ε}, where N is the input size, for any $0 \leq \varepsilon < \frac{1}{9}$ in polynomial time, unless P=NP. Our result holds even if restricted to instances with trees of height 2. On the other hand, we present an $O(N \log N)$-time heuristic for instances with $O(1)$ trees of height $O(1)$, yielding solutions within a constant factor of the optimum. Next, we prove that MICT is NP-complete and provide a simple heuristic for it which yields solutions within one third of the optimum in polynomial time. Finally, we give a more sophisticated polynomial-time heuristic for MILCT.

The second article is about ordered consensus trees. We develop an efficient decremental interval union algorithm to test if all of the constraints are compatible with each other, and show that for m constraints on n leaves, OLCT can be solved deterministically in time $O((m+n)\log n)$. We also show that MOLCT is solvable in cubic time by dynamic programming.

The third article is focused on HCP. We describe a randomized approximation algorithm with performance ratio $\frac{4}{3} + \varepsilon$, where $\varepsilon > 0$ can be made arbitrarily small. The algorithm runs in polynomial time as long as the Hamming radius of S is at least superlogarithmic in k. Furthermore, we show how to find in expected polynomial time a set B of $O(\log k)$ strings of length n such that for each string in S there is at least one string in B within Hamming distance not exceeding the radius of S.

References

[1] P. Buneman. The recovery of trees from measures of dissimilarity. In F.R. Hudson, D.G. Kendall, and P. Tautu, eds. *Mathematics in the Archaeological and Historical Sciences.* Edinburgh University Press, 1971, pp. 387–395.

[2] The Canterbury Tales Project. University of Sheffield, University of Oxford, De Montfort University, Brigham Young University, and Virginia Polytechnic Institute & State University (Virginia Tech). `http://www.shef.ac.uk/uni/projects/ctp/`

[3] G.D. Cohen, M.G. Karpovsky, H.F. Mattson, Jr., and J.R. Schatz. Covering Radius – Survey and Recent Results. *IEEE Transactions on Information Theory*, Vol. IT-31, 1985, pp. 328–343.

[4] R. Cole and R. Hariharan. An $O(n \log n)$ Algorithm for the Maximum Agreement Subtree Problem for Binary Trees. *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA'96), 1996, pp. 323–332.

[5] J. Felsenstein. PHYLIP – Phylogeny Inference Package (Version 3.2). *Cladistics*, Vol. 5, 1989, pp. 164–166.
See also `http://evolution.genetics.washington.edu/phylip.html`

[6] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.

[7] T. Jiang, P. Kearney, and M. Li. Orchestrating Quartets: Approximation and Data Correction. *Proceedings of the 39th Symposium on Foundations of Computer Science* (FOCS'98), 1998, pp. 416–425.

[8] S. Kannan, E. Lawler, and T. Warnow. Determining the Evolutionary Tree Using Experiments. *Journal of Algorithms* 21, 1996, pp. 26–50.

[9] W.-H. Li. *Molecular Evolution.* Sinauer Associates, Inc., Sunderland, 1997.

[10] J.C. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology.* PWS Publishing Company, Boston, 1997.

[11] The Tree of Life Project's web page on phylogenetic analysis computer programs. University of Arizona.
`http://phylogeny.arizona.edu/tree/programs/programs.html`

[12] M. Waterman. *Introduction to Computational Biology.* Chapman & Hall, Cambridge, 1995.

[13] The Willi Hennig Society's parsimony analysis software web page. http://www.vims.edu/~mes/hennig/software.html

Additional relevant references can be found at the end of each article.

On the Complexity of Constructing Evolutionary Trees

LESZEK GĄSIENIEC
Department of Computer Science, University of Liverpool, Peach Street, L69 7ZF, Liverpool, UK.
Email: `leszek@csc.liv.ac.uk`

JESPER JANSSON
Department of Computer Science, Lund Institute of Technology, Box 118, 221 00 Lund, Sweden.
Email: `Jesper.Jansson@dna.lth.se`

ANDRZEJ LINGAS
Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden.
Email: `Andrzej.Lingas@dna.lth.se`

ANNA ÖSTLIN
Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden.
Email: `Anna.Ostlin@dna.lth.se`

Abstract. In this paper we study a few important tree optimization problems with applications to computational biology. These problems ask for trees that are consistent with an as large part of the given data as possible. We show that the maximum homeomorphic agreement subtree problem cannot be approximated within a factor of N^ϵ, where N is the input size, for any $0 \leq \epsilon < \frac{1}{9}$ in polynomial time unless P=NP, even if all the given trees are of height 2. On the other hand, we present an $O(N \log N)$-time heuristic for the restriction of this problem to instances with $O(1)$ trees of height $O(1)$ yielding solutions within a constant factor of the optimum. We prove that the maximum inferred consensus tree problem is NP-complete, and provide a simple, fast heuristic for it yielding solutions within one third of the optimum. We also present a more specialized polynomial-time heuristic for the maximum inferred local consensus tree problem.

Keywords: algorithm, time complexity, evolutionary tree, homeomorphism, consensus tree

1 Introduction

An evolutionary tree models how different species in a given set have evolved. The leaves in an evolutionary tree correspond to species and internal nodes represent the species' ancestors.

The problem of constructing a reliable evolutionary tree has been studied extensively [2, 3, 4, 7, 8, 13, 14, 16, 17, 19, 20]. There are many different approaches, depending among other things on what kind of data that is available. Therefore, various versions of this problem arise in, for example, computational biology when one wants to deduce how different species are related, and comparative linguistics, where it is central to find out how different languages have evolved. In this paper, we look at some of these problems.

Given a set of alternative evolutionary trees describing possible evolutions for a fixed set of species, one might want to identify a subtree contained within every given tree such that the number of leaves labeled by species is maximized. This problem is known as *the maximum homeomorphic agreement subtree problem* (MHT) [16]. More formally, it is defined as follows. Given k rooted trees $T_1, T_2, ..., T_k$, each with n leaves labeled distinctly with elements chosen from a set A of cardinality n, find a maximum cardinality subset B of A such that the minimal homeomorphic subtrees of $T_1, T_2, ..., T_k$ (i.e., with all degree 2 nodes except for the root contracted) containing exactly the leaves labeled by B are isomorphic. To measure the input size of an instance of MHT, we let N denote the total number of nodes contained in the given trees. MHT restricted to instances with two trees is frequently called MAST; algorithms for MAST have been developed since 1985 [5]. It has been shown to be solvable in polynomial time, both for rooted trees [3, 4] and for UMAST, a variant of MAST with unrooted trees [3, 14, 17, 20]. In practice, however, the number of trees is often much larger than two [16]. For the special case of MHT in which at least one of the given trees has bounded degree, there exist polynomial-time algorithms [2, 16]. In contrast, MHT is known to be NP-complete even for instances with three trees of unbounded degree [16]. The first non-approximability result for MHT was published in [7]. It states that for three trees with unbounded degree, MHT cannot be approximated within ratio $2^{\log^\delta n}$ in polynomial time for any $\delta < 1$ unless NP $\subseteq$ DTIME$[2^{\text{polylog}\, n}]$. Here we prove that, unless P=NP, MHT cannot be approximated within a factor of N^ϵ, for any $0 \leq \epsilon < \frac{1}{9}$ in polynomial time even for instances containing only trees of height 2; see Section 2. On the other hand, in Section 3, we show that MHT for instances with $O(1)$ number of trees of height $O(1)$ can be approximated within a constant factor in time $O(N \log N)$. Similar results also hold for the unrooted version of MHT which is at least as hard as MHT (this can be seen by an argument analogous to that in [3] for MAST and UMAST).

Usually MHT instances do not admit a solution containing all the members of the species set. Therefore, in some applications, other methods may be preferred. One alternative approach is to attempt to construct an evolutionary tree from a set of constraints that relate the species to each other. Already during the early eighties, Aho et al. [1] studied the problem of inferring a tree from constraints on its lowest common ancestors in the context of relational data bases. They defined it as follows: Given a set of constraints of the form $\{i,j\} < \{k,l\}$, where $\{i,j,k,l\} \subset \{1,2,...,n\}$, if possible construct a tree on the set of leaves $\{1,2,...,n\}$ such that for each constraint of the aforementioned form, the lowest common ancestor of i and j is a proper descendant of the lowest common ancestor of k and l. Aho et al. showed how to decide whether an instance of this problem admits a solution, and if so, how to construct it, both in time $O(mn \log n)$, where m denotes the number of constraints.

Recently, many authors have studied the related problem of constructing the so-called consensus tree or local consensus tree [8, 13, 19]. For a set of binary rooted trees $\{T_1, T_2, ..., T_k\}$, each one leaf-labeled by a subset $L(T_i)$ of $\{1,2,...,n\}$, the consensus tree problem asks whether or not there is a tree T such that for $i = 1,2,...,k$, T_i is homeomorphic to the subtree of T induced by the nodes in $L(T_i)$ and their ancestors. If the input trees are of constant size it is termed the local consensus tree problem. A constraint of the form $\{i,j\} < \{i,k\}$ (denoted $(\{i,j\},k)$ for short) is easily seen to be equivalent to the constraint imposed by a full binary tree on the leaves i, j, k in the local consensus tree problem. For this reason, we shall call the tree inferring problem posed in [1] *the inferred consensus tree problem.*

Unfortunately, it is often impossible to construct an exact consensus tree. This creates a need for an optimization version of the inferred consensus tree problem whose objective is to find a consensus tree for an as large as possible subset of the input set of constraints of the form $\{i,j\} < \{k,l\}$. For brevity, we term this optimization problem *the maximum inferred consensus tree problem* (MICT for short). We also distinguish the restricted case of MICT where all constraints are of the form $(\{i,j\},k)$ and call it *the maximum inferred local consensus tree problem* (MILCT).

In Section 4 we provide an NP-completeness proof for MICT. Section 5 contains a simple $O((n+m)\log n)$-time heuristic for MICT yielding solutions within one third of the optimum and a more involved polynomial-time heuristic for MILCT. Both heuristics work equally well for the weighted versions of MICT and/or MILCT where the objective is to find a consensus tree for a subset of the input constraints of maximum total weight.

2 MHT is Hard to Approximate

Our main result in this section is the following theorem.

Theorem 1. *For any $0 \le \epsilon < \frac{1}{9}$, MHT, even if restricted to trees of height 2, cannot be approximated within a factor of N^ϵ in polynomial time, unless P=NP.*

Proof. First, we describe a reduction from the maximum independent set problem to MHT. Next, we show that if MHT can be approximated within a factor of N^ϵ in polynomial time then the problem of finding a maximum independent set in a graph with l nodes can be approximated within a factor of $l^{3\epsilon+o(1)}$. Finally, we apply known results about the inapproximability of the maximum independent set problem to get our result. In part, our reduction can be seen as a generalization of the reduction of three-dimensional perfect matching to MHT restricted to instances with three trees used in [16].

Let $G = (V, E)$ be a graph where $V = \{v_1, ..., v_l\}$ and $E = \{e_1, ..., e_k\}$ with $k > 1$. Construct k rooted trees $T_1, ..., T_k$ on $l + q$ labeled leaves that contain all the adjacency information about the nodes of G as follows. For each edge $e_i = (v_a, v_b) \in E$, build a rooted tree T_i on the set of leaves labeled by $w_1, ..., w_l, w_{l+1}, ..., w_{l+q}$. Let the root r_i of T_i be the parent of $(l-1)+q$ children, where the first child ("the non-leaf child") is a node with two children leaves labeled w_a and w_b, and the remaining children of r_i are leaves labeled by the elements in $\{w_j \mid 1 \le j \le l+q \text{ and } j \notin \{a, b\}\}$. Thus, r_i has exactly one pair of grandchildren, and we write $GC(T_i) = \{w_a, w_b\}$.

Now, let T be a maximum homeomorphic agreement subtree of $T_1, ..., T_k$. We choose q large enough to guarantee that each of the roots $r_1, ..., r_k$ will correspond to the root r of T. Actually, $q = 2$ is sufficient. (To see this, assume that the non-leaf child of r_i turned out to be the root for some i. By the construction above, all non-leaf children have two leaf children, so the number of leaves in this agreement subtree can be no larger than two. But we can always find an agreement subtree with three leaves by selecting r_i as root and including w_{l+1} and w_{l+2} in addition to the path from the root to a fixed leaf w_j, where $1 \le j \le l$.)

T has no non-leaf children because if it did, then there would exist some x

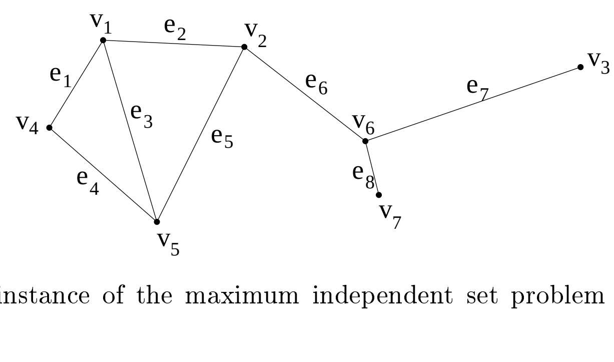

Figure 1. An instance of the maximum independent set problem with $l = 7$ and $k = 8$.

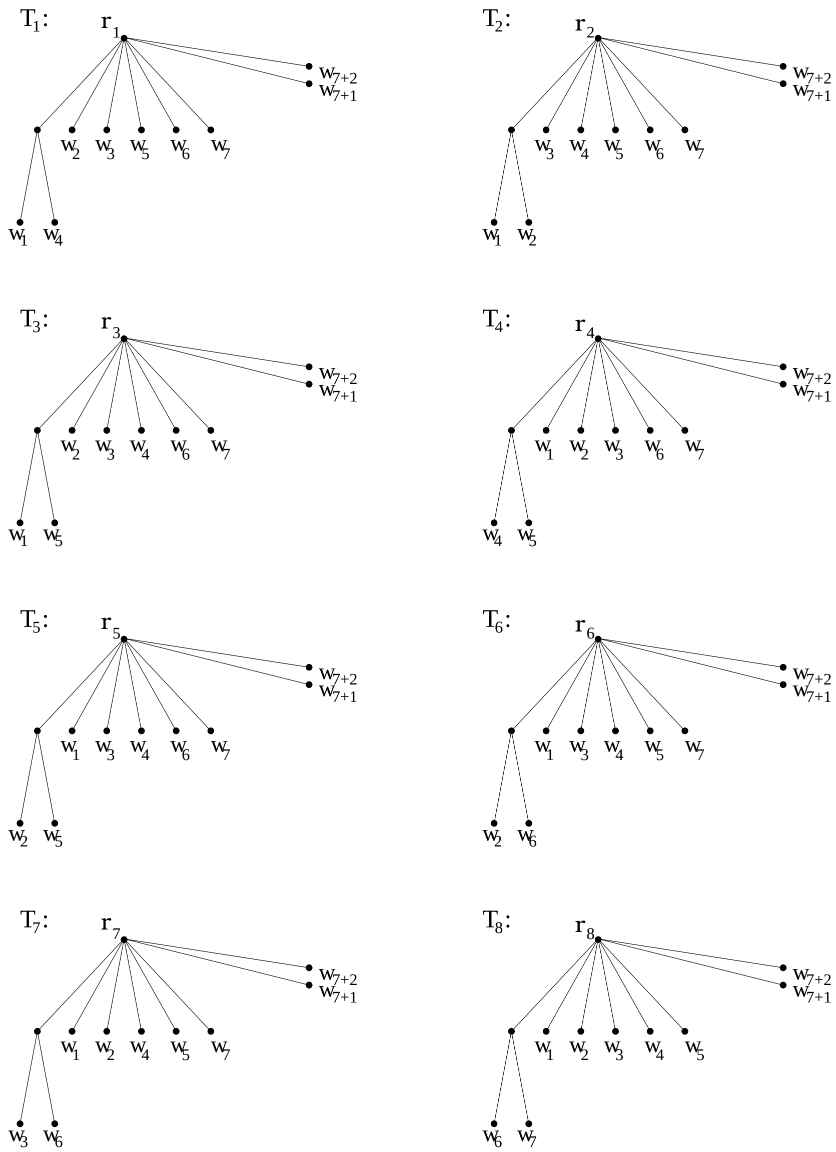

Figure 2. The trees T_i corresponding to the graph in Figure 1.

and y such that for each i, where $1 \leq i \leq k$, $GC(T_i)$ would be equal to $\{w_x, w_y\}$. Consequently, G would only have one edge, which contradicts $k > 1$. T's children are $m + q$ $(= m + 2)$ leaves labeled $w_{\mu_1}, w_{\mu_2}, ..., w_{\mu_m}, w_{l+1}, w_{l+2}$. If v_a is adjacent to v_b in G then at most one of w_a and w_b can be a child of T. Otherwise, $GC(T_i)$ wouldn't be equal to $\{w_a, w_b\}$ for any T_i, and $e_i \neq (v_a, v_b)$ would hold for all i, contradicting the adjacency of v_a and v_b in G. Thus, the nodes $v_{\mu_1}, v_{\mu_2}, ..., v_{\mu_m}$ form an independent set in G. Conversely, given an independent set I of nodes in G, we can easily construct an agreement

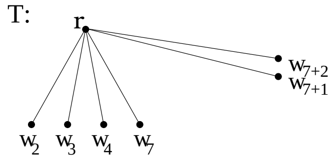

Figure 3. The maximum homeomorphic agreement subtree of $T_1, ..., T_8$ tells us that $\{v_2, v_3, v_4, v_7\}$ is a maximum independent set of the graph in Figure 1.

subtree T_I in the form of a rooted tree with $|I| + 2$ leaves uniquely labeled with w_j, where $v_j \in I$, and w_{l+1}, w_{l+2}. By the maximality of T, m equals the cardinality of the maximum independent set of G. Thus, an algorithm for MHT would immediately imply an algorithm for the maximum independent set problem. See Figures 1–3 for an example of the reduction.

The total size N of the trees $T_1, ..., T_k$ is $k \cdot O(l) = O(l^3) = l^{3+o(1)}$. Clearly, they can be constructed from G in polynomial time. Also, note that they are of height 2. Below, we only consider approximations that can be carried out in polynomial time. If MHT could be approximated within a factor of N^ϵ, then $\frac{OPT+2}{s+2} \leq N^\epsilon$, where $OPT + 2$ refers to the number of leaves in an optimal solution for a given instance of MHT and $s + 2$ is the number of leaves in its corresponding, approximative solution. For $s \geq 1$, it follows that $\frac{OPT}{s} \leq 3 \cdot \frac{OPT+2}{s+2} \leq 3N^\epsilon = l^{3\epsilon+o(1)}$, which would imply that the problem of finding a maximum independent set in a graph could be approximated within a factor of $l^{3\epsilon+o(1)}$. However, Håstad proved in [11] that this problem isn't approximable within $l^{1/3-\delta}$ for any $\delta > 0$, unless P=NP. Hence, if P $\neq$ NP, MHT cannot be approximated within a factor of N^ϵ for any $0 \leq \epsilon \leq \frac{1}{9} - o(1)$. Finally, since $\frac{1}{9} - o(1)$ can be made arbitrarily close to $\frac{1}{9}$ by choosing N large enough, there exist instances of MHT which cannot be approximated within a factor of N^ϵ for any constant $0 \leq \epsilon < \frac{1}{9}$ in polynomial time (unless P=NP). □

3 Approximations of MHT with $O(1)$ Trees of Height $O(1)$

We know that MHT is hard to approximate, both for instances with three trees [7] and for instances with an arbitrary number of trees of height 2 or more by Theorem 1. The natural question arises whether or not MHT for instances with a bounded number of trees, each one of bounded height, can be tightly approximated in polynomial time. The following result, together with Theorem 1, yields a characterization of the approximability of MHT restricted to instances with trees of $O(1)$ height.

Theorem 2. *MHT restricted to instances with k trees of height not exceeding h can be approximated within a factor of k^h in time $O(n \log n)$.*

To begin the proof of Theorem 2, we need to introduce the following notation. For a tree T, $V(T)$ stands for the set of nodes of T. Let v be a node of a rooted tree T. The minimal subtree of T rooted at v, including v and all its descendants is denoted by T_v. $L(T_v)$ stands for the set of labels of the leaves in T_v. The set of children of v in T is denoted by $C(v)$. Furthermore, by a *k-partite hypergraph* H we shall mean a pair $(V_1 \cup ... \cup V_k,\ E)$, where V_1 through V_k are pairwise disjoint sets and E is a subset of $V_1 \times ... \times V_k$. The elements of $V_1 \cup ... \cup V_k$ are called the *nodes* of H whereas the elements of E are called the *edges* of H. A *matching* of H is a subset of E in which no pair of edges includes a common node.

Let $T_1, ..., T_k$ be the input trees. For $(v_1, ..., v_k)$, where $v_i \in V(T_i)$ for $i = 1, ..., k$, let $Mht(v_1, ..., v_k)$ denote the maximum size of an agreement subtree of the trees $T_1, ..., T_k$ restricted to $B = L((T_1)_{v_1}) \cap ... \cap L((T_k)_{v_k})$. We can view $Mht(v_1, ..., v_k)$ as the solution of MHT for $(T_1)_{v_1}, ..., (T_k)_{v_k}$. Next, let $H(v_1, ..., v_k)$ denote the k-partite hypergraph $(C(v_1) \cup ... \cup C(v_k), C(v_1) \times ... \times C(v_k))$ whose edges $(w_1, ..., w_k)$ have weight $Mht(w_1, ..., w_k)$. Finally, let $Match(v_1, ..., v_k)$ be the maximum weight of a matching in hypergraph $H(v_1, ..., v_k)$ and $Diag(v_1, ..., v_k) = \max\{Mht(w_1, ..., w_k) \mid (w_1, ..., w_k) \in (\{v_1\} \cup C(v_1)) \times ... \times (\{v_k\} \cup C(v_k)) - \{(v_1, ..., v_k)\}\}$.

Intuitively, in the final agreement subtree of $(T_1)_{v_1}, ..., (T_k)_{v_k}$ either the roots of the trees, i.e., v_1 through v_k, are matched together which forces their children to be optimally matched together (*Match*), or only some of the roots are matched together with some children of the remaining roots (*Diag*). This yields the following lemma which is a straightforward generalization of the basic lemma in the dynamic programming approach to MAST in [20] (see also [3]).

Lemma 3. *For any* $(v_1, ..., v_k)$, *where* $v_i \in V(T_i)$ *for* $i = 1, ..., k$, *if at least one of the* v_i*'s is a leaf then* $Mht(v_1, ..., v_k) = |L((T_1)_{v_1}) \cap ... \cap L((T_k)_{v_k})|$, *else* $Mht(v_1, ..., v_k) = \max\{Match(v_1, ..., v_k),\ Diag(v_1, ..., v_k)\}$.

The recursive computation of $Mht(v_1, ..., v_k)$ for $(v_1, ..., v_k) \in V(T_1) \times ... \times V(T_k)$ in the lemma above can be bottom-up ordered by $Hs(v_1, ..., v_k) = \sum_{i=1}^{k} height((T_i)_{v_i})$. Hence, we have the following algorithm for MHT.

Algorithm 1

1. input $T_1, ..., T_k$
2. for each $(v_1, ..., v_k) \in V(T_1) \times ... \times V(T_k)$, in increasing order of $Hs(v_1, ..., v_k)$ do compute $Mht(v_1, ..., v_k)$ by using the expression in Lemma 3.
3. output $Mht(r_1, ..., r_k)$, where r_i is the root of T_i for $i = 1, ..., k$.

It is hard to compute the exact value of $Match(v_1, ..., v_k)$ in the expression of Lemma 3 since the problem of computing maximum matching in a 3-partite hypergraph is NP-complete [18]. For this reason, we shall rely on a greedy method for approximating $Match(v_1, ..., v_k)$ yielding an approximation of $Mht(v_1, ..., v_k)$. The greedy method consists of repeatedly picking the heaviest edge e and removing all edges overlapping e. It can be implemented easily using a priority queue. e can overlap with at most k edges in an optimum solution, and since their total weight $\leq k \cdot weight(e)$, we obtain:

Lemma 4. *Let $H = (V, E)$ be a k-partite hypergraph on m edges with positive integer weights. A matching in H of total weight within a factor k of the maximum can be constructed in a greedy fashion in time $O(k|E| + |V| + m \log m)$.*

Interestingly, in the unweighted case there are known (much slower, but still) polynomial-time heuristics yielding solutions within almost $\frac{k}{2}$ of the optimum [10].

Combining Algorithm 1 with the greedy method for approximating the value of $Match(v_1, ..., v_k)$, we obtain the following lemma, yielding Theorem 2.

Lemma 5. *For all $(v_1, ..., v_k) \in V(T_1) \times ... \times V(T_k)$, we can approximate $Mht(v_1, ..., v_k)$ within a factor of k^h, where $h = \max\{height((T_i)_{v_i}) \mid 1 \leq i \leq k\}$, in $O(n \log n)$ time.*

Proof. For $(v_1, ..., v_k) \in V(T_1) \times ... \times V(T_k)$, let $s(v_1, ..., v_k)$ denote the size of the intersections $L((T_1)_{v_1}) \cap ... \cap L((T_k)_{v_k})$. Clearly, we have $Mht(v_1, ..., v_k) \leq s(v_1, ..., v_k)$, and in particular if one of the v_i's is a leaf then $Mht(v_1, ..., v_k) = s(v_1, ..., v_k)$. For a leaf label j, we determine all k-tuples $(v_1, ..., v_k)$ for which $j \in L((T_1)_{v_1}) \cap ... \cap L((T_k)_{v_k})$ by finding, in each T_i, $i = 1, ..., k$, the nodes on the path of length $\leq h$ from the leaf labeled j to the root. It follows that the number of these tuples is $(h+1)^k$. Consequently, the set L of all k-tuples for which $s(v_1, ..., v_k) > 0$ has size not exceeding $n(h+1)^k$. To list L efficiently, we sort the pointers to leaves in T_1 through T_k by the leaf labels. Such a sorted list of pointers can be produced in time $O(|V(T_1)| + ... + |V(T_k)|)$. Using it, we can generate L by finding appropriate tree paths in time $O(|V(T_1)| + ... + |V(T_k)| + (h+1)^k)$.

For the k-tuples $(v_1, ..., v_k)$ in the set L that include at least one leaf, we have $s(v_1, ..., v_k) = 1$ and $Mht(v_1, ..., v_k) = 1$. To compute approximations of $Mht(v_1, ..., v_k)$ for the remaining k-tuples in L, we build a balanced search tree S_L for L, with respect to the lexicographic order of k-tuples in $V(T_1) \times ... \times V(T_k)$, in time $O(|L| \log |L|)$. Next, we follow the scheme of Algorithm 1 using the greedy method to approximate $Match(v_1, ..., v_k)$ in the hypergraph $H_L(v_1, ..., v_k)$ which is the hypergraph $H(v_1, ..., v_k)$ defined in Lemma 3 restricted to edges in L.

Each k-tuple $(w_1, ..., w_k) \in L$ occurs at most once as an edge in the hypergraphs $H_L(v_1, ..., v_k)$ for $(v_1, ..., v_k) \in L$ (only when $w_i \in C(v_i)$ for $i = 1, ..., k$). Hence, the hypergraphs $H_L(v_1, ..., v_k)$ for $(w_1, ..., w_k) \in L$, have no more than $|L|$ edges totally and can be constructed (without weights) by scanning L and using S_L in total time $O(|L| \log |L|)$. Clearly, each $H_L(v_1, ..., v_k)$ has at most $s(v_1, ..., v_k)$ edges with positive weights. For each of its edges $(w_1, ..., w_k)$, we have $Hs(w_1, ..., w_k) = Hs(v_1, ..., v_k) - k$ and $\max\{height(w_i) | 1 \leq i \leq k\} = h - 1$. Hence, we may inductively assume that we have already k^{h-1} approximations of $Mht(w_1, ..., w_k)$, i.e., of the weights of $(w_1, ..., w_k)$ in the hypergraph. Consequently, we obtain an approximation of $Match(v_1, ..., v_k)$ within a factor of k^h by applying the greedy method. Due to Lemma 4, the total time complexity of the greedy method is bounded by $O(k|L| + (\sum_{(v_1,...,v_k)\in L} s(v_1, ..., v_k)) \log n)$. By induction on $Hs(v_1, ..., v_k)$, we also obtain an approximation of $Diag(v_1, ..., v_k)$ within a factor of k^h by considering solely k-tuples $(w_1, ..., w_k)$ in $L \cap ((\{v_1\} \cup C(v_1)) \times ... \times (\{v_k\} \cup C(v_k)) - \{(v_1, ..., v_k)\})$. Each $(w_1, ..., w_k) \in L$ can contribute to the value of $Diag(v_1, ..., v_k)$ for at most $2^k - 1$ k-tuples $(v_1, ..., v_k) \in L$. Hence, the total size of the subsets of L contributing to $Diag(v_1, ..., v_k)$ over all $(v_1, ..., v_k) \in L$, and consequently the total cost of finding maxima of Mht-approximations over these subsets, is $O(2^k|L|)$. We can build these subsets, again by scanning L and using S_L, in total time $O(2^k|L| \log |L|)$.

Each of the trees T_1 through T_k has size not exceeding $2n$ by its binarity. Hence, by $|L| \leq (h+1)^k n$, $\sum_{(v_1,...,v_k)\in L} s(v_1, ..., v_k) \leq (h+1)^k n$, $h = O(1)$, $k = O(1)$, and straightforward calculations, we obtain the $O(n \log n)$ bound.

□

4 MICT is NP-Complete

The problem of deciding whether or not a 3-partite hypergraph (V, E) has a perfect matching (3PM), i.e., if V is covered by a subset of pairwise disjoint edges in E, is known to be NP-complete [18]. To show the NP-completeness of MICT, we provide a reduction of 3PM to MICT.

Let $H = (V, E)$ be a 3-partite hypergraph and k a parameter that will be specified later on. We let each vertex in V label one leaf. Also, for each edge $e \in E$, we introduce $k+2$ leaves labeled e_i, where $i = 0, ..., k+1$. Let C be the minimal set of constraints satisfying:

1. for each $e, f \in E$ with $e \neq f$: the contraints $(\{e_i, e_l\}, f_j) \in C$, where $i = 0, 1,\ \ l = 2, ..., k+1,\ $ and $j = 2, ..., k+1$.
2. for each $e = (a, b, c) \in E$: the three constraints $\{a, b\} < \{e_0, e_1\},\ \{a, c\} < \{e_0, e_1\},\ $ and $\{b, c\} < \{e_0, e_1\} \in C$.

Thus, C consists of $2k^2(|E|^2 - |E|)$ constraints of the first type and $3|E|$ constraints of the second type.

To characterize consensus trees for large subsets of C, we need the following definitions.

Definition 6. In a rooted tree T, the lowest common ancestor of a sequence of nodes $v_1, ..., v_m$ will be denoted by $lca(v_1, ..., v_m)$. Furthermore, the path from a node v to the root of T will be denoted by $R(v)$. The subtree of T induced by a sequence of nodes $v_1, ..., v_m$ is the smallest subtree of T including the paths $R(v_i)$, $i = 1, ..., m$.

Definition 7. The full binary tree on four leaves a, b, c, d, where $lca(a, b)$ and $lca(c, d)$ form the intermediate level, will be denoted by $B_4(a, b, c, d)$.

Lemma 8. *If T is a consensus tree for at least $|C| - k^2 + 1$ constraints in C, then for each $e, f \in E$ with $e \neq f$, the subtree induced of T by $\{e_0, e_1, f_0, f_1\}$ is homeomorphic to $B_4(e_0, e_1, f_0, f_1)$.*

Proof. By the assumption on the number of constraints satisfied by T, for each $e, f \in E$ with $e \neq f$, there are indices $l, j \in \{2, ..., k + 1\}$ such that for $i = 0, 1$, the constraints $(\{e_i, e_l\}, f_j)$, $(\{f_i, f_j\}, e_l)$ are satisfied by T. Because of $(\{e_0, e_l\}, f_j)$ and $(\{e_1, e_l\}, f_j)$, the path $R(lca(e_0, e_1, e_l))$ cannot be included in the path $R(f_j)$. Thus, $R(lca(e_0, e_1, e_l)) \not\subseteq R(lca(f_0, f_1, f_j))$. Similarly, by $(\{f_0, f_j\}, e_l)$ and $(\{f_1, f_j\}, e_l)$, we have $R(lca(f_0, f_1, f_j)) \not\subseteq R(lca(e_0, e_1, e_l))$. This means that the paths from $lca(e_0, e_1, e_l)$ to $lca(e_0, e_1, e_l, f_0, f_1, f_j)$ and from $lca(f_0, f_1, f_j)$ to $lca(e_0, e_1, e_l, f_0, f_1, f_j)$ must be edge-disjoint. □

Corollary 9. *Let T be a consensus tree for at least $|C| - k^2 + 1$ constraints in C. For each node $a \in V$ and two different edges $e, f \in E$, if T satisfies a constraint of the form $\{a, \cdot\} < \{e_0, e_1\}$ then T cannot satisfy any constraints of the form $\{a, \cdot\} < \{f_0, f_1\}$.*

Lemma 10. *Let $k > \sqrt{3|E| - |V|}$. The hypergraph H has a perfect matching iff there is a consensus tree for a subset of $2k^2(|E|^2 - |E|) + |V|$ constraints in C.*

Proof. Suppose first that H has a perfect matching M. We can construct a consensus tree T satisfying at least $2k^2(|E|^2 - |E|) + |V|$ of the constraints in C as follows. The root of T has $|E|$ children which are in one-to-one correspondence with the edges in E. For every $e \in E$, a subtree rooted in the corresponding child has as children the leaves $e_0, e_1, ..., e_{k+1}$. Furthermore, if $e = \{a, b, c\}$ is in M, then the subtree has another child which in turn is the parent of the leaves labeled a, b, c.

Suppose in turn that there is a consensus tree T satisfying $2k^2(|E|^2 - |E|) + |V|$ constraints in C. The total number of constraints in C is $2k^2(|E|^2 - |E|) + 3|E|$. It follows by $k > \sqrt{3|E| - |V|}$ that T satisfies at least $|C| - k^2 + 1$ constraints. Thus, by Corollary 9, for each node $a \in V$, there is at most one edge $e \in E$ such that some constraint of the form $\{a, \cdot\} < \{e_0, e_1\}$ is satisfied by T. On the other hand, for a given node a and a given edge e, at most two constraints of the form $\{a, \cdot\} < \{e_0, e_1\}$ can be satisfied by T by the construction of C. Consequently, V can be partitioned into three disjoint subsets V_r, $r = 0, 1, 2$, respectively consisting of nodes $a \in V$ for which T satisfies r constraints of the form $\{a, \cdot\} < \{\cdot, \cdot\}$. At most $|V_2| + \frac{|V_1|}{2}$ constraints of the form $\{\cdot, \cdot\} < \{\cdot, \cdot\}$ are satisfied by T, so since there are only $2k^2(|E|^2 - |E|)$ constraints of the form $(\{\cdot, \cdot\}, \cdot)$ in C, we conclude that V_2 has to be as large as possible, i.e., $V_2 = V$. It follows that for each edge $e \in E$, if a constraint of the form $\{\cdot, \cdot\} < \{e_0, e_1\}$ is satisfied by T, then all the three constraints of this form are satisfied by T. Hence, H has a perfect matching. □

The construction of C for k equal, say, to $|E|$ can easily be done in polynomial time. Hence, MICT is NP-hard by the NP-completeness of 3PM and Lemma 10. The membership of MICT in NP is obvious.

Theorem 11. *MICT is NP-complete.*

5 Approximation Heuristics for MICT

Our heuristics in fact work for the generalization of MICT where with each input constraint c a positive weight $w(c)$ is associated, and the objective is to construct a consensus tree for a subset of constraints of maximum total weight.

5.1 Heuristic 1

For a constraint $\{i, j\} < \{k, l\}$, where all the leaves are different, k and l are said to have an *upper occurrence* in the constraint, and i and j are said to have a *lower occurrence* in the constraint. For a constraint $\{i, j\} < \{i, k\}$, where i, j, k are different, i and j are said to have a lower occurrence in the constraint and k is said to have an upper occurrence in the constraint. The *total weight* of upper (or lower) occurrences for a leaf l is equal to the sum of the weights of all constraints in which l has upper (or lower) occurrences.

Lemma 12. *For any instance of MICT, the sum of all leaves' total weights of upper occurrences is at least one third (one half if all constraints contain four leaves) of the sum of all leaves' total weight of upper and lower occurrences.*

Heuristic 1

input: a set C of m weighted constraints on leaves 1 through n;

output: a consensus tree T for a subset of C whose weight is at least one third (one half if all constraints contain four leaves) of the total weight of the constraints in C;

1. $LEFT \leftarrow C$;
   $LEAVES \leftarrow \{1, ..., n\}$;
   $T \leftarrow \{v\}$;
2. if $LEFT = \emptyset$ then extend T by adding $|LEAVES|$ children to v, label them uniquely with elements in $LEAVES$, and return T;
3. pick a leaf y in $LEAVES$ which achieves the maximum ratio between the total weight of y's upper occurrences and the total weight of y's lower occurrences in the constraints in $LEFT$;
4. set Y to the set of constraints in $LEFT$ which contains y;
5. $LEFT \leftarrow LEFT \setminus Y$;
6. $LEAVES \leftarrow LEAVES \setminus \{y\}$;
7. extend T by adding two children to v; label the first child by y; set v to the second child;
8. go to 2

Theorem 13. *Heuristic 1 constructs a consensus tree for a subset of the input set of constraints C whose total weight is at least one third (one half if all constraints contain four leaves) of the total weight of C, in time $O((m+n)\log n)$.*

Proof. By Lemma 12 and the choice of y, the ratio between the total weight of upper occurrences and lower occurrences of y in the constraints in $LEFT$ is at least one third. All the constraints in Y in which y has an upper occurrence are satisfied by T by the construction of T.

To implement Steps 3 and 6 efficiently, we arrange $LEAVES$ in a priority queue partially ordered by the ratio between the total weight of their upper and lower occurrences in constraints in $LEFT$. All the priority queue operations, i.e., creating the priority queue, picking the y's, updating the priority queue after Step 5, take a total of $O((n + m)\log n)$ time.

To implement Steps 4 and 5, we lexicographically sort C four times according to four cyclic permutations of the four leaves in each constraint. For $i = 1, ..., 4$, the i-th permutation puts the i-th leaf as the first, the $(i + 1)$-st (in the cyclic order) as the second, etc. Next, four search trees are built on the basis of the sorted lists. Using the search trees, we can find Y in $LEFT$ and remove it from $LEFT$ in time $O(|Y|\log n)$. We conclude that Steps 4 and 5 totally take time $O((m + n)\log n)$ (including the preprocessing). □

The absolute factors of one third and half respectively provided by Heuristic 1 are worst-case optimal. For example, any consensus tree can satisfy at most one constraint from each consecutive triple of constraints in a sequence $(\{a_i, b_i\}, c_i)$, $(\{b_i, c_i\}, a_i)$, $(\{c_i, a_i\}, b_i)$, $i = 1, ..., k$. In case all constraints contain four different leaves, the sequence $\{a_i, b_i\} < \{c_i, d_i\}$, $\{c_i, d_i\} < \{a_i, b_i\}$, $i = 1, ..., k$, causes the lower bound $\frac{1}{2}$.

The consensus tree produced by Heuristic 1 has the form of a linear chain with singular leaves pending, where only the last chain node can have larger degree. It is easy to slightly modify Heuristic 1 to output a subset of the input constraints (a priori) satisfied by the tree. A minimum height consensus tree for at least one third of the input constraints is then obtained in time $O(mn \log n)$ by running the algorithm of Aho et al. [1] for the inferred consensus tree problem on this set.

In case the minimum number of constraints necessary to delete in order to build a consensus tree for the remaining part is very small, and the number m of constraints relative to the number of leaves is high (it is always $O(n^4)$), an approach different from that of Heuristic 1 might be more useful.

5.2 Heuristic 2

Heuristic 2 for MILCT simply mimics the algorithm of Aho et al. [1] for the inferred consensus tree problem restricted to constraints of the form $(\{i, j\}, k)$. Their basic idea is simple. The input set of leaves $1, 2, ..., n$ is partitioned into a minimal set of blocks satisfying the following requirement:

(*) If $(\{i, j\}, k)$ is a constraint then i and j are in the same block.

Now, if the number of blocks in the minimal set is at least two, the algorithm of Aho et al. creates the consensus tree by connecting the roots of the consensus trees recursively computed for the respective blocks with a common parent root node. Otherwise, the number is one, and it returns a null consensus tree.

For a subset S of leaves, let $G(S)$ denote the auxiliary graph on S where the edges are induced by the requirement (*), and their weights are equal to the total weight of the constraints inducing them.

Whenever the algorithm of Aho et al. is stuck at a non-divisible subset S of the set of leaves and has to return a null tree, Heuristic 2 simply finds a minimum weight edge cut of the auxiliary graph $G(S)$ (with respect to the current set of constraints). Next, the edges of the min-cut are deleted from $G(S)$ and the connected components of $G(S)$ are computed. Consequently, the constraints corresponding to the edges of the min-cut are also deleted. Finally, the approximation consensus trees for the connected components are recursively computed and connected by a common parent node.

By using recent dynamic data structures for graph connectivity, Henzinger et al. gave efficient implementations of the algorithm of Aho et al. restricted to constraints of the form $(\{i,j\},k)$ [8]. Their randomized implementation takes $O(m \log^3 n)$ expected time. They use an undirected graph U and a directed graph D defined as follows.

- $U = (V, E)$ with V equal to the set of leaves $\{1, 2, ..., n\}$, and where for each constraint $(\{a,b\},c)$ in C, edges $\{a,b\}$ and $\{b,c\}$ are in E.
- $D = (V', A)$, where for each constraint $(\{a,b\},c)$ in C, nodes $\{a,b\}$ and $\{b,c\}$ are in V', and the directed edge $\{a,b\} \to \{b,c\}$ is in A.

At the beginning, the graph U is colored yellow. U is used for finding yellow components. The consensus tree returned is found by combining the trees constructed for the yellow components. The graph D is used for finding edges in U that can be colored red; these edges correspond to the so-called maximal nodes in D.

A *maximal node* in D is a node with no outgoing edges, and a red edge whose endpoints are in different yellow components is called a *separable red edge*.

By slightly modifying the algorithm of Henzinger et al. and combining it with an algorithm for minimum weight edge cut [15], we can implement Heuristic 2 as follows.

Heuristic 2

1. Construct U and D. Add weights to the edges in U. The weight of an edge $\{a,b\}$ in U is equal to the sum of the weights of constraints of the form $(\{a,b\},\cdot)$. Color all nodes in D and edges in U yellow.
2. Identify maximal nodes in D. Recolor these nodes and the corresponding edges of U red.
3. If U has no edges, then return the consensus tree T with a root and all nodes in U children of the root. Otherwise, compute yellow components of U. If there is only one yellow component then find a minimum weight edge cut, delete the edges in the cut from U and the corresponding nodes from D, and recompute the yellow components. Let $\mathcal{C}_\infty, ..., \mathcal{C}_{||}$ be the current yellow components. Form a tree T by creating the root of T and connecting the root of the consensus trees recursively computed for the components $\mathcal{C}_\infty, ..., \mathcal{C}_{||}$ to it. For each current yellow component, identify the set E_{sep} of separable red edges incident to that new component. Delete these edges from U and the corresponding nodes from D. Go to Step 2.

Lemma 14. *Heuristic 2 can be implemented to run in $O(n^3 \log n + m \log^3 n)$ expected time.*

Proof. A minimum weight edge cut can be computed with high probability in time $O(n^2 \log n)$ [15]. In the worst case it has to be done n times; hence, the calls to minimum weight edge cut take a total of $O(n^3 \log n)$ expected time. All other operations can be performed in expected time $O(m \log^3 n)$ like in the algorithm of Henzinger et al. Thus, the total expected time is $O(n^3 \log n + m \log^3 n)$. $\square$

Lemma 15. *Let I be an instance of MICT, and let T be the tree produced by Heuristic 2 for I. The total weight of constraints in I not satisfied by T is at most height(T) times the minimum.*

Proof. Let J be a subset of I of minimum total weight such that $I \setminus J$ has a consensus tree. Next, let D be the set of connected components in the auxiliary graph where edges corresponding to the constraints in J are deleted. Suppose that Heuristic 2 at some stage finds a min-cut in a currently connected fragment C. Clearly, C cannot be a subset of a simple component in D since then there wouldn't exist a consensus tree for $I \setminus J$. Hence, there is a subset J_C of J such that the set of edges corresponding to the constraints in J_C disconnects $G(C)$ into disjoint components. Clearly, the total weight of J_C is not smaller than the weight of a min-cut of $G(C)$. Now, it is sufficient to observe that the subsets J_C for distinct C's on the same recursion level of Heuristic 2 are pairwise disjoint. $\square$

Theorem 16. *Let n, w, t be respectively the number of leaves, the total weight of constraints, and the minimum total weight of the constraints to remove in an instance I of MILCT. Heuristic 2 constructs a consensus tree for a subset of the constraints in I whose total weight is not smaller than $w - nt$.*

Note that the number of constraints in I might be cubic in n and that Heuristic 2 yields a better approximation factor than Heuristic 1 for MILCT whenever $t < \frac{2w}{3n}$.

6 Open Problems

We do not know whether or not it is possible to find a polynomial-time approximation scheme for instances of MHT with $O(1)$ trees of height $O(1)$.

It follows from Theorem 11 and the definition of MICT that MICT is strongly NP-complete. Hence, it cannot admit a fully polynomial-time approximation scheme [18]. However, it is an open question whether it admits a polynomial-time approximation scheme or at least a polynomial-time heuristic with a smaller approximation factor.

The computational complexity status of MILCT is also an interesting open question. If MILCT is NP-complete, does it admit a polynomial-time approximation scheme?

On a high level, the definitions of MICT and MILCT resemble those of MAX SAT and MAX k-SAT (see [9, 12]). In the design of Heuristic 1 we have utilized this similarity taking inspiration from the early heuristic for MAX k-SAT due to Johnson [12]. Recently, substantial progress in approximating MAX SAT and MAX k-SAT has been made by using linear programming, semidefinite programming, and randomized rounding [6, 9]. One of the main obstacles in applying these techniques to MICT is the complexity of "arithmetization" of the proper-descendant lowest-common-ancestor relation (the case of MILCT seems more promising).

7 Acknowledgements

The authors were supported in part by TFR (Swedish Research Council for Engineering Sciences) and in part by NUF-NAL (The Nuffield Foundation Awards to Newly Appointed Lecturers).

References

1. A.V. Aho, Y. Sagiv, T.G. Szymanski, and J.D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal of Computing*, Vol. 10, No. 3, 1981, pp. 405–421.
2. M. Farach, T. Przytycka, and M. Thorup. Computing the agreement of trees with bounded degrees. *Proc. of the 3rd ESA*, 1995, pp. 381–393.
3. M. Farach and M. Thorup. Fast Comparison of Evolutionary Trees. *Proc. of the 5th ACM-SIAM SODA*, 1994, pp. 481–488.
4. M. Farach and M. Thorup. Optimal evolutionary tree comparison by sparse dynamic programming. *Proc. of the 35th FOCS*, 1994, pp. 770–779.
5. C.R. Finden and A.D. Gordon. Obtaining common pruned trees. *Journal of Classification* 2, 1985, pp. 255–276.
6. M.X. Goemans and D.P. Williamson. New $\frac{3}{4}$-approximation algorithms for MAX SAT. *SIAM Journal of Discrete Mathematics*, 7, 1994, pp. 656–666.
7. J. Hein, T. Jiang, L. Wang, and K. Zhang. On the Complexity of Comparing Evolutionary Trees. *Discrete Applied Mathematics*, 71, 1996, pp. 153–169.
8. M.R. Henzinger, V. King, and T. Warnow. Constructing a Tree from Homeomorphic Subtrees, with Applications to Computational Biology. *Proc. of the 7th ACM-SIAM SODA*, 1996, pp. 333–340.

9. D.S. Hochbaum (editor). *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, Boston, 1997.
10. C.A.J. Hurkens and A. Schrijver. On the size of systems of sets every t of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems. *SIAM Journal of Discrete Mathematics*, Vol. 2, No. 1, 1989, pp. 68–72.
11. J. Håstad. Testing of the long code and hardness for clique. *Proc. of the 28th ACM STOC*, 1996, pp. 11–19.
12. D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9, 1974, pp. 256–278.
13. S. Kannan, T. Warnow, and S. Yooseph. Computing the Local Consensus of Trees. *Proc. of the 6th ACM-SIAM SODA*, 1995, pp. 68–77.
14. M.Y. Kao, T.W. Lam, T. Przytycka, W.K. Sung, H.F. Ting. General Techniques for Comparing Unrooted Evolutionary Trees. *Proc. of the 29th ACM STOC*, 1997, pp. 54–65.
15. D.R. Karger. Minimum Cuts in Near-Linear Time. *Proc. of the 28th ACM STOC*, 1996, pp. 56–63.
16. D. Keselman and A. Amir. Maximum agreement subtree in a set of evolutionary trees - Metrics and efficient algorithms. *Proc. of the 35th FOCS*, 1994, pp. 758–769.
17. T.W. Lam, W.K. Sung, and H.F. Ting. Computing the Unrooted Maximum Agreement Subtree in Sub-quadratic Time. *Proc. of the 5th SWAT*, 1996, pp. 124–135.
18. C.H. Papadimitriou. *Computational Complexity*, Addison-Wesley, Reading, 1994.
19. C. Phillips and T.J. Warnow. The Asymmetric Median Tree - A New Model for Building Consensus Trees. *Proc. of the 7th CPM*, LNCS 1075, 1996, pp. 234–252.
20. M. Steel and T. Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. *Information Processing Letters* 48, 1993, pp. 77–82.

Inferring Ordered Trees from Local Constraints

Leszek Gąsieniec[1], Jesper Jansson[2], Andrzej Lingas[2], and Anna Östlin[2]

[1] Department of Computer Science, University of Liverpool, Chadwick Building, Peach Street, Liverpool, L69 7ZF, UK; email:leszek@csc.liv.ac.uk
[2] Department of Computer Science, Lund University, Box 118, S-221 00 Lund, Sweden; email: {Jesper.Jansson, Andrzej.Lingas, Anna.Ostlin}@dna.lth.se

Abstract. We consider a problem of inferring an ordered tree from a set of local constraints on its leaves which we term *the ordered local consensus tree problem.* Using our efficient decremental interval union algorithm, we show that the ordered local consensus tree problem for m constraints on n leaves can be deterministically solved in time $O((m+n)\log n)$. We also show that the related optimization problem of constructing an ordered local consensus tree for the maximum number of 3-leaf constraints is solvable in cubic time.

1 Introduction

1.1 Background

Evolutionary trees are used to model tree-like evolution for species from some given set. The basic idea is to let the leaves correspond to species and to let the internal nodes represent the species' ancestors. Usually, all children in the tree are unordered, i.e., there is no distinction between left and right, since the intrinsic branching structure of the tree is what matters. The problem of constructing a reliable evolutionary tree, given some data, has been studied for a long time [2, 3, 4, 6, 9, 16, 17]. There are different approaches, depending among other things on what kind of data that is available. Also, depending on the nature of the underlying application, the resulting trees may be rooted or unrooted, and with or without weighted edges. Thus, various versions of this problem arise in, for example, computational biology when one wants to find out how different biological species or molecules are related, and comparative linguistics, where it is central to figure out how different languages have evolved. In this paper, we focus on a special case of the so-called local consensus tree problem for rooted, unweighted evolutionary trees in which a specified left-right ordering is imposed upon the species. We call this problem *the ordered local consensus tree problem.* The motivations for the ordered versions of the consensus tree problem come from tree drawing and from the desire to somehow order the leaves for ease of presentation, for example by the age of the species or languages involved, or just alphabetically.

1.2 Problem Description

For a set of binary rooted trees $\{T_1, ..., T_m\}$, each one leaf-labeled by a subset $L(T_i)$ of $\{1, 2, ..., n\}$ (the "species"), *the consensus tree problem* asks if there exists a tree T such that for $i = 1, ..., m$, T_i is homeomorphic to the subtree of T induced by the nodes in $L(T_i)$ and their ancestors, and if so, what it looks like. In other words, is there a consensus tree that contains all the structural information of $T_1, T_2, ..., T_m$ combined, or do some of the given trees contradict each other? If each of the input trees is known to be of size bounded by a constant, the problem is termed *the local consensus tree problem.* See Figure 1 for an example.

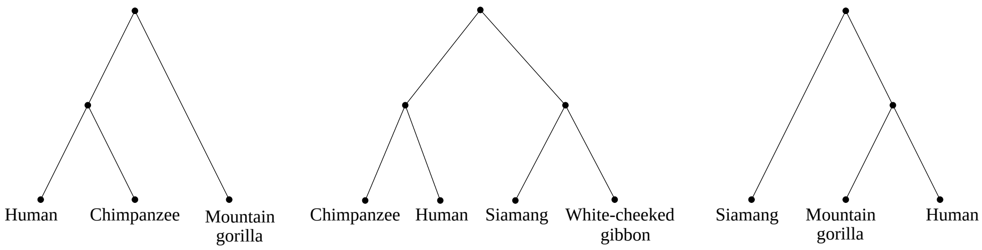

Figure 1. Are these evolutionary trees compatible? (The answer is yes!)

The problems of efficiently computing consensus trees have been intensively studied recently [10, 11, 15]. Already in the early eighties, Aho, Sagiv, Szymansky, and Ullman provided a quadratic-time solution for a variant of the local consensus tree problem [1]. Using new dynamic data structures for graph connectivity, Henzinger, King, and Warnow presented an $O(m\sqrt{n})$-time deterministic implementation and an $O(m \log^3 n)$ expected-time randomized implementation of the method of Aho et al. in [10].

Each one of the input trees can be regarded as a set of *constraints* on the possible phylogenetic relationships between species. A constraint of the form $(\{i, j\}, k)$ is called a *3-leaf constraint* or a *rooted triple*, and indicates that the lowest common ancestor of the leaves corresponding to species i and j is a proper descendant of the lowest common ancestor of the leaves corresponding to species i and k. For example, the leftmost tree in Figure 1 is equivalent to the constraint ({Human, Chimpanzee}, Mountain gorilla). Since any binary rooted tree on n leaves can be defined by a set of $(n - 2)$ 3-leaf constraints [16], we consider the problem of inferring a local consensus tree from a set of 3-leaf constraints rather than from a set of constant-sized trees.

Scanning the labels of the leaves of an ordered tree from left to right yields its *frontier*. If T, in addition to being in compliance with the given constraints, is required to be constructed in such a way that its frontier equals a certain ordering of the species, then we term the resulting problems *the ordered consensus tree problem* and *the ordered local consensus tree problem*, respectively. Figure 2 shows an ordered consensus tree for the species in Figure 1.

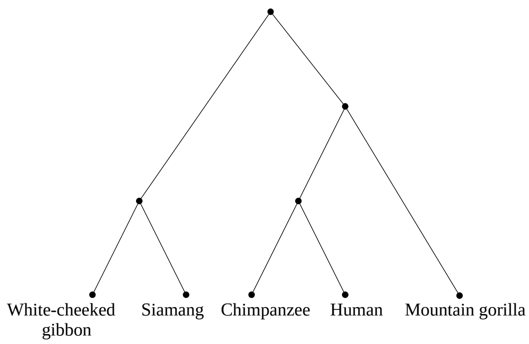

Figure 2. An ordered consensus tree for the trees depicted in Figure 1. Here, the species are ordered by size.

1.3 Our Results

Our key observation is that the basic dynamic graph used by Henzinger et al. in their fast algorithms for the local consensus tree problem becomes an *interval graph* if the tree under construction is required to be ordered. For interval graphs with known interval representation, efficient dynamic connectivity algorithms are simply implied by the known efficient dynamic interval union algorithms [5]. As we need only a decremental interval union algorithm, and the known efficient fully dynamic interval union algorithms are rather complicated, we provide a very simple and practical, decremental interval union algorithm matching the time performance of that from [5] in the decremental case.

Using our decremental interval union algorithm, we show that the ordered local consensus tree problem can be deterministically solved in time $O((m + n) \log n)$. We also show that the related optimization problem of constructing an ordered local consensus tree for the maximum number of given 3-leaf binary trees is solvable in cubic time.

1.4 Overview

The rest of the paper is organized as follows. In Section 2 we recall the definitions of and some known facts about segment and interval trees. In Section 3 we describe our simple decremental interval union algorithm. In Section 4 we present our fast algorithm for the construction of ordered local consensus trees, and in Section 5 we provide the cubic-time solution to the related optimization problem mentioned above.

2 Preliminaries

Segment trees, interval trees and interval tries will be used in our algorithms.

The following facts about segment trees and interval trees are shown in [13].

A segment tree is a useful data structure for storing a set S of intervals on the real line R together with some additional information. The leaves in a segment tree are $(-\infty, x_1)$, $[x_1, x_1]$, (x_1, x_2), $[x_2, x_2]$, ..., (x_N, ∞), where x_i, $i = 1, 2, ..., N$, are the legal endpoints for the segments and $x_i < x_{i+1}$ for all $i = 1, ..., N-1$.

Each node u of a segment tree is augmented by a node list $NL(u)$. To describe NL we will use $xrange(u)$, defined as follows. First, $xrange(root) = (-\infty, +\infty]$. Let r be the right child and l the left child of node u. The *split* value at node u is any value that is at least as large as the values at the leaves in the left subtree of u and smaller than the values at the leaves in the right subtree of u. Next, we define $xrange(r) = xrange(u) \cap (split(u), +\infty]$ and $xrange(l) = xrange(u) \cap (-\infty, split(u)]$. We set $NL(u)$ to $\{I \in S \mid xrange(u) \subseteq I \;\&\; xrange(parent(u)) \not\subseteq I\}$. Each interval can be stored in at most in $O(\log N)$ nodes.

Fact 1. [13] *A segment tree for a set of n intervals with both endpoints in a universe $\mathcal{U} \subset \mathcal{R}$ of size N can be constructed in time $O(n \log N)$. The depth of the tree is $O(\log N)$ and its size is $O(N)$. An interval with both endpoints in $\mathcal{U}$ can be inserted or deleted from the segment tree in time $O(g(n) \log N)$, where $g(n)$ is the time required to insert or delete an interval from a node list of size n.*

In our application $g(n) = O(1)$, since we shall use a node counter instead of a node list. The value $NC(u)$ of the node counter for a node u is just the cardinality of $NL(u)$, i.e., deletion or insertion corresponds to decreasing or increasing $NC(u)$ by one.

An interval tree is a leaf-oriented binary search tree that supports intersection queries for a set S of closed intervals on the real line in logarithmic time.

Fact 2. [13] *Suppose that the left endpoints of the intervals in S belong to a subset of R of size N and $|S| = n$. An interval tree T of depth $O(\log N)$ for S can be constructed in time $O(N + n \log Nn)$ using space $O(n + N)$. The insertion or deletion of an interval with left endpoint in $\mathcal{U}$ into T takes time $O(\log n + \log N)$. The intersection query is supported by T in time $O(\log N + r)$, where r is the number of the reported intervals.*

The interval trie is another data structure for maintaining a set S of intervals, efficiently supporting the stabbing query. The answer to it is the list of all intervals that contain the query point. The next fact following from [14] characterizes interval tries.

Fact 3. [14] *Let S be a set of n intervals with endpoints in $\{1, ..., N\}$ and let f be a function mapping natural numbers into natural numbers greater than 1. The interval trie data structure performs insertions and deletions of intervals in S in time $O(f(N) + \log\log N)$. The stabbing query can be answered in time $O(\frac{\log N}{\log f(N)} + r)$ where r is the number of reported component intervals. The total space required is $O(N\frac{\log N}{\log f(N)} + nf(N))$.*

3 A Simple Decremental Interval Union Algorithm

Here we consider the problem of maintaining the union of the set S of intervals under a sequence $s_1, ..., s_m$ of interval deletions from S, where for $i = 1, ..., m$, s_i is currently in S.

The first crucial step of our method consists of the construction of a special balanced segment tree T for S [13]. Let $e_1, ..., e_l$ be the consecutive endpoints of the intervals in S. The consecutive leaves in T are $(-\infty, e_1)$, $[e_1, e_1]$, (e_1, e_2), $[e_2, e_2]$, ..., (e_l, ∞). To each node u of T we associate the counter $NC(u)$ instead of the standard node list $NL(u)$. Recall that the value of the counter is just the cardinality of $NL(u)$. Note that $l \leq 2n$. Hence, the segment tree T can be constructed in time $O(n \log n)$ by Fact 1.

Additionally, for each node u of T, we set a special bit $r(u)$ to 1 iff the NC counters along the path from u to the root of T, including u and the root of T, are set to zero. Clearly, we can determine $r(u)$ for all nodes of T by traversing T in a top-down fashion in time proportional to the size of T, i.e., $O(n)$.

Remark 1. *There exist leaves l_1, l_2 and l_3 in the left-right order in T such that $r(l_1) = 0$, $r(l_2) = 1$ and $r(l_3) = 0$ if and only if the union of S is disconnected.*

Finally, we form a linear array A, to store the consecutive leaves of T, in linear time.

Besides the segment tree T, we initialize an interval tree P in order to maintain the component intervals in the union of the current S. We can find the component intervals of the initial S, e.g., by using a standard sweep-line technique in time $O(n \log n)$. Next, we can construct P on $O(n)$ leaves determined by the endpoints of the intervals in S and insert the initial component intervals in time $O(n \log n)$ by Fact 2.

To find the component intervals of the set resulting from the deletion of the interval s_1 from S, we proceed as follows.

To begin with, we find the component interval ci_1 that s_1 belongs to in time $O(\log n)$ by querying P with s_1.

Next, we find the $O(\log n)$ nodes u of T whose node lists would contain s_1, if we were keeping them, and decrease $NC(u)$ by one. This can be done in time $O(\log n)$ by Fact 1. If $NC(u)$ drops to 0 and $r(parent(u)) = 1$ then we set $r(u)$ to 1 and update $r(w)$ for the descendants w of u in a top-down and left-right fashion. We also produce the list $LL(u)$ of consecutive leaf-descendants w of u for which $r(w)$ become 1 due to the updating ($LL(u)$ can be empty). Further, we concatenate the lists $LL(u)$ in the order of the ranges of nodes u. Let L_1 denote the resulting list. Note that the updating of $r(w)$'s as well as the construction of the list L_1 takes time proportional to the number r_1 of nodes w that got $r(w)$ set to 1 for the first time.

Now the following observation is crucial: for any pair of consecutive leaves a, b in L_1 that are not neighbors in the array A of consecutive leaves of T, the right neighbor $rn(a)$ of a in A and the left neighbor $ln(b)$ of b in A yield the new component interval. Also, if the left endpoint of the old component interval ci_1 that s_1 belonged to is not on L_1, then it together with the left neighbor $ln(a)$ of the first leaf on L_1 yields another new component interval. Analogously, the right neighbor $rn(z)$ of the last leaf on L_1 together with the right endpoint of ic_1 might yield yet another new component interval. Hence, we need to insert the new component intervals into P and delete ci_1 from P. Any of the aforementioned insertions and deletions take $O(\log n)$ time.

We conclude that we can update T and P in order to maintain the union of the set of intervals resulting from the deletion of s_1 from S in time $O((k + 1)\log n + r_1)$, where k is the number of the new component intervals.

The next deletions d_i, $i = 2, ..., m$, are handled similarly. We define r_i for $i = 2, ..., m$ in the same way as we defined r_1 above. It follows that the sum $\sum_{i=1}^{m} r_i$ is bounded from above by the size of T, i.e., $O(n)$.

Summarizing, we obtain our main result in this section.

Theorem 1. *Let S be a set consisting of n closed intervals on the real line. After $O(n \log n)$ time preprocessing, the union of S can be maintained under a sequence of m interval deletions in time $O(m \log n + n + k \log n)$, where k is the final number of component intervals. After each deletion the newly created component intervals can be listed without increasing the asymptotic complexity. The intersection query can be answered in time $O(\log n + r)$, where r is the number of component intervals to report.*

If we need only a stabbing query and the endpoints of the intervals belong to $\{1, ..., N\}$ for a small N, we can improve our result using an interval trie instead of the interval tree P. By Fact 3, we obtain the following variant of the result.

Theorem 2. *Let S be a set of n closed intervals with endpoints in $\{1, ..., N\}$ and let $f(N)$ be an integer greater than 1. After $O(n \log n + N \frac{\log N}{\log f(N)} + nf(N))$-time preprocessing the union of S can be maintained under a sequence of m*

interval deletions in time $O(m \log n + n + k(f(N) + \log\log N))$, where k is the final number of component intervals. After each deletion the newly created component intervals can be listed without increasing the asymptotic complexity. The stabbing query can be answered in time $O(\frac{\log N}{\log f(N)} + r)$ where r is the number of reported intervals. The total space required is $O(N \frac{\log N}{\log f(N)} + nf(N))$.

Our simple algorithm for maintaining the union of a set S of intervals under a sequence of interval deletions can also be interpreted (and will be used in the next section) as an algorithm for the connected components of the interval graph induced by S under a sequence of vertex deletions.

If we have an interval graph G and don't have its interval representation S, we can form it in time proportional to the size of G, e.g., by ordering the maximal cliques of G so that for each vertex v of G the maximal cliques containing v are consecutive [8, 12].

4 Efficient Construction of an Ordered Consensus Tree

A binary rooted tree T_i on $O(1)$ leaves in the local consensus tree problem corresponds to $O(1)$ 3-leaf constraints of the form $(\{i, j\}, k)$ on the consensus tree T, where $(\{i, j\}, k)$ implies that the lowest common ancestor of the leaves i and j has to be a proper descendant of the lowest common ancestor of the leaves i and k [16]. For this reason, we can consider the problem of inferring a local consensus tree from a set of 3-leaf constraints instead of the local consensus tree problem.

By using recent dynamic data structures for graph connectivity, Henzinger et al. provide efficient algorithms for the aforementioned problem of inferring a local consensus tree. Their deterministic algorithm takes $O(m\sqrt{n})$ time whereas their randomized algorithm takes $O(m \log^3 n)$ expected-time, where m is the number of 3-leaf constraints on leaf-labels in $\{1, 2, ..., n\}$. In both cases, they use an undirected graph U and a directed graph D defined as follows.

- $U = (V, E)$ with V equal to the set of leaves $\{1, 2, ..., n\}$, and where for each constraint $(\{a, b\}, c)$ in C, edges $\{a, b\}$ and $\{b, c\}$ are in E.
- $D = (V', A)$, where for each constraint $(\{a, b\}, c)$ in C, nodes $\{a, b\}$ and $\{b, c\}$ are in V', and the directed edge $\{a, b\} \rightarrow \{b, c\}$ is in A.

The graph U is used for finding current (yellow in [10]) connected components. The consensus tree returned is found by combining the trees constructed for the current connected components. The graph D is used for finding edges in U that can be removed (colored red in [10]), these edges correspond to the so-called maximal nodes in D. A *maximal node* in D is a node with no outgoing edges.

Remark 2: *Whenever two leaves a and b are in the same current component of U then not only a and b, but also all other leaves in $[a,b]$ have to be in the same subtree of the ordered consensus tree representing this component.*

By Remark 2, the vertices of U can be interpreted as segments inducing an interval graph. Hence, instead of computing the current components of U we can compute the corresponding components of the interval graph utilizing our efficient decremental interval union algorithm.

By combining this idea with that of the algorithm of Henzinger et al., we can build an ordered consensus tree as follows.

Algorithm 1

1. $S \leftarrow \{[a,b] | (a,b) \in E\}$.
2. Construct D and initialize the interval trie Q on the universe $\{1,2,...,n\}$.
3. Identify maximal vertices in D. Move the corresponding intervals from S to Q.
4. If S is empty, then return the consensus tree O with a root and all nodes in U children of the root. Otherwise, compute the new component intervals of the union of S by using the decremental algorithm of Section 3. If there is no new component interval then return “no ordered consensus tree” and stop. Let $c_1, c_2, ..., c_k$ be the new component intervals in the order of their endpoints. For $i = 1, ..., k-1$, create the root of the ordered consensus tree to be computed for the component interval c_i and link it with that of the previous interval component including c_i if such one exists. For $i = 1, ..., k-1$, if c_i and c_{i+1} are included in the same old component interval then query Q with the median m_i of the right endpoint of c_i and the left endpoint c_{i+1}. Remove all reported (i.e., containing m_i) intervals from Q and the corresponding vertices from D. Go to Step 2.

Theorem 3. *For a set of m 3-leaf constraints on leaf-labels in $\{1, 2, ..., n\}$, we can decide whether there exists an ordered consensus tree, and if so, construct it, in time $O((m+n)\log n)$.*

Proof. (sketch). If Algorithm 1 produces an ordered consensus tree then it is equivalent to the minimal tree produced by Aho et al. algorithm or the tree produced by the faster algorithm of Henzinger et al. (Algorithm 1 in [10]). To see this note that the current component intervals of S correspond to the yellow components of the graph U in [10] and the intervals stored in Q correspond to the current red edges of U. Furthermore, an interval stored in Q corresponds to a red separable edge in [10] iff it overlaps with at least two new component intervals, i.e., if it is intersected by at least one of the medians m_i, $i = 1, ..., k-1$. If Algorithm 1 fails to produce an ordered consensus tree then there is no such a tree by Remark 1 and the correctness of the algorithms of Henzinger et al.

Analogously as in the algorithm of Henzinger et al., the construction of the graph D takes time $O(m)$ and the overall time taken by determining the maximal vertices is proportional to the size of D, i.e., $O(m)$. The total time-cost of recomputing the component intervals including the preprocessing is $O((m+n)\log n)$ by Theorem 1. The total cost of the $O(m)$ insertions from and deletions into Q, and the $O(n)$ stabbing queries for Q is $O((m+n)\frac{\log n}{\log\log n})$ by Fact 3 for $f(n) = \Theta(\frac{\log n}{\log\log n})$. □

5 A Cubic-Time Algorithm for the Optimization Problem

Unfortunately, it is often impossible to construct an exact consensus tree. Therefore, the optimization version of the problem of inferring a local consensus tree in which the objective is to find a consensus tree for an as large as possible subset of an input set of 3-leaf constraints was considered in [7]. Only approximation algorithms for this optimization problem were provided in [7], and the complexity status of this problem is still open. Here we show that the problem of finding an *ordered* consensus tree for a largest possible subset of the input set of constraints of the aforementioned form is solvable by dynamic programming in cubic time.

Theorem 4. *For a set of m 3-leaf constraints on leaf-labels in $\{1, 2, ..., n\}$, we can find an ordered consensus tree for the maximum number of the constraints in time $O(n^3)$.*

Proof. For each pair of i, j in $\{1, 2, ..., n\}$, we recursively construct the ordered consensus tree $T_{i,j}$ for the maximum number $m_{i,j}$ of the input constraints on leaf-labels in $\{i, i+1, ..., j\}$. For $i = 1, 2, ..., n$, the tree $T_{i,i}$ simply consists of the leaf labeled with i and its parent which is the root of the tree. To determine a tree $T_{i,j}$ where $i < j$, we compute, for $k = i, ..., j-1$, the number $w_{i,k,j}$ of the input constraints $(\{l_1, l_2\}, l_3)$ where $\{l_1, l_2, l_3\} \subset \{i, ..., j\}$ and either $\{l_1, l_2\} \subset \{i, ..., k\}$ & $l_3 \in \{k+1, ..., j\}$ or $l_3 \in \{i, ..., k\}$ & $\{l_1, l_2\} \subset \{k+1, ..., j\}$. Next, we find a $k \in \{i, ..., j\}$ which maximizes the sum $m_{i,k} + m_{k+1,j} + w_{i,k,j}$ and form the tree $T_{i,j}$ by creating its root and making the roots of $T_{i,k}$ and $T_{k,j}$ its children. It follows by induction on $|j-i|$ that $m_{i,j} = m_{i,k} + m_{k+1,j} + w_{i,k,j}$, which proves the correctness of the construction of $T_{i,j}$. Assuming that the numbers $w_{i,k,j}$ are precomputed and the trees $T_{i',j'}$ where $j' - i' < j - i$ are already determined, the determination of $T_{i,j}$ takes $O(j-i+1)$ time. The precomputation of all the numbers $w_{i,k,j}$, $i \leq k \leq j$, can be done by a single scan of the input constraint sequence in time $O(m+n^3)$, i.e., $O(n^3)$. The cubic upper time-bound follows. □

6 Final Remarks

It would be interesting to consider the more general situation when only a partial order of the leaf labels is given a priori. One could look here for efficient algorithms for constructing consensus trees that would take advantage of the input partial leaf-label order as much as possible.

Our cubic-time optimization algorithm can be generalized to include other more general forms of local tree constraints, e.g., constraints of the form "the lowest common ancestor of i and j has to be a proper descendant of the lowest common ancestor of k and l" considered by Aho et al. in [1].

References

1. A.V. Aho, Y. Sagiv, T.G. Szymanski, and J.D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal of Computing*, Vol. 10, No. 3, 1981, pp. 405–421.
2. H.J. Bandelt and A. Dress. Reconstructing the Shape of a Tree from Observed Dissimilarity Data. *Adv. Appl. Math.*, Vol. 7, 1986, pp. 309–343.
3. P. Buneman. The Recovery of Trees from Measures of Dissimilarity. In F.R. Hudson, D.G. Kendall, and P. Tautu (eds.), *Mathematics in the Archaeological and Historical Sciences*, Edinburgh Univ. Press, Edinburgh, 1971, pp. 387–395.
4. L.L. Cavalli-Sforza and A.W.F. Edwards. Phylogenetic Analysis: Models and Estimation Procedures. *Am. J. Hum. Genet.*, Vol. 19, 1967, pp. 233–257.
5. S.W. Cheng and R. Janardan. Efficient Maintenance of the Union of Intervals on a Line, with Applications. *Journal of Algorithms*, Vol. 12, 1991, pp. 57–74.
6. W.M. Fitch and E. Margoliash. Construction of Phylogenetic Trees. *Science*, Vol. 155, 1967, pp. 279–284.
7. L. Gąsieniec, J. Jansson, A. Lingas, and A. Östlin. On the Complexity of Computing Evolutionary Trees. In Proc. *3rd Annual International Computing and Combinatorics Conference* (COCOON'97), LNCS 1276, Springer-Verlag, 1997, pp. 134–145.
8. M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.
9. D. Gusfield. Efficient Algorithms for Inferring Evolutionary Trees. *Networks*, Vol. 21, 1991, pp. 19–28.
10. M.R. Henzinger, V. King, and T. Warnow. Constructing a Tree from Homeomorphic Subtrees, with Applications to Computational Biology. In Proc. *7th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA'96), 1996, pp. 333–340.
11. S. Kannan, T. Warnow, and S. Yooseph. Computing the Local Consensus of Trees. In Proc. *6th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA'95), 1995, pp. 68–77.

12. N. Korte and R.H. Möhring. An incremental linear time algorithm for recognizing interval graphs. *SIAM Journal of Computing*, Vol. 18, 1989, pp. 68–81.
13. K. Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*. EATCS Monographs on Theoretical Computer Science, Springer Verlag, Berlin, 1984.
14. M. Overmars. Computational geometry on a grid (an overview). *Theoretical Foundations of Computer Graphics and CAD*, NATO ASI Series, Vol. F40, 1987, pp. 167–184.
15. C. Phillips and T. Warnow. The Asymmetric Median Tree - A New Model for Building Consensus Trees, In Proc. *7th Annual Symposium on Combinatorial Pattern Matching* (CPM'96), LNCS 1075, Springer-Verlag, 1996, pp. 234–252.
16. M. Steel. The Complexity of Reconstructing Trees from Qualitative Characters and Subtrees. *Journal of Classification*, Vol. 9, 1992, pp. 91–116.
17. M.S. Waterman, T.F. Smith, M. Singh, and W.A. Beyer. Additive Evolutionary Trees. *J. Theor. Biol.*, Vol. 64, 1977, pp. 199–213.

Efficient Approximation Algorithms for the Hamming Center Problem

Leszek Gąsieniec[1] Jesper Jansson[2] Andrzej Lingas[3]

Abstract

The *Hamming center problem* for a set S of k binary strings, each of length n, asks for a binary string β of length n that minimizes the maximum Hamming distance between β and any string in S. The decision version of this problem is known to be NP-complete [6]. We provide several approximation algorithms for the Hamming center problem. Our main result is a randomized $(\frac{4}{3}+\varepsilon)$-approximation algorithm running in polynomial time if the Hamming radius of S is at least superlogarithmic in k. Furthermore, we show how to find in polynomial time a set B of $O(\log k)$ strings of length n such that for each string in S there is at least one string in B within Hamming distance not exceeding the radius of S.

1 Introduction

Let $\mathbb{Z}_2^n$ be the set of all strings of length n over the alphabet $\{0,1\}$. For any $\alpha \in \mathbb{Z}_2^n$ we use the notation $\alpha[i]$ to refer to the symbol placed at the ith position of α, where $i = 1, .., n$, and we let $\alpha[i..j]$ represent the substring of α starting at position i and ending at position j, where $1 \leq i < j \leq n$. The *Hamming distance* between $\alpha_1, \alpha_2 \in \mathbb{Z}_2^n$ is defined as the number of positions in which the strings differ, and is denoted by $d(\alpha_1, \alpha_2)$.

[1]Department of Computer Science, University of Liverpool, Peach Street, L69 7ZF, UK; leszek@csc.liv.ac.uk

[2]Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden; Jesper.Jansson@dna.lth.se

[3]Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden; Andrzej.Lingas@dna.lth.se

The *Hamming center problem* (HCP) is stated as follows: Given a set S of k binary strings $\alpha_i \in \mathbb{Z}_2^n$, where $i = 1,..,k$, find a string $\beta \in \mathbb{Z}_2^n$ such that the value

$$r = \max_{1 \leq i \leq k} d(\beta, \alpha_i)$$

is minimized.

Such a β is called a *center* of S, and the corresponding value of r is called the *radius* of S. Note that some instances of HCP (in fact most of them) have several centers. Throughout this paper, we will denote approximative solutions to HCP by β'.

The decision problem version of HCP is: Given a set S of k binary strings $\alpha_i \in \mathbb{Z}_2^n$, where $i = 1,..,k$, and a positive integer r_p, determine whether there exists a string $\beta \in \mathbb{Z}_2^n$ such that

$$\max_{1 \leq i \leq k} d(\beta, \alpha_i) \leq r_p.$$

There is a vast literature on graph and geometric center and multi-center problems modelling canonical facility location problems [15]. Unfortunately, because of the exponential size of $\mathbb{Z}_2^n$, its high dimension, and its discreteness, neither the graph approach nor the geometric approach work directly for HCP.

A variant of HCP has been studied by Berman et al. in [2], where they look for all maximal blocks within a set of aligned binary strings, having centers with radius of size at most 1. Their study is motivated by the detection of potentially important regions within closely related DNA sequences. They provide a linear time algorithm for their problem. In fact, there exists a straightforward polynomial-time algorithm for HCP as long as the radius is $O(1)$ (see Section 5). Berman et al. left the question open whether there exists an efficient algorithm for the case where the size of the radius is arbitrarily large.

In [6], Frances and Litman showed that the decision version of HCP, by them called the Minimum Radius Problem, as well as the dual Maximum Covering Radius Problem (MCR) are NP-complete; MCR and the decision version of HCP are in fact equivalent [3, 6]. MCR is defined as follows: Given a set $C \subseteq \mathbb{Z}_2^n$, called the set of *code words*, and a positive integer ρ, are all strings in $\mathbb{Z}_2^n$ within Hamming distance ρ of some code word in C? The *covering radius* is the smallest integer ρ that satisfies the above condition. It is a basic geometric parameter of a code which measures its quality. For example, if a code is used for data compression, the covering radius is a measure of the maximum distortion [1], and when it is used in error correction, it gives the maximum weight of a correctable random error [3]. Another related problem is the covering radius of a lattice in Euclidean space, which has applications to quantization and to coding for the Gaussian channel [4, 5].

Motivated by the intractability of HCP, we present here several approximation algorithms.* To start with, we give a very simple 2-approximation algorithm for HCP in Section 2. In Section 3, we provide an integer programming formulation of HCP. It yields polynomial time solutions for HCP if k (the number of input strings) or n (their lengths) is fixed. In the subsequent section, we observe that the method of randomized rounding applied to the linear programming relaxation of the integer programming formulation of HCP yields a solution with the additive error guarantee $O(\sqrt{n \log n})$ with probability greater than $1 - \frac{k}{n^2}$. In Section 5, we present our main result, i.e., the randomized $(\frac{4}{3} + \varepsilon)$-approximation algorithm for HCP running in polynomial time if the radius of set S is at least superlogarithmic in k. Finally, in Section 6 we show how to find in polynomial time a set B of $O(\log k)$ strings of length n such that for each string in S there is at least one string in B within Hamming distance not exceeding the radius of S.

2 A Simple Heuristic

We start with the following heuristic, showing that one can get a constant approximation ratio instantly:

> Set the approximative solution β' to α_l, where l is chosen arbitrarily from $\{1, .., k\}$.

The Hamming distance obeys the triangle inequality ([9], p. 424):

$$d(x, y) \leq d(x, z) + d(z, y), \quad \text{for all } x, y, z \in \mathbb{Z}_2^n. \tag{1}$$

Theorem 2.1 *The simple heuristic always yields a solution that is within a factor 2 of the optimal.*

Proof: Let β be an optimal solution. Due to inequality (1), $d(\beta', \alpha_i) \leq d(\beta', \beta) + d(\beta, \alpha_i)$ for every i in $\{1, .., k\}$. Next, $d(\beta', \beta) + d(\beta, \alpha_i) = d(\alpha_l, \beta) + d(\beta, \alpha_i) \leq r + r = 2r$ by the definition of r. □

*A preliminary version of this paper appeared in [7].

3 Integer Programming Formulation

The Hamming center problem is equivalent to a special case of the integer programming problem. Any given instance $(\alpha_1, .., \alpha_k, r_p)$ of the decision version of HCP, where $\alpha_i \in \mathbb{Z}_2^n$ for $1 \leq i \leq k$ and $r_p \in \mathbb{N}$, can be expressed as a system of k linear inequalities.

Let $x = (x_1, .., x_n) \in \{0,1\}^n$ represent a vector of 0-1-variables and let the ith inequality be

$$\sum_{\substack{\alpha_i[m]=0 \\ 1 \leq m \leq n}} x_m + \sum_{\substack{\alpha_i[m]=1 \\ 1 \leq m \leq n}} (1 - x_m) \leq r_p$$

The left-hand side is equal to the Hamming distance between α_i and x. (For each position m, if $\alpha_i[m] = 0$ then the sum is incremented by one if and only if $x_m = 1$, and conversely, if $\alpha_i[m] = 1$ then the sum is incremented by one if and only if $x_m = 0$.) The constraint '$\leq r_p$' ensures that the value of $d(\alpha_i, x)$ is smaller than or equal to the proposed radius.

A 0-1-solution x that is compatible with the system of inequalities corresponds directly to a solution β for the supplied instance, as can be seen by setting $\beta[m] = x_m$ for $1 \leq m \leq n$.

A system of k inequalities in n variables can be transformed into a system of k equalities in $n + k$ variables. The integer programming problem for a system of linear equalities on non-negative variables with coefficients unrestricted in sign admits pseudo-polynomial-time solutions when the number of equalities is constant [13]; this means that the running time of such an algorithm is bounded by a polynomial in n and the maximum among all the absolute values of coefficients involved in the equalities. Thus, HCP for $k = O(1)$ is solvable in time polynomial in its input length.

On the other hand, Lenstra, Lenstra, and Lovász [11] have shown that there is a polynomial (in k) time algorithm for testing whether a system of k linear inequalities with a fixed number of variables has an integer solution. So, HCP for $n = O(1)$ can also be solved in polynomial time.

Lemma 3.1 *HCP restricted to instances with $O(1)$ strings as well as HCP restricted to instances with strings of length $O(1)$ are solvable in polynomial time.*

4 Randomized Rounding

By relaxing the integer constraints on x in the 0-1-integer programming formulation of HCP, thus allowing each component x_m to be a real number in the interval $[0,1]$, we get a linear programming problem that can be solved in polynomial time by known methods [14]. Randomization can then be applied to the solution of the relaxed problem to get an approximative 0-1-solution β' to the original problem.

We use the following randomized rounding scheme: Let $\hat{x}_m$ be the value assigned to x_m in the solution to the linear program. For each m, where $1 \leq m \leq n$, set $\beta'[m]$ to 1 with probability $\hat{x}_m$ and to 0 with probability $1 - \hat{x}_m$.

To analyze this method, we can proceed analogously as in a standard analysis of the lattice approximation problem (see pp. 449–450 in [8]). We consider each inequality separately, and flip the probability from p_i to $1-p_i$ if $\alpha_i[m] = 1$. By using two variants of Chernoff's bounds (see pp. 449–450 in [8]) and defining the *generalized distance* between any $z_1, z_2 \in [0,1]^n$ as $d_G(z_1, z_2) = \sum_{m=1}^{n} |z_1[m] - z_2[m]|$, we obtain:

Lemma 4.1 *For an instance of HCP with k strings of length n, let r be the radius of the instance, and let β' be the approximative solution found by randomized rounding. Then the maximum Hamming distance between β' and any string in the instance is at most*

1. *$r + O(\sqrt{n \log n})$ with probability $\geq 1 - \frac{k}{n^2}$, and*
2. *$r(1+\varepsilon)$ with probability $\geq 1 - k \cdot 2\exp(-0.38\varepsilon^2 w)$, where w is the minimum generalized distance between a string in the instance and an optimal solution to the relaxed version of the instance.*

Corollary 4.1

1. *If the radius of the instance is larger than $q\sqrt{n \log n}$ then the maximum distance between β' and any string in the instance is at most $r + O(q^{-1})$ with probability $\geq 1 - \frac{k}{n^2}$.*
2. *If the minimum generalized distance w between a string in the instance and an optimal relaxed solution is at least $\frac{\ln(4k)}{0.38\varepsilon^2}$ then the maximum distance between β' and any string in the instance is at most $(1+\varepsilon)r$ with probability $\geq \frac{1}{2}$.*

In other words, the method of randomized rounding is likely to yield nearly optimal solutions if the radius r is substantially larger than $\sqrt{n \log n}$ and k is substantially smaller than n^2, or if the minimum generalized distance w is sufficiently large.

5 A Randomized $(\frac{4}{3}+\varepsilon)$-Approximation Algorithm

In this section, we give a randomized algorithm for HCP that with high probability yields answers which are within a factor of $(\frac{4}{3}+\varepsilon)$ of the optimum, where $\varepsilon > 0$ can be made arbitrarily small. The algorithm runs in polynomial time, provided the radius of the input instance is not very small.

Recall that r is defined as

$$r = \min_{\beta \in \mathbb{Z}_2^n} \left(\max_{1 \le i \le k} d(\beta, \alpha_i) \right)$$

where β is an optimal solution. The approximative solution found by our algorithm is called β', and we denote the value of $\max\limits_{1\le i\le k} d(\beta', \alpha_i)$ by r'.

We first describe the algorithm and then prove that $r' \le (\frac{4}{3}+\varepsilon)\, r$ with probability $\ge \frac{1}{2}$.

Description of the Algorithm

First of all, calculate the *diameter* f of the instance, where f is defined to be the maximum of all Hamming distances between any two of the given binary strings, i.e.,

$$f \;=\; \max_{1\le j\le k} \left(\max_{1\le i\le k} d(\alpha_i, \alpha_j) \right).$$

Next, compute $\frac{12.5\ln(4k)}{\varepsilon^3}$ using the specified value of ε.

If $f < \frac{12.5\ln(4k)}{\varepsilon^3}$ then let the algorithm branch to Case 1 below to obtain β'; if $f \ge \frac{12.5\ln(4k)}{\varepsilon^3}$ then obtain β' from Case 2.

<u>Case 1:</u> $f < \frac{12.5\ln(4k)}{\varepsilon^3}$

Note that $r \le f$ (since $r \le \min\limits_{1\le j\le k} \left(\max\limits_{1\le i\le k} d(\alpha_i, \alpha_j) \right) \le f$). Find an exact solution by brute force, i.e., by trying all of the $\sum\limits_{j=0}^{f} \binom{n}{j} = O(n^f)$ strings in $\mathbb{Z}_2^n$ within Hamming distance f of α_l, where l is chosen arbitrarily from $\{1, .., k\}$. Each such string can be evaluated in $O(kn)$ time; therefore, the time required is $O(k \cdot n^{f+1})$.

Case 2: $f \geq \frac{12.5 \ln(4k)}{\varepsilon^3}$

The second case is divided into two subcases: $f \leq \frac{4}{3}r$ and $f > \frac{4}{3}r$.

The algorithm can't know beforehand which subcase holds since r is still unknown. To get around this, the algorithm runs both procedures below and chooses the better one of the two solutions.

Subcase 2a: $f \leq \frac{4}{3}r$
Set $\beta' = \alpha_l$, where l is chosen arbitrarily from $\{1, .., k\}$.

Subcase 2b: $f > \frac{4}{3}r$
Rearrange the α_i's so that $d(\alpha_1, \alpha_k) = f$. Then, normalize the strings as follows:

old_$\alpha_1 := \alpha_1$
for $m := 1$ **to** n **do**
  if $\alpha_1[m] = 1$ **then**
    for $i := 1$ **to** k **do** $\alpha_i[m] := 1 - \alpha_i[m]$

This doesn't change any of the Hamming distances because whenever some bit in a string is changed, the corresponding bit in every other string is changed as well. Next, let some permutation σ act on the columns so that the f positions of α_k that contain 1s are located at $\alpha_k[1], .., \alpha_k[f]$ (see Figure 1). Note that this operation does not affect the Hamming distances either.

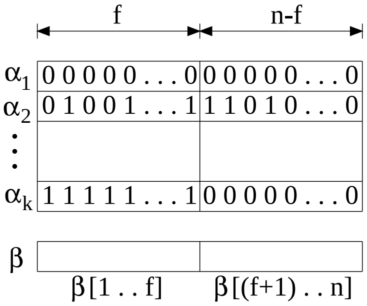

Figure 1: After normalization, the two strings that are the farthest apart are α_1 and α_k. They differ at precisely positions $1..f$.

Consider the 0-1-integer programming problem corresponding to the normalized instance with the integer constraints relaxed as described in Section 4. Add constraints which force the last $n - f$ positions of a solution to be 000..0. Let ρ be an optimal solution to this problem, and set γ to that $\alpha_i[1..f]\,0^{n-f}$, $i = 1, .., k$, which minimizes the generalized distance between γ and ρ. Next, apply the randomized rounding scheme from Section 4, and call the solution obtained μ. Set β' equal to the one of the two strings γ, μ with the smallest maximum distance to a string α_i, $i = 1, .., k$.

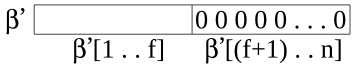

Figure 2: For Subcase 2b, the last $n-f$ positions of β' will always be 0.

Finally, to transform β' back to the original instance, apply σ^{-1} to β' and perform:

for $m := 1$ **to** n **do**
    if old_$\alpha_1[m] = 1$ **then**
        $\beta'[m] := 1 - \beta'[m]$

and Subcase 2b is done.

Performance Analysis

Now we analyze the algorithm's approximation ratio. We will show that for all cases, $r' \leq (\frac{4}{3} + \varepsilon)\, r$ with probability $\geq \frac{1}{2}$.

Case 1: $f < \frac{12.5 \ln(4k)}{\varepsilon^3}$

The algorithm finds an exact solution, so $r' = r$.

Subcase 2a: $f \geq \frac{12.5 \ln(4k)}{\varepsilon^3}$ and $f \leq \frac{4}{3} r$

Here $r' \leq f \leq \frac{4}{3} r$.

Subcase 2b: $f \geq \frac{12.5 \ln(4k)}{\varepsilon^3}$ and $f > \frac{4}{3} r$

First, we observe that an optimal solution β can't have a lot of ones in the last $n - f$ positions.

Lemma 5.1 $\beta[(f+1)..n]$ *contains less than* $\frac{1}{3} r$ *ones.*

Proof: Suppose $\beta[(f+1)..n]$ contains $\geq \frac{1}{3} r$ ones. Since $d(\beta, \alpha_1) \leq r$, the substring $\beta[1..f]$ must contain $\leq \frac{2}{3} r$ ones. Similarly, $d(\beta, \alpha_k) \leq r$ implies that $\beta[1..f]$ contains $\leq \frac{2}{3} r$ zeros. But this would mean that $f \leq \frac{2}{3} r + \frac{2}{3} r = \frac{4}{3} r$, which is a contradiction since $f > \frac{4}{3} r$. □

Next, define ζ to be a string in $\mathbb{Z}_2^n$ with zeros on the last $n - f$ positions that minimizes the value of $\max\limits_{1 \leq i \leq k} d(\zeta, \alpha_i)$.

Lemma 5.2 $\max\limits_{1\le i\le k} d(\zeta,\alpha_i) \le \frac{4r}{3}$.

Proof: Let ξ be the concatenation of the first f symbols of β and the string consisting of $n-f$ zeros, i.e., $\xi = \beta[1..f]\,0^{n-f}$. From the definitions and Lemma 5.1, we see that

$$\max_{1\le i\le k} d(\zeta,\alpha_i) \le \max_{1\le i\le k} d(\xi,\alpha_i) \le \max_{1\le i\le k}(d(\xi,\beta)+d(\beta,\alpha_i)) \le \frac{1}{3}r + r = \frac{4r}{3}.$$

□

Lemma 5.3 *If* $f \ge \frac{12.5\ln(4k)}{\varepsilon^3}$ *then* $r' \le (1+\frac{3\varepsilon}{4})\cdot \max\limits_{1\le i\le k} d(\zeta,\alpha_i)$ *with probability at least* $\frac{1}{2}$.

Proof: We have

$$r' = \max_{1\le i\le k} d(\beta',\alpha_i) \le \max_{1\le i\le k}(d_G(\beta',\rho)+d_G(\rho,\alpha_i)) \le d_G(\beta',\rho)+\max_{1\le i\le k} d(\zeta,\alpha_i).$$

Let w be the minimum generalized distance between ρ and $\alpha_i[1..f]\,0^{n-f}$, $i = 1,..,k$. If w is at most $\frac{3\varepsilon}{8}f$, then

$$d_G(\beta',\rho) \le d_G(\gamma,\rho) \le \frac{3\varepsilon}{8}f \le \frac{3\varepsilon}{4}r = \frac{3\varepsilon}{4}\cdot \max_{1\le i\le k} d(\beta,\alpha_i) \le \frac{3\varepsilon}{4}\cdot \max_{1\le i\le k} d(\zeta,\alpha_i)$$

since $f = d(\alpha_1,\alpha_k) \le d(\alpha_1,\beta)+d(\beta,\alpha_k) \le 2r$.

Otherwise, w satisfies $w > \frac{3\varepsilon}{8}f > \frac{\ln(4k)}{0.38(\frac{3\varepsilon}{4})^2}$. In this case, the lemma follows from the relation $\max\limits_{1\le i\le k} d(\beta',\alpha_i) \le \max\limits_{1\le i\le k} d(\mu,\alpha_i)$ along with a straightforward generalization of the second part of Corollary 4.1 to account for the constraint requiring zeros on the last $n-f$ solution positions, with ε set to $\frac{3\varepsilon}{4}$. □

Now, because $f \ge \frac{12.5\ln(4k)}{\varepsilon^3}$, it follows from Lemma 5.2 and Lemma 5.3 that

$$r' \le (1+\frac{3\varepsilon}{4})\cdot\frac{4r}{3} = (\frac{4}{3}+\varepsilon)\,r$$

with probability $\ge \frac{1}{2}$.

This concludes Subcase 2b. We have thus proved the main theorem of this section.

Theorem 5.1 *The approximative center* β' *found by the algorithm for an instance of HCP is within distance* $(\frac{4}{3}+\varepsilon)\,r$ *of every* α_i, $i = 1,..,k$, *with probability* $\ge \frac{1}{2}$. *If the diameter* f *of the input instance satisfies* $f \ge \frac{12.5\ln(4k)}{\varepsilon^3}$ *then* β' *will be constructed in polynomial time. If* $f < \frac{12.5\ln(4k)}{\varepsilon^3}$, *then* β' *is actually an optimal solution, and will be found in* $O(kn^{\frac{12.5\ln(4k)}{\varepsilon^3}+1})$ *time.*

Note that as long as f is superlogarithmic in k, the algorithm is guaranteed to run in polynomial time.

6 Another Type of Approximation

HCP can be approximated in other ways than described above. One is to find a set of strings B such that every given α_i, $i = 1,..,k$, is at a distance of at most r from some element in B, where r is the optimal radius. So, instead of looking for *one* optimal center β with radius r, we look for *a small set of* approximative centers that together are as good as β is by itself.

We can implement this idea as follows. Transform the given instance of HCP to its integer programming formulation as described in Section 3. By considering the linear programming relaxation of the formulation and using randomized rounding, we can find in polynomial time an approximative solution β'_1 that satisfies at least a quarter of the inequalities with probability greater than $\frac{1}{3}$. (To see this, let Z_1 be a random variable representing the number of inequalities that are not satisfied by β'_1. $\mathbf{E}[Z_1] = \frac{k}{2}$, so by the Markov inequality, $\mathbf{Pr}[Z_1 < \frac{3}{4}k] > \frac{1}{3}$.) After that, we remove all inequalities satisfied by β'_1, and iterate our method for the remaining inequalities to obtain a solution β'_2, remove all inequalities satisfied by β'_2, and so on. After K iterations, where $K = O(\log k)$, every inequality is satisfied by some β'_j ($j \in \{1,..,K\}$) with probability $\geq n^{-O(1)}$. Clearly, the set $B = \{\beta'_1,..,\beta'_K\}$ consists of a logarithmic (in k) number of strings that together fulfill the criterion $d(\beta'_j, \alpha_i) \leq r$, for some $j \in \{1,..,K\}$ for every $i \in \{1,..,k\}$.

By performing independent iterations of our algorithm until such a set B of cardinality $O(\log k)$ is found, we obtain the following theorem.

Theorem 6.1 *For an instance of HCP with binary strings α_i, $i = 1,..,k$, each of length n, we can find, in expected time polynomial in $n + k$, a set of binary strings β'_j, $j = 1,..,K$ with $K = O(\log k)$, such that for each α_i, $i = 1,..,k$, there is a $j \in \{1,..,K\}$ for which the Hamming distance between α_i and β'_j doesn't exceed the radius of the instance.*

7 Final Remarks

Special thanks go to Mike Paterson for pointing out the relation between the Hamming center problem and coding theory.

The randomized algorithm in Section 5 can be derandomized by using the method of conditional probabilities (see [12]).

Using techniques similar to the ones in this paper, Lanctot et al. [10] have independently found a $(\frac{4}{3} + \varepsilon)$-approximation algorithm for HCP that runs in polynomial time for all Hamming radii. By incorporating a simple modification inspired by their paper, our algorithm can also be made to run in polynomial time for all Hamming radii.

References

[1] T. Berger. *Rate Distortion Theory*. Prentice Hall, Englewood Cliffs, NJ, 1971.

[2] P. Berman, D. Gumucio, R. Hardison, W. Miller, and N. Stojanovic. A Linear-Time Algorithm for the 1-Mismatch Problem. *Proceedings of the 5th Workshop on Algorithms and Data Structures* (WADS'97), 1997, pp. 126–135.

[3] G.D. Cohen, M.G. Karpovsky, H.F. Mattson, Jr., and J.R. Schatz. Covering Radius – Survey and Recent Results. *IEEE Transactions on Information Theory*, Vol. IT-31, 1985, pp. 328–343.

[4] J.H. Conway and J.A. Sloane. Voronoi regions of lattices, second moments of polytopes, and quantization. *IEEE Transactions on Information Theory*, vol. IT-28, 1982, pp. 211–226.

[5] J.H. Conway and J.A. Sloane. Fast quantizing and decoding algorithms for lattice quantizers and codes. *IEEE Transactions on Information Theory*, vol. IT-28, 1982, pp. 227–232.

[6] M. Frances and A. Litman. On Covering Problems of Codes. *Theory of Computing Systems* **30**, 1997, pp. 113–119.

[7] L. Gąsieniec, J. Jansson, and A. Lingas. Efficient Approximation Algorithms for the Hamming Center Problem. *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA'99), 1999, pp. S905–S906.

[8] D.S. Hochbaum (editor). *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, Boston, 1997.

[9] B. Kolman, R. Busby, and S. Ross. *Discrete Mathematical Structures* [3rd ed.]. Prentice Hall, New Jersey, 1996.

[10] J.K. Lanctot, M. Li, B. Ma, S. Wang and L. Zhang. Distinguishing String Selection Problems. *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA'99), 1999, pp. 633–642.

[11] A.K. Lenstra, H.W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.* 261, 1982, pp. 515–534.

[12] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[13] C. Papadimitriou. On the Complexity of Integer Programming. *Journal of the ACM*, Vol. 28, No. 4, 1981, pp. 765–768.

[14] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, New York, 1986.

[15] S. Vishwanathan. An $O(\log^* n)$ Approximation Algorithm for the Asymmetric p-Center Problem. *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA'96), 1996, pp. 1–5.