Ernst W. Mayr (Ed.)

LNCS 9224 ARCoSS

# Graph-Theoretic Concepts in Computer Science

41st International Workshop, WG 2015
Garching, Germany, June 17–19, 2015
Revised Papers

Springer

**Lecture Notes in Computer Science** **9224**

*Commenced Publication in 1973*
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen



Advanced Research in Computing and Software Science
Subline of Lecture Notes in Computer Science

More information about this series at http://www.springer.com/series/7407

Ernst W. Mayr (Ed.)

# Graph-Theoretic Concepts in Computer Science

41st International Workshop, WG 2015
Garching, Germany, June 17–19, 2015
Revised Papers

Springer

*Editor*
Ernst W. Mayr
TU München Institut für Informatik
Garching
Germany

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-662-53173-0 ISBN 978-3-662-53174-7 (eBook)
DOI 10.1007/978-3-662-53174-7

Library of Congress Control Number: 2016947500

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues



Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer-Verlag GmbH Berlin Heidelberg

# Preface

The 41st International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2015) was held in Garching near Munich in Germany, during June 17–19, 2015. The WG conference series has a long tradition. Since 1975, it has taken place 23 times in Germany, four times in The Netherlands, three times in France, twice in Austria and in the Czech Republic, as well as once in each of Italy, Slovakia, Switzerland, Norway, the UK, Greece, and Israel. The WG conferences aim to connect theory and practice by demonstrating how graph-theoretic concepts can be applied to various areas of computer science and by extracting new graph problems from applications. Their goal is to present new research results and to identify and explore directions of future research. WG 2015 had 79 submissions. Each submission was carefully reviewed by three members of the Program Committee. The Program Committee then accepted 32 papers for presentation at WG 2015.

The WG 2015 Best Paper Award, sponsored by Springer, was awarded to Konstantinos Stavropoulos and his co-authors Martin Grohe, Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz, for their paper on "Colouring and Covering Nowhere Dense Graphs." The program also included three inspiring invited talks: Daniel Paulusma (Durham University, UK) gave a talk on "Open Problems on Graph Coloring for Special Graph Classes," Shmuel Zaks (Technion, Haifa, Israel) spoke "On the Complexity of Approximation and On-line Scheduling Problems with Applications to Optical Networks," and Rolf Niedermeier (TU Berlin, Germany) presented "Parameterized Algorithmics for Graph Modification Problems: On Interactions with Heuristics."

We would like to thank all the authors of the papers submitted to WG 2015, the speakers of the 32 contributed and the three invited talks, the members of the Program Committee, and all the 130 external reviewers. Special thanks also go to the Leibniz Supercomputing Centre (LRZ) of the Bavarian Academy of Sciences and Humanities for providing space and support for the sessions and the coffee breaks, and to the members of the local Organizing Committee and the members of the Chair for Efficient Algorithms of the Technical University of Munich (TUM), whose effort made the conference run smoothly and led to such a successful event. Finally, we want to express our thanks for the financial support we received from Springer for the best paper award and from Deutsche Forschungsgemeinschaft (DFG) for (most of) the conference participants.

May 2016 Ernst W. Mayr

# Organization

## Program Committee

| | |
|---|---|
| Hajo Broersma | Twente, The Netherlands |
| L. Sunil Chandran | Bangalore, India |
| Jianer Chen | College Station, USA |
| Victor Chepoi | Marseille, France |
| Pinar Heggernes | Bergen, Norway |
| Juraj Hromkovič | Zurich, Switzerland |
| Klaus Jansen | Kiel, Germany |
| Michael Kaufmann | Tübingen, Germany |
| Jan Kratochvíl | Prague, Czech Republic |
| Dieter Kratsch | Metz, France |
| Van Bang Le | Rostock, Germany |
| Ernst W. Mayr | München (Chair), Germany |
| Ross McConnell | Fort Collins, USA |
| Bojan Mohar | Burnaby, Canada |
| Haiko Müller | Leeds, UK |
| Christophe Paul | Montpellier, France |
| Dieter Rautenbach | Ulm, Germany |
| Dimitrios Thilikos | Montpellier/Athens, France/Greece |
| Oren Weimann | Haifa, Israel |

## Organizing Committee

| | |
|---|---|
| Ernst Bayer | TUM, München, Germany |
| Christine Lissner | TUM, München, Germany |
| Ernst W. Mayr | TUM, München (Chair), Germany |
| Helga Tyroller | LRZ, Garching, Germany |

The conference received ample support from the Department of Informatics of the Technical University of Munich (TUM) and also (and in particular) the Leibniz Supercomputing Centre (LRZ) of the Bavarian Academy of Sciences and Humanities.

## Additional Reviewers

| | |
|---|---|
| Patrizio Angelini | Michael Bekos |
| Jasine Babu | Maria Paola Bianchi |
| Kfir Barhum | Markus Blaeser |
| Julien Baste | Hans L. Bodlaender |

Hans-Joachim Boeckenhauer
Edouard Bonnet
Paul Bonsma
Magnus Bordewich
Marin Bougeret
Till Bruckdorfer
Roberto Bruni
Jeremie Chalopin
Dimitris Chatzidimitriou
Rajesh Chitnis
Basile Couëtoux
Bruno Courcelle
Konrad Kazimierz Dabrowski
Guillaume Ducoffe
Vida Dujmović
Martin Dyer
Thomas Erlebach
Bertrand Estellon
Josep Fàbrega
Stefan Felsner
Qilong Feng
Jiri Fiala
Valentin Garnero
Michael Gentner
Archontia Giannopoulou
Petr Golovach
Martin Golumbic
Laurent Gourves
Sathish Govindarajan
Alexander Grigoriev
Jiong Guo
Gregory Gutin
Frederic Havet
Danny Hermelin
Bart M.P. Jansen
Vít Jelínek
Felix Joos
Tomas Kaiser
Christos Kaklamanis
Frank Kammer
Mamadou Moustapha Kanté
Telikepalli Kavitha
Ralf Klasing
Kolja Knauer
Ekkehard Köhler
Dennis Komm
Miklós Krész
Murali Krishnan
Robert Krug
Sacha Krug
Piyush Kurur
O-Joung Kwon
Arnaud Labourel
Benjamin Leveque
Bernard Lidicky
Mathieu Liedloff
Andrzej Lingas
Giuseppe Liotta
Antoine Lobstein
Anna Lubiw
Christian Lwenstein
Jens Maberg
Frederic Maffray
Spyridon Maniatis
Martin Mares
Rogers Mathew
Klaus Meer
Daniel Meister
George Mertzios
Neeldhara Misra
Pranabendu Misra
Dieter Mitsche
Lalla Mouatadid
Shay Mozes
Guyslain Naves
Jaroslav Nesetril
Jan Obdrzalek
Pascal Ochem
Daniel Paulusma
Chris Pinkau
Alexandre Pinlou
Sheung-Hung Poon
Andrzej Proskurowski
Roman Rabinovich
Deepak Rajendraprasad
Michael Rao
Jean-Florent Raymond
Igor Razgon
Clément Requilé
Ignaz Rutter
Katarzyna Rybarczyk
Ignasi Sau

Ingo Schiermeyer
Chintan Shah
Feng Shi
Somnath Sikdar
Nitin Singh
Naveen Sivadasan
R. Sritharan
Björn Steffen
Lorna Stewart
Jayme Szwarcfiter
Jan Arne Telle
Ioan Todinca
Stefan Toman
Hanjo Täubig
Torsten Ueckerdt
Ali Vakilian
Petru Valicov
Erik Jan van Leeuwen
Rob van Stee
Yann Vaxès
Sundar Vishwanathan
Jan Vondrak
Jeremias Weihmann
Samuel Wilson
Steve Wismath
Gerhard J. Woeginger
Bang Ye Wu
Mingyu Xiao
Jie You
Christian Zielke

## Sponsors

We gratefully acknowledge generous support for the conference by Deutsche Forschungsgemeinschaft (DFG), project MA 890/14–1, and by Springer-Verlag for the Best Paper Award.

More details on the conference are available at
http://wwwmayr.in.tum.de/konferenzen/WG2015/

# Contents

# On the Tree Search Problem with Non-uniform Costs

Ferdinando Cicalese[1(✉)], Balázs Keszegh[2], Bernard Lidický[3], Dömötör Pálvölgyi[4], and Tomáš Valla[5]

[1] Department of Computer Science, University of Verona, Verona, Italy
cicalese@dia.unisa.it
[2] Rényi Institute, Budapest, Hungary
keszegh.balazs@renyi.mta.hu
[3] Department of Mathematics, Iowa State University, Ames, USA
lidicky@iastate.edu
[4] Eötvös University, Budapest, Hungary
dom@cs.elte.hu
[5] Faculty of Information Technology, Czech Technical University, Prague, Czech Republic
tomas.valla@fit.cvut.cz

**Abstract.** Searching in partially ordered structures has been considered in the context of information retrieval and efficient tree-like indices, as well as in hierarchy based knowledge representation. In this paper we focus on tree-like partial orders and consider the problem of identifying an initially unknown vertex in a tree by asking edge queries: an edge query $e$ returns the component of $T-e$ containing the vertex sought for, while incurring some known cost $c(e)$.

The Tree Search Problem with Non-Uniform Cost is the following: given a tree $T$ on $n$ vertices, each edge having an associated cost, construct a strategy that minimizes the total cost of the identification in the worst case.

Finding the strategy guaranteeing the minimum possible cost is an NP-complete problem already for input trees of degree 3 or diameter 6. The best known approximation guarantee was an $O(\log n/\log\log\log n)$-approximation algorithm of [Cicalese et al. TCS 2012].

B. Keszegh—Research supported by Hungarian National Science Fund (OTKA), under grant PD 108406 and under grant NN 102029 (EUROGIGA project GraDR 10-EuroGIGA-OP-003) and the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

B. Lidický—Research is partially supported by NSF grants DMS-1266016 and DMS-1600390.

D. Pálvölgyi—Research supported by Hungarian National Science Fund (OTKA), under grant PD 104386 and under grant NN 102029 (EUROGIGA project GraDR 10-EuroGIGA-OP-003) and the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

T. Valla—Supported by the Centre of Excellence – Inst. for Theor. Comp. Sci. (project P202/12/G061 of GA ČR).


E.W. Mayr (Ed.): WG 2015, LNCS 9224, pp. 90–102, 2016.
DOI: 10.1007/978-3-662-53174-7_7

We improve upon the above results both from the algorithmic and the computational complexity point of view: We provide a novel algorithm that provides an $O(\frac{\log n}{\log \log n})$-approximation of the cost of the optimal strategy. In addition, we show that finding an optimal strategy is NP-hard even when the input tree is a spider of diameter 6, i.e., at most one vertex has degree larger than 2.

## 1 Introduction

The design of efficient procedures for searching in a discrete structure is a fundamental problem in discrete mathematics [1,2] and computer science [10]. Searching is a basic primitive for building and managing operations of an information system as ordering, updating, and retrieval. The typical example of a search procedure is binary search which allows to retrieve an element in a sorted list of size $n$ by only looking at $O(\log n)$ elements of the list. If no order can be assumed on the list, then it is known that any procedure will have to look at the complete list in the worst case. Besides these two well characterized extremes, extensive work has also been devoted to the case where the underlying structure of the search space is a partial order. Partial orders can be used to model lack of information on the totally ordered elements of the search space [12] or can naturally arise from the relationship among the elements of the search space, like in hierarchies used to model knowledge representation [15], or in tree-like indices for information retrieval of large databases [3]. For more about applications of tree search see the end of this section.

In this paper, we focus on the case where the underlying search space is a tree-like partially ordered set and tests have nonuniform costs. We investigate the following problem.

The Tree Search Problem with non-uniform costs

*Input*: A tree $T = (V, E)$, $|V| = n$, with non-negative rational costs assigned to the edges defined by a $c : e \in E \mapsto c(e) \in \mathbb{Q}$.

*Output*: A strategy that minimizes (in the worst case) the cost spent to identify an initially unknown vertex $x$ of $T$ by using *edge queries*. An *edge query* $e = \{u, v\} \in E$ asks for the subtree $T_u$ or $T_v$ which contains $x$, where $T_u$ and $T_v$ are the (maximal connected) components of $T - e$, including the vertex $u$ and $v$ respectively. The cost of the query $e$ is $c(e)$. The cost of identifying a vertex $x$ is the sum of the costs of the queries asked.

More formally, a strategy for the Tree Search Problem with nonuniform costs over the tree $T$ is a *decision tree* $D$ which is a rooted binary tree with $|V|$ leaves where every leaf $\ell$ is associated with one vertex $v \in V$ and every internal node[1] $\nu \in V(D)$ is associated with one test $e = \{u, v\} \in E$. The outgoing edges from $\nu$ are associated with the possible outcomes of the query, namely, to the case

[1] For the sake of avoiding confusion between the input tree and the decision tree, we will reserve the term vertex for the elements of $V$ and the term *node* for the vertices of the decision tree $D$.

where the vertex to identify lies in $T_u$ or $T_v$ respectively. Every vertex has at least one associated leaf. The actual identification process can be obtained from $D$ starting with the query associated to the root and moving towards the leaves based on the answers received. When a leaf $\ell$ is reached, the associated vertex is output (see also Fig. 1 in the appendix for an example).

Given a decision tree $D$, for each vertex $v \in V(T)$, let $cost^D(v)$ be the sum of costs of the edges associated to nodes on the path from the root of $D$ to the leaf identifying $v$. This is the total cost of the queries performed when the strategy $D$ is used and $v$ is the vertex to be identified.

In addition, let the cost of $D$ be defined by

$$cost(D) = \max_{v \in V(T)} cost^D(v).$$

This is the worst-case cost of identifying a vertex of $T$ by the decision tree $D$. The optimal cost of a decision tree for the instance represented by the tree $T$ and the cost assignment $c$ is given by

$$OPT(T, c) = \min_D cost(D),$$

where the min is over all decision trees $D$ for the instance $(T, c)$.

**Previous Results and Related Work.** The Tree Search Problem has been first studied under the name of tree edge ranking [5,7,9,11,13], motivated by multi-part product assembly. In [11] it was shown that in the case where the tests have uniform cost, an optimal strategy can be found in linear time. A linear algorithm for searching in a tree with uniform cost was also provided in [14]. Independently of the above articles, the first paper where the problem is considered in terms of searching in a tree is [3], where the more general problem of searching in a poset was also addressed.

The variant considered here in which the costs of the tests are non-uniform was first studied by Dereniowski [6] in the context of edge ranking. In this paper, the problem was proved NP-complete for trees of diameter at most 10. Dereniowski also provided an $O(\log n)$ approximation algorithm. In [4] Cicalese et al. showed that the tree search problem with non-uniform costs is strongly NP-complete already for input trees of diameter 6, or maximum degree 3, moreover, these results are tight. In fact, in [4], a polynomial time algorithm computing the optimal solution is also provided for diameter 5 instances and an $O(n^2)$ algorithm for the case where the input tree is a path. For arbitrary trees, Cicalese et al. provided an $O(\frac{\log n}{\log \log \log n})$-approximation algorithm.

**Our Result.** Our contribution is both on the algorithmic and on the complexity side. On the one hand, we provide a new approximation algorithm for the tree search problem with non-uniform costs which improves upon the best known guarantee given in [4]. In Sect. 3 we will prove the following result.

**Theorem 1.** *There is an $O(\log n / \log \log n)$-approximation algorithm for the Weighted Tree Search Problem that runs in polynomial time in $n$.*

A *spider graph* (henceforth simply referred to as a *spider*) is a tree with at most one vertex of degree larger than 2.

In this paper, we also show that the tree search problem with non-uniform costs is NP-hard already when the input tree is a *spider* of diameter 6.

**More About Applications.** We discuss some scenarios in which the problem of searching in trees with non-uniform costs naturally arises.

Consider the problem of locating a buggy module in a program in which the dependencies between different modules can be represented by a tree. For each module we can verify the correct behavior independently. Such a verification may consist in checking, for instance, whether all branches and statements in a given module work properly. For different modules, the cost of using the checking procedure can be different (here the cost might refer to the time to complete the check). In such a situation, it is important to device a debugging strategy that minimizes the cost incurred in order to locate the buggy module in the worst case.

Checking for consistency in different sites keeping distributed copies of tree-like data structures (e.g., file systems) can be performed by maintaining at each node some check sum information about the subtree rooted at that node. Tree search can be used to identify the presence of "buggy nodes", and efficiently identifying the inconsistent part in the structure, rather than retransmitting or exhaustively checking the whole data structure. In [3], an application of this model in the area of information retrieval is also described.

Another example comes from a class of problems which is in some sense dual to the previous ones: deciding the assembly schedule of a multi-part device. Assume that the set of pairs of parts that must be assembled together can be represented by a tree. Each assembly operation requires some (given) amount of time to be performed and while assembling two pieces, the same pieces cannot be involved in any other assembly operations. At any time different pairs of parts can be assembled in parallel. The problem is to define the schedule of assembly operations which minimize the total time spent to completely assembly the device. The schedule is an edge ranking of the tree defined by the assembly operations. By reversing the order of the assembly operation in the schedule we obtain a decision tree for the problem of searching in the tree of the assembly operation where each edge cost is equal to the cost of the corresponding assembly.

## 2 Basic Lower and Upper Bounds

In this section we provide some preliminary results which will be useful in the analysis of our algorithm presented in the next section. We introduce some lower bounds on the cost of the optimal decision tree for a given instance of the problem. We also recall two exact algorithms for constructing optimal decision trees which were given in [4]. The first is an exponential time dynamic programming algorithm which works for any input tree. The second is a quadratic time algorithm for instances where the input tree is a path. Finally, we show a construction of 2-approximation decision trees for spider graphs.

Let $T$ denote the input tree and $c$ the cost function. It is not hard to see that, given a decision tree $D$ for $T$, we can extract from it a decision tree for the instance of the problem defined on a subtree $T'$ of $T$ and the restriction of $c$ to the vertices in $T'$. For this, we can repeatedly apply the following operation: if in $D$ there is a node $\nu$ associated with an edge $e = \{u, v\}$, such that $T_u$ (resp. $T_v$) is included $T - T'$, then remove the node $\nu$ together with the subtree rooted at the child of $\nu$ corresponding to the case where the vertex to identify is in $T_u$ (resp. $T_v$). Let $D'$ be the resulting decision tree when the above step cannot be performed any more. Then, clearly $cost(D', c) \leq cost(D, c)$. We have shown the following (also observed in [4]).

**Lemma 1.** *Let $T'$ be a subtree of $T$. Then, $OPT(T, c) \geq OPT(T', c)$.*

Another immediate observation is that for a given input tree $T$, the value $OPT(T, c)$ is monotonically non-decreasing with respect to the cost of any edge. This is recorded in the following.

**Lemma 2.** *Let $c$ and $c'$ be cost assignments on a tree $T$ such that $c'(e) \leq c(e)$ for every $e \in E(T)$. Then, $OPT(T, c) \geq OPT(T, c')$.*

The next proposition shows that subdividing an edge cannot decrease the cost of the optimal decision tree.

**Proposition 1.** *Let $c$ be a cost assignment on a tree $T$. Let $v \in V(T)$ have exactly two neighbors $u_1, u_2 \in V(T)$. If $T'$ is obtained from $T - v$ by adding the edge $\{u_1, u_2\}$ and $c'$ is obtained from $c$ by setting $c'(u_1u_2) = \min\{c(u_1v), c(u_2v)\}$, then $OPT(T, c) \geq OPT(T', c')$.*

*Proof.* Let $D$ be an optimal decision tree for the instance $(T, c)$. Let us assume without loss of generality that in $D$ the node $\nu_1$ associated with $e_1 = \{u_1, v\}$ is an ancestor of the node $\nu_2$ associated with $e_2 = \{u_2, v\}$. Notice that one of the children of $\nu_2$ is a leaf associated with the vertex $v$. Let $\tilde{D}$ be the subtree of $D$ rooted at the non-leaf child of $\nu_2$.

Let $D'$ be the decision tree obtained from $D$ by associating the node $\nu_1$ to the edge $e = \{u_1, u_2\}$ and replacing the subtree rooted at $\nu_2$ with the subtree $\tilde{D}$.

It is not hard to see that $D'$ is a proper decision tree for $T'$. We also have that for any vertex $z$ of $T'$ which is associated to a leaf in $\tilde{D}$ it holds that $cost^{D'}(z) = cost^{D}(z) - c(e_1) - c(e_2) + c'(u_1u_2)$, and for any other vertex $z$ of $T'$ we have $cost^{D'}(z) = cost^{D}(z) - c(e_1) + c'(u_1u_2)$ or $cost^{D'}(z) \leq cost^{D}(z)$. It follows that $OPT(T', c') \leq cost(D') \leq cost(D) = OPT(T, c)$. □

The following two results from [4] provide exact algorithms for the construction of optimal strategies. More precisely, Proposition 2 provides an exponential dynamic programming based algorithm for general trees. Theorem 2 gives an $O(n^2)$ time algorithm for the special case where the input tree is a path and will be useful in the analysis of our main algorithm and also in Lemma 3 regarding the spider tree.

**Proposition 2** [4]. *Let $T$ be an edge-weighted tree on $n$ vertices. Then an optimal decision tree for $T$ can be constructed in $O(2^n n)$ time.*

The following theorem was proved by Cicalese et al. in [4] and will be useful later in the analysis of our algorithm and also in the following lemma regarding the spider tree.

**Theorem 2** [4]. *There is an $O(n^2)$ time algorithm that constructs an optimal decision tree $D$ for a given weighted path on $n$ vertices.*

Note that for a star $T$ any decision tree $D$ has the same cost, since all the edges have to be asked in the worst case. Hence, for a tree $T$ such that there is only one node with degree greater than 1 we have $OPT(T, c) = \sum_{e \in E(T)} c(e)$, for any cost function $c$.

**Definition 1.** *A tree $T$ is a* spider *if there is at most one vertex in $T$ of degree greater than two. We refer to this vertex as the* head *(or* center*) of the spider. In the special case when all vertices have degree at most 2 then an arbitrarily chosen vertex of degree 2 is designated to be the* head *of the spider. Moreover, each path from the head of the spider to one of the leaves will be referred to as a* leg *of the spider.*

**Lemma 3.** *Let $T$ be a spider. Then there is an algorithm which computes a 2-approximate decision tree $D$ for $T$ and runs in time $O(n^2)$.*

*Proof.* If $T$ is a path, then by Theorem 2 there exists an algorithm computing the optimal decision tree in $O(n^2)$ time. Assume $T$ is not a path. Then $T$ contains exactly one vertex $v$ of degree at least three. Let $S_v$ be the star induced by $v$ and the vertices adjacent to $v$. Let us denote by $w_1, \ldots, w_k$ the vertices adjacent to $v$, where $k = \deg(v)$. By Theorem 2, for every $i \in \{1, \ldots, k\}$ we construct the optimal decision tree $D_i$ for the path component $C_i$ of $T - v$ containing $w_i$ in time $O(|C_i|^2)$. Note that the total running time for construction of $D_1, \ldots, D_k$ is $O(n^2)$. Finally, for $S_v$ we compute the optimal decision tree $D_v$ (in $O(n)$ time). The decision tree $D$ for $T$ is obtained from $D_v$ by replacing the node corresponding to $w_i$ by the root of $D_i$ for every $i \in \{1, \ldots, k\}$. Clearly, the algorithm runs in $O(n^2)$ time and $cost(D) \leq OPT(S_v, c) + \max_{1 \leq i \leq k}\{OPT(C_i, c)\} \leq 2OPT(T, c)$. The last inequality follows because by Lemma 1 both $OPT(S_v, c)$ and $\max_{1 \leq i \leq k}\{OPT(C_i, c)\}$ are lower bounds on $OPT(T, c)$. □

## 3 The Algorithm

In this section we present Algorithm TS for the tree search problem—we also provide a pseudocode of the algorithm in the appendix.

Let $n$ be the size of the input tree and $t = 2^{\lfloor \log \log n \rfloor + 2}$ be a parameter fixed for the whole run of the algorithm. It holds that $2 \log n \leq t \leq 4 \log n$.

The basic idea of our algorithm is to construct a subtree $S$ of the input tree $T$ such that: (i) we can construct a decision tree for $S$ whose cost is at most a

constant times the cost of an optimal decision tree for $S$; (ii) each component of $T - S$ has size not larger than $|T|/t$.

This will allow us to build a decision tree for $T$ by assembling the decision tree for $S$ with the decision trees recursively constructed for the components of $T - S$. The constant approximation guarantee on $S$ and the fact that, due to the size of the subtrees on which we recur, we need at most $O(\frac{\log n}{\log\log n})$ levels of recursion to show that our algorithm gives an $O(\frac{\log n}{\log\log n})$ approximation.

**The Subtree *S*.** We iteratively build subtrees $S_0 \subset S_1 \subset \cdots \subset S_t \subseteq T$. Starting with the empty tree $S_0$, in every iteration $i \in \{1, \ldots, t\}$ we pick a centroid[2] $x_i$ of the largest component of the forest $T - S_{i-1}$. The subtree $S_i$ is set to be the minimal subtree containing $x_i$ and $S_{i-1}$. If for some $i$ we have that $S_i = T$, then we set $S = S_i = T$ and we stop the iterations. If all $t$ iterations are completed, then we set $S = S_t$.

We have the following lemma—which establishes (ii) above.

**Lemma 4.** *If $H$ is a component of $T - S$, then $|H| \leq |T|/\log|T|$.*

*Proof.* We prove by induction on $k$ that after $2^k$ iterations all components of $T - S_{2^k}$ have size at most $|T|/2^{k-1}$. Let $k = 0$. We observe that by the definition of centroid, after $1 = 2^k$ iteration all components of $T - S_1$ have size at most $|T|/2 \leq 2|T| = |T|/2^{k-1}$. This establishes the basis of our induction.

Now fix some $k > 0$ and assume (induction hypothesis) that after $2^{k-1}$ iterations all components of $T - S_{2^{k-1}}$ have size at most $|T|/2^{k-2}$. Among these there are $p \leq 2^{k-1}$ components that have size at least $|T|/2^{k-1}$. In the next $p$ iterations we will choose a centroid in each of these components, one by one. Choosing a centroid in a component $H$ splits $H$ into parts that have size at most half of $H$, thus after $2^k = 2^{k-1} + p$ steps all components of $T - S_{2^k}$ have size at most $|T|/2^{k-1}$.

Thus, if the process of constructing $S$ is stopped after $t = 2^{\lfloor \log\log n \rfloor + 2}$ iterations all components have size at most $|T|/2^{\lfloor \log\log n \rfloor + 1} \leq |T|/\log n$. On the other hand, if the process of constructing $S$ is stopped at some iteration $i < t$, then it means that $S = T$ and trivially $|H| = 1$. □

**The Decision Tree for *S*.** Let $X$ contain all $x_i$ for $i \in \{1, \ldots, t\}$ and vertices of degree at least three in $S$. Note that $|X| \leq 2t - 2$ for $t \geq 2$ and $|X| = 1 = 2t - 1$ for $t = 1$. Indeed, by induction on $k$, this is true for $k = 1, 2$. Adding a new $x_i$, $S_i$ is a tree which is the union of $S_{i-1}$ and a path reaching to $x_i$, thus $S_i$ has at most one more vertex of degree at least three than $S_{i-1}$. Together with $x_i$, $|X|$ increases by at most two in a step. Let $P_{u,v}$ be the path of $T$ whose endpoints are vertices $u$ and $v$.

We define an auxiliary tree $Y$ on the vertex set $X$ in which the paths of $T$ between the vertices of $X$ are replaced by 'shortcut' edges. Vertices $u, v \in X$ form an edge of $Y$ if $u$ and $v$ are the only vertices of $X$ of the path $P_{u,v}$ in

[2] Recall that a *centroid* of a tree $T$ is a vertex $v$ such that any component of $T - v$ has size at most $|T|/2$.

$T$ with endpoints $u$ and $v$. Let $e_{uv} = \arg\min_{e \in P_{u,v}} c(e)$ (the edge of $P_{u,v}$ with minimal cost) and $c_Y(uv) = c(e_{uv})$. Let $Z = \bigcup_{uv \in E(Y)} e_{uv}$. By Proposition 2, we can compute an optimal decision tree $D_Y$ for $Y$ in time $O(2^{2t}t)$ which is polynomial in $n$ (Fig. 2).

Let $D_X$ be obtained from $D_Y$ by changing the label of every internal node from $uv$ to $e_{uv}$, for each $uv \in E(Y)$. The tree $D_X$ is not a decision tree for $S$, however, leaves of $D_X$ correspond to components of $S - Z$. Notice that $cost(D_X) = cost(D_Y) = OPT(Y, c_Y)$.

Since every component $C$ of $S - Z$ contains at most one vertex of degree at least three, every such component is a spider. By Lemma 3, a decision tree $D_C$ for each such component $C \in S - Z$ can be computed in $O(n^2)$ time with approximation ratio 2.

We can now obtain the decision tree $D_S$ for $S$ by replacing each leaf in $D_X$ with the decision tree for the corresponding component in $S - Z$. We have

$$\frac{cost(D_S)}{OPT(S,c)} \leq \frac{cost(D_X) + \max_{C \in S-Z} cost(D_C)}{OPT(S,c)} \leq \frac{cost(D_X)}{OPT(Y,c_Y)} + \max_{C \in S-Z} \frac{cost(D_C)}{OPT(C,c)} \leq 3, \quad (1)$$

where the second inequality holds because a repeated application of Proposition 1 implies $OPT(Y, c_Y) \leq OPT(S, c)$ and Lemma 1 implies $OPT(C, c) \leq OPT(S, c)$.

**Assembling the Pieces in the Decision Tree for $T$.** Let $v$ be a vertex in $S$ with a neighbor not in $S$, let $S_v$ be the star induced by $v$ and its neighbors outside $V(S)$.

Let $D_v$ be a decision tree for $S_v$ (notice that they all have the same cost). For every neighbor $w \notin V(S)$ of $v$ we compute recursively the decision tree $D_w$ for the component $H_w$ of $T - S$ containing $w$ and replace the leaf node of $D_v$ associated to $w$ with the root of $D_w$. The result is a decision tree $D'_v$ for the subtree of $T$ including $S_v$ and all the components of $T - S$ including some neighbor $w$ of $v$.

In order to obtain a decision tree $D_T$ for $T$ we now modify $D_S$ as follows: for each vertex $v$ in $S$ with a neighbor not in $S$, replace the leaf in $D_S$ associated with $v$ with the decision tree $D'_v$ computed above.

**The Approximation Guarantee for $D_T$.** Let $APP(T) = \frac{cost(D_T)}{OPT(T,c)}$ denote the approximation ratio obtained by Algorithm TS on the instance $(T, c)$. Let $APP(k) = \max_{|T| \leq k} APP(T)$.

**Lemma 5.** *For any tree $T$ on $n$ vertices and any cost assignment $c$, we have $APP(T) \leq 4 \log n / \log \log n$.*

*Proof.* For every $1 \leq k \leq n$ let $f(k) = \max\{1, 4 \log k / \log \log n\}$. We shall prove by induction on $k$ that $APP(k) \leq f(k)$, which implies the statement of the lemma.

If $|T| \leq t$, then our algorithm builds an optimal decision tree, thus $APP(k) = 1 \leq f(k)$ for $k \leq t$. This establishes the induction base.

Choose a tree $T$ as in the statement of the lemma such that $APP(T) = APP(n)$. Let $S$ and $Y$ be the substructures of $T$ built by the algorithm as described above. Let $\tilde{V}$ be the set of vertices of $S$ with some neighbor not in $S$. For each $w \notin V(S)$ let $H_w$ be the component of $T - S$ containing $w$. Let $\mathcal{H}$ be the set of components of $T - S$. Then, by construction, we have

$$APP(T) = \frac{ALG(T)}{OPT(T)} \tag{2}$$

$$\leq \frac{cost(D_S) + \max_{v \in \tilde{V}} cost(D_v) + \max_{w \notin V(S)} cost(D_w)}{OPT(T, c)} \tag{3}$$

$$\leq \frac{cost(D_S)}{OPT(S, c)} + \max_{v \in \tilde{V}} \frac{cost(D_v)}{OPT(S_v, c)} + \max_{w \notin V(S)} \frac{cost(D_w)}{OPT(H_w, c)} \tag{4}$$

$$\leq 4 + \max_{H \in \mathcal{H}} \frac{ALG(H)}{OPT(H, c)} = 4 + \max_{H \in \mathcal{H}} \{APP(H)\} \tag{5}$$

$$\leq 4 + \max_{H \in \mathcal{H}} f(|H|) \leq 4 + f(|T|/\log n) \tag{6}$$

$$= 4 + f(n/\log n) = 4 + \frac{4 \log \frac{n}{\log n}}{\log \log n} = \frac{4 \log n}{\log \log n}, \tag{7}$$

where

- (4) follows from (3) because of $OPT(S, c), OPT(S_v, c), OPT(H_w, c) \leq OPT(T, c)$ (Lemma 1)
- (5) follows from (4) because of (1) we have $\frac{cost(D_S)}{OPT(S,c)} \leq 3$ and because any decision tree for a star $S_v$ has the same cost, hence also equal to $OPT(S_v, c)$
- in (6) the first inequality follows by induction and the second inequality by Lemma 4
- (7) follows from (6) because of $|T| = n$ and the definition of $f(\cdot)$. □

**Lemma 6.** *For a tree $T$ on $n$ vertices, the Algorithm TS builds the decision tree $D_T$ in time polynomial in $n$.*

*Proof.* If $|T| \leq t$, then the algorithm builds an optimal decision tree for $T$ in time $O(2^t \cdot t) = O(n^4)$ using the construction from Proposition 2. Otherwise, every iteration needed to build the subtree $S$ (lines 7–11 of the algorithm) introduces one new vertex $x_i$ and at most one other vertex of degree at least three, thus $|X| \leq 2t - 1$. Proposition 2 then implies that an optimal decision tree $D_Y$ for $Y$ can be computed in time $O(2^{2t} \cdot 2t)$ which is polynomial in $n$. By Lemma 3, the 2-approximation decision tree $D_H$ for $H$ can be computed in $O(n^2)$ time. Building the decision tree $D_v$ for the stars $S_v$ takes $O(|S_v|)$ time (line 30). The rest of the algorithm, not counting the recursion on line 34, needs time $O(n^2)$. As the recursion is for a graph whose size is at most half of the original, the overall algorithm running time is polynomial in $n$. □

Lemmas 5 and 6 now imply Theorem 1.

## 4 Tree Search with Non-uniform Costs is NP-hard on Spider Graphs

In this section we provide a new hardness result which contributes to refining the separation between hard and polynomial instances of the tree search problem with non-uniform costs. We show that the problem of finding a minimum cost decision tree is hard even for instances where the input graph is a spider and the length of every leg is three.

Our reduction is from the NP-complete Balanced Partition problem [8, A3.2], a special case of the Partition problem. The input of the Partition problem is given by a set of numbers, $\{a_i \mid i \in [m]\}$, and our goal is to find an index set $I$ such that $\sum_{i\in I} a_i = \sum_{i\notin I} a_i$. In the Balanced Partition problem it is further required that $|I| = m/2$, i.e., there are the same number of numbers in both parts of the partition. (This implies that for a non-trivial input $m$ has to be even.) Because of this, we can also suppose that all the numbers have roughly the same size as adding a contant to each will not affect the set of solutions. This implies that we can suppose that $a_i < 2a_j$ and $\sum_{i\in I} a_i < \sum_{i\notin I} a_i$ for any $|I| < m/2$.

From a set of numbers $\{a_i \mid i \in [m]\}$ with the above properties, we construct an instance $(S, c)$ for the tree search problem with non-uniform costs, where $S$ is a spider. Each leg will correspond to a number. Therefore, we will speak of the $i$th leg as the leg corresponding to the $i$th number. For each $i \in [m]$, the $i$th leg will consist of three edges: the one closest to the head will be called *femur* (and referred to as $f_i$), the middle edge will be called *tibia* (and referred to as $t_i$), the end will be called the *tarsus* (and referred to as $s_i$). The cost function is defined as follows: For each $i \in [m]$, we set $c(f_i) = 2a_i$; $c(t_i) = a_i$ and $c(s_i) = N$, with $N = \sum_{i\in[m]} a_i$.

It is easy to see that in an optimal strategy, for each $i \in [m]$ the tarsus is always queried last among the edges on the $i$th leg. Given a decision tree $D$, we denote by $F$ the set of indices of the legs for which, in $D$, the node associated with the query to the tibia is an ancestor[3] of the node associated with the query to the femur. Then, we have the following proposition, whose proof is omitted from this extended abstract.

**Proposition 3.** *There is an optimal decision tree $D$ with $F \neq \emptyset$ and such that:*

*(i) for any $i \in F$ and $j \in [m] \setminus F$ the node of $D$ associated with the $j$th femur is an ancestor of the node associated with the $i$th tibia.*

*(ii) for any $i, j \in F$ the node of $D$ associated with the $i$th tibia is an ancestor of the node associated with the $j$th femur.*

By this proposition, we can assume that in the optimal decision tree $D$ for at least one leg of the spider the first edge queried is a tibia. In addition, in $D$, there is a root to leaf path where first all femora not in $F$ are queried, then all tibiae in $F$, and finally all femora in $F$. Then, the cost of such a decision tree is given

[3] A node $\nu$ is an ancestor of another node $\nu'$ if $\nu$ lies on the path connecting $\nu'$ to the head.

by the maximum of the cost of the above mentioned path ($\sum_{i\notin I} 2a_i + \sum_{i\in I} 3a_i$) and the costs of the paths to the leaves on the legs, which either start with a femur with index not in $F$ (cost $\leq \sum_{i\notin I} 2a_i + \max_{i\notin I} a_i + N$) or with a tibia with index in $F$ (cost $\leq \sum_{i\notin I} 2a_i + \sum_{i\in I} a_i + N$). This last value is indeed attained for the leaf of the last leg starting with a tibia. Using the fact that for all $i, j$ we have $a_i < 2a_j$, this last value, $\sum_{i\notin I} 2a_i + \sum_{i\in I} a_i + N$, is always greater than the cost for leaves on legs starting with femora, $\sum_{i\notin I} 2a_i + \max_{i\notin I} a_i + N$ if $m \geq 4$. Therefore, the cost of the optimal solution is given by the following expression

$$OPT(S,c) = \min_{\emptyset\subset I\subseteq[m]} \max\left\{ \sum_{i\notin I} 2a_i + \sum_{i\in I} a_i + N; \sum_{i\notin I} 2a_i + \sum_{i\in I} 3a_i \right\}.$$

Using $N = \sum_{i\in[m]} a_i$, the two sums are equal if and only if $\sum_{i\notin I} a_i = \sum_{i\in I} a_i$. Therefore $OPT(S,c) \geq \frac{5}{2}\sum_{i\in[m]} a_i$ and equality holds if and only if $\sum_{i\notin I} a_i = \sum_{i\in I} a_i$, which is only possible if $|I| = m/2$ (recall that we could suppose $\sum_{i\in I} a_i < \sum_{i\notin I} a_i$ for any $|I| < m/2$). This is equivalent to having a Balanced Partition of the numbers, so we have finished the reduction.

**Acknowledgment.** We are very grateful to Balázs Patkós for organizing 5th Emléktábla Workshop where we collaborated on this paper.

## Appendix

### Figures

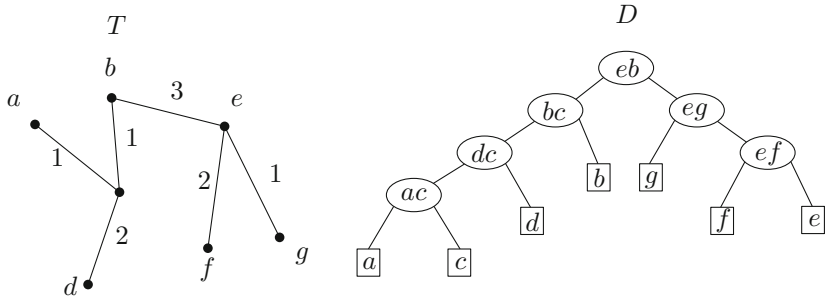


**Fig. 1.** An example of the tree search problem, $T$ is the input tree and $D$ is a decision tree with $cost(D) = 7 = cost^D(a) = cost^D(c)$. If the vertices of the tree $T$ represent the parts of a device to assemble, the decision tree corresponds to the assembly procedure that at time 0 joins $e$ with $b$; then at time 3 joins $b$ with $c$ and $e$ with $g$. At time 4 the joining of $d$ with $c$ and $e$ with $f$ is started. Finally, at time 6 part $a$ is joined with part $c$ and the procedure ends by time 7.

**Algorithm TS.** Tree Search Algorithm

**function** MAIN(tree $T$, cost $c$)
  $t \leftarrow 2^{\lfloor \log \log |T| \rfloor + 2}$
  **Output** $D \leftarrow$TREESEARCH$(T, c, t)$
**end function**
**function** TREESEARCH(tree $T$, costs $c, t$)
  **if** $|T| \leq t$ **then return** optimal decision tree $D_X$ for $T$ computed by Proposition 2
  $S_0 \leftarrow \emptyset$
  **for all** $i = 1, \ldots, t$ **do**
    $x_i \leftarrow$ centroid of a maximum size component of $T - S_{i-1}$
    $S_i \leftarrow$ smallest subtree containing $x_i$ and $S_{i-1}$
  **end for**
  $S \leftarrow S_t$
  $X \leftarrow \{x_i |\ i = 1, \ldots, t\} \cup \{v \in V(S) |\ \deg_S(v) \geq 3\}$
  $Y \leftarrow$ tree on vertex set $X$, $uv \in E(Y)$ iff $X \cap P_{u,v} = \{u, v\}$
  **for all** $uv \in E(Y)$ **do**
    $c_Y(uv) \leftarrow \min_{e \in P_{u,v}} c(e)$
    $e_{uv} \leftarrow$ edge of $P_{u,v}$ with minimum cost
  **end for**
  $Z \leftarrow \bigcup_{uv \in E(Y)} e_{uv}$
  Compute optimal decision tree $D_Y$ for $(Y, c_Y)$ by Proposition 2
  **for all** $uv \in E(Y)$ **do**
    Replace label of $uv$ in $D_Y$ by $e_{uv}$
  **end for**
  **for all** components $H$ of $Y - Z$ **do**
    ▷ $H$ contains at most one vertex of degree 3 or more, i.e., $H$ is a spider
    Compute 2-approximate decision tree $D_H$ for $H$ by Lemma 3
    replace the leaf $k \in D_Y$ corresponding to $H$ by the root of $D_H$
  **end for**
  **for all** $v \in V(S)$ with a neighbor not in $S$ **do**
    $S_v \leftarrow$ star induced by $v$ and its neighbors outside of $V(S)$
    Construct decision tree $D_v$ for $(S_v, c)$
    **for all** $w \in S_v \setminus \{v\}$ **do**
      $U \leftarrow$ component of $T - S$ containing $w$
      $D_w \leftarrow$ TREESEARCH$(U, c, t)$
      leaf of $D_v$ corresponding to $w \leftarrow$ root of $D_w$
    **end for**
    replace the leaf of $D_Y$ associated to $v$ by the root of $D_v$
  **end for**
  **return** $D_Y$
**end function**

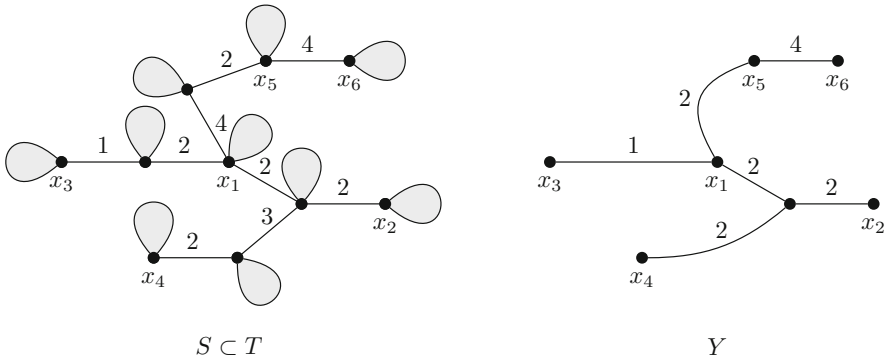


**Fig. 2.** An example of the tree $S$, the important set of vertices $X$ and the auxiliary tree $Y$ in the construction of Sect. 3

## References


1. Ahlswede, R., Wegener, I.: Search Problems. Wiley, Chichester-New York (1987)
2. Aigner, M.: Combinatorial Search. Wiley-Teubner, New York-Stuttgart (1988)
3. Ben-Asher, Y., Farchi, E., Newman, I.: Optimal search in trees. SIAM J. Comput. **28**(6), 2090–2102 (1999)
4. Cicalese, F., Jacobs, T., Laber, E., Valentim, C.: The binary identification problem for weighted trees. Theor. Comput. Sci. **459**, 100–112 (2012)
5. de la Torre, P., Greenlaw, R., Schäffer, A.: Optimal edge ranking of trees in polynomial time. Algorithmica **13**(6), 592–618 (1995)
6. Dereniowski, D.: Edge ranking of weighted trees. Discrete Appl. Math. **154**, 1198–1209 (2006)
7. Dereniowski, D.: Edge ranking and searching in partial orders. Discrete Appl. Math. **156**(13), 2493–2500 (2008)
8. Garey, M.R., Johnson, D.S.: Computer and Intractability. W.H. Freeman & Co., New York (1979)
9. Iyer, A.V., Ratliff, H.D., Vijayan, G.: On an edge ranking problem of trees and graphs. Discrete Appl. Math. **30**(1), 43–52 (1991)
10. Knuth, D.: Searching and Sorting. The Art of Computer Programming, vol. 3. Addison-Wesley, Reading (1998)
11. Lam, T.W., Yue, F.L.: Optimal edge ranking of trees in linear time. In: Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1998, pp. 436–445, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics (1998)
12. Linial, N., Saks, M.: Searching order structures. J. Algorithms **6**, 86–103 (1985)
13. Makino, K., Uno, Y., Ibaraki, T.: On minimum edge ranking spanning trees. J. Algorithms **38**, 411–437 (2001)
14. Mozes, S., Onak, K., Weimann, O.: Finding an optimal tree searching strategy in linear time. In: Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2008), pp. 1096–1105 (2008)
15. Wermelinger, M.: Searching Efficiently in Posets. New University of Lisbon, Topics in Programming Technology (1993)