A Note on Finding the Minimum With Non-Uniform Comparison Costs

Downloaded from: https://research.chalmers.se, 2026-09-05 04:47 UTC

Citation for the original published paper (version of record):
Damaschke, P. (2026). A Note on Finding the Minimum With Non-Uniform Comparison Costs. Matematica, 5(2). http://dx.doi.org/10.1007/s44007-026-00216-x

N.B. When citing this work, cite the original published paper.

research.chalmers.se offers the possibility of retrieving research publications produced at Chalmers University of Technology. It covers all kind of research output: articles, dissertations, conference papers, reports etc. since 2004. research.chalmers.se is administrated and maintained by Chalmers Library

(article starts on next page)

La Matematica (2026) 5:36
https://doi.org/10.1007/s44007-026-00216-x

ORIGINAL RESEARCH ARTICLE

A Note on Finding the Minimum With Non-Uniform Comparison Costs

Peter Damaschke[1]

Received: 29 July 2025 / Revised: 23 March 2026 / Accepted: 10 April 2026
© The Author(s) 2026

Abstract
Let us consider the following seemingly simple scenario: A finite set is given, which has a hidden total order of its elements. A player called the searcher can freely select any two elements and compare them. Furthermore, these comparisons have different positive costs that are known to the searcher in advance. The goal of the searcher is simply to find the minimum element at a minimum total cost. This combinatorial search game naturally appears as a subproblem in an approach to exact solutions of small instances of certain hard combinatorial optimization problems. We show in this note that a simple greedy strategy minimizes the worst-case total cost of the search. While this result as such might be quite expected, interestingly, the proof is not so obvious. We apply an exchange argument within a game-theoretic setting.

Keywords combinatorial search · minimum · two-person game · non-uniform costs · greedy strategy

Mathematics Subject Classification 68Q17 · 68W40 · 91A05 · 91A46

1 Introduction

1.1 The Problem

Consider the following scenario: We are given a set of n objects among which some unknown total order relation is defined. For every pair of objects, the fixed (and positive) cost of comparing the two objects is known. The problem is simply to find the minimum object in the total order, by doing pairwise comparisons with minimum total cost. More precisely, we wish to devise a comparison strategy that guarantees the smallest total comparison cost in the worst case, that is, maximized over all $n!$ possible

✉ Peter Damaschke
ptr@chalmers.se

[1] Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg, SE-41296 Gothenburg, Sweden

Published online: 06 May 2026

total orderings of the objects. While this question as such appears very natural, we came across the problem also in a practical setting; see the Appendix.

1.2 Related Work

A number of fundamental problems like searching, selection, sorting, as well as other combinatorial search problems, have been intensively studied in scenarios where comparisons or other operations are unreliable. Various models of unreliable tests have been considered, e.g.: random errors occur [13], a malicious "liar" can give wrong answers on purpose, where the number of lies is limited in some way [1, 14], or systematically recurring errors are present [3, 4]. On another front, many non-trivial results are known for the exact (i.e., not only the asymptotic) number of comparisons for selection (order statistics) and sorting, for various fixed numbers of elements [5, 10–12].

Our minimum-search problem is trivial if all comparisons have the same cost: $n-1$ comparisons are necessary and sufficient for finding the minimum of n objects. But we are not aware of an earlier treatment of this problem and related search problems (as mentioned above) with different individual comparison costs that are known to the searcher in advance. We call them *non-uniform costs*.

The search problem in [6, 7] differs from ours in that some *fixed* order of objects is known to the searcher, along with non-uniform costs of binary search steps telling whether an unknown object is to the left or to the right of the queried object. The goal is to find the said unknown object. Linear orders are naturally generalized to tree orders in [2]: A query to an edge e of the tree reveals in which of the two subtrees separated by e an unknown object is located.

1.3 Contribution

A natural greedy strategy for solving our minimum-search problem is the following. Initially, all objects are candidates for the minimum. We always pick two candidates whose comparison is cheapest, and then discard the larger of these two objects. This is repeated until only one candidate remains, which is then obviously the minimum.

One might expect that this also guarantees the worst-case minimum cost. It is tempting to argue that every comparison yields the same amount of information (ruling out one candidate), and hence one should always go for the cheapest option. But this information argument is not conclusive, since it completely ignores that the remaining instance and its cost structure depends on the comparisons chosen so far, and on their results. It is a typical beginners' fallacy to infer that greedy steps always yield a minimum total cost.

More specifically, it might happen that there exist very cheap comparisons between candidates and already discarded objects, and by transitivity they might allow some indirect comparisons between candidates that are cheaper than any immediate comparisons between candidates. Therefore, a cost optimality proof for the greedy algorithm must also show that comparisons involving discarded objects are not beneficial for the search cost in the worst case; see Lemma 4 and Theorem 5.

A brighter optimality proof attempt is to try and relate the problem to the optimality of Kruskal's minimum spanning tree algorithm. But on a second thought, the similarity turns out to be superficial and not helpful; see the example in Section 3.1.

Our result is that the above greedy algorithm indeed minimizes the total cost of comparisons in the worst case. While this may not be surprising as such, the optimality argument is not that obvious and is slightly technical.

2 Preliminaries

In a combinatorial search problem, a player Q (questioner) wants to figure out some information about a hidden instance I by using certain questions that are available and have certain costs. The total cost $c(S, I)$ of the search depends on the instance I and on the chosen strategy S. Player Q wants to use a strategy that minimizes the *worst-case cost* of the search, which is $\arg\min_S \max_I c(S, I)$. One can view every such problem as a game against another player A (answerer, or adversary) who can decide on I and has the goal to maximize the search cost. In the case of deterministic strategies, once A knows the chosen strategy S, A can decide on an instance I that maximizes $c(S, I)$. Hence it is optimal for Q to choose S which minimizes this maximum. The result is called the value of the game.

In our case, I is some of the $n!$ possible orders of n objects, the desired information is merely the smallest object in I, and the available questions are the pairwise comparisons.

From a practical angle it would be infeasible to enumerate all instances I and strategies S, simulate the executions of all S to all I, and then determine the value of the game in a huge matrix. Instead one can take advantage of the fact that the actions in S are single questions, and turn the game into one with alternating moves: Player Q does not reveal S but just follows the steps of the "secret" strategy S, and player A does not have to decide on I right from the beginning but just needs to answer the questions consistently, i.e., the answers given until every moment must be correct for at least one I. One can also represent the possible courses of the game as a decision tree, and the value of the game is achieved if both players apply the well-known minimax algorithm there. This equivalent view allows to use specific structural features of the search problem to determine its value.

Definition 1 Let $\mathcal{M}$ be the following game between two players Q (questioner) and A (answerer). Some set of n objects is given, with pairwise comparison costs that are known to both Q and A. Player Q wants to find the minimum object. Player Q can select, in every step, the pair of objects (u, v) to compare. We also say that Q asks a comparison question and pays the known cost for it. Player A answers either $u < v$ or $u > v$. But player A must answer consistently, that is, A's answers must always remain compatible with at least one possible total order of the objects.

The goal of player Q (A) is to minimize (maximize) the total cost of comparisons used. More precisely, player Q wants to guarantee the smallest upper bound on the cost that can be achieved under any possible consistent answers of player A, and player A wants to force player Q to really pay this cost, and not a smaller one. This minimum *worst-case cost* is called the value $c(\mathcal{M})$ of the game.

If both Q and A play optimally, then player Q will pay exactly the value $c(\mathcal{M})$.

The requirement of giving consistent answers can be characterized as follows. The answers given by player A until any moment naturally define a directed graph G on the set of objects, in that any answer $x < y$ creates a new directed edge from x to y. A topological order of a directed graph is a total order relation $<$ of its vertices where $x < y$ holds for every directed edge from x to y. Thus, the answers of player A are compatible exactly with those total orders that are topological orders of G. This also yields:

Proposition 2 *Player A answers consistently if and only if the directed graph of answers never contains directed cycles.*

Proof "only if": Directed cycles cannot appear in an order relation.

"if": It is well known that the absence of directed cycles in a directed graph implies the existence of a topological order. □

Finally, we introduce some terms that describe the status of each element. At every moment during the execution of a search, an object v is said to be *discarded* if it has been compared earlier to another object u, with the result $u < v$. The not yet discarded elements are called *candidates* (as each of them could eventually turn out to be the minimum element).

3 Analysis

We will state the greedy strategy and prove its optimality as follows. We use the interpretation of search problems as games. First, we define some auxiliary game where comparisons with discarded objects are made non-informative (Definition 3). Only the adversary gets restricted there, such that the greedy strategy is still applicable. Then, we show optimality of the greedy strategy in this auxiliary game by an exchange argument (Lemma 4). The reason for introducing the auxiliary game is that that it simplifies this exchange argument. Finally, we relate the the worst-case optimal search costs in the auxiliary game and in the original problem (Theorem 5).

3.1 Greedy Strategy for the Questioner

It is natural to consider the following simple strategy for player Q:

Greedy strategy:
| repeat
| pick some pair of candidates with lowest comparison cost
| compare them
| discard the larger candidate
| until one candidate remains
| return the candidate

Example As a warm-up and an illustration of the game $\mathcal{M}$ we describe the smallest example. Let x, y, z be three objects with the following comparison costs:

a between x and y,
b between x and z,
c between y and z,
where $a \leq b \leq c$.
The greedy algorithm would first compare x and y at cost a. If the answer is $y < x$, it would then compare y and z at cost c. Hence the total cost is $a + c$. Obviously, the answer $y < x$ is also the worst case for the greedy algorithm. Furthermore, no other algorithm can beat $a + c$ in the worst case: The searcher can always be forced to compare y and z, if not in the first step then in the second step. Hence the cost c plus the cost of another comparison can be enforced, and their sum is at least $a + c$.

Already the case of three objects also demonstrates that the problem is not related to a minimum spanning tree: Its cost in the edge-weighted clique would be $a+b$ which can be arbitrarily small compared to $a + c$.

3.2 Auxiliary Game

For analyzing the game $\mathcal{M}$, we define an auxiliary game $\mathcal{M}'$ with a less powerful adversary:

Definition 3 Let $\mathcal{M}'$ be the following game between two players Q (questioner) and A (answerer). Some set of n objects is given, with pairwise comparison costs that are known to both Q and A. Player Q wants to find the minimum object. Player Q can select, in every step, the pair of objects (u, v) to compare. We also say that Q asks a comparison question and pays the known cost for it. Player A answers either $u < v$ or $u > v$. But player A must answer in a restricted way: Whenever player Q compares some candidate u and some discarded object v, then player A answers $u < v$. Whenever player Q compares two discarded objects, then player A answers arbitrarily but still makes sure that the answers are consistent.

The goal of player Q (A) is the same as in $\mathcal{M}$, and the value $c(\mathcal{M}')$ of the game is defined accordingly.

Note that, also here, it is always possible for A to answer consistently (remember the characterization in Proposition 2): Player A may decide to make all discarded objects larger than all candidates, and fix some hidden total order relation among the discarded objects, thus avoiding the creation of directed cycles involving them. Among the candidates, A's restrictions are the same as in $\mathcal{M}$.

More formally, this defines two acyclic directed graphs G_c and G_d representing the comparison results between the candidates and between the discarded objects, respectively, which are then connected by directed edges between the pairs of candidates and discarded objects that are compared. The combined graph G is still acyclic, since the concatenation of any topological orders of G_c and G_d is a topological order of G.

3.3 Worst-Case Optimality

First we show optimality of the greedy strategy against the restricted adversary. This is the more technical part.

Lemma 4 *In the auxiliary game $\mathcal{M}'$, player Q can guarantee a total comparison cost of at most $c(\mathcal{M}')$ by applying the greedy strategy. In other words, the greedy strategy is worst-case optimal in $\mathcal{M}'$.*

Proof We proceed by induction on the number n of objects. The induction base with $n = 2$ is trivial, since there is only one comparison to do. As our induction hypothesis, suppose that the greedy strategy is optimal for n objects. Below we will prove this claim:
(*) An optimal strategy for player Q, for $n + 1$ objects, can always start with some cheapest comparison between candidates.
If this is true, we can do the inductive step as follows. After this first comparison, n candidates remain. By the induction hypothesis, it is optimal to continue with the greedy strategy. But also the first comparison has followed the greedy strategy, hence, overall this means to apply the greedy strategy to the given $n + 1$ objects.

Now we prove assertion (*). For an arbitrary object u we can state:
(**) Every execution of any strategy for $\mathcal{M}'$ requires u to be involved in at least one comparison with a candidate.

Claim (**) holds since u might be the minimum and cannot be recognized as such, if it never participates in any comparison. Moreover, comparisons with already discarded objects are not informative in $\mathcal{M}'$.

For proving (*) we will transform some assumed optimal strategy S of player Q into a strategy that begins with the cheapest comparison, without increasing the total cost.

Let Q play the game $\mathcal{M}'$ as follows. First, player Q makes some cheapest comparison. Say, Q compares some objects u and v, and the answer is $u < v$. After this first step, Q continues by applying strategy S, pretending that this first comparison was not made. In particular, Q still considers v a candidate and ignores the knowledge that $u < v$.

Due to statement (**), in some step the strategy S requests a comparison between the candidate v and some other candidate w. But "in reality", v is already discarded (due to $u < v$). The rules of $\mathcal{M}'$ enforce that player A must answer $w < v$ (since w is a candidate while v is not). At this moment, we interrupt the application of strategy S by player Q. Instead, we let player Q now discard v and skip the requested comparison of w and v. (Remember that Q actually knows already that v is not the minimum, due to $u < v$ and without having to learn that $w < v$.)

Now, player Q has obtained exactly the same candidate set that would have been produced by applying the strategy S right from the beginning. After this moment, Q continues following the strategy S again. The total cost of comparisons made by player Q is no higher than in S, since Q has made one additional comparison first, but also saved one comparison later, and the new first comparison was a cheapest one.

It follows that the modified strategy of player Q for $\mathcal{M}'$, starting with some cheapest comparison between candidates, is still optimal, which establishes (*). □

Relating the two games to each other yields the final result:

Theorem 5 *We have $c(\mathcal{M}) = c(\mathcal{M}')$. Furthermore, the greedy strategy is worst-case optimal for the game $\mathcal{M}$, and it runs in $O(n^2 \log n)$ time for n objects.*

Proof Every possible strategy of player Q for $\mathcal{M}$ is, trivially, also applicable to $\mathcal{M}'$, since by Definition 3, player Q has the same actions available (i.e., Q can compare arbitrary objects), and only the adversary A has restrictions in $\mathcal{M}'$, compared to $\mathcal{M}$.

In particular, when playing $\mathcal{M}'$, player Q may even apply an optimal strategy for $\mathcal{M}$. Hence Q can guarantee a total cost of at most $c(\mathcal{M})$ in the game $\mathcal{M}'$. This shows $c(\mathcal{M}) \geq c(\mathcal{M}')$.

By Lemma 4, the greedy strategy is worst-case optimal in $\mathcal{M}'$, in other words, it guarantees a total comparison cost of at most $c(\mathcal{M}')$ in $\mathcal{M}'$.

On the other hand, the greedy strategy compares only candidates, that is, the restrictions applying to player A have no effect. Thus, A playing optimally gives the same answers in $\mathcal{M}$ as in $\mathcal{M}'$. Hence the greedy strategy guarantees the same total comparison cost of at most $c(\mathcal{M}')$ also when $\mathcal{M}$ is played. This shows the opposite inequality $c(\mathcal{M}) \leq c(\mathcal{M}')$, and consequently $c(\mathcal{M}) = c(\mathcal{M}')$. Finally, the greedy strategy is optimal also in $\mathcal{M}$, as it guarantees this cost.

For the time bound, think of the input as a clique of n nodes that represent the objects, and edge costs that represent the comparison costs. We merely have to sort the $O(n^2)$ comparison costs in $O(n^2 \log n)$ time, pick $n-1$ times the minimum-cost edge, and delete all edges incident to the discarded node. These $O(n^2)$ dictionary operations can be done in $O(n^2 \log n)$ time as well. □

We remark that the time for sorting and dictionary operations might be improved to $O(n^2)$ if the comparison costs are integers in an $O(n)$ range.

4 Conclusions

We have shown that a greedy algorithm for searching for the minimum in a set with non-uniform comparison costs between the elements guarantees optimal total costs in the worst-case sense. One natural follow-up question is whether the searcher can achieve expected search costs that beat these optimal worst-case costs, by some randomized comparison strategy against an oblivious adversary. That is, the adversary may know the randomized algorithm beforehand, but not the random decisions that will actually be made. Furthermore, one may generalize the game to a "hypergraph" setting where, for some fixed $k \geq 2$, the minimum of any k objects can be determined at some given cost. Finally, similar problems can be raised also for selection and sorting with different comparison costs.

Appendix: Motivation from a Discrete Optimization Problem

A central aspect of factory layout planning is to place machines in a factory hall in such a way that related machines are close to each other. The input is described as an edge-weighted graph, where the vertices represent machines, and the weight of an edge models the strength of the relationship of the two machines. For example, the weight can simply be the amount of material to be moved per time unit from one machine to the other one. The goal is to place the machines on grid points so as to minimize the weighted sum of all pairwise Manhattan distances [8, 9]. Limiting the positions to the

grid points of a square grid ensures that there remain free aisles between the machines in two orthogonal directions. But still, the optimal solution on the grid can be further modified when other constraints come in, like the sizes and shapes of machines.

In a long-term investigation we have been studying this NP-hard optimization problem from a practical point of view, for small numbers of machines, which is typical for real use cases. (NP-hardness follows, e.g., from results in [15].) We found that, up to symmetries, surprisingly few non-congruent sets of grid points exist that can host optimal solutions, and many permutations of the vertices therein can be quickly discarded as non-optimal. One simple way to find the optimum among the remaining options is then to perform successive comparisons between selected pairs of solutions and always discard the one with the larger weighted sum of distances (or either one, if the sums are equal). However, the number of operations for a comparison of two expressions depends on their similarity, as it would be pointless to evaluate terms that appear in both expressions. The question arises in which ordering we should compare the expressions in this approach, in order to minimize the total number of arithmetic operations in the worst case, that is, maximized over all weighted graphs with the same number of vertices. Abstraction of this setting leads to the problem introduced in this paper. Note that, for the number of operations, only the pairwise comparison costs and the unknown total ordering of the values are relevant, but not the values themselves.

The following toy example may illustrate the origin of our search problem more concretely. For four machines there exist, up to symmetries, only two types of optimal grid layouts (we skip the easy proof here). They are displayed below, where the circles indicate the positions of the machines on the grid points.

Let w_{ij} denote the weight of the edge ij (where $i, j \in \{1, 2, 3, 4\}$ and $i \neq j$). Since the weighted sum of the Manhattan distances shall be minimized, some of the following seven essentially different grid layouts is optimal. The numbers are simply the indices of the four machines.

13	12	12	3	3	2	2
42	43	34	214	124	134	143

In these layouts, the weighted sum of Manhattan distances is $w_{12} + w_{13} + w_{14} + w_{23} + w_{24} + w_{34}$ plus one of these sums:

$w_{12} + w_{34}$, $w_{13} + w_{24}$, $w_{14} + w_{23}$, called "2-sums", or $w_{23} + w_{24} + w_{34}$, $w_{13} + w_{14} + w_{34}$, $w_{12} + w_{14} + w_{24}$, $w_{12} + w_{13} + w_{23}$, called "3-sums".

In the following, we consider the number of arithmetic operations (specifically, additions and comparisons of real numbers) needed to find the minimum. First computing all the 7 sums and then finding their minimum by 6 comparisons needs $3 \cdot 1 + 4 \cdot 2 + 6 = 17$ operations. It is smarter to use the fact that many summands are common. Comparing any two 2-sums costs 3 operations (2 additions and 1 comparison), and the same holds for any two 3-sums, since they always have one common summand. Comparing any 2-sum with any 3-sum costs only 2 operations (1 addition and 1 comparison), also due to a common summand. The question is in which order we should compare the sums, and this is where our search problem arises. The greedy strategy tells us to compare a 2-sum to a 3-sum and to discard the larger sum in every

step, as long as possible. In the worst case, it is always the 2-sum that gets discarded. Then, we do 3 steps with 2 operations, and the remaining 3 steps need 3 operations each, which are $3 \cdot 2 + 3 \cdot 3 = 15$ operations in total.

By further preprocessing tricks not shown here (comparing certain single weights, re-using precomputed sums, etc.), this can be further reduced to 13 operations. However, for more machines and candidate layouts it becomes very subtle to design cheaper search strategies, or even to minimize the number of operations. The demonstrated greedy elimination of sums provides a simple general strategy that already saves enough operations in order to make somewhat larger instances of the layout problem feasible.

Acknowledgements The author would like to thank Fredrik Ekstedt and Raad Salman from the Fraunhofer-Chalmers Research Centre for Industrial Mathematics, Gothenburg, for extensive discussions of the practical problem that inspired this note, and anonymous reviewers for their constructive comments.

Author Contributions Peter Damaschke: Conceptualization, Methodology, Formal Analysis, Writing

Funding Open access funding provided by Chalmers University of Technology. No funding was received to assist with the preparation of this manuscript.

Data availability No data have been used in this work.

Declarations

Competing Interests The author has no competing interests to declare that are relevant to the content of this article.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Borgstrom, R.S., Rao Kosaraju, S.: Comparison-based Search in the Presence of Errors. In: Rao Kosaraju, S., Johnson, D.S., Aggarwal, A. (Eds.), 25th ACM Symp. Theory of Comp. STOC 1993, 130–136 (1993). https://doi.org/10.1145/167088.167129
2. Cicalese, F., Keszegh, B., Lidický, B., Pálvölgyi, D., Valla, T.: On the Tree Search Problem With Non-uniform Costs. Theor. Comp. Sci. **647**, 22–32 (2016)
3. Geissmann, B., Mihalák, M., Widmayer, P.: Recurring Comparison Faults: Sorting and Finding the Minimum. In: Kosowski, A., Walukiewicz, I.: 20th Int. Symp. on Fundamentals of Comp. Theory FCT 2015. LNCS 9210, 227–239 (2015). https://doi.org/10.1007/978-3-319-22177-9_18
4. Geissmann, B., Penna, P.: Inversions from Sorting with Distance-based Errors. In: Min Tjoja, A. et al. (Eds.) SOFSEM 2018. LNCS 10706, 508–522 (2018). https://doi.org/10.1007/978-3-319-73117-9_36
5. Knuth, D.: The Art of Computer Programming, vol. 3. Addison-Wesley, Sorting and Searching (1968)
6. Laber, E.S., Milidiú, R.L., Pessoa, A.A.: On Binary Searching with Nonuniform costs. SIAM J. Comp. **31**, 1022–1047 (2002a)

7. Laber, E.S., Milidiú, R.L., Pessoa, A.A.: A Strategy for Searching with Different Access Costs. Theor. Comp. Sci. **287**, 571–584 (2002b)
8. Mårdberg, P., Fredby, J., Engström, K., Li, Y., Bohlin, R., Berglund, J., Carlson, J.S., Vallhagen, J.: A Novel Tool for Optimization and Verification of Layout and Human Logistics in Digital Factories. Procedia CIRP **72**, 545–550 (2018). https://doi.org/10.1016/j.procir.2018.03.158
9. Muther, R., Wheeler, J.D.: Simplified Systematic Layout Planning. Management and Industr. Res. Publ., Univ. of Michigan (1994)
10. Peczarski, M.: New Results in Minimum-Comparison Sorting. Algorithmica **40**, 133–145 (2004). https://doi.org/10.1007/s00453-004-1100-7
11. Peczarski, M.: The Ford-Johnson Algorithm Still Unbeaten for Less Than 47 Elements. Info. Proc. Letters **101**, 126–128 (2007). https://doi.org/10.1016/j.ipl.2006.09.001
12. Peczarski, M.: Towards Optimal Sorting of 16 Elements. CoRR arXiv:abs/1108.0866 (2011)
13. Pelc, A.: Searching with Known Error Probability. Theor. Comp. Sci. **63**, 185–202 (1989). https://doi.org/10.1016/0304-3975(89)90077-7
14. Pelc, A.: Searching Games with Errors - Fifty Years of Coping with Liars. Theor. Comp. Sci. **270**, 71–109 (2002). https://doi.org/10.1016/S0304-3975(01)00303-6
15. de Sá, V.G.P., da Fonseca, G.D., Machado, R., de Figueiredo, C.M.H.: Complexity Dichotomy on Partial Grid Recognition. Theor. Comput. Sci. **412**, 2370–2379 (2011). https://doi.org/10.1016/j.tcs.2011.01.018

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.