Applications of Algorithmic Metatheorems to Space Complexity and Parallelism

Till Tantau

Institute for Theoretical Computer Science, Universität zu Lübeck, Germany
tantau@tcs.uni-luebeck.de

Abstract

Algorithmic metatheorems state that if a problem can be described in a certain *logic* and the inputs are *structured* in a certain way, then the problem can be solved with a certain *amount of resources*. As an example, by Courcelle's Theorem all monadic second-order ("in a certain logic") properties of graphs of bounded tree width ("structured in a certain way") can be solved in linear time ("with a certain amount of resources"). Such theorems have become a valuable tool in algorithmics: If a problem happens to have the right structure and can be described in the right logic, they immediately yield a (typically tight) upper bound on the time complexity of the problem. Perhaps even more importantly, several complex algorithms rely on algorithmic metatheorems internally to solve subproblems, which considerably broadens the range of applications of these theorems. The talk is intended as a gentle introduction to the ideas behind algorithmic metatheorems, especially behind some recent results concerning space classes and parallel computation, and tries to give a flavor of the range of their applications.

1998 ACM Subject Classification F.4.1 Mathematical Logic, G.2.2 Graph Theory

Keywords and phrases Algorithmic metatheorems, Courcelle's Theorem, tree width, monadic second-order logic, logarithmic space, parallel computations

Digital Object Identifier 10.4230/LIPIcs.STACS.2017.4

Category Invited Talk

1 Talk Summary[1]

Alice, a first-year student of computer science, has an evil homework assignment: Devise an efficient algorithm for the vertex cover problem (at least I feel that tasking first-year students with solving NP-complete problems is a trifle unfair). I guess *you* are right now mentally weighing the different tools at your disposal from the vast machinery developed in complexity theory for attacking such problems – but what would a first-year student do? Being smart, Alice tries to apply the arguably most important and ubiquitous algorithmic approach in computer science: divide-and-conquer. After all, the approach lies at the heart of the fundamental algorithms she just learned about, including merge sort, quick sort, and binary search.

Solving Vertex Cover Using Divide-And-Conquer? Naturally, Alice soon notices that the divide-and-conquer approach fails quite miserably when applied to finding small vertex covers.

[1] This summary contains some revised material from the introduction of the paper [9]. That paper is a good starting point for a more detailed introduction to algorithmic metatheorems for logarithmic space and circuit classes.

The problem lies in the *dividing phase:* She finds no way of dividing, for instance, a clique into parts. Alice learns an important lesson here: divide-and-conquer is only applicable to problems whose inputs are "amenable" to dividing them into parts. After telling her professor about her difficulties, she relents and makes the problem (much) easier by restricting the input to *trees.* Now, clearly, dividing the input is no longer a problem: For a tree T with root r we can recurse on the subtrees T_1 to T_m rooted at the children c_1 to c_m of the root.

Alice still has to tackle the *merging phase* in her divide-and-conquer algorithm: How does one assemble optimal vertex covers for the T_i into a vertex cover for the whole tree T? Clearly, this is not a trivial task since optimal vertex covers for each subtree do not suffice to build an optimal overall vertex cover. The trick is to compute *two* optimal vertex covers for each subtree: one for the case that the root is part of the vertex cover and one for the case that it is not. This yields a divide-and-conquer algorithm for the vertex cover problem on trees that runs in linear time.

The Question of Why. Algorithmic metatheorems, which this talk is about, help us in understanding *why* the vertex cover problem behaves the way it does with respect to the divide-and-conquer approach. Why does the division phase fail? Why does the merging phase work? The first question has a fairly easy answer: arbitrary graphs do not have any "decomposition property" at all. On the other hand, trees certainly *can* be decomposed very well, and it turns out that this is still the case when the graph is "nearly" a tree, namely a graph of bounded tree width. The second question seems harder: The answer "because solutions can be assembled using a trick" does not generalize very well. It took the research community quite some time to find a better answer: In 1990, Bruno Courcelle found that the merging phase works "because the vertex cover problem can be described in monadic second-order logic."

The general pattern underlying algorithmic metatheorems is as follows: If a problem can be described in a certain logic ("are amenable to merging" for the right logic) and instances can be decomposed in a certain tree-wise fashion ("are amenable to division"), they can be solved within a certain amount of time. The first and most famous of these theorems is the just-mentioned Theorem of Courcelle [1]: *All monadic second-order properties of graphs of bounded tree width can be decided in linear time.* A long line of further theorems have later been obtained by varying the three "parameters" of algorithmic metatheorems: the logic, the instance structure, and the required resources. By weakening one of them, one can often strengthening another. For instance, for problems describable in *first*-order logic, we can change the requirement on the decomposition property to, for instance, nowhere dense graphs (a much larger class of graphs than those of bounded tree width) and still obtain a (near) linear time bound [7] or to planar graphs and still obtain a linear time bound [4]. In another direction, when we increase the time bound to polynomial (rather than linear) time, we can broaden the class of graphs to graphs of bounded clique-width [2]. In yet another direction, which this talk will mainly be about, instead of the classical sequential worst-case time bounds, one can look at the space complexity or the parallel complexity. One important result in that direction [3] is that Courcelle's Theorem also holds when "linear time" is replaced by "logarithmic space."

The Range of Applications of Algorithmic Metatheorems. The power of algorithmic metatheorems lies in their ease of application. Had Alice known about Courcelle's Theorem, she could have finished her homework much more quickly: The vertex cover problem can be described in monadic second-order logic and trees are clearly very "tree-like," so the theorem

tells her (and us) that there is a linear-time algorithm for the problem. Admittedly, devising a linear time algorithm for the vertex cover problem on trees is not all that hard – but by the logspace version of Courcelle's Theorem, we also get a logspace algorithm for this problem for free. You are cordially invited to try to come up with such an algorithm directly (or, failing that, make it a homework assignment for your students).

To make the vertex cover problem accessible to algorithmic metatheorems, we had to insist that all input graphs are trees (a ridiculously strong restriction) or, at least, that they are tree"-like" (no longer a ridiculous restriction, but still a strong restriction). It is thus somewhat surprising that algorithmic metatheorems *can* be used in contexts where the inputs are *not* tree-like graphs. The underlying algorithmic approach is quite ingenious: On input of a graph, if the graph is tree-like, apply an algorithmic metatheorem; and if the graph is not tree-like, it must be "highly cyclic internally," which we may be able to use to our advantage in solving the problem.

One deceptively simple problem where the just-mentioned approach works particularly well is the *even cycle problem,* which just asks whether there is a cycle of even length in a graph. It is not difficult to show that, like the vertex cover problem and just about any other interesting problem, the even cycle problem can be described in monadic second-order logic. Thus, it can be solved efficiently on tree-like graphs. Now, what about those "highly cyclic" graphs that are *not* tree-like? Intuitively, these many internal cycles might very well make it easier to decide whether the graph has an even cycle. Indeed, they make it *very* easy: such graphs *always* have an even cycle [10]. In other words, we can solve the even cycle problem on *arbitrary* graphs as follows: If the input graph is not tree-like, simply answer "yes," otherwise apply Courcelle's Theorem to it.

We will not always be so lucky that the to-be-solved problem more or less disappears for non-tree-like graphs, but in the talk several interesting problems are presented where algorithmic metatheorem play a key role in the internals of algorithms.

Related Work. As the title suggests, this talk focuses on algorithmic metatheorems for *space classes* and *parallel* computation (mainly in the form of circuits of polylogarithmic depth). In contrast, most algorithmic metatheorems in the literature concern *time classes.* There are a number of excellent surveys on algorithmic metatheorems and their applications regarding time-efficient computations [5, 6, 8]. An interesting aspect of the theorems covered in the talk is that they can be used to establish *completeness results* for many problems for which the classical algorithmic metatheorems do not yield an exact complexity-theoretic classification: Using Courcelle's Theorem and the tricks hinted at earlier, the even cycle problem can be solved in linear time and, clearly, this is also a tight lower bound. However, from a structural complexity-theoretic point of view, the problem is most likely not *complete* for linear time; indeed, it *is* complete for logarithmic space and the algorithmic metatheorems presented in the talk lie at the heart of the proof.

References

1 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

2 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

3 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010)*, pages 143–152, 2010. doi:10.1109/FOCS.2010.21.

4 Markus Frick and Martin Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, November 2001. doi:10.1145/504794.504798.

5 Martin Grohe. Logic, graphs, and algorithms. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives*, volume 2 of *Texts in Logic and Games*, pages 357–422. Amsterdam University Press, 2007. URL: http://www2.informatik.hu-berlin.de/~grohe/pub/meta-survey.pdf.

6 Martin Grohe and Stephan Kreutzer. Methods for algorithmic meta theorems. In *Model Theoretic Methods in Finite Combinatorics*, volume 558 of *Contemporary Mathematics*, pages 181–206. American Mathematical Society, 2011. URL: http://www2.informatik.hu-berlin.de/~grohe/pub/grokre11.pdf.

7 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC'14, pages 89–98, New York, NY, USA, 2014. ACM. doi:10.1145/2591796.2591851.

8 Stephan Kreutzer. Algorithmic meta-theorems. In Javier Esparza, Christian Michaux, and Charles Steinhorn, editors, *Finite and Algorithmic Model Theory*, volume 379 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 2011. URL: http://logic.las.tu-berlin.de/Kreutzer/Publications/amt-survey.pdf.

9 Till Tantau. A gentle introduction to applications of algorithmic metatheorems for space and circuit classes. *Algorithms*, 9(3):44, 2016. URL: http://www.mdpi.com/1999-4893/9/3/44, doi:10.3390/a9030044.

10 Carsten Thomassen. On the presence of disjoint subgraphs of a specified type. *Journal of Graph Theory*, 12(1):101–111, 1988. doi:10.1002/jgt.3190120111.