Practical Algorithms for MSO Model-Checking on Tree-Decomposable Graphs[☆]

Alexander Langer, Felix Reidl, Peter Rossmanith, Somnath Sikdar

Theoretical Computer Science,
Department of Computer Science,
RWTH Aachen University,
52074 Aachen, Germany

Abstract

In this survey, we review practical algorithms for graph-theoretic problems that are expressible in monadic second-order logic. Monadic second-order (MSO) logic allows quantifications over unary relations (sets) and can be used to express a host of useful graph properties such as *connectivity*, *c-colorability* (for a fixed c), *Hamiltonicity* and *minor inclusion*. A celebrated theorem in this area by Courcelle states that any graph problem expressible in MSO can be solved in linear time on graphs that admit a *tree-decomposition* of constant width. Courcelle's Theorem has been used thus far as a theoretic tool to establish that linear-time algorithms exist for graph problems by demonstrating that the problem in question is expressible by an MSO formula. A straightforward implementation of the algorithm in the proof of Courcelle's Theorem is useless as it runs into space-explosion problems even for small values of treewidth. Of late, there have been several attempts to circumvent these problems and we review some of these in this survey. This survey also introduces the reader to the notions of tree-decompositions and the basics of monadic second order logic.

Keywords: Monadic Second-Order Logic, Tree decompositions, Courcelle's Theorem

[☆]Parts of this survey appeared in the PhD thesis of Alexander Langer [124]. The authors were supported by the DFG grant RO 927/8.

Email addresses: langer@cs.rwth-aachen.de (Alexander Langer), reidl@cs.rwth-aachen.de (Felix Reidl), rossmani@cs.rwth-aachen.de (Peter Rossmanith), sikdar@cs.rwth-aachen.de (Somnath Sikdar)

Contents

1 Introduction **3**
1.1 Notation and Problems 9

2 Treewidth and tree decompositions **10**
2.1 Computing Tree Decompositions 13

3 Logic and Graphs **14**
3.1 Propositional Logic 15
3.2 First-order Logic 16
3.2.1 FO Model-Checking 17
3.3 MSO Logic 19
3.3.1 MSO-definable Properties and Graph Problems 22
3.3.2 Extended MSO 24
3.3.3 MSO and Semiring Homomorphisms 32
3.3.4 MSO Model-Checking 37

4 Courcelle's Theorem for Treewidth **39**
4.1 Proving Courcelle's Theorem 41
4.1.1 Alternative Proofs of Courcelle's Theorem 42
4.2 On hidden Constants 43

5 Courcelle's Theorem in Practice **44**
5.1 Not implemented or no results known 44
5.2 Negative results and problems. 45
5.3 Precomputed and Nondeterministic Automata 48
5.4 On-the-fly Construction of Automata 50
5.5 Reduction to Monadic Datalog 51
5.6 A Game-theoretic Approach 55

6 Beyond Treewidth **56**
6.1 Width Measures for Dense Graphs 57
6.1.1 Clique-width 57
6.1.2 Rank-width 60
6.1.3 Boolean width 61
6.2 Variants of Treewidth 62
6.3 Avoiding the non-elementary blow-up 62
6.4 Lower Bounds 63

7 Conclusions **63**
7.1 Further Reading 63
7.2 Acknowledgements 64

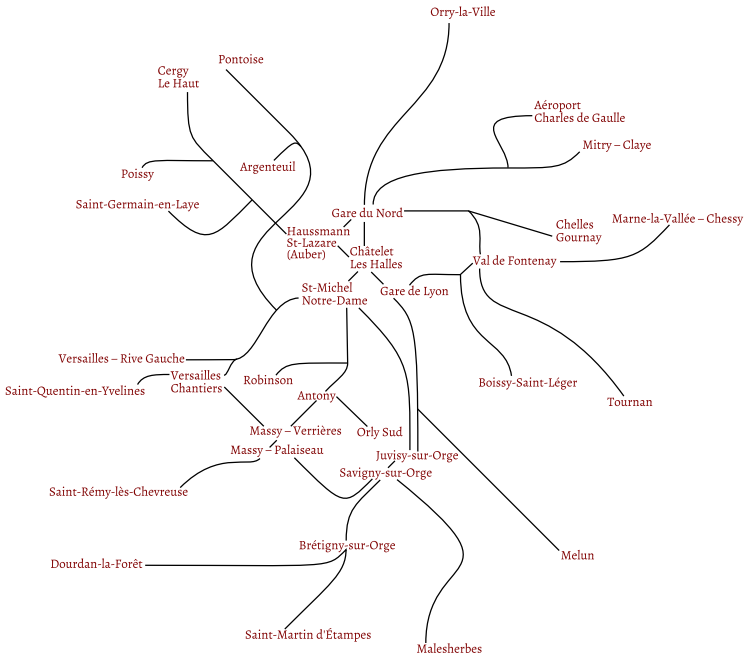


Figure 1: A slightly simplified geographical view of the Paris railway system.

1. Introduction

Imagine yourself as a consultant of the Paris railway network. Your task is to improve upon the existing infrastructure, which currently consists of 246 stations and a little less than 600 km of rails, by adding new stations to accommodate a shift in population density. The new stations have to be added to the network with the objective of maximizing accessibility of the network by potential customers. Figure 1 depicts the most important railway stations of this network in an approximate geographical representation.

Worryingly, the task at hand is the well-known NP-complete Station Location problem [182, 117, 134] and a brute-force approach is clearly infeasible. Worse, the real-world cost associated with each new stations makes non-optimal solutions in the best case costly and in the worst case unaffordable.

Let us step back and see what factors besides efficiency influence the choice of an appropriate algorithm in such a situation:

- **Optimality**: Besides the high cost associated with non-optimal solutions, an additional motivation for obtaining an optimal solution is that they provide an estimate of the bare minimum cost required.
- **Resources**: An estimate of the resources available plays a role in determining which algorithms can be used.
- **Generality**: An algorithm is more useful if it can be used for a reasonable number of real-world optimization problems.

- **Usability**: Lastly, the algorithm should be usable by a layman with a minimum amount of training or supervision.

Mixed integer linear program solvers are often a very good match. In this survey, we focus on another technique, that of MSO model-checking, which under well-defined circumstances provides an alternative angle of attack. Before continuing, let us clarify what we mean by this term.

Model checking is more commonly used in the field of *system verification*. Here logic model checking is used to verify the functional properties of complex systems such as hardware and software systems. The focus lies on the underlying complex system that has to be verified. It is sometimes considered an art to find a *model* that sufficiently captures the underlying system at the right level of abstraction. Logic is then used to express the desired properties of that model such as deadlock freeness or timing behavior. See, e.g., [8] for an introductory textbook.

In *algorithms* and *computational complexity* the term is used somewhat differently: the focus is usually on testing some fixed *property*, the goal being to design efficient algorithms that quickly check whether an *arbitrary* input satisfies the property. In this context, the Model Checking problem is to *check* whether a given input is a *model* for a—usually fixed—formula that defines the property of interest. With this understanding, model checking in particular encompasses all optimization problems that can be expressed in a certain logic system as a property of the input.

Let us briefly return to the Paris railway system. If we disentangle the network, the picture depicted in Figure 2 emerges. Not only is the underlying graph of the system planar, it contains only a few cycles—which, given the nature of the network, is not surprising. Many local railway networks have a generic star-like structure connecting a central station with nearby suburban stations. This structure can be exploited as outlined in the survey [182] by Wagner: one decomposes the original Station Location problem into sub-problems that are modeled by only a few *line segments* and are efficiently solvable [83, 134]. The solutions to these sub-problems are later combined, possibly by relaxing some of the optimization constraints.

An important feature of the network in question is that it *almost* looks like a tree. The fact that many NP-hard problems become easy on trees was noticed in the 1970s and was later quantified by Johnson [110] in a survey article where he notes that:

> *... (the class of trees) are by far the champion at rendering NP-hard problems solvable in polynomial time ...* [110].

This statement is corroborated by the fact that this survey article considered the complexity of several important graph problems including Independent Set, Clique, Chromatic Number, Hamiltonian Cycle, Dominating Set, MaxCut, Steiner Tree, and Graph Isomorphism on a number of graph classes. These include trees and forests, series-parallel graphs, partial k-trees, outerplanar graphs, grid graphs, planar graphs, genus-k graphs, Halin graphs, perfect graphs, chordal graphs, bipartite graphs, cographs, interval graphs, and claw-free graphs.

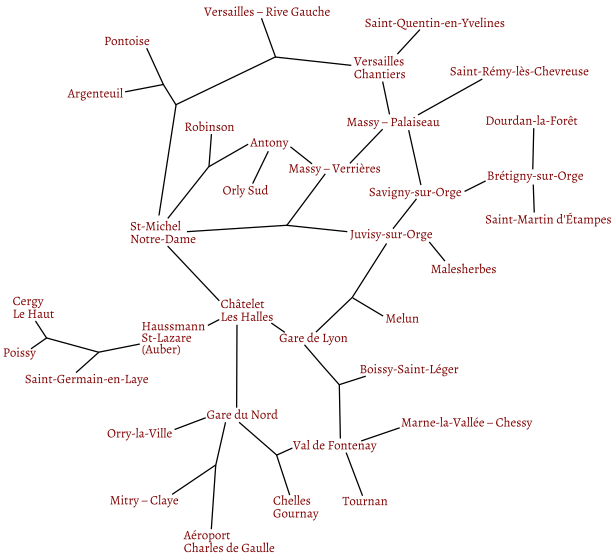


Figure 2: A topological view of the Paris railway system.

Johnson also notes that on trees many problems can be solved using a generic algorithm design technique, namely, by using dynamic programming [83]. But does this scale up to more general graphs? It turns out that many of these problems can indeed be solved in polynomial time when restricted to graphs that exhibit exactly those properties of trees which makes them amenable to dynamic programming, such as series-parallel graphs, outerplanar graphs, partial k-trees, Halin graphs, or interval graphs. See [110] and the references therein for further details.

Given this state of affairs, a natural question is how one can define the property of being "tree-like" in a mathematical context. This leads us to the notion of treewidth originally introduced by Halin [96] and later rediscovered by Robertson and Seymour [153] as part of their Graph Minors Project, cf. [55, 70, 53]. The treewidth of a graph is an integer that essentially measures how far removed the graph is from a tree. Trees and forests have constant treewidth one; $n \times n$-grids with n^2 vertices have treewidth n; and complete graphs on n vertices have treewidth $n-1$, which is the maximum possible width for an n-vertex graph. The graph classes on the high end of the spectrum are more interesting for theoretical purposes, but indeed many important and rich graph classes fall into the low end, i.e., their treewidth is bounded by a constant. For instance, series-parallel and outerplanar graphs have treewidth at most two; Halin graphs have treewidth at most five; and partial k-trees are precisely those graphs of treewidth k [159, 185, 180]. Of course, to judge the applicability of this restriction, it is even more interesting to know that many instances that arise in practice have low treewidth. For example, the control-flow graphs of `goto`-free C programs have treewidth at most six [174], and one can show that many train and road networks have low treewidth (the Paris railway from Figure 2 has treewidth 3). The surveys by Bodlaender [18, 21] and by Bodlaender and Koster [15] list many more real-world examples of graphs with small treewidth.

A central property of graphs of treewidth k is that they can be recursively decomposed into smaller subgraphs of treewidth k. Such graphs are called *tree-decomposable* or, simply, *decomposable*. The corresponding *tree decomposition* can be viewed as a *parse tree* that explains how a treewidth k graph can be recursively generated from smaller treewidth k graphs. This property makes it possible to run dynamic programming algorithms that use the parse tree as a guide.

By the mid-80s it was known that several problems including Vertex Cover, Independent Set, Dominating Set, and k-Colorability for every fixed k, admit linear-time algorithms on graphs of bounded treewidth. These results are summarized in the surveys by Arnborg [4] and the one by Johnson [110]. In a 1994 survey, Hedetniemi already lists more than 200 references [98] (see also the searchable online bibliography [99]).

There were also several attempts at a general design paradigm for problems on graphs on bounded treewidth. One of the first of these general results is by Takamizawa, Nishizeki, and Saito [171, 170] who showed that all problems characterized by a finite number of forbidden graphs admit linear-time algorithms on series-parallel graphs. This was followed by the works of Wimer, Hedetniemi, and Laskar [186], and of Arnborg and Proskurowski [7], both of which describe linear time dynamic programming algorithms. A common aspect of both these approaches is that they need to be tailored to specific problems. Even today, the majority of algorithms for graphs of bounded treewidth essentially follow the dynamic programming framework established in [7], see, e.g., [51, 181].

Bern, Lawler, and Wong [11] considered the problem of finding an optimal subgraph H in a given weighted graph G. They devised a general approach for constructing linear-time algorithms for this problem when the graph G is defined by certain rules of decomposition and the desired subgraph H satisfies a property that is "regular" (has finite state) with respect to these rules of decomposition.

Their regularity property has also been used later in [132, 27, 26]. Bodlaender [17] defined two classes of graph decision problems and introduced polynomial time algorithms for these problems on graphs with bounded treewidth. Scheffler and Seese [162, 160] formalize graph properties that can be expressed in a certain local logical calculus called "L-existential locally verifiable", which also yields linear time algorithms for such graphs. All these frameworks already included many important graph problems for which algorithms for graphs of bounded treewidth were known, including, e.g., Vertex Cover, 3-Colorability, and Hamiltonian Cycle. By the late 1980s, Courcelle was able to unify many previous results by showing that every problem that is definable in monadic second-order logic (MSO), a logical calculus with huge expressive power, can be decided in linear time on graphs of bounded treewidth. This celebrated result became known as *Courcelle's Theorem*.

Theorem 1 (**Courcelle's Theorem** [37]). *Fix an MSO-definable graph property* Π *and a positive integer* w. *There is an algorithm that for every graph* $G = (V, E)$ *of order* $n := |V|$ *and of treewidth at most* w *decides whether* $G \in \Pi$ *in time* $O(n)$.

Courcelle's Theorem was actually stated for Counting MSO (CMSO), an extension

of MSO by predicates that can count modulo integers, but differences can be neglected here. We note that the approach of Bern, Lawler, and Wong [11] resembles that of Courcelle. In particular, their technique of constructing a finite-state tree automaton that recognizes the tree-decomposition of the input graph is also commonly used to prove Courcelle's Theorem (see Section 4.1). However, their approach still lacks the syntactic formalism of MSO required to formally capture the class of problems that admit the desired regularity property. A result very similar to Courcelle's Theorem was later obtained by Borie, Parker, and Tovey [27] utilizing the framework of Bern, Lawler, and Wong. Their approach works for optimization problems such as Minimum Vertex Cover, which cannot be expressed in MSO. At this point, we note that the language of MSO can express only graph properties, which translates into decision problems. In particular, optimization problems cannot be expressed in the framework of MSO logic.

Not much later, Courcelle's Theorem was extended by Arnborg, Lagergren, and Seese [6], who proved that the result can be lifted to a large class of decision, optimization and counting problems. For this they define an extension of MSO called Extended MSO (EMSO) where one can specify a property using an MSO-formula along with a function that *evaluates* the solutions to the MSO-formula. In this way, decision problems with integer numbers as a part of the input as well as optimization and counting problems can be expressed on top of MSO. We will describe this useful concept in Section 3.3.2. Hohberg and Reischuk [103] used homomorphisms into suitable rings to model such evaluations. Courcelle and Mosbah [48] showed that if problem can be expressed as a semiring homomorphism that maps satisfying assignments of the MSO-formula into a semiring of choice (see Section 3.3.3 for details), then the problem can be solved in linear or polynomial time on graphs of bounded treewidth. This unifies many of the previous results such as the ones in [170, 11, 37]. We note that the results of Arnborg, Lagergren, and Seese [6] and Courcelle and Mosbah [48] are orthogonal in the sense that each framework includes problems that cannot be expressed in the other [48]. Finally Flum, Frick, and Grohe [69] addressed the problem of how the linear running time predicted by Courcelle's Theorem can be obtained in the RAM model by using more sophisticated data structures and algorithms.

Courcelle's Theorem, which is a key ingredient in many papers, is one of those results that are referred to nowadays as a *meta-theorem*. These are results that hold not just for a few isolated problems but for a whole *class* of problems. Examples of other meta-theorems can be found in [148, 23, 71, 64]. We also refer the reader to the surveys on algorithmic meta-theorems by Grohe and Kreutzer [92, 119, 93]. It is important to realize that while meta-theorems are immensely useful in quickly establishing whether a problem at hand admits an algorithm of a particular type, they are usually not of much use in providing algorithms that are usable. In fact, it is considered folklore that Courcelle's Theorem does not lead to efficient algorithms that can actually be used in practice. For instance, Niedermeier [144] writes in his well-known textbook on parameterized algorithms

> *It must be emphasized, however, that the now described methodology is of purely theoretical interest because the associated running times suffer from*

> *huge constant factors and combinatorial explosions with respect to the parameter treewidth.* [...] *After establishing fixed-parameter tractability in this way, as a second step one should then head for a concrete, problem-specific algorithm with improved efficiency* [144, p. 169f].

Similar statements by other authors can be found in, e.g., [88, 91].

This pessimism is with good reason. First, the standard proof for Courcelle's Theorem requires one to construct a finite-state tree automaton that recognizes the tree-decomposition. While the construction of finite (tree) automata from MSO formulas is standard and particularly well-understood in theory [28, 65, 172, 54, 173], it remains a challenging task in practice due to problems of space-explosion, cf., [115, 136].

Second, although Courcelle's Theorem states that graph properties expressible in MSO can be checked in linear time, theoretical lower bounds on the *constant* in the running time expression show that it is simply too large to render the algorithm useful. For example, even for *trees* the constant is a tower of twos $2^{2^{2^{\cdots}}}$ whose height depends on the formula:

$$\left.2^{2^{\cdot^{\cdot^{\cdot^{2^2}}}}}\right\}\text{height}$$

Frick and Grohe show that these constants cannot be improved unless P $\neq$ NP [76] (see Section 4.2 for details). In other words, for arbitrary formulas it is simply impossible to prove any *efficient* running time bounds for Courcelle's Theorem. Of course, these lower bounds do not hold for all input formulas. Nevertheless, even for *fixed* formulas (consider a fixed problem like 3-Colorability, Hamiltonian Cycle, for instance) it remains a tedious task to show concrete upper (or lower) bounds for the automata-theoretic approach, see, e.g., [44, Chapter 6] or [40]. Finally, actual experiments reveal that the problems "predicted" theoretically, in particular the space-explosion problems, occur in practice, cf. [112, 167, 88, 87, 40] and Section 5.

This explains why the trend has now shifted from generic algorithms to specialized algorithms where the dependency on the treewidth is explicitly specified. This dependency was usually neglected in previous papers where the treewidth is considered a constant. One of the first papers where the running times include this dependency on treewidth is by Arnborg and Proskurowski [7], where they give some "crude performance estimates." For example, they (implicitly) give a running time of $O(9^w n)$ for the Minimum Dominating Set problem, but it is easy to see that with minor modifications their algorithm runs in $O(5^w n)$ time (w denotes the treewidth). The focus in their paper was on devising a design approach for linear-time algorithms for graphs of bounded treewidth, and no effort was made to improve the dependency for particular algorithms. Nevertheless, many of the algorithms presented in their work, including the $O(2^w n)$ algorithm for Minimum Vertex Cover and the $O(3^w n)$ algorithm for 3-Colorability, remain the best-known algorithms for graphs of bounded treewidth to date.

Current research in this area is mainly towards improving the running time dependency on the treewidth w. For example, for Minimum Dominating Set, the running time was first improved to $O(4^w \cdot n)$ in [3] and to $O(3^w \cdot w^2 n)$ in [181]. For further information on such specialized algorithms for graphs of bounded treewidth, we refer

to the surveys of Bodlaender [20] and Bodlaender and Koster [15] and the references therein.

We mention two recent break-through results in this area. First, Lokshtanov, Marx, and Saurabh [131] showed that the running times of several algorithms for problems on graphs of bounded treewidth, including the $O(2^w n)$ and $O(3^w n)$ algorithms for Minimum Vertex Cover and 3-Colorability in [7], are essentially the best possible. Any improvement in the running times would directly imply faster algorithms for the well-researched SAT problem which is considered unlikely by many. Second, it has been a long-standing open question whether several problems that obey a *connectivity constraint* on the solution set (Steiner Tree, Hamiltonian Cycle, Minimum Connected Dominating Set, ...) allow for algorithms with a running time of $O(c^w poly(n))$ for a constant c, i.e., with a single-exponential dependency on the treewidth. This has been answered in the affirmative by Cygan, Nederlof, Pilipczuk, Pilipczuk, van Rooij, and Wojtaszczyk [51] by using a novel, randomized technique called *Cut-and-Count* that extends to a lot of problems with connectivity constraints.

Beside these improvements in specialized algorithms for specific problems, there is an increasing interest in making the general approach outlined by Courcelle's Theorem feasible in practice. A couple of groups (including the authors of this survey) have worked on implementations of Courcelle's Theorem. With some considerable effort there are ways to actually obtain algorithms that can be used in practice. In the following, we survey work and advances in practical aspects of MSO model checking on decomposable graphs.

1.1. Notation and Problems

Throughout this survey, we consider simple, loop-free, undirected graphs $G = (V, E)$ only. We always denote by $n := |V|$ the number of vertices and by $m := |E|$ the number of edges of G. Let us formally define a few well-known and important graph problems that we shall use for examples throughout this work.

We consider the three NP-complete [83] decision problems Vertex Cover, Dominating Set, and 3-Colorability. These are to decide whether a graph has a small vertex cover, a small dominating set, or whether a graph is three-colorable. We first introduce these three graph theoretic notions. Let $G = (V, E)$ be a graph.

- A vertex set $U \subseteq V$ is called a *vertex cover* of G if for every edge $\{u, v\} \in E$ at least one of its endpoints u or v is contained in U.
- A vertex set $U \subseteq V$ is called a *dominating set* of G if each vertex is contained in U or has at least one neighbor that is contained in U.
- A partition R, G, B of the vertex set V is called a *three-coloring* if no two adjacent vertices share the same color. We say G is *three-colorable*, if the vertex set of G can be partitioned into a three-coloring R, G, B.

Note that the vertex set V is always a vertex cover and a dominating set of G. It is, however, hard to decide whether the graph admits, respectively, a small vertex cover or

a small dominating set, or whether a graph is three-colorable. The following classical decision problems are NP-complete, cf. [83].

Vertex Cover	
Input:	A graph $G = (V, E)$ and an integer k
Question:	Is there a vertex cover $U \subseteq V$ of G with $\|U\| \leq k$?

Dominating Set	
Input:	A graph $G = (V, E)$ and an integer k
Question:	Is there a dominating set $U \subseteq V$ of G with $\|U\| \leq k$?

3-Colorability	
Input:	A graph $G = (V, E)$
Question:	Is G three-colorable?

We also consider the NPO-complete optimization variants of Vertex Cover and Dominating Set, where the solution size is not part of the input:

Minimum Vertex Cover	
Input:	A graph $G = (V, E)$
Solutions:	Vertex covers $U \subseteq V$
Goal:	Minimize $\|U\|$

Minimum Dominating Set	
Input:	A graph $G = (V, E)$
Solutions:	Dominating sets $U \subseteq V$
Goal:	Minimize $\|U\|$

2. Treewidth and tree decompositions

We have mentioned that many problems become easier on trees and on graphs that have a structure similar to trees. The notion of treewidth, which is a measure of how close an undirected graph is to a tree, was first introduced by Halin [96] and later rediscovered by Robertson and Seymour [153] as a crucial part of their Graph Minors Project. The formal definition of treewidth is a little cumbersome and we therefore propose to introduce it via an equivalent game-theoretic characterization.

The game in question is called the cops-and-robber game [128, 164]. There are two parties in this game: a robber who lives on the vertices of a graph $G = (V, E)$ and cops who move about in helicopters. The cops and the robber can see each other and the goal of the cops is to catch the robber; the goal of the robber is to evade capture. The robber is allowed to move about in the graph using its edges at an infinite speed. The point of the helicopters is that the cops are not constrained to move along edges, but their

speed is finite. Moreover, the only way the cops can catch the robber is by landing a helicopter on the vertex occupied by the robber. The robber cannot run through a cop. The point of the unbounded speed of the robber is that he can see where the cops are going to land and can react before the cops have actually landed. Thus, the only way for the cops to capture the robber is by blocking all escape routes, that is, by placing cops on all vertices that are neighbors of the vertex occupied by the robber. We now define the treewidth of a graph G as the minimum number of cops required to catch a robber in G *minus one*. The "minus one" is cosmetic and ensures trees have treewidth one.

In order to get a feeling for this cop-and-robber game, it is a good idea to come up with cop or robber strategies for graphs such as trees, circles, grids, and cliques. We will assume that the graphs we will deal with have at least two vertices as otherwise only one cop is sufficient to catch the robber. If the graph is connected and has at least two vertices then one cop cannot catch the robber because the latter can always move to a cop-free vertex. For trees, it is actually quite easy to see that two cops are both necessary and sufficient. One of the cops lands on an arbitrary vertex u of the tree T and this forces the robber to move to a subtree T_i of $T - u$. The second cop then moves to the unique neighbor of u in T_i and the game continues with the roles of cops exchanged. The two cops effectively push the robber to a leaf of the tree and trap him there.

For a cycle C_n on $n \geq 3$ vertices, two cops do not suffice. Here is a robber strategy that demonstrates this fact. For any two vertices u, v chosen by the cops, the robber moves to a vertex in the larger component of $C_n - \{u, v\}$ before the second cop lands. For $n \geq 3$, a free vertex is always available and the robber can always elude capture. Three cops, however, can catch the robber as follows. Place two cops at diametrically opposite vertices of C_n. The robber must move into a path $P \in C_n - \{u, v\}$. Place the third cop on a vertex that is in the middle of P and repeat. This cop strategy halves the size of the connecting component inhabited by the robber at each stage. The process is repeated until the robber is caught.

A clique on n vertices requires n cops as $n - 1$ cops always leaves a cop-free vertex that can be used by the robber. For an $n \times m$-grid one can show that $\min\{n, m\} + 1$ cops suffice. Place $\min\{n, m\}$ cops on a row or column (depending on which is shorter) and use the last cop so that the row (or column) of cops "sweep across" the grid. Showing that $\min\{n, m\} + 1$ cops are necessary is more difficult and we will not prove it here. The interested reader may refer to [53].

Before we state the formal definition of treewidth, let us consider once again the cop-strategy for trees. The cop-strategy in this case crucially used the fact that every (internal) vertex of a tree is a cut-vertex. A tree can (trivially) be decomposed into cut-vertices. The definition of treewidth essentially replicates this idea with cut-vertices being replaced by vertex-sets which are called *bags*. Removing a bag decomposes the graph into smaller pieces similar to what happens when an internal vertex of a tree is removed. These smaller components can themselves be recursively decomposed into bags. The treewidth of the graph is essentially the maximum size of the bags in the decomposition. Of course, to ensure that the bags are actually cut-sets, they must satisfy

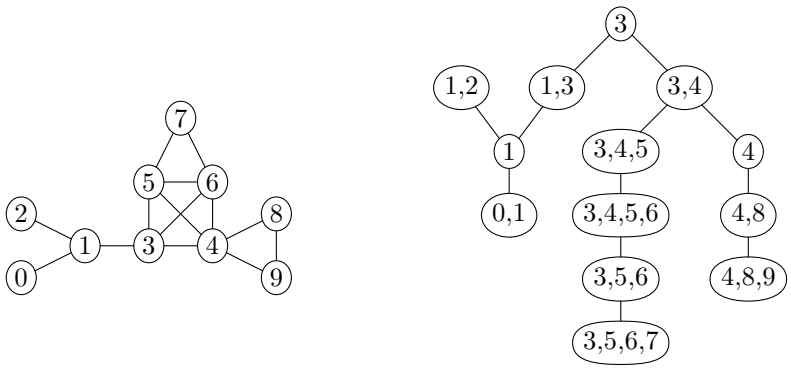


Figure 3: A simple graph and an optimal tree decomposition.

some additional properties which are specified in the formal definition that follows. For a proof that the game-theoretic definition is equivalent to the one given below see [164].

Definition 2 (Tree Decomposition)**.** A *tree decomposition* of a graph $G = (V, E)$ is a pair $(T, \mathcal{X})$, where $T = (I, F)$ is a tree, and $\mathcal{X} = \{ X_i \mid i \in I \}$ is a family of subsets of V, one for each node $i \in I$, such that

- for all vertices $v \in V$ there exists $i \in I$ with $v \in X_i$,
- for all edges $\{v, w\} \in E$ there exists $i \in I$ with $u, v \in X_i$, and
- for each vertex $v \in V$, the set of nodes $\{ i \in I \mid v \in X_i \}$ is connected in T.

The vertices of the tree T are usually referred to as *nodes* and the sets X_i are called *bags*. The *width* of a tree decomposition is the size of the largest bag minus one. The *treewidth* of a graph G is the minimum width over all possible tree decompositions of G.

Example 3. Figure 3 shows a small graph and a corresponding tree decomposition of width 3. This is optimal because one also requires at least four cops to catch the robber on the graph induced by the vertices $3, 4, 5, 6$ alone. The bags of the tree decomposition can be understood as the description of a cop strategy. For example, if in the very first step the first cop lands on the cut-vertex 3 of the graph, as seen in the topmost bag of the tree decomposition, then the robber is either in the left component induced by the vertices $0, 1, 2$ or in the right component induced by the vertices $4, \ldots, 9$. The tree decomposition then tells the cops which vertex is to be blocked next; in this case, vertex 1 or 4, respectively. The cops continue to occupy the vertices as described by the tree decomposition, until eventually the robber is caught either on vertex 0 (with cops on vertices $0, 1$), on vertex 2 (with cops on vertices $1, 2$), on vertex 7 (with cops on vertices $3, 5, 6, 7$), or on vertex 9 (with cops on vertices $4, 8, 9$).

We mention another equivalent definition of treewidth using what are known as partial k-trees. Partial k-trees were introduced in the 1960s [13, 14, 157], but their connection with treewidth was established much later.

Definition 4 (k-Trees and Partial k-Trees)**.** The class of k-trees is inductively defined as follows.

1. A complete graph on k vertices is a k-tree.
2. If H is a k-tree on n vertices with $k \leq n$, adding a new vertex and connecting it to all the vertices of a k-clique in H creates a k-tree on $n+1$ vertices.

A partial k-tree is a graph obtained by deleting some edges (but not vertices) of a k-tree.

We have the following result.

Theorem 5 ([159, 185, 180])**.** *A graph has treewidth k if and only if it is a partial k-tree.*

It is useful to be acquainted with several equivalent definitions of treewidth because in a given situation one definition might prove to be more useful than the others. For example, as an application of the above theorem, we show that graphs of treewidth k are *sparse* in the sense that the number of edges is a linear function of the number of vertices:

$$m \leq k \cdot n + \binom{k}{2}$$

To see this, simply observe that a k-tree on n vertices has exactly $\binom{k}{2} + k(n-k)$ edges. In fact, the partial k-tree formulation shows that graphs of treewidth k are k-degenerate (every subgraph of it has a vertex of degree at most k). This fact is not very easy to see using the cops-and-robber game formulation. Sparse graphs, in general, are an interesting topic in themselves and is an active area of research. See the recent book by Nešetřil and Ossona de Mendez [143].

Several important graph classes occurring in practice can be shown to have small treewidth. An oft-quoted example is the control-flow graphs of `goto`-free C programs that have treewidth at most six as shown by Thorup [174]. Similar results were obtained for Java [95] and Ada [30]. For more background on treewidth, we refer the reader to surveys such as [18, 21].

2.1. Computing Tree Decompositions

A typical algorithm that makes use of dynamic programming on graphs of bounded treewidth requires a tree-decomposition of the graph to be supplied as input. Therefore being able to compute tree-decompositions of small width efficiently is crucial for practical applications. Unfortunately, the problem of deciding whether a graph has treewidth at most width w (the TREEWIDTH problem) is NP-complete [5].

For *fixed* values of w, the first polynomial-time algorithm for the problem was by Arnborg, Corneil, and Proskurowski [5]. Their algorithm runs in time $O(n^{w+2})$. Moreover, the problem admits an algorithm with running time $O(f(w) \cdot n^{O(1)})$, where f is

a function of w alone. That is, TREEWIDTH is fixed-parameter tractable [55, 70, 144] with respect to the width w as parameter. The fixed-parameter tractability of TREEWIDTH was first shown by Robertson and Seymour using results from their Graph Minors Project [153, 155]. However their result is not constructive. In 1996, Bodlaender published an algorithm that decides whether a given graph has treewidth at most w in time $O(2^{43 \cdot w^3} \cdot n)$ and if so, constructs a tree-decomposition of width at most w [19]. However this algorithm uses too much time even for $w = 4$ [18, 156]. In fact, it is a challenging open problem to design a more efficient algorithm that computes an optimal tree-decomposition.

A survey of Bodlaender and Koster [15] lists a number of algorithms that have been implemented and tested in practice. These include polynomial-time algorithms for bounded width, branch-and-bound algorithms, and moderately-exponential time algorithms. Unfortunately, all these algorithms are either completely unusable in practice or infeasible for non-trivial instances of reasonable sizes (larger than, say, 100 vertices). Fortunately, in practice it usually suffices to use a heuristic for computing the tree-decomposition, since it is used only as a guide for the dynamic programming algorithm. Using a non-optimal tree-decomposition affects only the running time of the algorithm, which will nevertheless output the correct solution. Several heuristics for obtaining small tree-decompositions are known and they seem to work well in a practical setting. Often, nearly optimal tree-decompositions can be computed [16, 24] and thus it can actually be considered preferable to use heuristics for this step of the computation. Given that heuristics are fast compared to the actual dynamic programming algorithm that uses the tree-decomposition, in some situations it makes sense to use several heuristics to compute a tree-decomposition of the input graph and use the one with the smallest width.

The aforementioned paper [15] surveys many approximation algorithms with strict performance guarantees and classic best-effort heuristics. The reader interested in these algorithms are referred to [15], [44, Section 6.2] and the references therein. An experimental study of the performance of a large number of treewidth heuristics can be found in [16, 24].

3. Logic and Graphs

The main purpose of this section is to introduce the reader to logic and, in particular, to monadic second-order logic, which is the formalism used in Courcelle's Theorem and its extensions. Our plan is to begin with the very basics and this approach makes this section rather long. The reader familiar with this area may move ahead to the next section.

Let us motivate the need for a logical formalism in computer science in general and model-checking in particular. One can describe graph properties or graph problems in some natural language, but natural languages are inherently ambiguous. Therefore if we want to specify a problem as input to an algorithm, for instance, we need a formal language that adheres to certain *syntactic rules* so that statements in that language

can be parsed by the algorithm. Furthermore, we need to ascribe meaning (*semantics*) to the words of this formal language so that the algorithm can, in a sense, understand statements in that language. These criteria are fulfilled by *mathematical logics*.

A *logic* is a formal language whose words are called *formulas* or *statements*, and they are given a meaning or semantic by defining when statements *hold*, or are *true* in the logic. The field of mathematical logic has natural connections to philosophical logics and artificial languages, cf., [158]. We refer the interested reader to introductory texts such as [25, 66, 138, 165]. In what follows, we introduce the logical calculus that is relevant to this survey.

3.1. Propositional Logic

Propositional logic is concerned with propositions and their interrelationships. A proposition is statement that has the property of being either *true* or *false*. Propositions may be combined in various ways to create more complicated propositions in a way such that the truth or falsity of resulting proposition is determined by the truth or falsity of its component propositions.

There are three basic operations on propositions. The simplest of these is the *negation* operation. Given a proposition A, the negation of A, denoted by $\neg A$, is a proposition which is true if and only if A is false. The *conjunction* of two propositions A and B is denoted by $A \wedge B$. The proposition $A \wedge B$ is true if and only if both A and B are true. The *disjunction* of A and B is denoted by $A \vee B$ and is true if and only if at least one of A or B is true.

To formally describe the *syntax* of propositional logic, we start with a set $\mathcal{A} = \{ A_1, \ldots, A_i \mid i \in \mathbf{N} \}$ of *propositions*. A *propositional formula* is then defined as follows.

- Every proposition is an (atomic) formula.
- If F and G are formulas, then $(F \vee G)$ and $(F \wedge G)$ are formulas.
- If F is a formula, then $\neg F$ is a formula.

Here is how one assigns meaning to formulas (we follow [161]): Let $\Phi_{\mathcal{A}}$ be the set of all propositional formulas having atomic formulas in $\mathcal{A}$, and $\varphi \in \Phi_{\mathcal{A}}$ a propositional formula. Let $\alpha \colon \mathcal{A} \to \{0, 1\}$ be a function assigning truth values to propositions, where we use the integer 0 for *false* and 1 for *true*. We call α an *assignment* of $\mathcal{A}$. We extend α to $\alpha \colon \Phi_{\mathcal{A}} \to \{0, 1\}$ via:

- $\alpha(\neg F) = 1 - \alpha(F)$.
- $\alpha(F \vee G)$ is 1 if $\alpha(F) + \alpha(G) > 0$ and 0 otherwise.
- $\alpha(F \wedge G)$ is 1 if $\alpha(F) \cdot \alpha(G) > 0$ and 0 otherwise.

If $\varphi \in \Phi_{\mathcal{A}}$ is a formula and $\alpha \colon \mathcal{A} \to \{0, 1\}$ is an assignment of $\mathcal{A}$, then we call α a *model* for φ iff $\alpha(\varphi) = 1$.

Using truth tables, one can show that, for instance, $\neg(A_i \wedge A_j)$ is semantically equivalent to $\neg A_j \vee \neg A_i$ and that the *implications* $(F \to G)$ and $(F \leftrightarrow G)$ can be expressed

by the equivalent formulas $(\neg F \vee G)$ and $(F \to G) \wedge (G \to F)$, respectively, which sometimes are convenient abbreviations when expressing properties.

3.2. First-order Logic

First-order logic (FO) builds on propositional logic by adding the quantifiers *for all* ($\forall$) and *there exists* ($\exists$). In FO, we are interested in whether a formula holds on certain *structures*. In this survey, we restrict ourselves to graphs which makes the logical foundations more accessible. We follow [60].

We first fix a countably infinite set of *individual variables* $x_1, x_2, \ldots$ that we want to use as place-holders for vertices in the problem specification. The *formulas* of FO are strings that are obtained from finitely many applications of the following rules:

1. If t_1 and t_2 are individual variables then $t_1 = t_2$ is a formula.
2. If t_1, t_2 are individual variables, then $\mathrm{adj}(t_1, t_2)$ is a formula.
3. If φ is a formula then $\neg\varphi$ is a formula.
4. If $\varphi_1, \ldots, \varphi_k$ are formulas then $(\varphi_1 \vee \cdots \vee \varphi_k)$ and $(\varphi_1 \wedge \cdots \wedge \varphi_k)$ are formulas.
5. If φ is a formula and x is an individual variable then $\exists x\varphi$ and $\forall x\varphi$ are formulas.

Variables that are not within the scope of a quantifier are called *free*. Free variables can be used to test properties w.r.t. these variables; to this end, their values will be "provided" to the formula. A formula without free variables is called a *sentence*. By $\mathrm{free}(\varphi)$ we denote the set of free variables of φ.

We now assign meanings to the logical symbols by defining the *satisfaction relation* $\models$. Let $G = (V, E)$ be a graph. We first need to provide values for the free variables of a formula. An *assignment* in G is a function $\alpha\colon \mathrm{free}(\varphi) \to V$ that assigns individual variables values in V. For an individual variable x and an assignment α, we let $\alpha[x/a]$ denote an assignment that agrees with α except that it assigns the value $a \in V$ to x. We define the relation $G \models \varphi[\alpha]$ (φ is *true* in G under α) as follows:

$G \models t_1 = t_2[\alpha]$	iff	$\alpha(t_1) = \alpha(t_2)$
$G \models \mathrm{adj}(t_1, t_2)[\alpha]$	iff	$\{\alpha(t_1), \alpha(t_2)\} \in E$
$G \models \neg\varphi[\alpha]$	iff	not $G \models \varphi[\alpha]$
$G \models (\varphi_1 \vee \cdots \vee \varphi_k)[\alpha]$	iff	$G \models \varphi_i[\alpha]$ for some $1 \leq i \leq k$
$G \models (\varphi_1 \wedge \cdots \wedge \varphi_k)[\alpha]$	iff	$G \models \varphi_i[\alpha]$ for all $1 \leq i \leq k$
$G \models \exists x\varphi[\alpha]$	iff	there is an $v \in V$ such that $G \models \varphi[\alpha[x/v]]$
$G \models \forall x\varphi[\alpha]$	iff	for all $v \in V$ it holds that $G \models \varphi[\alpha[x/v]]$

If φ is a sentence and $G \models \varphi[\alpha]$, where $\alpha\colon \emptyset \to V$, then we say G is a *model* for φ and just write $G \models \varphi$.

Definition 6. A graph property Π is called FO-definable if there is an FO-sentence φ such that for a graph $G = (V, E)$ it holds

$$G \in \Pi \qquad \text{iff} \qquad G \models \varphi.$$

Example 7. Let $k \geq 0$ be an integer.

- The following FO-sentence expresses that a graph has a vertex cover of size at most k:

$$\exists v_1, \ldots, \exists v_k \forall u \forall v \big(\mathrm{adj}(u,v) \rightarrow (u = v_1 \vee v = v_1 \vee \cdots \vee u = v_k \vee v = v_k) \big)$$

  In the formula, we let existential quantifiers guess k vertices of the vertex cover, and then verify that for all edges $\{u, v\}$ at least one of the endpoints is one of the k selected vertices.

- The following FO-sentence expresses that a graph has a dominating set of size at most k:

$$\exists v_1, \ldots, \exists v_k \forall u \exists v \big((\mathrm{adj}(u,v) \vee u = v) \wedge (v = v_1 \vee \cdots \vee v = v_k) \big) \big)$$

  Again, we let existential quantifiers guess k vertices for the dominating set, and then verify that all vertices are dominated.

Consequently, the graph properties of having a vertex cover of size at most k or a dominating set of size at most k are FO-definable.

3.2.1. FO Model-Checking

We are now interested in the complexity of the following problem: We are given a formula φ, a graph $G = (V, E)$ and an assignment α of the free variables of the formula in this graph, and we are asked whether $G \models \varphi[\alpha]$. This is known as the FO Model-Checking problem:

FO Model-Checking	
Input:	An FO-formula φ, a graph $G = (V, E)$, and an assignment $\alpha \colon \mathrm{free}(\varphi) \to V$
Question:	Does $G \models \varphi[\alpha]$?

This problem is easily seen to be PSPACE-complete: Hardness follows from the fact that quantified Boolean formulas (QBF) can naturally be written as first-order logic formulas with equality $x = y$, and QBF Satisfiability is a classical PSPACE-complete problem, cf. [168, 187]. The reduction from QBF Satisfiability to FO Model-Checking works even if the graph (that is an input to the latter problem) has only two vertices. This indicates that the "hardness" of FO Model-Checking lies in the formula itself. To see that FO Model-Checking is in PSPACE we can use Algorithm 1, which recursively tests whether the formula holds by following the definition of the satisfaction relation $\models$. If $k := \|\varphi\|$ denotes the length of the input formula, it is not hard to see that Algorithm 1 uses at most n^k recursive calls, and hence its running time is $O(n^{k+c})$, where c is a constant that depends on the polynomial overhead for each recursive call.

In particular, if the formula is fixed then FO Model-Checking can be solved in polynomial time.

Algorithm 1 FO-Model Checking Algorithm $A(\varphi, G, \alpha)$

Input: An FO-formula φ, a graph $G = (V, E)$, and an assignment $\alpha\colon \text{free}(\varphi) \to V$

if $\varphi = t_1 = t_2$ **then return** 1 iff $\alpha(t_1) = \alpha(t_2)$ and 0 otherwise
if $\varphi = \text{adj}(t_1, t_2)$ **then return** 1 iff $\{\alpha(t_1), \alpha(t_2)\} \in E$ and 0 otherwise
if $\varphi = \neg\psi$ **then return** $1 - A(\psi, G, \alpha)$
if $\varphi = (\varphi_1 \vee \cdots \vee \varphi_k)$ **then**
    for $i = 1, \ldots, k$ **if** $A(\varphi_i, G, \alpha) = 1$ **then return** 1
    return 0
if $\varphi = (\varphi_1 \wedge \cdots \wedge \varphi_k)$ **then**
    for $i = 1, \ldots, k$ **if** $A(\varphi_i, G, \alpha) = 0$ **then return** 0
    return 1
if $\varphi = \exists x\, \psi$ **then**
    for each $v \in V$ **if** $A(\psi, G, \alpha[x/v]) = 1$ **then return** 1
    return 0
if $\varphi = \forall x\, \psi$ **then**
    for each $v \in V$ **if** $A(\psi, G, \alpha[x/v]) = 0$ **then return** 0
    return 1

Theorem 8 (Folklore)**.** *Fix an FO-definable graph property* Π*. There is an algorithm that for every graph* $G = (V, E)$ *of order* $n := |V|$ *decides whether* $G \in \Pi$ *in time* $O(poly(n))$*.*

Hence, for every fixed FO-formula φ the following problem is in P:

φ-FO Model-Checking	
Input:	A graph $G = (V, E)$, an assignment $\alpha\colon \text{free}(\varphi) \to V$
Question:	Does $G \models \varphi[\alpha]$?

Recall that even though Vertex Cover and Dominating Set are NP-complete, checking whether a graph has a vertex cover or a dominating set of at most k, respectively, is in P *for each fixed integer* k, since there are only $\binom{n}{k}$ subsets of size k, which for every fixed k can be enumerated in polynomial time.

If we leave the realm of general graphs and consider more restricted graph classes, then the φ-FO Model-Checking problem can often be solved in linear time. For example, a result by Seese states that the problem can be solved in linear time on graphs of bounded degree.

Theorem 9 (Seese [163])**.** *Fix an FO-definable graph property* Π *and an integer* $d > 0$*. There is an algorithm that for every graph* $G = (V, E)$ *of order* $n := |V|$ *and maximum degree at most* d *decides whether* $G \in \Pi$ *in time* $O(n)$*.*

The underlying reason here is, roughly speaking, that first-order formulas of size k can only express *local* properties: An FO-formula of bounded size cannot distinguish

two vertices u_1, u_2 of distance large enough from each other if their local neighborhoods of radius k are isomorphic, since each node on a path between two vertices requires one quantifier, and thus only paths of bounded size can be defined (which corresponds to the radius). This property is formalized in a result that is known as Gaifman's Locality Theorem [77]. Furthermore, on graphs of bounded degree d, the size of the neighborhood within a constant radius r has size at most $d^r + 1$. On graphs of bounded degree, to decide whether the formula holds it then suffices to count the number of different local neighborhoods up to a radius of k by a result of Hanf [97] (the Hanf-Sphere-Lemma).

Similar results were shown for several graph classes, all of which are *locally* restricted in some sense. The running time of the algorithms in each of these cases is either linear or almost linear: $O(n^{1+\epsilon})$ for an arbitrary $\epsilon > 0$. For example, Frick and Grohe [75] showed that φ-FO Model-Checking admits a linear-time algorithm for locally tree-decomposable graphs and an almost linear-time algorithm for graphs of locally bounded treewidth. Locally tree-decomposable graphs include planar, bounded degree, and bounded treewidth graphs. Dawar, Grohe, and Kreutzer give polynomial time algorithms for graphs locally excluding an arbitrary but fixed minor, where the degree of the polynomial does not depend on the formula [52]. All these results were generalized in [59] to graphs of bounded expansion and locally bounded expansion. Graphs of bounded expansion include, in particular, graphs of bounded treewidth, all proper minor-closed graph classes, and graphs of bounded degree. Graphs of locally bounded expansion include graphs of locally bounded treewidth and those locally excluding a minor [59].

Theorem 10 (Dvořák, Král, and Thomas [59])**.** *Fix an FO-definable graph property* Π*. There is an algorithm that for every graph* $G = (V, E)$ *of order* $n := |V|$ *and bounded expansion decides whether* $G \in \Pi$ *in time* $O(n)$*.*

Theorem 11 (Dvořák, Král, and Thomas [59])**.** *Fix an FO-definable graph property* Π *and a number* $\epsilon > 0$*. There is an algorithm that for every graph* $G = (V, E)$ *of order* $n := |V|$ *and locally bounded expansion decides whether* $G \in \Pi$ *in time* $O(n^{1+\epsilon})$*.*

In this survey, we focus on monadic second-order logic, a much richer language than FO.

3.3. MSO Logic

By Theorem 8, FO-definable graph properties can be decided in polynomial time. Assuming P $\neq$ NP, graph properties that are NP-hard to decide cannot be defined in FO. One such problem is 3-Colorability, which cannot be expressed in FO. Indeed, one would intuitively expect that a logic in which 3-Colorability can be expressed provides methods to express that there *exists* a *partition* of the vertex set into colors *sets red*, *green*, *blue*, such that no two adjacent vertices have the same color, which requires some means to fix/guess subsets (such as colors) and check properties w.r.t. these subsets (such as no two adjacent vertices share the same color).

Monadic second-order logic (MSO) is an extension of first-order logic which allows quantification over sets of vertices and edges. In second-order logic, relations of arbitrary

arity can be subject to quantification, and MSO is the *unary* fragment of second-order logic: sets are relations of arity one. To define MSO, we first fix variables that we use as place-holders for objects and sets in the graph. We fix a countably infinite set of *individual variables* $x_1, x_2, \ldots$ and a countably infinite set of *set variables* $X_1, X_2, \ldots$. Every variable x, X is said to have a *type* $\mathrm{tp}(x), \mathrm{tp}(X) \in \{1, 2\}$ denoting whether it refers to vertices ($\mathrm{tp}(x) = 1$), edges ($\mathrm{tp}(x) = 2$), vertex sets ($\mathrm{tp}(X) = 1$), or edge sets ($\mathrm{tp}(X) = 2$), respectively. The *formulas* of MSO are strings that are obtained from finitely many applications of the following rules:

1. If x_1 and x_2 are individual variables with $\mathrm{tp}(x_1) = \mathrm{tp}(x_2)$, then $x_1 = x_2$ is a formula.
2. If x_1, x_2 are individual variables with $\mathrm{tp}(x_1) = \mathrm{tp}(x_2) = 1$, then $\mathrm{adj}(x_1, x_2)$ is a formula.
3. If x_1, x_2 are individual variables with $\mathrm{tp}(x_1) = 1$ and $\mathrm{tp}(x_2) = 2$, then $\mathrm{inc}(x_1, x_2)$ is a formula.
4. If x is an individual variable and X is a set variable with $\mathrm{tp}(x) = \mathrm{tp}(X)$, then $x \in X$ is a formula.
5. If φ is a formula then $\neg\varphi$ is a formula.
6. If $\varphi_1, \ldots, \varphi_k$ are formulas then $(\varphi_1 \vee \cdots \vee \varphi_k)$ and $(\varphi_1 \wedge \cdots \wedge \varphi_k)$ are formulas.
7. If φ is a formula and x is an individual variable then $\exists x\varphi$ and $\forall x\varphi$ are formulas.
8. If φ is a formula and X is a set variable then $\exists X\varphi$ and $\forall X\varphi$ are formulas.

The formulas obtained by 1–4 above are called *atomic formulas*. If the formulas in 8 use edge set variables ($\mathrm{tp}(X) = 2$), we call this quantification an *edge set quantification*, which will be of interest later. The free variables of a MSO-formula φ are denoted by $\mathrm{free}(\varphi)$. If φ has free set variables only, we say φ is a *relational* formula. A formula without free variables is called a *sentence*.

The *quantifier rank* $\mathrm{qr}(\varphi)$ of a formula φ is the maximum number of nested quantifiers occurring in it.

$$\begin{aligned}
\mathrm{qr}(\varphi) &:= 0, \text{ if } \varphi \text{ is atomic;} & \mathrm{qr}(\exists x\varphi) &:= \mathrm{qr}(\varphi) + 1;\\
\mathrm{qr}(\neg\varphi) &:= \mathrm{qr}(\varphi); & \mathrm{qr}(\exists X\varphi) &:= \mathrm{qr}(\varphi) + 1;\\
\mathrm{qr}(\varphi \vee \psi) &:= \max\{\mathrm{qr}(\varphi), \mathrm{qr}(\psi)\}; & \mathrm{qr}(\forall x\varphi) &:= \mathrm{qr}(\varphi) + 1;\\
\mathrm{qr}(\varphi \wedge \psi) &:= \max\{\mathrm{qr}(\varphi), \mathrm{qr}(\psi)\}; & \mathrm{qr}(\forall X\varphi) &:= \mathrm{qr}(\varphi) + 1,
\end{aligned}$$

As in the case of FO, we assign meanings to the logical symbols by defining the *satisfaction relation* $G \models \varphi$. Let $G = (V, E)$ be a graph. We first extend the notion of an assignment to set variables: An *assignment* in G is a function α that assigns the variables of φ values in G. For an individual variable x and an assignment α, we let $\alpha[x/a]$ denote an assignment that agrees with α except that it assigns the value a to x, where $a \in V$ if $\mathrm{tp}(x) = 1$ and $a \in E$ if $\mathrm{tp}(x) = 2$. Similarly, for a set variable X and an assignment α, we let $\alpha[X/A]$ denote an assignment that agrees with α except that it assigns the value A to X, where $A \subseteq V$ if $\mathrm{tp}(X) = 1$ and $A \subseteq E$ if $\mathrm{tp}(X) = 2$.

If φ is a relational MSO-formula with free set variables $\text{free}(\varphi) = \{X_1, \ldots, X_l\}$ and $G = (V, E)$ is a graph, we let

$$\text{assignments}(\varphi, G) := \big\{\, \alpha[X_1/U_1, \ldots, X_l/U_l] \;\big|\; U_i \subseteq V \text{ if } \text{tp}(X_i) = 1 \text{ and } U_i \subseteq E \text{ else} \,\big\},$$

the set of assignments to variables of φ over G. Here, α is the empty assignment. It will be convenient to also identify the set

$$\big\{\, (\alpha(X_1), \ldots, \alpha(X_l)) \mid \alpha \in \text{assignments}(\varphi, G) \,\big\}$$

with the assignments $\text{assignments}(\varphi, G)$.

We define the *satisfaction relation* $\models$ as follows:

$G \models t_1 = t_2[\alpha]$	iff	$\alpha(t_1) = \alpha(t_2)$
$G \models \text{adj}(t_1, t_2)[\alpha]$	iff	$\{\alpha(t_1), \alpha(t_2)\} \in E$
$G \models \text{inc}(t_1, t_2)[\alpha]$	iff	$\alpha(t_1) \in V$ and $\alpha(t_2) \in E$ and $\alpha(t_1) \in \alpha(t_2)$
$G \models t \in X[\alpha]$	iff	$\alpha(t) \in \alpha(X)$
$G \models \neg\varphi[\alpha]$	iff	not $G \models \varphi[\alpha]$
$G \models (\varphi_1 \vee \cdots \vee \varphi_k)[\alpha]$	iff	$G \models \varphi_i[\alpha]$ for some $1 \leq i \leq k$
$G \models (\varphi_1 \wedge \cdots \wedge \varphi_k)[\alpha]$	iff	$G \models \varphi_i[\alpha]$ for all $1 \leq i \leq k$
$G \models \exists x\varphi[\alpha]$	iff	$\text{tp}(x) = 1$ and $\exists v \in V$ such that $G \models \varphi[\alpha[x/v]]$, or $\text{tp}(x) = 2$ and $\exists e \in E$ such that $G \models \varphi[\alpha[x/e]]$
$G \models \forall x\varphi[\alpha]$	iff	$\text{tp}(x) = 1$ and $\forall v \in V$, $G \models \varphi[\alpha[x/v]]$, or $\text{tp}(x) = 2$ and $\forall e \in E$, $G \models \varphi[\alpha[x/e]]$
$G \models \exists X\varphi[\alpha]$	iff	$\text{tp}(X) = 1$ and $\exists U \subseteq V$ such that $G \models \varphi[\alpha[X/U]]$, or $\text{tp}(X) = 2$ and $\exists F \subseteq E$ such that $G \models \varphi[\alpha[X/F]]$
$G \models \forall X\varphi[\alpha]$	iff	$\text{tp}(X) = 1$ and $\forall U \subseteq V$, $G \models \varphi[\alpha[X/U]]$, or $\text{tp}(X) = 2$ and $\forall F \subseteq E$, $G \models \varphi[\alpha[X/F]]$

If $G \models \varphi[\alpha]$, we say α satisfies φ on G or φ is *true* in G under α. Again, if φ is a sentence and $G \models \varphi[\alpha]$, then we say G is a *model* for φ and just write $G \models \varphi$. We let

$$\text{sat}(\varphi, G) := \{\, \alpha \in \text{assignments}(\varphi, G) \mid G \models \varphi[\alpha] \,\}$$

be the set of satisfying assignments of φ on G. We can use the following convenient abbreviations:

- $\varphi_1 \to \varphi_2$ short-hand for $(\neg\varphi_1 \vee \varphi_2)$
- $X_1 \subseteq X_2$ short-hand for $\forall x(x \in X_1 \to x \in X_2)$

Courcelle [37] considers an extension of MSO called Counting MSO (CMSO), by which formulas are able to count modulo integers. Here, we additionally allow atomic formulas of the form $\text{card}_{n,p}(X)$ for a set variable X and integers $0 \leq n < p$ and p prime, which is satisfied if and only if $|\alpha(X)| \equiv n \bmod p$. Most of the techniques and algorithms discussed in the remainder of this section can be extended to CMSO. For simplicity, we stick to MSO in the following.

3.3.1. MSO-definable Properties and Graph Problems

We can now introduce MSO-definable properties and graph problems.

Definition 12. A graph property Π is called MSO-definable if there is an MSO-sentence φ such that for a graph $G = (V, E)$ it holds

$$G \in \Pi \qquad \text{iff} \qquad G \models \varphi.$$

A graph decision problem is MSO-definable if it can be stated as deciding membership of an MSO-definable graph property. A vertex set property is MSO-definable if there is a relational MSO-formula φ with $\text{free}(\varphi) = \{X\}$ and $\text{tp}(X) = 1$ such that for a graph $G = (V, E)$ and any $U \subseteq V$ it holds that U has the property if and only if $G \models \varphi[X/U]$.

Similarly, we define MSO-definable edge set properties and properties on tuples of multiple vertex and edge sets and the corresponding membership decision problems.

Let us give some examples.

Example 13. The following properties are MSO-definable:

- The sets R, G, B partition the vertex set:

$$\begin{aligned} \mathit{part}(R, G, B) = \forall x \big(& (x \in R \vee x \in G \vee x \in B) \wedge \\ & \neg(x \in R \wedge x \in G) \wedge \neg(x \in R \wedge x \in B) \wedge \neg(x \in G \wedge x \in B) \big) \end{aligned}$$

- The sets R, G, B are a three-coloring:

$$\begin{aligned} \mathit{3col}'(R, G, B) = \mathit{part}(R, G, B) \wedge \forall x \, \forall y \, \Big(& \text{adj}(x, y) \rightarrow \\ & \big(\neg(x \in R \wedge y \in R) \wedge \neg(x \in G \wedge y \in G) \wedge \neg(x \in B \wedge y \in B) \big) \Big) \end{aligned}$$

- The graph is three-colorable:

$$\mathit{3col} = \exists R \, \exists G \, \exists B \; \mathit{3col}'(R, G, B)$$

- The vertex set X is a vertex cover:

$$\mathit{vc}(X) = \forall e \, \exists x \, \big(\text{inc}(x, e) \wedge x \in X \big)$$

  Alternatively, we can avoid edge-quantification and write:

$$\mathit{vc}'(X) = \forall x \, \forall y \, \big(\text{adj}(x, y) \rightarrow (x \in X \vee y \in X) \big)$$

- The vertex set X is a dominating set:

$$\mathit{ds}(X) = \forall x \, \big(x \in X \vee \exists y \, (y \in X \wedge \text{adj}(x, y)) \big)$$

- The vertex set X is connected by the edge set F, via expressing that for each bipartition $Y, X \setminus Y$ of X with $|Y|, |X \setminus Y| > 0$ there is an edge in F that connects both parts.

$$\begin{aligned} conn(X,F) = \forall Y \Big(\big(\forall y (y \in Y \to y \in X) \wedge \exists y (y \in Y) \wedge \\ \exists x (x \notin Y \wedge x \in X) \big) \to \\ \exists e \, \exists x \, \exists y \, (e \in F \wedge \mathrm{inc}(x,e) \wedge \mathrm{inc}(y,e) \wedge \\ x \in X \wedge x \notin Y \wedge y \in Y) \Big) \end{aligned}$$

- The vertex set X is connected:

$$conn'(X) = \exists F \, conn(X,F)$$

- The vertex set X is a *connected* dominating set:

$$cds(X) = ds(X) \wedge conn'(X)$$

- The edge set F is an Hamiltonian cycle through the vertex set X, via expressing that F connects X and each vertex in X has exactly two incident edges in F:

$$\begin{aligned} ham'(F,X) = \Big(& conn(X,F) \wedge \\ & \forall x \, \exists e_1 \, \exists e_2 \, \Big(e_1 \in F \wedge e_2 \in F \wedge \mathrm{inc}(x,e_1) \wedge \mathrm{inc}(x,e_2) \wedge \\ & \forall e_3 \, \big((\mathrm{inc}(x,e_3) \wedge e_3 \in F) \to (e_3 = e_1 \vee e_3 = e_2) \big) \Big) \Big) \end{aligned}$$

- Let

$$ham''(F) = \forall X \, \big((\forall x \, x \in X) \to ham'(F,X) \big)$$

  express that $F \subseteq E$ is a Hamiltonian Cycle through all vertices. Then

$$ham = \exists F \, ham''(F)$$

  expresses that the graph is Hamiltonian.

- The fixed graph H is contained as a minor: Writing the exact formula for particular graphs H is rather tedious when done by hand, but can easily be done by a computer program: The resulting formula states that for each vertex v of H, there is a connected set of vertices X_v, such that there is an edge from X_v to X_u if there is an edge from u to v in H. In other words, the formula *guesses* the so-called *model* of H.

- As a corollary, we immediately get that planarity can be defined in MSO by excluding the graphs K_5 and $K_{3,3}$ as minors.

Note that MSO-formulas often resemble the description of properties expressed in a natural language. For example, the formula $vc(X)$ for vertex cover in plain English reads

"For all edges e, there is a vertex x that is incident to e and contained in X,"

which is close to our definition of a vertex cover in Section 1.1. The formulas for connectivity are not so intuitive at first glance, but these only need be defined once and can be used as a black-box afterwards. For example, we can add the connectivity constraint to the dominating set formula $ds(X)$ and obtain the formula *cds* for the *connected dominating set* property. On the contrary, the ILP-formulation of [141] for Minimum Connected Dominating Set, for example, guarantees connectivity of the solution by requiring a flow between the nodes of the solution, and is quite tedious and cumbersome to generate, which has to be done for each individual input graph. Similarly, the formula for Hamiltonian cycles is quite intuitive when $conn(X, F)$ is used as a black-box.

We conclude that MSO is a powerful language, which allows one to express graph properties in a *natural* way. Nevertheless, this natural expression can be parsed — and "understood" — by an *algorithm*. This makes MSO a powerful and convenient tool for expressing graph problems in an algorithmic context.

The next result shows that in fact many important graph decision problems are MSO-definable:

Theorem 14 (Arnborg, Lagergren, and Seese [6])**.** *The problems listed in Table 1 are MSO-definable graph problems.*

Finally, we note that MSO as in our definition allows to quantify over *sets of edges*. In the literature this is sometimes referred to MSO_2 or two-sorted MSO, and MSO without quantification over sets of edges, i.e., with quantification over sets of vertices only is called MSO_1 or one-sorted MSO. MSO_2 is strictly more powerful than MSO_1. For instance, expressing that a graph G contains an Hamiltonian Cycle requires to quantify over sets of edges as in formula *ham* of Example 13, cf. [60].

3.3.2. Extended MSO

As seen in the previous section, a large number of *graph properties* can be defined in MSO. In particular, we can define properties for which the corresponding membership problem is NP-hard or even PSPACE-hard to decide. However, problems whose input instances consist of more than just a graph cannot be defined with a single MSO-formula. For instance, in the Vertex Cover problem we are given a graph G *and* an integer k and we are asked whether there is a vertex cover of size at most k. MSO does not provide constructs that "speak" about integers. Since k is part of the input, we need different formulas, one for each k — just as in Example 7, where we used a different FO-formula for every k. Similarly, non-decision problems such as counting or optimization problems cannot be defined in MSO at all.

To overcome these limitations, we now present an extended methodology that uses MSO to define properties of feasible solutions, that are then *evaluated* in an appropriate

Problem	*G&J*
Domatic Number for fixed k	GT3
Chromatic Number (k-Colorability) for fixed k	GT4
Achromatic Number for fixed k	GT5
Monochromatic Triangle	GT6
Partition into Triangles	GT11
Partition into Isomorphic Subgraphs for fixed connected H	
Partition into Hamiltonian Subgraphs	GT13
Partition into Forests for fixed k	GT14
Partition into Cliques for fixed k	GT15
Partition into Perfect Matching for fixed k	GT16
Covering by Cliques for fixed k	GT17
Covering by Complete Bipartite Subgraphs for fixed k	GT18
Induced Subgraph with MSO-property P and fixed k	GT21
Induced Connected Subgraph with MSO-property P and fixed k	GT22
Induced Path for fixed k	GT23
Cubic Subgraph	GT32
Hamiltonian Completion for fixed k	GT34
Hamiltonian Cycle	GT37
Hamiltonian Path	GT39
Subgraph Isomorphism for fixed subgraph H	GT48
Graph Contractability for fixed graph H	GT51
Graph Homomorphism for fixed graph H	GT52
Path with Forbidden Pairs for fixed n	GT54
Kernel	GT57
Degree Constrained Spanning Tree for fixed k	
Disjoint Connecting Path for fixed k	ND40
Chordal Graph Completion for fixed k	
Chromatic Index for fixed k	

Table 1: A selection of MSO-definable problems [6, Theorem 3.5] and their number in Garey & Johnson [83]

manner. For a motivating example, recall that we defined the MINIMUM VERTEX COVER optimization problem as follows:

MINIMUM VERTEX COVER	
Input:	A graph $G = (V, E)$
Solutions:	Vertex covers $U \subseteq V$
Goal:	Minimize $\lvert U \rvert$

That is, we specify a set of *feasible solutions* and an optimization goal. Using the $vc(X)$ formula of Example 13 above, we can rephrase this as:

MINIMUM VERTEX COVER	
Input:	A graph $G = (V, E)$
Solutions:	sets $U \subseteq V$ with $G \models vc[X/U]$
Goal:	Minimize $\lvert U \rvert$

Here we use MSO to define feasible solutions, namely those $U \subseteq V$ that satisfy $vc[X/U]$. The optimization goal remains the same.

To formally capture this concept, Arnborg, Lagergren, and Seese [6] introduce *Extended MSO* (EMSO), where MSO is used to define the feasible solutions, and *evaluations* are used to map sets into the domain of integers or reals. Those can then be compared to each other and any number that is part of the input. It is also possible to optimize them against a target function or to count the number of solutions. A similar approach to solve optimization and counting problems can be found in the aforementioned work by Borie, Tovey, and Parker [27], which includes an extension to this kind of problems. We now formally introduce the evaluations as given in [6].

Definition 15 (Evaluation Term, Evaluation Relation)**.** An *evaluation term* over the real variables $y_1, \ldots, y_k$ is a function $t\colon \mathbf{R}^k \to \mathbf{R}$ built using the arithmetic operators $-$, $+$, $\times$, the real numbers, and a subset of the variables $y_1, \ldots, y_k$. Let $\mathcal{T}_{y_1,\ldots,y_k}$ denote the set of all evaluation terms over $y_1, \ldots, y_k$. An *evaluation relation* over variables $y_1, \ldots, y_k$ is a propositional formula over the atomic formulas

$$\mathcal{A}_{y_1,\ldots,y_k} = \{\, t \leq 0, t = 0 \mid t \in \mathcal{T}_{y_1,\ldots,y_k} \,\}$$

Note that $t > 0$ can be expressed as $\neg(t \leq 0)$ and $t \geq 0$ as $t > 0 \vee t = 0$.

Example 16. The following are evaluation relations:

$$y_1 - y_2 \leq 0 \qquad\qquad 3y_1 - y_2 = 0$$

$$2y_1 - 3y_2 + 28 \leq 0 \quad \wedge \quad 2y_3^2 - y_1 > 0$$

For real numbers $r_1, \ldots, r_k$ we let $\alpha(r_1, \ldots, r_k)\colon \mathcal{A}_{y_1,\ldots,y_k} \to \{0, 1\}$ be an assignment to $\mathcal{A}_{y_1,\ldots,y_k}$ such that

$$\alpha(r_1, \ldots, r_k)(\, t(y_1, \ldots, y_k) \leq 0 \,) = 1 \qquad \text{iff} \qquad t(r_1, \ldots, r_k) \leq 0,$$

and

$$\alpha(r_1, \ldots, r_k)(\, t(y_1, \ldots, y_k) = 0\,) = 1 \qquad \text{iff} \qquad t(r_1, \ldots, r_k) = 0,$$

i.e., the function t evaluated at positions $r_1, \ldots, r_k$ is at most zero or equal to zero, respectively. As in Section 3.1, we extend $\alpha(r_1, \ldots, r_k)$ to all propositional formulas over $\mathcal{A}_{y_1,\ldots,y_k}$.

Example 17.

$$\alpha(5,7)(\, y_1 - y_2 \leq 0\,) = 1$$
$$\alpha(389,389)(\, y_1 - y_2 \leq 0\,) = 1$$
$$\alpha(27,15)(\, y_1 - y_2 \leq 0\,) = 0$$
$$\alpha(4,12)(\, 3y_1 - y_2 = 0\,) = 1$$
$$\alpha(3,12)(\, 3y_1 - y_2 = 0\,) = 0$$
$$\alpha(4,12,2)(\, 2y_1 - 3y_2 + 28 \leq 0 \quad \wedge \quad 2y_3^2 - y_1 > 0\,) = 1$$
$$\alpha(3,12,0)(\, 2y_1 - 3y_2 + 28 \leq 0 \quad \wedge \quad 2y_3^2 - y_1 > 0\,) = 0$$

To simplify notation, for $\psi \in \mathcal{A}_{y_1,\ldots,y_k}$ we simply write $\psi(r_1, \ldots, r_k) = \mathit{true}$ to express that $\alpha(r_1, \ldots, r_k)(\psi) = 1$ holds.

We are now ready to define EMSO-definable problems. Roughly speaking, an EMSO-definable problem consists of an MSO-formula and an evaluation relation such that yes-instances are precisely those for which there exists an assignment that satisfies both the formula and the evaluation relation.

Definition 18 (Arnborg, Lagergren, and Seese [6])**.** A graph problem P is called *EMSO-definable* if there is a relational MSO-formula φ with l free set variables $\text{free}(\varphi) = \{X_1, \ldots, X_l\}$, integers $m, t \in \mathbf{N}$, and an evaluation relation ψ over variables $y_1, \ldots, y_{lm+t}$ such that for a graph $G = (V, E)$, weight functions $w_1, \ldots, w_m \colon V \cup E \to \mathbf{R}$ and integers $z_1, \ldots, z_t$ we have:

$(G, w_1, \ldots, w_m, z_1, \ldots, z_t)$ is a "yes"-instance of P if and only if there is $(U_1, \ldots, U_l) \in \text{assignments}(\varphi, G)$ such that

$$G \models \varphi[X_1/U_1, \ldots, X_l/U_l]$$

and

$$\psi\Big(\sum_{u \in U_1} w_1(u), \ldots, \sum_{u \in U_l} w_1(u), \ldots, \sum_{u \in U_1} w_m(u), \ldots, \sum_{u \in U_l} w_m(u), z_1, \ldots, z_t \Big) = \mathit{true}.$$

The integers $z_1, \ldots, z_t$ and the weight functions are part of the input instances and are given as arguments to a multi-variate evaluation relation. This formal definition, particularly the last line, is somewhat cumbersome. The last line simply means that we are applying each of the m weight functions to each element of the l set $U_1, \ldots, U_l$ ($l \cdot m$ arguments), and the t additional arguments correspond to the t integers $z_1, \ldots, z_t$ given as part of the input. Most natural problems have simple and straight-forward evaluation relations. Let us give some examples.

Example 19.

- The Vertex Cover decision problem, where the input is a graph $G = (V, E)$ and an integer k, is EMSO-definable using the MSO-formula $vc(X)$ of Example 13, $m = t = 1$ with $z_1 = k$ and the evaluation relation $\psi = y_1 - y_2 \leq 0$. The necessary weight function $w_1 \colon V \to \mathbf{R}$ of Definition 18 is implicitly given via $w_1(v) := 1$ for all $v \in V$.

- In the Weighted Dominating Set decision problem, we are given a graph $G = (V, E)$, an number k and a weight function $w_1 \colon V \to \mathbf{R}$ and we have to decide whether there is a dominating set $U \subseteq V$ for G that has total weight $\sum_{u \in U} w_1(u) \leq k$.

  This problem is EMSO-definable using the MSO-formula $ds(X)$ of Example 13, $m = t = 1$ with $z_1 = k$ and the evaluation relation $\psi = y_1 - y_2 \leq 0$.

- The 3-Colorability problem is MSO-definable and does not require the EMSO machinery. Suppose therefore that for some reason we are interested in only particular three-colorings, where one color set has exactly two times the *weight* of the second color set. In this problem, we are given a graph $G = (V, E)$ and a weight function $w_1 \colon V \to \mathbf{R}$, and we are to decide whether there is a three-coloring R, G, B of G with $\sum_{r \in R} w_1(r) = 2 \sum_{g \in G} w_1(g)$.

  This problem is EMSO-definable using the MSO-formula $\mathit{3col}'(R, G, B)$ of Example 13, $m = 1$, $t = 0$, and the evaluation relation $\psi = y_1 - 2y_2 = 0$.

We can now extend this concept to optimization problems. For EMSO-definable optimization problems, in addition to an MSO-formula and an evaluation relation, one has an objective function. Given an input, the goal is to find an assignment that not only satisfies the formula and the evaluation relation but maximizes (or minimizes) the objective function.

Definition 20 (Arnborg, Lagergren, and Seese [6])**.** A problem P is an *EMSO-definable optimization problem* if there exists a relational MSO-formula φ with free set variables $\text{free}(\varphi) = \{X_1, \ldots, X_l\}$, integers $m, t \in \mathbf{N}$, real variables $y_1, \ldots y_{lm+t}$, an evaluation term $t(y_1, \ldots, y_{lm+t})$ and an evaluation relation ψ over $y_1, \ldots, y_{lm+t}$ such that P can be expressed as finding the maximum (or minimum) of the evaluation term

$$t\Big(\sum_{u \in U_1} w_1(u), \ldots, \sum_{u \in U_l} w_1(u), \sum_{u \in U_m} w_m(u), \ldots, \sum_{u \in U_m} w_m(u), z_1, \ldots, z_t\Big)$$

over all $(U_1, \ldots, U_l) \in \text{assignments}(\varphi, G)$ subject to

$$G \models \varphi[X_1/U_1, \ldots, X_l/U_l]$$

and

$$\psi\Big(\sum_{u \in U_1} w_1(u), \ldots, \sum_{u \in U_l} w_1(u), \ldots, \sum_{u \in U_1} w_m(u), \ldots, \sum_{u \in U_l} w_m(u), z_1, \ldots, z_t\Big) = \mathit{true}.$$

Here $G = (V, E)$ is a graph, $w_1, \ldots, w_m \colon V \cup E \to \mathbf{R}$ are weight functions and $z_1, \ldots, z_t$ are integers, all of which are part of the input to P.

P is called a *linear* EMSO-definable optimization problem (also called *LinMSO optimization problem*) if $t(y_1, \ldots, y_{lm+t})$ is a linear function in $y_1, \ldots, y_{lm}$ and ψ is always *true*.

Example 21. We give some examples.

- The Minimum Vertex Cover optimization problem is linear EMSO-definable using the MSO-formula $vc(X)$ of Example 13, $m = 1$, $t = 0$, and the task is to minimize the value of the evaluation term $t(y_1) = y_1$. The weight function $w_1 \colon V \to \mathbf{R}$ is implicitly given via $w_1(v) := 1$ for all $v \in V$.

- The Minimum Weighted Dominating Set optimization problem for a graph and a weight function $w_1 \colon V \to \mathbf{R}$ is linear EMSO-definable using the MSO-formula $ds(X)$ of Example 13, $m = 1$, $t = 0$, and the task is to minimize the value of the evaluation term $t(y_1) = y_1$.

- The Traveling Sales Person optimization problem is linear EMSO-definable using the MSO-formula $ham''(F)$ of Example 13, $m = 1$, $t = 0$, and the task is to minimize the value of the evaluation term $t(y_1) = y_1$. Here, the weight function $w_1 \colon E \to \mathbf{R}$ is part of the input.

- Suppose we want to find a three-coloring on a graph with two weight functions $w_1, w_2 \colon V \to \mathbf{R}$ that maximizes the term

$$\left(\sum_{r \in R} w_2(r) \right)^2 - 3 \sum_{b \in B} w_2(b),$$

  but still maintains the property $\sum_{r \in R} w_1(r) = 2 \sum_{g \in G} w_1(g)$ of the previous example.

  This problem is EMSO-definable using the MSO-formula $3col'(R, G, B)$ of Example 13, $m = 2$, $t = 0$, and the evaluation relation $\psi = y_1 - 2y_2 = 0$. The task is to maximize the value of the evaluation term $t(y_1, y_2, y_3, y_4, y_5, y_6) = y_4^2 - 3y_6$.

Finally, Arnborg et al. [6] consider *counting problems*: For a fixed relational MSO-formula φ with free set variables $\mathrm{free}(\varphi) = \{X_1, \ldots, X_l\}$ and an input graph $G = (V, E)$, one has to compute the number of solutions $|\mathrm{sat}(\varphi, G)|$.

A large number of classical and important graph optimization problems are EMSO-definable or even linear EMSO-definable.

Theorem 22 (Arnborg, Lagergren, and Seese [6])**.** *The problems listed in Tables 2 and 3 are linear EMSO-definable optimization problems.*

Theorem 23 (Arnborg, Lagergren, and Seese [6])**.** *The problems listed in Table 4 are EMSO-definable problems.*

Problem	*G&J*
Minimum Vertex Cover	GT1
Minimum Dominating Set	GT2
Minimum Feedback Vertex Set	GT7
Partial Feedback Edge Set for fixed maximum cycle length l	GT9
Minimum Maximal Matching	GT10
Partition into Cliques	GT15
Maximum Clique	GT19
Maximum Independent Set	GT20
Induced Subgraph with MSO-property P	GT21
Induced Connected Subgraph with MSO-property P	GT22
Induced Path	GT23
Balanced Complete Bipartite Subgraph	GT24
Bipartite Subgraph	GT25
Degree-bounded Connected Subgraph for fixed d	GT26
Planar Subgraph	GT27
Transitive Subgraph	GT29
Uniconnected Subgraph	GT30
Minimum k-connected Subgraph for fixed k	GT31
Hamiltonian Completion	GT34
Multiple Choice Matching for fixed J	GT55
k-closure	GT58
Path Distinguishers	GT60
Maximum Leaf Spanning Tree	ND2
Minimum Edge/Vertex Deletion for any MSO-property P [162]	
Unit Weight Steiner Tree	
Unit Weight Longest Path	
Unit Weight Longest Cycle	

Table 2: A selection of linear EMSO-definable optimization problems with weight functions bounded by a constant [6, Theorem 3.6] and their number in Garey & Johnson.

Problem	*G&J*
Minimum Weighted Vertex Cover	
Minimum Weighted Dominating Set	
Maximum Weighted Independent Set	
Bounded Diameter Spanning Tree for fixed D	ND4
Steiner Tree	ND12
Maximum Cut	ND16
Longest Cycle	ND28
Longest Path	ND29

Table 3: A selection of linear EMSO-definable optimization problems with integer valued weight functions [6, Theorem 3.7] and their number in Garey & Johnson.

Problem	*G&J*
Partition into Isomorphic Subgraphs for fixed H	GT12
Partition into Perfect Matchings	GT16
kth Best Spanning Tree for fixed k	ND9
Bounded Component Spanning Forest for fixed k	ND10
Minimum Cut into Bounded Sets	ND17
Shortest Weight-Constrained Path	ND30
kth Shortest Path for fixed k	ND31

Table 4: A selection of EMSO-definable problems [6, Theorem 3.8] and their number in Garey & Johnson.

3.3.3. MSO and Semiring Homomorphisms

The EMSO-machinery extends over plain MSO by allowing numerical input parameters, numerical evaluations and the definition of optimization and counting problems. Yet, there are several "non-numerical" problems that do not fit into this framework. For example, there are problems where one has to compute a list of tuples of vertices that satisfy certain constraints. One may also consider multiple optimization goals at once to output a list of Pareto-optimal solutions. The machinery considered by Courcelle and Mosbah [48] extends Courcelle's Theorem to this kind of problems.

To introduce this machinery, we first observe that a graph problem P can also be understood as the task of computing the value of $g_P(G)$ for an input graph G. Following the terminology of [48], g_P is an *evaluation* that maps the graph G into an appropriate domain. For instance, we usually have $g_P \colon G \to \{true, false\}$ for decision problems, $g_P \colon G \to \mathbf{R} \cup \{-\infty, +\infty\}$ for optimization problems, or $g_P \colon G \to \mathbf{N}$ for counting problems. However, we may also have much different evaluations, as in, say,

$$g_P \colon G \mapsto \{(v_{1,1}, \ldots, v_{1,l}), \ldots, (v_{s,1}, \ldots, v_{s,l})\},$$

which maps the graph $G = (V, E)$ into a set of l-tuples of vertices $v_{i,j} \in V$, or $g_P \colon G \mapsto 2^{\mathbf{R} \times \mathbf{R}}$, which might map a graph G into a set of Pareto-optimal solution sizes.

As we are concerned with MSO-definable problems in this survey, we are interested in those evaluations for which we have

$$g(G) = h(\mathrm{sat}(\varphi, G)),$$

where φ is the MSO-formula, $\mathrm{sat}(\varphi, G)$ is the set of satisfying assignments of φ on G, and h is some suitable function. For example, in the case of linear EMSO optimization problems the solutions $(U_1, U_2, \ldots, U_k) \in \mathrm{sat}(\varphi, G)$ are mapped into the integers or real numbers via the evaluation term, and among all such numbers the optimal value *opt* is chosen. Hence, in this case, $g_P(\mathrm{sat}(\varphi, G)) := opt$.

In many cases, g_P maps graphs into the universe of some *semiring*, e.g., $\mathbf{N}$ or $\mathbf{R}$ with the standard addition and multiplication operations. It was shown by Courcelle and Mosbah in [48] that, if h resembles an homomorphism into such an semiring, then $g_P(G) = h(\mathrm{sat}(\varphi, G))$ can be computed efficiently on graphs of bounded treewidth.

This is a powerful machinery as it applies to a rich class of problems that cannot be captured in EMSO. In order to define it formally, we first need a couple of further definitions.

Definition 24 (Semiring)**.** A *semiring* is a tuple $R = (U_R, \oplus, \otimes, \hat{0}, \hat{1})$, where U_R is a non-empty set called the *universe* of R, and $\oplus$ and $\otimes$ are the binary *addition* and *multiplication* operations with the following properties:

1. $(U_R, \oplus, \hat{0})$ is a commutative monoid with neutral element $\hat{0}$ (zero): for all $a, b, c \in U_R$,
   (a) $(a \oplus b) \oplus c = a \oplus (b \oplus c)$
   (b) $a \oplus \hat{0} = \hat{0} \oplus a = a$

   (c) $a \oplus b = b \oplus a$.
2. $(U_R, \otimes, \hat{1})$ is a monoid with neutral element $\hat{1}$ (one): for all $a, b, c \in U_R$,
   (a) $a \otimes (b \otimes c) = (a \otimes b) \otimes c$
   (b) $a \otimes \hat{1} = \hat{1} \otimes a = a$.
3. multiplication distributes over addition: for all $a, b, c \in U_R$,
   (a) $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$
   (b) $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$.
4. $\hat{0}$ annihilates under $\otimes$, i.e., for all $u \in U_R$ it holds: $\hat{0} \otimes u = u \otimes \hat{0} = \hat{0}$.

Example 25. We list a few natural semirings frequently used for graph problems:

- Boolean := $(\{\mathit{false}, \mathit{true}\}, \vee, \wedge, \mathit{false}, \mathit{true})$ for decision problems.
- Counting := $(\mathbf{N}, +, \cdot, 0, 1)$, i.e., the natural numbers with the standard addition and multiplication.
- MinCard := $(\mathbf{N} \cup \{\infty\}, \min, +, \infty, 0)$ for optimization problems. In this ring the $+$ is the *multiplication* with neutral element 0.

  Note that the conditions of semirings are indeed satisfied: Both operations provide monoids over $\mathbf{N}$, $\mathbf{N}$ is annihilated by ∞ under the multiplication $+$, since $a + \infty = \infty$ for all $a \in \mathbf{N}$, and for any three integers $a, b, c \in \mathbf{N}$ we have $a + \min\{b, c\} = \min\{a + b, a + c\}$, i.e., $+$ distributes over min.

  More generally, MinWeight := $(\mathbf{R} \cup \{\infty\}, \min, +, \infty, 0)$ can be used for optimization problems with real weights.

Example 25 gives some prominent examples of semirings commonly used. For instance, MinCard and MinWeight are often implicitly used when finding solutions for minimization problems. For a comprehensive list of algorithmically useful semirings see [48, Section 4].

We now define the semiring that corresponds to the domain of the mapping h to be used. Note that the domain of h contains *sets* of assignments, since we are interested in the value of $h(\mathrm{sat}(\varphi, G))$, where $\mathrm{sat}(\varphi, G) \subseteq \mathrm{assignments}(\varphi, G)$ is the set of satisfying assignments. The universe of the ring is therefore the power set of $\mathrm{assignments}(\varphi, G)$. We define the semiring

$$\mathrm{assignring}(\varphi, G) := (A, \cup, \hat{\cup}, \emptyset, \hat{\emptyset}),$$

where $A = 2^{\mathrm{assignments}(\varphi, G)}$. The addition $\cup$ in the semiring is the standard union of sets with the neutral element $\emptyset$. This is because if $A, B \subseteq S$ are two sets of assignments, then also $A \cup B$ is a set of assignments. The multiplication in this ring is the following operation $\hat{\cup}$. Given two tuples $(U_1, \ldots, U_l)$ and $(U'_1, \ldots, U'_l)$ with $U_1, \ldots, U_l, U'_1, \ldots, U'_l \subseteq V \cup E$, we first let

$$(U_1, \ldots, U_l) \,\hat{\cup}\, (U'_1, \ldots, U'_l) := (U_1 \cup U'_1, \ldots, U_l \cup U'_l).$$

That is, $a \hat{\cup} b$ for two l-tuples a, b is an element-wise (set) union. We extend this to *sets* of l-tuples A, B via a Cartesian product of the form

$$A \hat{\cup} B := \{\, a \hat{\cup} b \mid a \in A \wedge b \in B \,\}.$$

The neutral element of $\hat{\cup}$ is $\hat{\emptyset} = \{(\emptyset, \ldots, \emptyset)\}$, i.e., a set that contains a single tuple, whose elements are the empty set. Note that indeed the empty set (the neutral element of the addition) annihilates w.r.t. the multiplication $\hat{\cup}$, because taking the Cartesian product with an empty set results in the empty set.

The intention behind the $\hat{\cup}$-operation becomes apparent once we return to the field of *decomposable* graphs, where a graph G can recursively be decomposed into subgraphs G_1 and G_2. One can, roughly speaking, combine assignments for G_1 with assignments for G_2 in an Cartesian style manner, i.e., one can inductively compute the set of satisfying assignments for the input graph G from those of its subgraphs. In fact, the main result of [48] states that for many mappings h the value of $h(\mathrm{sat}(\varphi, G))$ can be computed in the same inductive manner on decomposable structures as it can be done [37] for $\mathrm{sat}(\varphi, G)$ [48, Theorem 2.10].

Definition 26. An *homomorphism* from a semiring $(U_1, \oplus_1, \otimes_1, \hat{0}_1, \hat{1}_1)$ into a semiring $(U_2, \oplus_2, \otimes_2, \hat{0}_2, \hat{1}_2)$ is a map $h\colon U_1 \to U_2$ such that for all $a, b \in U_1$ it holds:

- $h(a \oplus_1 b) = h(a) \oplus_2 h(b)$;
- $h(a \otimes_1 b) = h(a) \otimes_2 h(b)$; and
- $h(\hat{1}_1) = \hat{1}_2$.

Let G be a graph and φ an MSO-formula. We call two sets $A, B \subseteq \mathrm{assignments}(\varphi, G)$ *separated* if $U_i \cap U'_j = \emptyset$ for all $(U_1, \ldots, U_l) \in A$, for all $(U'_1, \ldots, U'_l) \in B$ and all $1 \leq i, j \leq l$.

A *weak homomorphism* from $\mathrm{assignring}(\varphi, G)$ into a semiring $(U_R, \oplus, \otimes, \hat{0}, \hat{1})$ is a map $h\colon 2^{\mathrm{assignments}(\varphi, G)} \to U_R$ such that for all $A, B \subseteq \mathrm{assignments}(\varphi, G)$ it holds:

- $h(A \cup B) = h(A) \oplus h(B)$ if $A \cap B = \emptyset$;
- $h(A \hat{\cup} B) = h(A) \otimes h(B)$ if A and B are separated;
- $h(\emptyset) = \hat{0}$; and
- $h(\hat{\emptyset}) = \hat{1}$.

We can now define MSO-evaluation problems:

Definition 27 (cf. Courcelle, Mosbah [48])**.** A graph problem P is an *MSO-evaluation problem* if there is a relational MSO-formula φ and a semiring $R = (U_R, \oplus, \otimes, \hat{0}, \hat{1})$ such that P can be stated as computing $h(\mathrm{sat}(\varphi, G)) \in U_R$, where a graph $G = (V, E)$ and the weak semiring homomorphism h between $\mathrm{assignring}(\varphi, G)$ and R are part of the input.

Remark 1. A few words on the input size are in order. The domain of h is huge compared to G, since it is the power set of the set of l-tuples over subsets of V and E. Fortunately, h is a weak semiring homomorphism, i.e., we have $h(\bar{U}_1 \cup \bar{U}_2) = h(\bar{U}_1) \oplus h(\bar{U}_2)$ for disjoint sets of assignments $\bar{U}_1, \bar{U}_2 \subseteq \text{assignments}(\varphi, G)$. We can therefore recursively split a set $\bar{U} \subseteq \text{assignments}(\varphi, G)$ with $|\bar{U}| > 1$ into disjoint subsets $\bar{U} = \bar{U}_1 \cup \bar{U}_2$ until eventually both have size 1. The value of $h(\bar{U})$ for arbitrary $\bar{U} \subseteq \text{assignments}(\varphi, G)$ can then be computed from the values $h(\bar{U}')$ of the *singleton sets* $\bar{U}' \subseteq \text{assignments}(\varphi, G)$ with $|\bar{U}'| = 1$ by a number of applications of the addition operation $\oplus$ in the target semiring. However, there still is an exponential number of singleton sets $\bar{U}' \subseteq \text{assignments}(\varphi, G)$, since each element in $\bar{U}'$ is a tuple of subsets of $V \cup E$. For these, we can use the same recursive decomposition using the multiplication $\hat{\cup}$, since $\{(U_1, \ldots, U_l)\}$ with $|U_i| > 1$ for some $1 \leq i \leq l$ can recursively be separated as

$$\{(U_1, \ldots, U_l)\} = \{(U_1', \ldots, U_l')\} \,\hat{\cup}\, \{(U_1'', \ldots, U_l'')\},$$

until all the sets U_i have size at most 1. This leaves us with only $O(|V \cup E|^l)$ different singleton sets, for which we need to provide the value of h as an input to the algorithm. Using a number of addition and multiplication operations in the semiring, we can then compute $h(\bar{U})$ for arbitrary $\bar{U} \subseteq \text{assignments}(\varphi, G)$.

The concept of MSO-evaluation problems and semirings will be much clearer with some concrete examples.

Example 28. Consider again the formula $\varphi = vc(X)$ expressing that the set X is a vertex cover for the input graph G. We shall now define several variants of the classical Vertex Cover problem by simply considering different semirings and suitable weak homomorphisms h. Note that by Remark 1 the weak homomorphism h is sufficiently described by defining its value on the elementary elements $\{(U)\}$, where $U \subseteq V(G)$ with $|U| \leq 1$.

1. Let $h_1 \colon 2^{\text{assignments}(\varphi,G)} \to \text{MinCard}$ such that h_1 maps a set A of assignments to the minimum size of all assignments in A, and to ∞ if A is empty. More formally, let $h_1(A) = \min\{\, |U| \mid (U) \in A \,\}$ and $h_1(\emptyset) = \infty$. We verify that h_1 is a weak homomorphism from assignring(φ, G) to MinCard:
   - $h_1(\emptyset) = \infty$ is the zero in MinCard.
   - $h_1(\hat{\emptyset}) = h_1(\{(\emptyset)\}) = \min\{|\emptyset|\} = 0$ is the one in MinCard.
   - For $A, B \subseteq \text{assignments}(\varphi, G)$, we have

$$\begin{aligned} h_1(A \cup B) &= \min\{\, |U| \mid (U) \in A \cup B \,\} \\ &= \min\Big\{ \min\{\, |U| \mid (U) \in A \,\}, \min\{\, |U| \mid (U) \in B \,\} \Big\} \\ &= \min\{h_1(A), h_1(B)\}. \end{aligned}$$

- For separated $A, B \subseteq \text{assignments}(\varphi, G)$, we have

$$\begin{aligned} h_1(A \mathbin{\hat\cup} B) &= \min\{\, |U| \bigm| (U) \in A \mathbin{\hat\cup} B \,\} \\ &= \min\{\, |U_A \cup U_B| \bigm| (U_A) \in A, (U_B) \in B \,\} \\ &= \min\{\, |U_A| + |U_B| \bigm| (U_A) \in A, (U_B) \in B \,\} \\ &= \min\{\, |U_A| \bigm| (U_A) \in A \,\} + \min\{\, |U_B| \bigm| (U_B) \in B \,\} \\ &= h_1(A) + h_1(B). \end{aligned}$$

Since $h_1(\text{sat}(\varphi, G)) = \min\{\, |U| \bigm| (G, \alpha[X/U] \models \varphi) \,\}$, where α is the empty assignment, we have that $h_1(\text{sat}(\varphi, G))$ is the size of a minimum vertex cover for G. The Minimum Vertex Cover optimization problem is therefore an MSO-evaluation problem.

2. More generally, for a weight function $w\colon V \to \mathbf{R}$, let $h_2\colon 2^{\text{assignments}(\varphi,G)} \to \text{MinWeight}$ such that h_2 maps a set A of assignments to the minimum *weight* of all assignments in A.
   The Minimum Weighted Vertex Cover optimization problem, where $w\colon V \to \mathbf{R}$ is part of the input, can be stated as computing $h_2(\text{sat}(\varphi, G))$, where $h_2\colon \{(u)\} \mapsto w(u)$, $\emptyset \mapsto \infty$, is part of the input (cf., Remark 1).
3. Let $h_3\colon 2^{\text{assignments}(\varphi,G)} \to \text{Counting}$ such that $h_3(A) = |A|$ is the number of assignments in A. Then h_3 is a weak homomorphism from $\text{assignring}(\varphi, G)$ to Counting:
   - $h_3(\emptyset) = 0$ is the zero in Counting.
   - $h_3(\hat\emptyset) = |\{(\emptyset)\}| = 1$ is the one in Counting.
   - For disjoint $A, B \subseteq \text{assignments}(\varphi, G)$, we have

$$h_3(A \cup B) = |A \cup B| = |A| + |B| = h_3(A) + h_3(B).$$

   - For separated $A, B \subseteq \text{assignments}(\varphi, G)$, we have

$$\begin{aligned} h_3(A \mathbin{\hat\cup} B) &= |A \mathbin{\hat\cup} B| = \bigl|\{\, a \mathbin{\hat\cup} b \mid a \in A, b \in B \,\}\bigr| \\ &= |A| \cdot |B| \qquad \text{(since } A \text{ and } B \text{ are separated)} \\ &= h_3(A) \cdot h_3(B) \end{aligned}$$

   Since $h_3(\text{sat}(\varphi, G))$ is the number of solutions to the MSO-formula φ in G, the #P-hard #Vertex Cover counting problem [178] is an MSO-evaluation problem.
4. For a more sophisticated example, let's assume we want to compute at once, for each size k, the *number* of vertex covers of size k. For, we first define the following semiring:

$$\text{CardCounting} := (U_R, \oplus, \otimes, \hat 0, \hat 1),$$

   where elements of $U_R = \mathbf{N} \times \mathbf{N} \times \cdots$ are infinite sequences of natural numbers. An entry $(n_i)_{i\geq 0} = (n_0, n_1, \ldots) \in U_R$ means there are n_0 solutions of size 0, n_1 solutions of size 1 and so forth. The addition $\oplus$ is the element-wise addition defined via

$$(n_i)_{i\geq 0} \oplus (n'_i)_{i\geq 0} := (n_i + n'_i)_{i\geq 0}$$

with neutral element $\hat{0} = (0, 0, \ldots)$ (all zeros). The multiplication $\otimes$ is defined via

$$(n_i)_{i\geq 0} \otimes (n'_i)_{i\geq 0} := \Big(\sum_{\substack{a,b\geq 0 \\ a+b=i}} n_a \cdot n'_b \Big)_{i\geq 0}$$

with neutral element $\hat{1} = (1, 0, 0, \ldots)$. We routinely confirm that the semiring properties are fulfilled; we particularly note the annihilation of $\otimes$ under $\hat{0}$.
Now let $h_4 \colon 2^{\text{assignments}(\varphi,G)} \to \text{CardCounting}$ be the weak homomorphism such that $h_4(\emptyset) = \hat{0}$ and for all $(U) \in \text{assignments}(\varphi, G)$ we have

$$h_4(\{(U)\}) = (n_i)_{i\geq 0} \quad \text{with} \quad n_i = \begin{cases} 1 & i = |U| \\ 0 & i \neq |U| \end{cases}$$

Computing $h_4(\text{sat}(\varphi, G)) = (n_i)_{i\geq 0}$ then yields the desired list $(n_i)_{i\geq 0}$ of numbers n_i, where an entry n_i is the number of vertex covers of G of size i. Note that we have $n_i = 0$ for $i > |V|$, and hence $(n_i)_{i\geq 0}$ has a finite and small description.

This example demonstrates the versatility of the MSO-evaluations approach, where a single fixed formula $vc(X)$ defining the vertex cover property of sets X can be used to define several variants of the Vertex Cover problem by simply changing the homomorphism used. Courcelle and Mosbah [48] remark that the homomorphism machinery has previously been used in [103], but there without the backing of the general MSO framework. They also write that the homomorphism-approach can be seen as a syntactic characterization of the regular sets of Borie, Tovey, and Parker [27]. We mention again that the homomorphism approach is orthogonal to the EMSO approach in the sense that there are problems that can be expressed as an MSO-evaluation but are not EMSO-definable and vice versa. For example, the cardinality counting problem in Example 28-4 is not EMSO-definable.

3.3.4. MSO Model-Checking

As in the case of FO, we are now concerned with the following problem: We are given an MSO-formula φ, a graph $G = (V, E)$ and an assignment to the free variables of the formula α, and we are asked whether $G \models \varphi[\alpha]$. This is known as the MSO Model-Checking problem:

MSO Model-Checking	
Input:	An MSO-formula φ, a graph $G = (V, E)$, and an assignment α to the free variables of φ
Question:	Does $G \models \varphi[\alpha]$?

Since MSO contains all of FO, this problem is PSPACE-hard (cf., Section 3.2.1). By a slight modification of Algorithm 1, depicted in Algorithm 2, we also get that MSO Model-Checking is in PSPACE. Let again $k = \|\varphi\|$ be the length of a suitable encoding of the input formula and $n = |V|$ the size of the graph. Algorithm 2 uses polynomial

Algorithm 2 MSO-Model Checking Algorithm $A(\varphi, G, \alpha)$

Input: An MSO-formula φ, a graph $G = (V, E)$, and an assignment α to the free variables of φ

if $\varphi = t_1 = t_2$ **then return** 1 iff $\alpha(t_1) = \alpha(t_2)$ and 0 otherwise
if $\varphi = t \in X$ **then return** 1 iff $\alpha(t) \in \alpha(X)$ and 0 otherwise
if $\varphi = \mathrm{adj}(t_1, t_2)$ **then return** 1 iff $\{\alpha(t_1), \alpha(t_2)\} \in E$ and 0 otherwise
if $\varphi = \neg\psi$ **then return** $1 - A(\psi, G, \alpha)$
if $\varphi = (\varphi_1 \vee \cdots \vee \varphi_k)$ **then**
    return 1 iff there is $1 \leq i \leq k$ with $A(\varphi_i, G, \alpha) = 1$ and 0 otherwise
if $\varphi = (\varphi_1 \wedge \cdots \wedge \varphi_k)$ **then**
    return 0 iff there is $1 \leq i \leq k$ with $A(\varphi_i, G, \alpha) = 0$ and 1 otherwise
if $\varphi = \exists x\psi$ and $\mathrm{tp}(x) = 1$ **then**
    return 1 iff there is $v \in V$ with $A(\psi, G, \alpha[x/v]) = 1$ and 0 otherwise
if $\varphi = \forall x\psi$ and $\mathrm{tp}(x) = 1$ **then**
    return 0 iff there is $v \in V$ with $A(\psi, G, \alpha[x/v]) = 0$ and 1 otherwise
if $\varphi = \exists x\psi$ and $\mathrm{tp}(x) = 2$ **then**
    return 1 iff there is $e \in E$ with $A(\psi, G, \alpha[x/e]) = 1$ and 0 otherwise
if $\varphi = \forall x\psi$ and $\mathrm{tp}(x) = 2$ **then**
    return 0 iff there is $e \in E$ with $A(\psi, G, \alpha[x/e]) = 0$ and 1 otherwise
if $\varphi = \exists X\psi$ and $\mathrm{tp}(x) = 1$ **then**
    return 1 iff there is $U \subseteq V$ with $A(\psi, G, \alpha[X/U]) = 1$ and 0 otherwise
if $\varphi = \forall X\psi$ and $\mathrm{tp}(x) = 1$ **then**
    return 0 iff there is $U \subseteq V$ with $A(\psi, G, \alpha[X/U]) = 0$ and 1 otherwise
if $\varphi = \exists X\psi$ and $\mathrm{tp}(x) = 2$ **then**
    return 1 iff there is $F \subseteq E$ with $A(\psi, G, \alpha[X/F]) = 1$ and 0 otherwise
if $\varphi = \forall X\psi$ and $\mathrm{tp}(x) = 2$ **then**
    return 0 iff there is $F \subseteq E$ with $A(\psi, G, \alpha[X/F]) = 0$ and 1 otherwise

space since the recursion depth is bounded by k and the assignment α can be encoded with $O(kn^2)$ bits. The algorithm therefore needs polynomial space only. However, if the formula contains q nested set quantifiers the algorithm requires more than 2^{qn} recursive calls, and hence it is clearly not feasible for practical applications on any non-trivial instance.

A simple modification of Algorithm 2 can be used to solve most of the EMSO-definable problems and MSO-evaluations in polynomial space. For example, if we are to solve an EMSO-definable problem, we simply enumerate all assignments in the way Algorithm 2 iterates through sets for quantifiers $\forall X$ and $\exists X$, and then checks whether the evaluation relation is *true*. For optimization problems, we simply optimize over all such solutions, and counting problems can equally easy be solved. Hence, most of these problems are in PSPACE as well. Note, however, that MSO-evaluations can be used to output, say, the list $\mathrm{sat}(\varphi, G)$ of all satisfying assignments, whose representation is *not* polynomial in the input size.

Consider now the case that the MSO-formula is fixed. This is a natural assumption from an algorithmically focused viewpoint since (E)MSO-definable problems and MSO-evaluations use fixed MSO-formulas. For every fixed MSO-formula φ we define the following problem.

φ-MSO Model-Checking	
Input:	A graph $G = (V, E)$, and an assignment α to the free variables of φ
Question:	Does $G \models \varphi[\alpha]$?

For example, if $\varphi = \textit{3col}$ of Example 13, then *3col*-MSO Model-Checking is simply the standard 3-Colorability problem, and *ham*-MSO Model-Checking is the classical Hamiltonian Cycle problem. Since these are NP-complete, there is little hope that the φ-MSO Model-Checking problem can be solved in polynomial time on general graphs. On the other hand, a well-known result by Doner [54] and Thatcher and Wright [172] states that the φ-MSO Model-Checking problem can be solved in *linear time* on labeled binary trees. The proofs actually show that a tree-language L is regular if and only if it is MSO-definable, and therefore one can construct a finite-state tree automaton that recognizes the language L. The construction of the tree automaton depends on the formula φ only, which is fixed, and can hence done in constant time. The simulation of a run of the automaton on the input tree T takes time linear in the number of nodes in T. MSO-model-checking is therefore computationally easy on trees. As we will see in the next section, the linear-time automata approach for labeled trees can be lifted to tree-*like* graphs — more precisely, to graphs of bounded treewidth.

4. Courcelle's Theorem for Treewidth

As we have already seen in the introduction, Courcelle's Theorem unified a large number of results on efficient algorithms for tree-decomposable graphs. We restate the theorem and its extensions for easy reference.

Theorem 29 (Courcelle's Theorem [37, Proposition 4.14])**.** *Let P be an MSO problem defined by an MSO-formula φ and let w be a positive integer. There is an algorithm A and a function $f\colon \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ such that for every graph $G = (V, E)$ of order $n := |V|$ and treewidth at most w, A solves P on input G in time $f(\|\varphi\|, w) \cdot n$, where $\|\varphi\|$ is the length of φ.*

The function $f(\|\varphi\|, w)$ is an iterated exponential of height $\Theta(\|\varphi\|)$, that is,

$$f(\|\varphi\|, w) = 2^{2^{\cdot^{\cdot^{\cdot^{2^w}}}}} \Big\} \Theta(\|\varphi\|).$$

Furthermore, Frick and Grohe have shown that unless P = NP, the function f cannot be upper bounded by an iterated exponential of bounded height in terms of φ

and w [76]. Note that for a fixed MSO-definable graph property Π and constant treewidth w, Courcelle's Theorem states that membership in Π can be decided in linear time, since $f(\|\varphi\|, w)$ is a constant "hidden" in the $O(n)$ of Theorem 1.

This result has been generalized to EMSO-definable problems by Arnborg, Lagergren, and Seese in 1991 and to MSO-evaluations (with homomorphisms) by Courcelle and Mosbah in 1993.

Theorem 30 (Arnborg et al. [6]). *Let P be an EMSO-definable problem or an EMSO-definable optimization problem with weight functions bounded by a constant, and $w \in \mathbf{N}$ an integer. Then one can solve P on graphs $G = (V, E)$ of order $n := |V|$ and treewidth at most w in time $O(f_P(w) \cdot n)$.*

Theorem 31 (Arnborg et al. [6]). *Let P be an EMSO-definable counting problem or an EMSO-definable optimization problem with integer-valued weight functions, and $w \in \mathbf{N}$ an integer. Then one can solve P on graphs $G = (V, E)$ of order $n := |V|$ and treewidth at most w in time $O(f_P(w) \cdot poly(n))$.*

Theorem 32 (Courcelle and Mosbah [48]). *Let P be an MSO-evaluation problem. Then one can solve P on graphs $G = (V, E)$ of order $n := |V|$ and treewidth at most w in time $O(f_P(w) \cdot poly(n))$.*

We remark that the polynomial time increase for the integer-valued EMSO-definable problems stems from the time required to process numbers of unbounded size, which requires time $\omega(1)$. For *linear* EMSO-definable optimization problems one can improve this to time $O(n \log n)$ in the logarithmic cost measure and to time $O(n)$ in the uniform cost measure, since here arithmetic operations take constant time. See the discussion in [6]. Similarly, many MSO-evaluations of practical interest can actually be solved in time $O(n)$ or time linear *in the input size* plus *the output size*. For example, the list of numbers n_k of vertex covers of size k as in Example 28-4 requires a representation of size $\omega(n)$, since the n numbers can range from 0 to 2^n. It can, however, be computed in time linear in the input size plus the resulting output size. In [48, Section 4], the complexity of several practically important MSO-evaluations is discussed.

From the theorems above, we immediately get the following result.

Corollary 1. *On graphs of bounded treewidth,*

1. *the problems depicted in Tables 1 and 2 can be solved in linear time.*
2. *the problems depicted in Tables 3 and 4 can be solved in polynomial time.*
3. *the problems depicted in Table 3 can be solved in linear time in the uniform cost measure.*

Furthermore, it has been shown that the linear time requirement in Courcelle's Theorem can be replaced by logarithmic space.

Theorem 33 (Elberfeld, Jakoby, and Tantau [62]). *Let Π be an MSO-definable graph property and $w \in \mathbf{N}$ an integer. Then one can decide membership in Π for input graphs $G = (V, E)$ of order $n := |V|$ and treewidth at most w in space $O(\log n)$.*

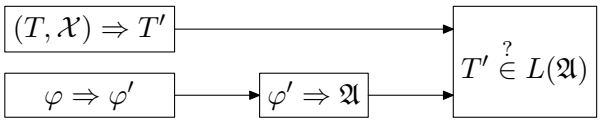


Figure 4: The proof. Left column: MSO-interpretation of G in T' and the corresponding conversion of the formula. Middle column: MSO-to-FTA conversion. Right column: Automata run.

Theorem 33 can be extended to counting problems [62, 63].

The true beauty of the MSO based approach lies in the fact that it suffices to define the *graph problem* (for general graphs) in MSO, and yet in many cases no further work has to be spent on the details of how the problem can actually be solved.

In particular, Courcelle's Theorem states that there is at least one algorithm that can solve it in linear time for graphs of bounded treewidth. In practice, however, it can be left to the implementation to choose the actual strategy that is used to solve the particular MSO Model-Checking instance at hand. For example, the polynomial space Algorithm 2 is very fast for small inputs and the approaches presented in the following exploit the *decomposability* of the input graphs.

The versatility that lies within the separation of problem definition and concrete solving makes the MSO approach a powerful method in the algorithmist's toolbox.

4.1. Proving Courcelle's Theorem

The literature is rich in different techniques to prove Courcelle's Theorem. It is safe to say that the standard method to prove Courcelle's Theorem is a reduction of the φ-MSO Model-Checking problem for graphs of bounded treewidth to the φ'-MSO Model-Checking for labeled binary trees. The formula φ' can be easily obtained from φ by applying a number of somewhat technical, but nevertheless straight-forward rewriting rules. In the world of logic, such a reduction is called an *interpretation of theories* [150, 33]. It is well-known [172, 54] that the latter problem can be solved in linear time by constructing a suitable finite-state tree automaton (FTA). The advantage is that the underlying methods (MSO-to-FTA conversion) and the MSO interpretation techniques are well understood and widely covered in the literature.

An accessible exposition of this proof can be found in Kreutzer's survey on algorithmic meta-theorems [119]. For readers who are also interested in details on the MSO-to-FTA conversion for binary trees we can recommend the detailed and self-contained exposition in [70, Chapters 10 and 11].

Since the literature is rich in proofs of Courcelle's Theorem, we shall only give a brief description of this reduction method. For simplicity, we prove it for MSO-definable graph properties only, i.e., MSO sentences without free variables, but the extension to multiple free variables, and hence to EMSO-definable problems or MSO-evaluations with homomorphisms is not hard, see, e.g., the original proof for EMSO-problems in [6], or [44, Section 6.3].

The overall strategy is depicted in Figure 4 and explained in the following. First fix an MSO-sentence φ and an integer $w \in \mathbf{N}$. For an input graph $G = (V, E)$ of treewidth

at most w, we first need to obtain a tree-decomposition $(T, \mathcal{X})$ of width w. This tree-decomposition can be obtained in time $O(n)$ using Bodlaender's Algorithm [19]. Next, we identify the tree-decomposition with a labeled binary tree T' over a fixed alphabet Σ_w. This alphabet is quite large but has bounded size. At a node $v \in V(T)$, a label of the binary tree encodes, for instance, which vertices in X_v are adjacent. The resulting labeled binary tree T' completely describes the input graph G. In other words, we have found a way to *interpret* the graph G in a labeled binary tree T' with labels from Σ_w. From the MSO-interpretation theory we therefore get that we can convert φ into a formula φ' such that $G \models \varphi$ if and only if $T' \models \varphi'$. This conversion can easily be performed by a computer program in constant time (since φ is fixed). Once we have done this, we can use the standard MSO-to-FTA conversion due to Doner–Thatcher–Wright to construct a finite-state tree automaton $\mathfrak{A}$ with $T' \in L(\mathfrak{A})$ iff $T' \models \varphi$.

This concludes the proof since a run of the automaton $\mathfrak{A}$ on T' can be simulated in linear time. We also mention in this context [69], where much care has been taken to prove how the linear running for this automata simulation can in fact be obtained. Similarly, it is shown in [62] that Bodlaender's Algorithm as well as the simulation of the run of the automaton can be done in logarithmic space, yielding Theorem 33.

4.1.1. Alternative Proofs of Courcelle's Theorem

There are several alternative ways to prove Courcelle's Theorem. From a broad perspective, the overall "spirit" of the proofs is remarkably similar: at a high level, all algorithms rely on a finite table-lookup strategy that resembles the finite-state automata approach. One might therefore argue that, essentially, all known proofs of Courcelle's Theorem explicitly or implicitly emulate the automata-theoretic approach. However, the low-level proof details have a huge impact w.r.t. practical applications, as we will see in Section 5. We therefore briefly survey other proof techniques.

The technique we outlined above, i.e., a reduction to the classical model checking problem for MSO on labeled trees, has been described by many authors, see, e.g., [6, 74, 69, 70, 119]. A direct and explicit construction of the tree automata from the original formula φ, i.e., one that avoids the interpretability machinery of the previous section, is described in, e.g., [183] or [44, Section 6.3].

Courcelle's original paper proves [37, Proposition 4.14] the existence of such an automaton, cf. [37, Propositions 4.3 and 1.6], but the construction is not immediate. Similarly, in [2, 55] a Myhill-Nerode type argument is used to show that the treewidth parse tree operators admit a right congruence with finitely many congruence classes, i.e., the tree language is finite-state (regular) and can be recognized by an FTA. The method of *test sets* can be used to construct the tree automata of [37, 2, 55]. See, e.g., [55, Section 6.1] for details.

A model-theoretic variant to prove Courcelle's Theorem is based on the Feferman–Vaught Theorem [67] (also called *Splitting Theorem*), which can be extended to MSO, cf., [37, 94, 133]. The Splitting Theorem essentially states that if a graph G can be decomposed into two subgraphs G_1 and G_2, then from the input formula φ one can construct a *reduction sequence* of finitely many MSO-formulas that holds in G_1 and G_2

if and only if φ holds in G, cf., [37, 44, 94, 133]. These new MSO-formulas have the same quantifier rank as φ. Since there are only finitely many formulas of quantifier rank q, eventually these formulas must repeat in the process (but their actual number is huge!). By induction, one can therefore use dynamic programming on the tree decomposition to compute the *q-theory* of G, i.e., set of formulas of quantifier rank at most q that hold in G (cf., [74, 92, 133]). See [92] for an accessible exposition.

Similarly, Courcelle and Mosbah [48] show how a combination of the Splitting Theorem [37] and a dynamic programming algorithm can be used to compute the set of satisfying assignments for the input formula [48]: First, one traverses the tree decomposition of the input graph top-down and applies the Splitting Theorem at every single node, which yields a reduction sequence of finitely many MSO-formulas. In the leaves, we evaluate these formulas, and then use bottom-up dynamic programming to finally evaluate whether the formula φ is true on G.

Gottlob, Pichler, and Wei [88] show that the set of satisfying assignments of φ on G can be described in *monadic Datalog*, and this description can effectively be obtained from the MSO input formula, see [88, Theorem 4.6]. Their proof does not rely on the Splitting Theorem but is based on Ehrenfeucht-Fraïssé games [88, Lemmas 3.5-3.7]. The monadic Datalog approach of [88] has also been studied in [72, 73]. We will revisit this approach and its practical utility in Section 5.5.

Another approach [116] is based on model-checking games [100, 89, 90]. This game-theoretic approach does not rely on automata or the Splitting Theorem. Rather, it is essentially a variant of Algorithm 2 that uses dynamic programming on the tree-decomposition to compute the result. It is shown in [116] that the number of entries in the dynamic programming algorithm's tables is bounded by a constant, which then yields the desired linear running time. We will cover this approach in Section 5.6.

4.2. On hidden Constants

Unfortunately, the function f in the running time bound given in Theorem 29 grows extremely fast. That is, the constants hidden in the O-notation for the running time bound given in Theorem 1 are very large. MSO logic has huge expressive power in that already very short formulas can be used to express hard problems. Indeed, the size of the automaton for an MSO-formula in terms of the formula cannot be bounded by an elementary function [169, 151]. Frick and Grohe prove in [76] non-elementary worst-case lower bounds for the multiplicative constants in the linear running time. A function $f\colon \mathbf{C}\times\cdots\times\mathbf{C}\to\mathbf{C}$ is *elementary* if it can be written in a "closed" form over the complex numbers, the elementary functors $-$, $+$, $\times$ and a constant number of exponentials and logarithms.

Theorem 34 (Frick, Grohe [76])**.** *Assume that* P $\neq$ NP*. Then there is no algorithm that, given an MSO sentence φ and a tree T decides whether $T\models\varphi$ in time $f(\|\varphi\|)n^{O(1)}$, where f is an elementary function and $n=|V(T)|$.*

This latter theorem makes it impossible to prove any efficient running time bounds for Courcelle's Theorem, and no general algorithm can asymptotically be better than the

automata approach. Note, however, that the proof of Theorem 34 uses a reduction from an NP-hard problem and therefore the constructed MSO-formulas are not very natural. More precisely, they will most probably not appear in typical practical applications. For particular problems, say Minimum Vertex Cover, Minimum Dominating Set, or 3-Colorability, one can indeed show that these problems can be solved much faster on graphs of bounded treewidth, as stated in the introduction. Typical problems that admit efficient algorithms for graphs of bounded treewidth, including these three problems, admit formulas with few nested quantifiers. In his PhD thesis, Weyer [183] studies the growth of the constants in the O-notation in terms of the number of nested quantifier alternations of the input formula. The upper and lower bounds obtained [183, Chapter 6] provide a good explanation for the actual complexity of problems observed in practical experiments. Nevertheless, the MSO Model-Checking problem remains problematic in practical applications even for formulas of small or moderate quantifier rank. This will be covered in the next section.

5. Courcelle's Theorem in Practice

Courcelle and Engelfriet mention in [44, p. 504] that MSO model-checking for decomposable graphs (e.g., [37, 47]) has been cited in different fields of applied computer science, such as, e.g., computational biology [130], computational linguistics [111], database querying [9], logistics [140], telecommunications [137], quantum computing [179], or the aforementioned [174] on register allocation for compilers, to name a few. It is therefore of great interested to make Courcelle's Theorem useful in practice.

For weak second-order logic of two successors (WS2S), which for finite graphs coincides with MSO, there is a powerful and established software called MONA [114]. MONA was developed over the course of many years by Klarlund, Møller, and Schwartzbach and essentially implements the Doner–Thatcher–Wright MSO-to-FTA conversion. It contains many tricks and improvements such as formula reductions, *guided* tree automata, eager minimization, BDD-based automata representations, or cache-conscious data structures [115]. It has found a wide practical application in different areas such as hardware and program verification or natural languages, see [115] and the references therein.

Nevertheless, using the reduction procedure outlined in Section 4.1 to generate input instances for MONA does not yield practically usable algorithms for the MSO Model-Checking problem. In this section we survey not only the advances but also the negative results in this research area.

5.1. Not implemented or no results known

We are not aware of any implementations of Courcelle's Theorem based on the Splitting Theorem approach. The generation of all possible reduction sequences for MSO-formulas "obviously is not practical" [133, Section 1.6], as their number grows too fast with the quantifier rank. The algorithms presented in [74, 133] are therefore infeasible in practice. However, from [48] we get that computing the particular reduction sequence

for the input formula φ suffices. Some lower bounds are known for the necessary conversions into disjunctions [139], but it would still be interesting to see how this approach behaves in practice. We consider this as an interesting direction for future research.

It is mentioned in [55] that a Myhill–Nerode based program implementing the method of test sets has been developed as part of an M.Sc. thesis, which unfortunately does not seem to be available any more. The aforementioned thesis of Sloper [166] describes an implementation of the method of test sets with practical applications in mind, but it is presented for *word* automata only. We are not aware of any extensions to tree automata and corresponding experiments for decomposable graphs.

5.2. Negative results and problems.

We start with reporting on negative results, because they will help us understand design decisions that led to progress and faster implementations.

It turns out that the major limiting factor in practical applications of Courcelle's Theorem are the space (memory) requirements of known algorithms. Recall that the majority of the known algorithms for bounded treewidth graphs use the tree-decomposition as a guide for dynamic programming. For most problems, an exponential or even super-exponential number of entries have to be stored in the dynamic programming table. This includes the majority of known, specialized algorithms for specific problems. For instance, the $\Theta(2^w n)$-time algorithm for Minimum Vertex Cover [7] requires space $\Theta(2^w)$, where w is the width of a given tree decomposition for the input graph. Similarly, the algorithms for 3-Colorability [7] and Minimum Dominating Set [181] use tables with $\Theta(3^w)$ entries, and the algorithms based on the Cut-and-Count technique [51] use tables of size $\Theta(c^w)$, where the constant c depends on the particular problem. Due to the nature of these algorithms, the table entries are accessed very frequently. More precisely, *all* of them are accessed at least once for every node of the tree decomposition. Swapping out entries from memory to slower second-tier storage (e.g., hard disks or SSDs) is therefore usually not an option as the huge performance penalty on I/O to the second-tier storage renders the known algorithms useless in practice. We would therefore consider it a major improvement to find an algorithm for, say, Minimum Vertex Cover with a running time of $O(c^w poly(n))$ for, say, $c < 10$, and subexponential space $O(2^{o(w)})$.

Space requirements are also the major problem for *all* proof techniques we outlined in the previous section, particularly for the automata theoretic approach. As a somewhat surprising fact, space is even a problem for the *logspace* algorithm of [62]. For practical applications the main goal is therefore to make the space requirements feasible. In fact, many of the "implementation secrets" for MONA [115] are concerned with memory requirements, such as the usage of a BDD representation or of so-called *guides* that can be used as additional hints for the automaton in order to decrease automata size.

In the automata based approach, every quantifier alternation ($\forall\exists$ or $\exists\forall$) in the formula induces a *power set construction* during the construction of the finite-state tree automaton. This is because for a formula $\forall x\varphi$, the FTA construction described by Doner–Thatcher–Wright first constructs the non-deterministic automaton $\mathfrak{A}$ for the formula $\exists x\neg\varphi$, and then converts it into the complement automaton $\mathfrak{A}'$ that recognizes the

complement language $L(\mathfrak{A}')$ of $L(\mathfrak{A})$ (note that $\forall x\varphi = \neg\exists x\neg\varphi$). The complementation is typically preceded with *determinization* using a power set construction, because for deterministic automata we can simply invert the set of accepting states. Each such power set construction may yield an deterministic automaton that has size *exponential* in the size of the previous non-deterministic automaton. A formula with k quantifier-alternations can therefore require a deterministic automaton, whose size is a k-times iterated exponential. It is well-known [169, 151] that this blow-up is not avoidable. Furthermore, as a consequence of the Grohe and Frick lower-bound [76] (see Section 4.2), *no* general algorithm can perform better than the automata approach.

In practice, of course, not all resulting automata are that large. However, even when the final deterministic automaton is "small" and fits into memory, the power set construction of an intermediate non-deterministic automaton might be prohibitively expensive. Unfortunately, this is in fact observed in practical experiments. The space required to construct the automata, both using MONA and with direct constructions, causes major problems in applications [112, 167, 88, 136].

In his thesis [167], Soguet systematically studied the sizes of the finite-state tree automata corresponding to various problems for graphs of small *clique-width*.[1] The automata were generated with MONA. Among the problems considered are properties that can be decided in polynomial time even on general graphs, and properties that are NP-hard on general graphs but polynomial-time solvable in the considered graph classes. The following is a list of some of the problems considered by Soguet in his thesis:

$\Delta(G) \leq i$?	Is the maximum degree $\Delta(G)$ of G at most $i \in \{3, 4\}$?
x, y connected?	Are the two vertices x and y connected by a path in G?
Connected	Is the graph connected?
2-Disjoint Paths	Are there two disjoint paths in G that connect, respectively, the vertices s_1 and t_1 as well as the vertices s_2 and t_2?
$vc'(X)$	The set X is a vertex cover for G (cf., Example 13).
$\mathit{clique}(X)$	The set X is a clique in G.
i-Clique	Does G contain a clique of size at least $i \in \{3, 4\}$?
i-Vertex Cover	Does G have a vertex cover of size at most $i \in \{3, 4\}$?
Bipartite	Is G bipartite (two-colorable)?
3-Colorability	Is G three-colorable?

Table 5 shows the sizes of the finite-state tree automata Soguet constructed for graphs of clique-width 2 and 3. We list here the sizes of the BDD representations for the

[1] Like treewidth, clique-width is a width measure for decomposable graphs, but the tree automata are easier to construct; the results do also shed light on the automata-approach for graphs of bounded treewidth. See Section 6 for more details on clique-width and the MSO Model-Checking problem for graphs of bounded clique-width.

Problem	*Complexity in general graphs*	$cw = 2$		$cw = 3$	
		states	*nodes*	*states*	*nodes*
$\Delta(G) \leq 3$?	$O(n)$	91	1341	*fail*	
$\Delta(G) \leq 4$?	$O(n)$	231	4178	*fail*	
x, y connected?	$O(n+m)$	26	444	*fail*	
Connected	$O(n+m)$	11	96	*fail*	
2-Disjoint Paths	$O(n^2)$ [113]	*fail*		*fail*	
$clique(X)$	$O(n^2)$	5	22	9	84
$vc'(X)$	$O(n^2)$	5	30	9	155
3-Clique	$O(n^3)$	11	81	41	1510
4-Clique	$O(n^4)$	21	189	153	8673
3-Vertex Cover	$O(n^4)$	63	635	414	23845
4-Vertex Cover	$O(n^5)$	11	1223	1037	75700
Bipartite	$O(n+m)$	11	81	57	2934
3-Colorability	NP-complete	21	189	*fail*	

Table 5: Sizes of the finite-state tree automata generated by Soguet [167, Figures 4.8, 4.9] with MONA for graphs of clique-width 2 and 3. *States* is the size of the automata, *nodes* are the number of nodes in the corresponding Binary Decision Diagrams (BDDs). The entry *fail* means the computation aborted with "out-of-memory" errors.

adjacency relation entitled "Adj1" in Soguet's thesis.[2] As one observes, the sizes of the resulting automata are surprisingly small. Particularly the automata for the properties $clique(X)$ and $vc'(X)$ of the EMSO-definable optimization problems Maximum Clique and Minimum Vertex Cover are very small (5 vs. 9 states). On the other hand the automata for some properties that are trivially solvable even in general graphs cannot be constructed anymore. Further inspection of the MSO-formulas corresponding to the problems considered by Soguet reveals that the culprit is probably a large number of *nested* quantifiers that are required to *count* in MSO, or that are required to express that a set is connected (cf., Example 13). Note that in order to count, say, four distinct neighbors, one requires four nested quantifiers.

Similar numbers were obtained by Durand with her `Autowrite` software tool [56, 57] as reported in [40, 44].

Motivated by applications in knowledge representation and reasoning, Gottlob, Pichler, and Wei also used MONA for the MSO to FTA-conversion and report the same state-

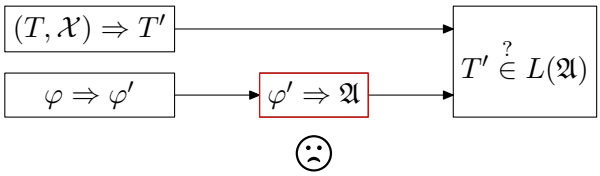
[2]For graphs of clique-width 2, Soguet also studied two alternative BDD representations for the adjacency relation and lists their (positive) impact on the size of the resulting automata. Numbers for graphs of clique-width 3 are not reported. For the sake of better comparability we only quote the numbers for the first representation here.

explosion problems described above [88, 87]. They also investigate in [86, 87] reasons why MONA experiences these problems:

> *An analysis of the various components of our program has revealed that MONA is the weak point of the program. In fact, the way how MONA evaluates an MSO-formula φ^* over a tree T^* is very problematical (and, in a sense, contradicts the spirit of model checking). [...] MONA undoubtedly has its merits in other areas, notably in verification. However, with its current strategy of considering the input structure as part of the formula (and, therefore, mixing up data complexity and query complexity), MONA is not suited for the KR & R problems studied here.* [86] (and similarly in [87, p. 127])

Consequently, they also tried a direct implementation of the MSO-to-FTA conversion, but the state explosion problems "led to failure before we were able to feed any input data to the program" [88, p. 4].

Therefore, at least three independent implementations (MONA, `Autowrite` and the direct construction of Gottlob et al.) reveal that the size of the resulting automata poses a problem in practical applications, which even advanced tools such as MONA or `Autowrite` cannot easily handle. We therefore conclude:



Finally, we note that even the logspace-algorithm of [62] might face these problems, since the size of the automaton is considered a constant in their work; the logarithmic space improvement mainly concerns Bodlaender's algorithm for constructing the tree decomposition and the simulation of the run of the automaton in logarithmic space, all of which are independent of the problematic automata construction.

In what follows, we will give some high-level, but nevertheless detailed exposure of the current advances in making Courcelle's Theorem more useful in practical applications.

5.3. Precomputed and Nondeterministic Automata

Since the power set construction for intermediate automata is problematic, Courcelle and Durand [41, 42, 43, 40] (see also [44, Section 6.3]) suggest the following combined approach:

1. Avoid the power set construction for *determinization* by avoiding quantifier alternation and allowing *non-deterministic* FTAs, runs of which can easily be simulated.
2. Reduce the *quantifier nesting depth* of the formula by providing additional atomic predicates expressing commonly used, powerful properties such as connectivity or subset relations.

3. *Precompile* these provided predicates into *small* deterministic automata to avoid large intermediate automata.

It is a standard-technique to simulate runs of a non-deterministic automata by keeping track of the states the automata *possibly* is in. Since in each run the automaton $\mathfrak{A}$ must be in one of the $|\mathfrak{A}|$ states, this simulation algorithm therefore only needs to handle at most $|\mathfrak{A}|$ states at once. This technique naturally entails a small increase of the running time, but the space required is typically much less compared to what is needed to construct the deterministic automaton in the first place. Note, for example, that the power set construction does require $\Theta(2^{|\mathfrak{A}|})$ space, and this blow-up is not avoidable since in general the size of the minimal *deterministic* automaton $\mathfrak{A}'$ with $L(\mathfrak{A}) = L(\mathfrak{A}')$ is $|\mathfrak{A}'| = \Theta(2^{|\mathfrak{A}|})$.

The simulation of non-deterministic automata naturally extends to standard operations on languages recognized by automata, particularly the intersection (for formulas with $\wedge$), the union (for formulas with $\vee$), and *projection*, a technique used to construct automata for formulas with existential quantification. Complementation, however, which is required for formulas with negation and, therefore, also universal quantification, cannot be approached with this non-deterministic simulation technique.

Unfortunately, in most cases one cannot avoid negation or universal quantification to express basic properties. The workaround proposed by Courcelle and Durand is to provide a large list of atomic formulas for commonly used properties. One example for such a basic property is $\mathit{Path}(X, Y)$ expressing that X has exactly two vertices that are linked by a path in $G[Y]$. Another example are *Boolean Set Terms* which can be used to express properties of sets of sets that require many quantifiers but typically yield small automata, such as $X \subseteq Y$ for two set variables X, Y or $\mathit{Partition}(X_1, \ldots, X_m)$ expressing that $(X_1, \ldots, X_m)$ is a partition of the vertex set. See [44, pp. 471ff] for a large list of predicates for which small precompiled automata can be provided and how they are constructed.

It can be shown [40, 44], however, that the minimal deterministic automaton deciding the Connected problem via clique-width expressions has more than $2^{2^{k/2}}$ states, where k is the clique-width of the graphs considered. The construction presented in [40, 44] does yield an automaton with $2^{2^{O(k)}}$ states. Table 6.12 in [44, p. 474] lists several examples with a similar flavor of growth. Courcelle and Durand [40] describe another technique of using special tree *annotations* of the clique-width parse tree and how they can be used to further shrink the size of the automata. For example, a non-deterministic automaton for annotated inputs for the Connected problem has only $2^{O(k \log k)}$ states.

Since universal quantification poses problems due to the required negation, Courcelle and Durand consider an *existential* fragment of MSO that contains only formulas of the form $\exists X_1 \cdots \exists X_p \varphi$, where φ is a Boolean combination of atomic formulas with free set variables in $\{X_1, \ldots, X_p\}$. The atomic formulas are from a large set of predicates for which precomputed automata are provided by the implementation. This fragment of MSO is rich enough to express many important graph problems such as graph partition problems like 3-Colorability (and, more general, k-Colorability) or whether a

fixed graph H is a minor of the input graph.

Their approach has been implemented on top of the `Autowrite` software developed by Durand [56, 57], which is written in Common Lisp and implements bottom-up tree-automata (term-automata) and most of the standard operations on automata such as union, intersection, determinization, minimization, and complementation. By a language extension, this implementation is furthermore able to solve decision problems such as Vertex Cover or Dominating Set, where the solution size is part of the input [40, Figure 4].

Several experimental results with the implementation reported in [42, 41, 40, 44] reveal that using the techniques described in this section do in some extend help to construct the automata. Unfortunately, with growing clique-width, the construction of the automata still soon causes memory problems and can no longer be done in practice. An approach to conquer this problem is described in the next section.

5.4. On-the-fly Construction of Automata

In conclusion of the previous section, the construction of the complete automaton is not feasible in practice since the huge number of states is beyond what we can handle on real hardware.

What can be done? Clearly, some representation of the automaton is required in order to simulate runs of the automaton. The natural question is if one actually needs to keep a representation of the complete automaton in memory in order to simulate a run on a given input. This leads to the idea to not construct the automaton before computation, but rather construct the automaton on the fly as needed, while simulating the run of the automaton.

Early work for graphs of bounded treewidth utilizing this idea was reported at a Dagstuhl seminar [184] in 2001.[3] White reports that the *test set* for the Hamiltonian Cycle problem on graphs of bounded treewidth contains $2^w w!$ elements, and that the naive method to construct the corresponding tree automaton using the method of test sets then yields an intermediate automaton with $2^{2^w w!}$ states (which should then be minimized). Recall in this context that the power set construction also has exponential space requirements for the intermediate automata. Walker then proceeds to show that one can *dynamically* construct the automaton on-the-fly, which avoids the exponential blow-up and yields an algorithm for Hamiltonian Cycle with a running time of only $O(2^w w! n)$. Such an on-the-fly construction of automata via the method of test sets is also described in the Master's thesis of Sloper [166], but we are not aware of experiments for tree automata or graph problems.

Courcelle and Durand have extended their work described in the previous section to the on-the-fly construction of the corresponding automata [40]. The idea of their approach is that the transitions are no longer stored explicitly in precomputed form, say in a table, but are rather represented implicitly as a small set of "meta-rules" from which an algorithm can compute the transitions as needed. The transitions are therefore

[3] Unfortunately this work has never been published in other form.

computed *on-the-fly* while simulating the run of the automaton. The time required to compute transitions is naturally larger than a simple table-lookup, but the space requirements shrink significantly: Only a suitable representation of the current state has to be kept in memory, which typically is much less than the space required to construct the complete automaton.

Courcelle and Durand call these automata *fly-automata* as opposed to the classic, precomputed automata that they call *table automata*. Fly-automata have been implemented in `Autowrite`, and promising experiments with this implementation are presented in [40]. For example, they compare the running times of table automata with fly-automata and analyze the penalty on the evaluation function vs. the simpler table-lookup for transitions. For paths P_n on n vertices (clique-width 3 if $n \geq 4$) they compare the computation times of both automata types for increasing values of n for the Connected problem. The penalty of fly- versus table automata on these instances is an approx. factor 4 in the running time [40, Figure 5], which is surprisingly low considering the significantly smaller memory footprint and the improved feasibility for larger clique-widths. The experiments furthermore show that the computation time is indeed roughly linear with respect to n (as expected).

They also ran experiments for the 3-Colorability problem on grid graphs. While bipartite grid graphs are trivially three-colorable, such graphs have the advantage that it is easy to construct the corresponding graph decompositions (tree decomposition or clique-width decomposition) of arbitrary width and graph sizes. On $6 \times n$ grids (clique-width 8 if $n \geq 6$) the running time does in fact increase linearly. They report a running time of approx. $5000s$ on a 6×100 grid, of $12000s$ on a 6×600 grid, and of approx. $18000s$ for a 6×1000 grid [40, Figure 9]. On the contrary, the classic table-automaton could not be constructed for these graphs.

Several further experiments of the same spirit in [40] indicate feasibility of the fly-automata approach. At the moment, the biggest limitation seems to be a lack of good algorithms or heuristics that compute small clique-width decompositions for arbitrary input graphs. They therefore conclude:

> *We did not reach any limitation using fly-automata which we tried up to cwd* $= 18$*. We could run the automata on terms representing terms on any graph we had a term representation for. Our problem right now is to find big graphs with their clique-decomposition in order to perform tests.* [42]

However, we note that the techniques of [40] can be adapted for graphs of bounded tree-width, where several suitable heuristics are known. Such an extension is already planned by Courcelle and Durand [40, p. 405], and would be of tremendous value for practical applications.

5.5. Reduction to Monadic Datalog

A completely different approach to conquer the space problems with the FTA-to-MSO conversion is proposed by Gottlob, Pichler, and Wei [88, 87, 149]. They describe an automatic translation of the MSO-formula into a set of monadic Datalog predicates.

A Datalog program is a built from function-free, definite Horn clauses and consists of a set of *facts* and a set of *rules* (see, e.g., [1, 31, 177]). Monadic Datalog is the fragment of Datalog where certain predicates are required to be unary. (Monadic) Datalog has the advantage that due to its applications in database theory it is well-studied from a practical viewpoint and there are several fast implementations (e.g., the DLV system [129]) that use sophisticated optimization techniques, see, e.g., those mentioned in [88, Section 6.3] and the references therein.

It was previously known that on trees monadic Datalog has the same expressive power as MSO [85]. Gottlob et al. show [88] how an MSO-formula for a graph G can be translated into a monadic Datalog program for the tree decomposition of the input graph. This conversion builds upon the well-known fact that an MSO-formula of bounded quantifier rank q cannot distinguish two graphs iff the *duplicator* has a winning-strategy in the *Ehrenfeucht-Fraïssé game* over q rounds (see, e.g., [60]). One can therefore define an equivalence relation $\equiv_q^{MSO}$ on graphs as follows: For two graphs G_1 and G_2, we let $G_1 \equiv_q^{MSO} G_2$ if and only if for every MSO-formula φ of quantifier rank at most q we have $G_1 \models \varphi \Leftrightarrow G_2 \models \varphi$, i.e., if no MSO-formula with bounded quantifier rank can *distinguish* G_1 and G_2. The index of $\equiv_q^{MSO}$ is finite but extremely large, namely growing non-elementary in q. The authors then essentially show that each equivalence class of $\equiv_q^{MSO}$ can be described by a set of rules in monadic Datalog. This results in a monadic Datalog program that can be used to check whether the original input formula holds in the input graph. The extension to counting, enumeration and optimization problems is possible [149, 72, 73].

In the worst-case, this conversion must result in a Datalog program that has (super-) exponential size in the original MSO-formula — recall the lower bounds for the hidden constants discussed in Section 4.2. In fact, an algorithm that does implement the construction described in [88] would be infeasible in practice even for very small values of q, since there are simply too many equivalence classes that have to be described as Datalog rules. However, the general approach outlined in the proof can serve as an inspiration for the *manual* construction of monadic Datalog programs for specific problems. This manual adaption has been done for several problems, including the 3-Colorability graph problem, for the Primality problem for relational schemas or the classical SAT problem for CNF-formulas.

The resulting programs are much smaller than the space required to construct the corresponding tree automata, and the available highly optimized Datalog solvers can be used to solve the problem on specific input instances. Several experiments have been carried out by Gottlob et al. and other authors. The results are very promising and show that the monadic Datalog approach is indeed feasible for practical applications.

For example, in [88], the authors report on experiments for the Primality problem of relational schemas (testing if some attribute in a relational schema is part of a key). We list their results [88, Table I] in Table 6. The monadic Datalog approach clearly outperforms the MSO-to-FTA translation approach backed by MONA by an order of magnitude, while maintaining the linear running time in the number of tree nodes. MONA, in turn, soon failed early with "out-of-memory" errors. Very similar numbers

treewidth	*#attr.*	*#func. deps.*	*#tree nodes*	*Mon. Datalog*	*MONA*
3	3	1	3	0.1	650
3	6	2	12	0.2	9210
3	9	3	21	0.4	17930
3	12	4	34	0.5	*fail*
3	21	7	69	0.8	*fail*
3	33	11	105	1.0	*fail*
3	45	15	141	1.2	*fail*
3	57	19	193	1.6	*fail*
3	69	23	229	1.8	*fail*
3	81	27	265	1.9	*fail*
3	93	31	301	2.2	*fail*

Table 6: Processing times in "milliseconds [sic]" for the monadic Datalog approach vs. MONA for the PRIMALITY problem [88, Table I].

treewidth	*#Vars*	*#Clauses*	*#tree nodes*	*Mon. Datalog*	*MiniSat*
3	5	9	24	0.04	< 10
3	31	48	195	0.2	< 10
3	347	522	2157	0.8	< 10
3	3955	5934	23793	8.1	< 10
3	32765	49149	207438	65.5	40
3	337859	506790	2120361	643.9	430

Table 7: Processing times in "milliseconds" [sic] of the monadic Datalog approach vs. MiniSat (with a granularity of 10ms) for the SAT problem [87, Table 6].

are presented in [87] for the SOLVABILITY problem.

In [87], they furthermore compare their monadic Datalog approach to the well-known open-source SAT solver MiniSat [61] for the classical SAT problem. The treewidth of an CNF-formula is defined as the treewidth of the graph that contains vertices for clauses and variables, and edges between a clause C and a variable x iff x occurs in C. We list their results [87, Table 6] in Table 7. It is quite remarkable that the rather generic monadic Datalog approach achieves running times that are comparable to those of an highly optimized SAT solver that was specifically tuned to solve SAT instances.

Their approach has furthermore been applied to a Σ_2^p-complete problem in the area of answer set programming by Jakl, Pichler, and Woltran [108] and scaled well up to treewidth 7 and 1000 nodes in the tree decomposition. Their experiments reveal, too, that for low treewidth the monadic Datalog approach is competitive compared to state of-the-art systems tuned to solve such problems (in this case, DLV [129]). Another set of experiments is reported in [107], where the monadic Datalog approach was able to solve input instances of treewidth up to 10.

At the moment the largest drawback of the monadic Datalog approach is the lack of a feasible *automatic* translation of an MSO-formula into the corresponding monadic Datalog program. In all of the cases above, the monadic Datalog programs have been created manually by the authors of the papers and directly encode the *dynamic programming strategy* to solve the problem on the tree decomposition. Consequently, for the manual construction of the monadic Datalog program one also needs to manually come up with an inductive description of the problem at hand, i.e., with a dynamic programming algorithm for the problem. In particular, additional manual work is required to prove that this formulation is indeed correct. This requires some decent background in dynamic programming algorithms for decomposable graphs and further proofs, see, e.g., the proof for the monadic Datalog program for the PRIMALITY problem in [88, p. 34–42]. Hence, the versatility of MSO that lies in the separation of problem definition and concrete solving is lost if we consider monadic Datalog programs that describe an inductive solution strategy for tree decompositions.

Given the promising experimental results for the manually constructed formulas, it therefore would be of tremendous utility to have an feasible algorithm that given an MSO-formula automatically generates a small monadic Datalog program corresponding to the input formula, which can then be fed into one of the fast Datalog solvers. One can actually observe that the manually constructed monadic Datalog programs do somewhat resemble the MSO-formulas obtained by applying the Splitting Theorem to the MSO-formula as in the top-down approach of Courcelle and Mosbah [48]. For example, the monadic Datalog rules for the 3-COLORABILITY problem in [88, Figure 5] essentially express that if the input graph G decomposes into subgraphs G_1 and G_2, then a partition R, G, B of the vertex set is a three-coloring for G if and only if $(R \cap V(G_i), G \cap V(G_i), B \cap V(G_i))$ is a three coloring for G_i, where $i = 1, 2$. This matches what we get from applying the Splitting Theorem to the formula expressing 3-COLORABILITY. We therefore believe that an algorithm inspired by [48] can be used to construct the desired monadic Datalog programs. We consider this an interesting

direction for future work with powerful applications.

5.6. A Game-theoretic Approach

Another different approach is proposed by the authors of this survey [116, 125]. It can be understood as a dynamic programming variant of Algorithm 2 and does not explicitly construct a tree automaton either. It is called *game-theoretic* because one can understand the recursive call tree of Algorithm 2 as the *game tree* (see [10]) of a two-player pebble game called the *model-checking game*, cf. [100, 89, 90]. It is sometimes convenient to use such a game characterization, because several useful properties and concepts of transition systems and pebble games, particularly bisimulation or winning strategies, have been studied independently and are generally well-understood, see, e.g., [8]. Recall, for instance, that the proof of the monadic Datalog approach [88] was based on the Ehrenfeucht-Fraïssé game, which is closely related to the model-checking game. In what follows, we shall avoid the game-theoretic characterization in order to keep the presentation clean from further definitions.

As mentioned in Section 3.3.4, on general graphs the running time of Algorithm 2 is not feasible for practical applications, since each set quantification requires an enumeration of the 2^n subsets. However, it was shown in [116] that one can use dynamic programming on the tree decomposition to simulate runs of Algorithm 2 in a way that for graphs of bounded treewidth the total running time remains linear in n.

This is achieved by introducing a third possible return value to Algorithm 2 that we may call "*don't know*". This return value essentially means that the currently considered subgraph does not provide enough information to decide whether $G \models \varphi[\alpha]$ (*true*) or whether $G \not\models \varphi[\alpha]$ (*false*) on the input graph. A dynamic programming approach is then used to eventually turn all *don't know* states into the desired *true* or *false* output. For, the algorithm stores all recursive calls that returned *don't know* in a large table, and revisits each such recursive call at every node of the tree decomposition, until eventually all *don't know*'s become *true* or *false*.

For bounded treewidth and a fixed formula, there is only a constant number of non-equivalent *don't know* calls that have to be stored, which is the reason why this approach is also able to obtain linear running time. Again, this number can in general not bounded by an elementary function, which of course cannot be avoided due to the lower bounds discussed in Section 4.2.

However, in practice the actual number of entries that have to be stored is typically much smaller and also heavily depends on the input graph. The game-theoretic approach works for all of MSO, and extensions to EMSO-problems [6] as well as MSO evaluation problems [48] are considerably straight-forward. The proof in [116] is for linear EMSO optimization problems.

The game-theoretic approach has been implemented in C++. We evaluated its practical utility experimentally [125] and compared it to the commercial ILP solver CPLEX [50]. In many cases CPLEX is much faster, as one would expect from a highly optimized commercial optimization package. CPLEX easily outperformed our approach on small to medium sized instances and sparse graphs. On the other hand, our approach was faster

than CPLEX for, e.g., the Minimum Dominating Set problem on medium to large sized grid-like graphs. This particularly stems from the fact that the running time of the tree-width based approach does scale linearly with the graph size as stated by Courcelle's Theorem, which typically is not the case for a ILP-based methods.

Furthermore, the MSO-based approach can show its full strength for problems that require to model "global" constraints such as a global connectivity constraint as in the connected version of the Minimum Dominating Set problem. Connectivity of the solution set can be easily expressed in MSO, cf., Example 13, but is non-trivial to model in an ILP. For example, the ILP-formulation of [141] for Minimum Connected Dominating Set guarantees connectivity of the solution by requiring a flow between the nodes of the solution, and is quite tedious to generate.

In the case of the Minimum Connected Dominating Set problem, our implementation could significantly outperform CPLEX on graphs of small treewidth. For instance, on a large input instance (673 vertices, 1445 edges, treewidth ≤ 8) that was constructed from the Hannover urban railway network from the data available in the OpenStreetMap project[4], an optimal connected dominating set of size 319 was found by our MSO solver in about 3761 seconds and with 299 MB of memory usage. On the same instance, we stopped CPLEX after 20945 seconds real time computation. At that point, the best integer feasible solution found so far was 358. Our MSO solver has been improved further since and can now solve the same instance in about 45 seconds. Interested readers can try it out online on the project homepage [126].

6. Beyond Treewidth

Treewidth is a width measure for sparse graphs in the sense that graphs of bounded treewidth have only a linear number of edges: Recall from Section 2 that for a graph $G = (V, E)$ of treewidth k, we have $|E| \leq k|V| + k^2$. For the algorithms presented in the previous sections, dense graphs with unbounded treewidth do imply a severe, typically exponential or even super-exponential, increase in the running time and, worse, space requirements. Note on the other hand, that many of the aforementioned problems are easy on complete graphs in the same way they are easy on trees. For example, Minimum Vertex Cover, Minimum Dominating Set, and 3-Colorability are trivially solved on such instances. This naturally bears the question when problems become easy for dense graphs.

In what follows, we briefly survey width measures that remain small for several dense graphs and their utility regarding the MSO Model-Checking problem. We also list some recent results on the general complexity of the MSO approach on graphs of bounded treewidth and related graph classes.

[4] `http://www.openstreetmap.org/browse/relation/54023`

6.1. Width Measures for Dense Graphs

It has been shown by Courcelle, Makowsky, and Rotics [47] that a theorem similar to Courcelle's Theorem for bounded treewidth can also be shown for graphs of bounded clique-width. Clique-width, like treewidth, is a width measure for graphs, but in contrast to treewidth, it is also small for several classes of dense graphs. This extension to dense graphs, however, comes at a price: The set of problems to which the result applies is strictly smaller than the set of problems for treewidth (MSO_1-definable vs. MSO_2-definable).

Two other width measures have since been introduced that aim to provide algorithmic utility for dense graphs: Rank-width and Boolean width. On undirected graphs, all three measures are somewhat related and particularly within bounds of each other, i.e., if one of them is bounded on some graph class, all of them are. The result of [47] therefore directly extends to graphs of bounded rank-width and Boolean width.

Hliněný, Oum, Seese, and Gottlob [102] survey several width measures and their algorithmic implications, particularly including clique-width and rank-width (but not Boolean width, which has been defined afterwards). We shall therefore only briefly cover width measures for dense graphs in the following, with a focus on the practical aspects of the MSO Model-Checking problem for these graph classes.

6.1.1. Clique-width

We mentioned clique-width several times before, in particular since the experimental evaluation of the automata approach by Soguet [167] (Section 5.2) and the fly-automata approach by Courcelle and Durand (Section 5.4) have been implemented for the clique-width measure.

The notion of clique-width has been introduced by Courcelle, Engelfriet, and Rozenberg [45] in terms of a graph grammar. Both, treewidth and clique-width, can be defined in terms of graph grammars, *hyperedge replacement grammars* for treewidth, and *vertex replacement grammars* for clique-width, cf. [44, Chapter 2] for details.

To define clique-width, we consider a set of graph operations on labeled graphs. A *labeled graph* is a tuple (V, E, lab), where (V, E) is a graph and $lab\colon V \to \mathbf{N}$ assigns a label to each vertex. A p-graph is a labeled graph with labels in $\{1, \ldots, p\}$. With every graph $G = (V, E)$ we may identify the 1-graph (V, E, lab), where $lab\colon V \to \{1\}$.

We can now define the four graph operations that are used to define clique-width. Fix some integer $p \in \mathbf{N}^+$.

- We let $\odot$ denote the p-graph with a single vertex labeled 1.
- For a p-graph $G = (V, E, lab)$, we let $\mathrm{lab}_{i\to j}(G)$ be the graph $G' = (V, E, lab')$ obtained from G by relabeling all i-labeled vertices to j, i.e.,

$$lab'(v) = \begin{cases} j & \text{if } lab(v) = i \\ lab(v) & \text{else.} \end{cases}$$

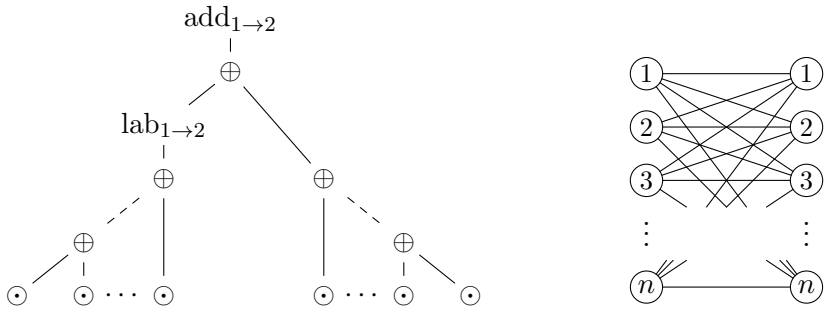


Figure 5: A 2-labeled clique-width parse tree generating the complete bipartite graph $K_{n,n}$

- For a p-graph $G = (V, E, lab)$, we let $\mathrm{add}_{i\to j}(G)$ be the graph obtained from G by connecting all vertices labeled with i with those labeled with j:

$$\mathrm{add}_{i\to j}(G) = (V, E \cup \{\, (u, v) \mid lab(u) = i \wedge lab(v) = j \,\}, lab)$$

- For two p-graphs $G_1 = (V_1, E_1, lab_1)$ and $G_2 = (V_2, E_2, lab_2)$, we let $G_1 \oplus G_2$ be the *disjoint union* $G = (V, E, lab)$ of G_1 and G_2, defined via $V = V_1 \cup V_2$, $E = E_1 \cup E_2$, and

$$lab(v) = \begin{cases} lab_1(v) & \text{if } v \in V_1 \\ lab_2(v) & \text{if } v \in V_2 \end{cases}$$

  Here, we w.l.o.g. assume that $V_1 \cap V_2 = \emptyset$, since we can rename vertices as needed.

A *p-labeled parse tree* T (also called *p-expression* in the literature) is a finite, ordered, rooted subcubic tree (with the root of degree at most two), such that

- all leaves of T are labeled with the $\odot$ symbol;
- all internal nodes of T with one child are labeled with $\mathrm{add}_{i\to j}$ or $\mathrm{lab}_{i\to j}$, where $i, j \in \{1, \ldots, p\}$; and
- all internal nodes of T with two children are labeled with $\oplus$.

A parse tree T *generates* the p-graph G that is obtained by the successive leaves-to-root application of the operators that label the nodes of T. A graph $G = (V, E)$ has clique-width p, if there is a p-labeled parse tree generating (V, E, lab) for some labeling $lab\colon V \to \{1, \ldots, p\}$. A 2-labeled parse tree generating a complete bipartite graph $K_{n,n}$ is shown in Figure 5. Note that a $K_{n,n}$ has treewidth n and n^2 edges.

Several interesting graph classes have bounded clique-width. For example, the class of *cographs* (P_4-free graphs) coincides with the graphs of clique-width bounded by 2 [49], and trees and, more generally, distance-hereditary graphs have clique-width at most three [84]. Furthermore, if a graph has treewidth k, then it has clique-width $2^{k+1}+1$ [36]. However, just as in the case of treewidth, the clique-width grows if the graphs become "moderately" dense. For instance, square grids with n vertices have treewidth and clique-width $\Theta(\sqrt{n})$.

Courcelle, Makowsky, and Rotics showed that on graphs of bounded clique-width every linear EMSO_1-definable problem can be solved in linear time if a p-expression is part of the input [47]. When we use the $O(|V|^3)$ time approximation algorithm by Oum [145] to compute a corresponding $f(w)$-expression, we get the following general result for graphs of bounded clique-width:

Theorem 35 (Courcelle, Makowsky, Rotics [47], Oum [145])**.** *Let P be an linear $EMSO_1$-definable optimization problem, $w \in \mathbf{N}$ an integer and $f\colon \mathbf{N} \to \mathbf{N}$. Then one can solve P on graphs $G = (V, E)$ of order $n := |V|$ and clique-width at most w in time $O(n^3)$ and in time $O(n)$ if an $f(w)$-expression is part of the input.*

As a consequence, we immediately get that many well-known NP-hard problems including the following can be solved in polynomial time on several graph classes containing dense graphs.

Corollary 2 (Courcelle, Makowsky, Rotics [47])**.** *On graphs of bounded clique-width, the following problems can be solved in polynomial time:* Minimum Vertex Cover *(GT1),* Minimum Dominating Set *(GT2),* Domatic Number *for fixed k (GT3), k-*Colorability *for fixed k (GT4),* Partition into Cliques *for fixed k (GT15),* Maximum Clique *(GT19),* Maximum Independent Set *(GT20), and* Induced Path *(GT23).*

However, the utility of MSO for dense graphs comes at a price: Unless EXPTIME = NEXPTIME, Theorem 35 cannot be extended to MSO_2, even if we allow the running time bound to be a polynomial:

Theorem 36 (Courcelle, Makowsky, Rotics [47])**.** *If* EXPTIME $\neq$ NEXPTIME*, then there is an MSO_2-definable decision problem over the class of cliques which is not solvable in polynomial time.*

As the experimental evaluation of Courcelle and Durand shows (cf., Section 5.4), algorithms based on the clique-width decompositions of the input graphs are furthermore also feasible in practical applications. A major problem for clique-width based algorithms currently lies in the lack of efficient algorithms to compute the necessary clique-width expressions, cf. Section 5.4. The problem of deciding whether a given input graph has clique-width at most k when k is part of the input is NP-complete [68]. It is in P for $k = 2$ [35] and for $k = 3$ [34], but its classification remains open for larger fixed values of k. Johansson presented in [109] an approximation algorithm for clique-width, which achieves an approximation ratio of $2k \log |V|$. Due to its dependency on $|V|$, this algorithm cannot be used to derive the time bounds of Theorem 35. In order to design an approximation algorithm for clique-width with an approximation ratio independent of $|V|$, Oum and Seymour introduced in [146] a new width measure named *rank-width*. The $O(|V|^3)$ time bound given in Theorem 35 is due to the running time of the rank-width approximation algorithm [145], which also approximates the clique-width.

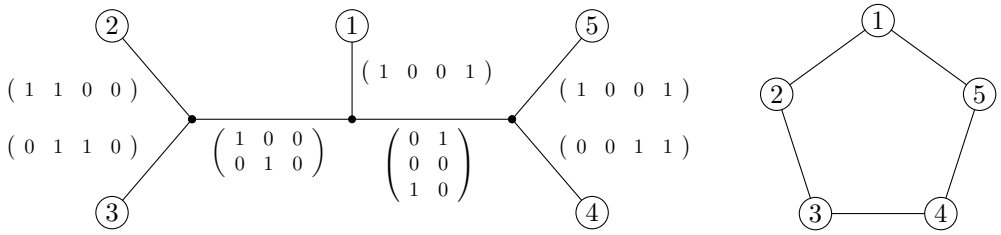


Figure 6: A rank-decomposition of the C_5 [81]

6.1.2. Rank-width

Rank-width was introduced by Oum and Seymour in order to study clique-width [146]. They defined *rank-width* as the *branch-width* [154] of the *cut-rank* function. *Branch-width* and *branch-decompositions* were originally introduced by Robertson and Seymour [154] and provide another notion of a recursive graph decomposition. A *rank-decomposition* of a graph $G = (V, E)$ is a pair (T, μ), where T is a subcubic tree and $\mu\colon V \to \{\, t \mid t \text{ is a leaf of } T \,\}$ is a bijection. Every edge of a rank-decomposition tree defines a *cut* of the graph, which is a partition of the vertex set into disjoint sets V_1, V_2. With every cut (V_1, V_2), we can associate a $|V_1| \times |V_2|$-adjacency matrix over $GF(2)$, where an entry $e_{i,j} = 1$ iff the ith vertex of V_1 is adjacent to the jth vertex of V_2. The width of a rank-decomposition is the maximum *rank* over all cuts induced by its edges (the *cut-rank*), and the *rank-width* of a graph G is the minimum width over all rank-decompositions of G. See Figure 6 for an example of a rank-decomposition. For the exact definitions, we refer the reader to [146] or the survey [102].

The definition in terms of branch-width allows one to prove several properties of rank-width including the fact that rank-width and cliquewidth are equivalent width measures in the sense that a class of undirected graphs has bounded rank-width if and only if it has bounded cliquewidth [146]. More precisely, for every (undirected) graph G, we have

$$rwd(G) \leq cwd(G) \leq 2^{rwd(G)+1} - 1,$$

where $rwd(G)$ and $cwd(G)$ are, respectively, the rank-width and clique-width of G. However, this definition in terms of branch-width is not very intuitive and also not always useful from an algorithmic point-of-view. This prompted Courcelle and Kanté [46] to introduce an equivalent formulation of rank-width in terms of algebraic operations on labeled graphs that are very similar to those of the clique-width operations defined in the previous section. These operations were restated by Ganian and Hliněný [81] in terms of labeling joins and *t-labeled parse trees*. A width t rank-decomposition can easily be transformed into a t-labeled parse tree and vice versa [81]. The precise definitions of the parse tree operations are somewhat technical and it doesn't add much value to this survey to restate them here. The reader interested in these notions is referred to their definitions in [46, 81].

In [81], Ganian and Hliněný show that Myhill–Nerode type arguments for t-labeled parse trees can be used to prove Theorem 35 directly in terms of rank-width, therefore essentially outlining how a finite tree automaton that recognizes the parse tree can

be constructed. Another proof for rank-width, which is related to the game-theoretic approach described in Section 5.6, can be found in [127]. None of these approaches has been implemented yet, so their actual practical feasibility remains open.

As usual, all of these algorithms do require a rank-width decomposition (or a t-labeled parse tree) as input. In order to achieve the cubic running time of Theorem 35, we therefore depend on polynomial algorithms that compute a small rank-decomposition for a given input graph. It is NP-hard to decide whether a graph G has rank-width at most k when k is part of the input [101]. For fixed k, there is an $O(|V|^3)$ algorithm by Hliněný and Oum based on matroid methods, which computes an rank-decomposition of width k if such a rank-decomposition exists. It is, however, not feasible for practical applications. As in the case of treewidth, it suffices to use heuristics to compute a small rank-decomposition. There are some approximation algorithms by Oum and Seymour [147] and Oum [145] with different running times and performance guarantees, but we are not aware of any implementations. The lack of a suitable decomposition algorithm with proven practical feasibility therefore currently remains the biggest obstacle to using rank-width based approaches in practice. This gap is hopefully closed soon once suitable heuristics become available [12].

6.1.3. Boolean width

Boolean width is another width measure for dense graphs introduced by Bui-Xuan, Telle, and Vatshelle [29]. It is related to rank-width in the sense that we do not use the *rank* of the adjacency matrix as a measure for the cut, but rather use the *number of different neighborhoods* induced by the cut. More precisely, for a cut $C = (V_1, V_2)$, let

$$num(C) := \big|\{\, U_2 \subseteq V_2 \mid \exists U_1 \subseteq V_1 \wedge U_2 = N(V_1) \cap V_2 \,\}\big|,$$

which is the number of different neighborhoods in V_2 over all subsets of V_1. The *Boolean dimension* of C is then defined as $booldim(C) := \log_2 |num(C)|$, and the Boolean width of a branch decomposition of G is the maximum Boolean dimension over all of its cuts. The *Boolean width* of a graph G is then the minimum Boolean width over all branch-decompositions of G.

As shown in [29], the Boolean width of an undirected graph is always polynomially upper-bounded by its clique-width and its rank-width and is often much smaller. Since furthermore the algorithms presented in [29] have a rather small, i.e., single-exponential, dependency on the Boolean width, Boolean width might be a well-suited width-measure for practical applications. Unfortunately, owing the lack of a algorithmically useful notion of a *parse tree* based on graph operations as those for clique-width and rank-width, no theorem in the spirit of Theorem 35 has been shown yet using a Boolean width decomposition directly. The detour over a rank-width or clique-width parse trees that is currently needed might render this approach infeasible in practice due to the non-avoidable blow-up induced by switching width measures.

Hvidevold, Sharmin, Telle, and Vatshelle presented [104] a heuristic for Boolean width and an *experimental evaluation* on a large set of graphs from the Treewidth-LIB [175] graph library. For a large number of graphs, the best known bound on the

treewidth is at least twice as big as the Boolean width bound computed by their heuristic. This gives hope that in future algorithms based on width measures for dense graphs could be equally successful as those for small treewidth.

6.2. Variants of Treewidth

There are several other width measures that are related to treewidth. They typically share the property of treewidth that MSO_2-definable problems can be solved in linear time, as opposed to the width measures for dense graphs, where this is possible for MSO_1 only.

Branch-width was introduced by Robertson and Seymour [154] and is always within a factor 1.5 of treewidth. There are a few restricted variants of treewidth that only play a minor role nowadays, e.g., *strong treewidth* [162] or *domino treewidth* [22]. A more commonly used restriction of treewidth is *pathwidth*, also introduced by Robertson and Seymour [152]. It is defined in the same way as treewidth, but we use a *path* instead of a tree. Since every path is also a tree, algorithms for tree decomposition can be used for path-decompositions without modifications. On the other hand, algorithms that only operate on path decompositions are significantly simpler—and often faster—than those for tree decompositions [39, 44], since one does not need to handle the complex *fusion* (*gluing*) operation for subgraphs. The price we pay for this is that the pathwidth is general larger than its treewidth, up to a factor of $O(\log n)$. For example, the pathwidth of a path is 1, but becomes $\Theta(\log n)$ for trees; $n \times m$ grids have pathwidth and treewidth $\min\{n, m\}$, and every graph has pathwidth $\leq n - 1$.

Courcelle introduced *special treewidth* [38, 39]. Special treewidth lies between pathwidth and treewidth, but the corresponding automata are exponentially smaller than those for treewidth, which is beneficial for the space-explosion problems observed in practical applications. No algorithms that construct small special tree decompositions are known yet.

6.3. Avoiding the non-elementary blow-up

A recent paper by Lampis [123] initiated the quest for graph parameters for which the non-elementary blow-up of the constants in the running time of algorithms solving the MSO Model-Checking problem can be avoided.

Lampis [123] considers graphs that admit a small vertex cover or a small maximum-leaf spanning tree. Both measures are typically *larger* than the treewidth of a graph and can be hence be understood as *restrictions* of treewidth, i.e., the class of bounded *vertex cover number* k is strictly contained in the class of graphs of bounded treewidth k. Lampis shows that for these graph classes and MSO_1-definable properties, the non-elementary lower bounds shown by Frick and Grohe [76] can be avoided and replaced by a double-exponential dependency on k.

Ganian [79] introduces the notion of a *twin cover*, which is defined similar to a vertex-cover but remains small on several dense graphs. The corresponding *twin cover number* of a graph lays between the size of its minimal vertex-cover and its clique-width and rank-width. Ganian proves that Lampis' result can be generalized from vertex cover to twin

cover. This has been further generalized to graphs of bounded tree-*depth* and bounded *shrub-depth* by Gajarský and Hlinený [78]. *Tree-depth* was introduced by Nešetřil and Ossona de Mendez in [142]; *shrub-depth* is further studied in [80].

6.4. Lower Bounds

From a more theoretically motivated standpoint, we may also ask on how much we can extend Courcelle's Theorem beyond graphs of bounded treewidth. In a series of papers, Kreutzer [118, 120] and Kreutzer and Tazari [122, 121] gave a corresponding complexity lower-bound for Courcelle's Theorem. Roughly speaking, they show that, modulo a certain complexity-theoretical assumption (the Exponential Time Hypothesis [106]), the MSO Model-Checking problem *cannot* be solved in time $O(poly(n))$ on $\mathcal{C}$, where $poly(n)$ is a polynomial whose degree depends on the input formula φ, and $\mathcal{C}$ is a class of graphs that has *strongly poly-logarithmically unbounded treewidth* and is either closed under coloring [120, 122], or under taking subgraphs [121]. A related result for the MSO_1 Model-Checking problem was proven in [82]. We note that there are, indeed, classes of strongly poly-logarithmically unbounded treewidth that admit polynomial time algorithms for the MSO_1 Model-Checking problem, e.g., classes of bounded clique-width or rank-width, but those are not closed under taking subgraphs.

7. Conclusions

For a long time, Courcelle's Theorem was used only as a theoretical tool to establish that linear-time algorithms exist for some problems and deemed useless in practical situations. An aim of this survey was to show that it is indeed possible to design a generic algorithm based on this meta-theorem that actually works in practice.

While it is still true that hand-crafted algorithms, tailored to specific problems, are much faster than the generic approaches presented in this survey, it is important to note that coming up with an error-free implementation of an algorithm specifically designed for a particular (MSO-expressible) problem is time-consuming. One can often solve such problems much faster (i.e., total time including development) using one of the generic tools presented in this survey.

Today, optimization suites like CPLEX or Gurobi often outperform hand-crafted or manually implemented algorithms to the extent that they have become the software of choice for solving many real world optimization problems. There is hope that with more research, generic tools for MSO-expressible problems will be a weapon of choice in solving such problems.

7.1. Further Reading

For the reader who is interested in the theoretical foundations of this area, we suggest the following introductory surveys: [92, 119, 93, 133, 102]. Monographs that give an in-depth exposition of algorithms for the MSO Model-Checking problem from an algorithmic viewpoint are [70, 44, 55].

7.2. Acknowledgements

We thank Stephan Kreutzer for pointing out a mistake in Example 28.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] K. A. Abrahamson and M. R. Fellows. Finite automata, bounded treewidth, and well-quasiordering. In *Proc. of Graph Structure Theory, Contemporary Mathematics 147*, pages 539–564. American Mathematical Society, 1991.

[3] J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier. Fixed parameter algorithms for dominating set and related problems on planar graphs. *Algorithmica*, 33(4):461–493, 2002.

[4] S. Arnborg. Efficient algorithms for combinatorial problems with bounded decomposability - a survey. *BIT*, 25(1):2–23, 1985.

[5] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8:277–284, 1987.

[6] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.

[7] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Appl. Math.*, 23(1):11–24, 1989.

[8] C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[9] C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo. Querying business processes with BP-QL. *Inf. Syst.*, 33(6):477–507, 2008.

[10] E. R. Berlekamp, J. H. Conway, and R. K. Guy. *Winning Ways for Your Mathematical Plays*. A.K. Peters, 1982.

[11] M. W. Bern, E. L. Lawler, and A. L. Wong. Linear-time computation of optimal subgraphs of decomposable graphs. *J. Algorithms*, 8(2):216–235, 1987.

[12] M. Beyß. Fast algorithm for rank-width. In *Proceedings of the 8th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS)*. Springer, 2012. To appear.

[13] L. W. Bieneke and R. E. Pippert. The enumeration of labeled 2-trees. *Notices of the American Mathematical Society*, 15:384, 1968.

[14] L. W. Bieneke and R. E. Pippert. The number of labeled k-dimensional trees. *Journal of Combinatorial Theory*, 6:200–205, 1969.

[15] H. Bodlaender and A. M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008.

[16] H. Bodlaender and A. M. C. A. Koster. Treewidth computations I. Upper bounds. *Inf. Comput.*, 208(3):259–275, 2010.

[17] H. L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In *Proceedings of the 15th International Colloquium on Automata, Languages, and Programming (ICALP)*, number 317 in Lecture Notes in Computer Science, pages 105–118. Springer, 1988.

[18] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.

[19] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25:1305–1317, 1996.

[20] H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In I. P. and P. Ruzicka, editors, *Proceedings of the 22nd Conference on Mathematical Foundations of Computer Science (MFCS)*, volume 1295 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 1997.

[21] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Comput. Sci.*, 209:1–45, 1998.

[22] H. L. Bodlaender and J. Engelfriet. Domino treewidth. *J. Algorithms*, 24(1):94–123, 1997.

[23] H. L. Bodlaender, F. V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D. M. Thilikos. (Meta) Kernelization. In *Proceedings of the 50th IEEE Symposium on Foundations of Computer Science (FOCS)* [105], pages 629–638.

[24] H. L. Bodlaender and A. M. C. A. Koster. Treewidth computations II. Lower bounds. *Inf. Comput.*, 209(7):1103–1119, 2011.

[25] G. Boolos and J. Burgess. *Computability and Logic*. Cambridge University Press, 4th edition, 2002.

[26] R. B. Borie. Generation of polynomial-time algorithms for some optimization problems on tree-decomposable graphs. *Algorithmica*, 14(2):123–137, 1995.

[27] R. B. Borie, R. G. Parker, and C. A. Tovey. Automatic Generation of Linear-Time Algorithms from Predicate Calculus Descriptions of Problems on Recursively Constructed Graph Families. *Algorithmica*, 7(1):555–581, 1992.

[28] J. R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

[29] B. Bui-Xuan, J. A. Telle, and M. Vatshelle. Boolean-width of graphs. *Theor. Comput. Sci.*, 412(39):5187–5204, 2011.

[30] B. Burgstaller, J. Blieberger, and B. Scholz. On the tree width of Ada programs. In *9th Ada-Europe International Conference on Reliable Software Technologies*, volume 3063 of *Lecture Notes in Computer Science*, pages 78–90. Springer, 2004.

[31] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer, 1990.

[32] M. Charikar, editor. *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2010.

[33] K. J. Compton and C. W. Henson. A uniform method for proving lower bounds on the computational complexity of logical theories. *Ann. Pure Appl. Logic*, 48(1):1–79, 1990.

[34] D. G. Corneil, M. Habib, J.-M. Lanlignel, B. A. Reed, and U. Rotics. Polynomial-time recognition of clique-width $\leq$ 3 graphs. *Discrete Applied Mathematics*, 160(6):834–865, 2012.

[35] D. G. Corneil, Y. Perl, and L. K. Stewart. A linear recognition algorithm for cographs. *SIAM J. Comput.*, 14(4):926–934, 1985.

[36] D. G. Corneil and U. Rotics. On the relationship between clique-width and tree-width. *SIAM J. Comput.*, 34(4):825–847, 2005.

[37] B. Courcelle. The Monadic Second-Order Theory of Graphs. I. Recognizable Sets of Finite graphs. *Information and Computation*, 85:12–75, 1990.

[38] B. Courcelle. Special tree-width and the verification of monadic second-order graph properties. In *Proceedings of the 30th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13–29, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[39] B. Courcelle. On the model-checking of monadic second-order formulas with edge set quantifications. *Discrete Applied Mathematics*, 160(6):866–887, 2012.

[40] B. Courcelle and I. Durand. Automata for the verification of monadic second-order graph properties. *J. Applied Logic*, 10(4):368–409, 2012.

[41] B. Courcelle and I. A. Durand. Tractable constructions of finite automata from monadic second-order formulas, 2010. Presented at *Logical Approaches to Barriers in Computing and Complexity*, Greifswald, Germany.

[42] B. Courcelle and I. A. Durand. Verifying monadic second-order graph properties with tree automata. In *3rd European Lisp Symposium*, pages 7–21, 2010. Informal proceedings edited by C. Rhodes.

[43] B. Courcelle and I. A. Durand. Fly-automata, their properties and applications. In *Implementation and Application of Automata - 16th International Conference, CIAA 2011*, volume 6807 of *Lecture Notes in Computer Science*, pages 264–272. Springer, 2011.

[44] B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language Theoretic Approach*. Number 138 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, June 2012.

[45] B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46(2):218–270, 1993.

[46] B. Courcelle and M. M. Kanté. Graph operations characterizing rank-width. *Discrete Applied Mathematics*, 157(4):627–640, 2009.

[47] B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique width. *Theory of Computing Systems*, 33:125–150, 2000.

[48] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comput. Sci.*, 109(1-2):49–82, 1993.

[49] B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

[50] IBM ILOG CPLEX Optimizer, version 12.4.0.0. `http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/`. Visited 2012-09-17.

[51] M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. van Rooij, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings of the 52nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 150–159. IEEE Computer Society, 2011.

[52] A. Dawar, M. Grohe, and S. Kreutzer. Locally excluding a minor. In *Proceedings of the 22 Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 270–279. IEEE Computer Society, 2007.

[53] R. Diestel. *Graph Theory*. Springer-Verlag, Heidelberg, 4th edition, 2010.

[54] J. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4:406–451, October 1970.

[55] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[56] I. Durand. Autowrite: A tool for checking properties of term rewriting systems. In S. Tison, editor, *Proceedings of the 13th International Conference on Rewriting Techniques and Applications (RTA)*, volume 2378 of *Lecture Notes in Computer Science*, pages 371–375. Springer, 2002.

[57] I. Durand. A tool for term rewrite systems and tree automata. *Electr. Notes Theor. Comput. Sci.*, 124(2):29–49, 2005.

[58] C. Dürr and T. Wilke, editors. *Proceedings of the 29th Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 14 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[59] Z. Dvorak, D. Král, and R. Thomas. Deciding first-order properties for sparse graphs. In Trevisan [176], pages 133–142.

[60] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1999.

[61] N. Een and N. Sörensson. The MiniSat page. `http://minisat.se`. Visited 2012-04-11.

[62] M. Elberfeld, A. Jakoby, and T. Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In Trevisan [176], pages 143–152.

[63] M. Elberfeld, A. Jakoby, and T. Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:62, 2010.

[64] M. Elberfeld, A. Jakoby, and T. Tantau. Algorithmic Meta Theorems for Circuit Classes of Constant and Logarithmic Depth. In Dürr and Wilke [58], pages 66–77.

[65] C. Elgot. Decision problems of finite-automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–51, 1961.

[66] H. Enderton. *A mathematical introduction to logic*. Academic Press, 2nd edition, 2001.

[67] S. Feferman and R. Vaught. The first order properties of algebraic systems. *Fund. Math*, 47:57–103, 1959.

[68] M. R. Fellows, F. A. Rosamond, U. Rotics, and S. Szeider. Clique-width is NP-complete. *SIAM J. Discrete Math.*, 23(2):909–939, 2009.

[69] J. Flum, M. Frick, and M. Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, 2002.

[70] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

[71] F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Bidimensionality and kernels. In Charikar [32], pages 503–510.

[72] E. Foustoucos and L. Kalantzi. The monadic second-order logic evaluation problem on finite colored trees: a database-theoretic approach. *Fundam. Inform.*, 92(3):193–231, 2009.

[73] E. Foustoucos and L. Kalantzi. Automata-theoretic and datalog-based solutions of monadic second-order logic evaluation problems over structures of bounded-treewidth, 2011. Techreport. Available at `http://nemertes.lis.upatras.gr/jspui/handle/10889/4327`.

[74] M. Frick. *Easy Instances for Model Checking*. PhD thesis, Universität Freiburg, 2001.

[75] M. Frick and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001.

[76] M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. *Annals of Pure and Applied Logic*, 130(1–3):3–31, 2004.

[77] H. Gaifman. On local and non-local properties. In *Proceedings of the Herbrand Symposium, Logic Colloquium '81*. North-Holland, 1982.

[78] J. Gajarský and P. Hlinený. Deciding graph MSO properties: Has it all been told already? *CoRR*, abs/1204.5194, 2012.

[79] R. Ganian. Twin-cover: Beyond vertex cover in parameterized algorithmics. In Marx and Rossmanith [135], pages 259–271.

[80] R. Ganian, P. Hlinený, J. Nesetril, J. Obdrzálek, P. O. de Mendez, and R. Ramadurai. When trees grow low: Shrubs and fast MSO1. In B. Rovan, V. Sassone, and P. Widmayer, editors, *Proceedings of the 37th Conference on Mathematical Foundations of Computer Science (MFCS)*, number 7464 in Lecture Notes in Computer Science, pages 419–430. Springer, 2012.

[81] R. Ganian and P. Hliněný. On parse trees and Myhill–Nerode–type tools for handling graphs of bounded rank-width. *Discrete Applied Mathematics*, 158(7):851–867, 2010.

[82] R. Ganian, P. Hliněný, A. Langer, J. Obdrzálek, P. Rossmanith, and S. Sikdar. Lower bounds on the complexity of MSO1 model-checking. In Dürr and Wilke [58], pages 326–337.

[83] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, San Francisco, 1979.

[84] M. C. Golumbic and U. Rotics. On the clique-width of some perfect graph classes. *Int. J. Found. Comput. Sci.*, 11(3):423–443, 2000.

[85] G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for web information extraction. *J. ACM*, 51(1):74–113, 2004.

[86] G. Gottlob, R. Pichler, and F. Wei. Bounded treewidth as a key to tractability of knowledge representation and reasoning. In *Proc. of AAAI 2006*, pages 250–256. AAAI Press, 2006.

[87] G. Gottlob, R. Pichler, and F. Wei. Bounded treewidth as a key to tractability of knowledge representation and reasoning. *Artif. Intell.*, 174(1):105–132, 2010.

[88] G. Gottlob, R. Pichler, and F. Wei. Monadic datalog over finite structures of bounded treewidth. *ACM Trans. Comput. Logic*, 12(1):3:1–3:48, 2010.

[89] E. Grädel. Finite model theory and descriptive complexity. In *Finite Model Theory and Its Applications*, pages 125–230. Springer, 2007.

[90] E. Grädel. Back and forth between logics and games. In K. R. Apt and E. Grädel, editors, *Lectures in Game Theory for Computer Scientists*, pages 99–145. Cambridge University Press, 2011.

[91] M. Grohe. Descriptive and parameterized complexity. In J. Flum and M. Rodríguez-Artalejo, editors, *Proceedings of the 13th international Workshop on Computer Science Logic (CSL)*, number 1683 in Lecture Notes in Computer Science, pages 14–31. Springer, 1999.

[92] M. Grohe. Logic, graphs, and algorithms. In J. Flum, E. Grädel, and T. Wilke, editors, *Logic and Automata: History and Perspectives*, pages 357–422. Amsterdam University Press, Amsterdam, 2007.

[93] M. Grohe and S. Kreutzer. Methods for algorithmic meta-theorems. In *Model Theoretic Methods in Finite Combinatorics*, volume 588 of *Contemporary Mathematics*. American Mathematical Society, 2011.

[94] Y. Gurevich. Modest Theory of Short Chains. I. *J. Symb. Log.*, 44(4):481–490, 1979.

[95] J. Gustedt, O. A. Mæhle, and J. A. Telle. The treewidth of Java programs. In D. M. Mount and C. Stein, editors, *ALENEX*, volume 2409 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 2002.

[96] R. Halin. S-functions for graphs. *Journal of Geometry*, 8:171–186, 1976.

[97] W. Hanf. Model-theoretic methods in the study of elementary logic. In *Theory of Models (Proc. 1963 Internat. Sympos., Berkeley)*, pages 132–145. North-Holland, 1965.

[98] S. Hedetniemi. References on partial k-trees. *Discrete Applied Mathematics*, 54:281–290, 1994.

[99] S. T. Hedetniemi, A. Proskurowski, and S. Arnborg. Bibliography on partial k-trees. `http://liinwww.ira.uka.de/bibliography/Theory/partial.k.trees.html`, 2012. Visited 2012-03-28.

[100] J. Hintikka. *Logic, Language-Games and Information: Kantian Themes in the Philosophy of Logic*. Clarendon Press, 1973.

[101] P. Hliněný and S. Oum. Finding branch-decomposition and rank-decomposition. *SIAM J. Comput.*, 38:1012–1032, 2008.

[102] P. Hliněný, S. Oum, D. Seese, and G. Gottlob. Width parameters beyond tree-width and their applications. *Comput. J.*, 51(3):326–362, 2008.

[103] W. Hohberg and R. Reischuk. A framework to algorithms for optimization problems on graphs. Technical report, Technische Hochschule Darmstadt, Germany, 1990.

[104] E. M. Hvidevold, S. Sharmin, J. A. Telle, and M. Vatshelle. Finding good decompositions for dynamic programming on dense graphs. In Marx and Rossmanith [135], pages 219–231.

[105] IEEE Computer Society. *Proceedings of the 50th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2009.

[106] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

[107] M. Jakl, R. Pichler, S. Rümmele, and S. Woltran. Fast counting with bounded treewidth. In *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 5330 of *Lecture Notes in Computer Science*, pages 436–450. Springer, 2008.

[108] M. Jakl, R. Pichler, and S. Woltran. Answer-set programming with bounded treewidth. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 816–822, 2009.

[109] Ö. Johansson. $\log n$-approximative $\mathrm{NLC_k}$-decomposition in $o(n^{2k+1})$ time. In *WG*, volume 2204 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2001.

[110] D. S. Johnson. The NP-completeness column: An ongoing guide. *J. Algorithms*, 6(3):434–451, 1985.

[111] S. Kepser. Querying linguistic treebanks with monadic second-order logic in linear time. *Journal of Logic, Language and Information*, 13(4):457–470, 2004.

[112] S. Kepser. Using MONA for querying linguistic treebanks. In *Proceedings of HLT/EMNLP 2005*. The Association for Computational Linguistics, 2005.

[113] S. Khuller, S. G. Mitchell, and V. V. Vazirani. Processor efficient parallel algorithms for the two disjoint paths problem and for finding a Kuratowski homeomorph. *SIAM J. Comput.*, 21(3):486–506, 1992.

[114] N. Klarlund and A. Møller. *MONA Version 1.4 User Manual*. BRICS, Dept. of Comp. Sc., University of Aarhus, January 2001. Available from `http://www.brics.dk/mona/`.

[115] N. Klarlund, A. Møller, and M. I. Schwartzbach. MONA Implementation Secrets. In *Proc. of CIAA00*, pages 182–194. Springer-Verlag, 2001.

[116] J. Kneis, A. Langer, and P. Rossmanith. Courcelle's Theorem – a game-theoretic approach. *Discrete Optimization*, 8(4):568–594, 2011.

[117] E. Kranakis, P. Penna, K. Schlude, D. Taylor, and P. Widmayer. Improving customer proximity to railway stations. In *Proceedings of the 5th Italian Conference on Algorithms and Complexity*, volume 2653 of *Lecture Notes in Computer Science*, pages 264–276. Springer-Verlag, 2003.

[118] S. Kreutzer. On the parameterised intractability of monadic second-order logic. In *Proceedings of the 23rd international Workshop on Computer Science Logic (CSL)*, number 5771 in Lecture Notes in Computer Science, pages 348–363. Springer, 2009.

[119] S. Kreutzer. Algorithmic meta-theorems. In *Finite and Algorithmic Model Theory*, number 379 in London Mathematical Society Lecture Notes. Cambridge University Press, 2011.

[120] S. Kreutzer. On the parameterized intractability of monadic second-order logic. *Logical Methods in Computer Science*, 8(1), 2012.

[121] S. Kreutzer and S. Tazari. Lower bounds for the complexity of monadic second-order logic. In *Proceedings of LICS'10*, pages 189–198, 2010.

[122] S. Kreutzer and S. Tazari. On brambles, grid-like minors, and parameterized intractability of monadic second-order logic. In Charikar [32], pages 354–364.

[123] M. Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 2011. To appear.

[124] A. Langer. *Fast Algorithms for Decomposable Graphs*. PhD thesis, RWTH Aachen University, 2013.

[125] A. Langer, F. Reidl, P. Rossmanith, and S. Sikdar. Evaluation of an MSO-solver. In D. A. Bader and P. Mutzel, editors, *Proceedings of ALENEX'12*, pages 55–63. Society for Industrial and Applied Mathematics, 2012.

[126] A. Langer, F. Reidl, P. Rossmanith, and S. Sikdar. Sequoia homepage. `http://sequoia.informatik.rwth-aachen.de/sequoia/`, 2012. Visited 2012-09-16.

[127] A. Langer, P. Rossmanith, and S. Sikdar. Linear-time algorithms for graphs of bounded rankwidth: A fresh look using game theory (extended abstract). In *TAMC'11*, volume 6648 of *LNCS*, pages 505–516. Springer, 2011.

[128] A. S. LaPaugh. Recontamination does not help to search a graph. *J. ACM*, 40(2):224–245, 1993.

[129] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.

[130] C. Liu. *Tree Decomposable Models for Efficient Bioinformatics Algorithms.* PhD thesis, University of Georgia, Athens, GA, USA, 2006.

[131] D. Lokshtanov, D. Marx, and S. Saurabh. Slightly superexponential parameterized problems. In D. Randall, editor, *Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 760–776. SIAM, 2011.

[132] S. Mahajan and J. G. Peters. Regularity and locality in k-terminal graphs. *Disc. Appl. Math.*, 54:229–250, 1994.

[133] J. A. Makowsky. Algorithmic uses of the Feferman-Vaught Theorem. *Ann. Pure Appl. Logic*, 126(1-3):159–213, 2004.

[134] M. F. Mammana, S. Mecke, and D. Wagner. The station location problem on two intersecting lines. *Electr. Notes Theor. Comput. Sci.*, 92:52–64, 2004.

[135] D. Marx and P. Rossmanith, editors. *Proceedings of the 6th International Workshop on Parameterized and Exact Computation (IWPEC)*, number 7112 in Lecture Notes in Computer Science. Springer, 2011.

[136] H. Maryns. On the implementation of tree automata: Limitations of the naive approach. In *Proc. 5th Int. Treebanks and Linguistic Theories Conference (TLT 2006)*, pages 235–246, 2006.

[137] C. McDiarmid and B. A. Reed. Channel assignment on graphs of bounded tree-width. *Discrete Mathematics*, 273(1-3):183–192, 2003.

[138] E. Mendelson. *Introduction to Mathematical Logic.* Chapman & Hall, 4th edition, 1997.

[139] P. B. Miltersen, J. Radhakrishnan, and I. Wegener. On converting CNF to DNF. *Theor. Comput. Sci.*, 347(1-2):325–335, 2005.

[140] L. S. Moonen and F. C. R. Spieksma. Exact algorithms for a loading problem with bounded clique width. *INFORMS Journal on Computing*, 18(4):455–465, 2006.

[141] M. Morgan and V. Grout. Finding optimal solutions to backbone minimisation problems using mixed integer programming. In *Proceedings of the 7th International Network Conference (INC 2008)*, pages 53–64. University of Plymouth, 2008.

[142] J. Nešetřil and P. O. de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European J. Combin.*, 27(6):1024–1041, 2006.

[143] J. Nešetřil and P. O. de Mendez. *Sparsity: Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012.

[144] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[145] S. Oum. Approximating rank-width and clique-width quickly. *ACM Transactions on Algorithms*, 5(1), 2008.

[146] S. Oum and P. D. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory Series B*, 96(4):514–528, 2006.

[147] S. Oum and P. D. Seymour. Testing branch-width. *Journal of Combinatorial Theory, Series B*, 97(3):385–393, 2007.

[148] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43:425–440, 1991.

[149] R. Pichler. Exploiting bounded treewidth with datalog (a survey). In *Datalog Reloaded - First International Workshop*, volume 6702 of *Lecture Notes in Computer Science*, pages 88–105. Springer, 2011.

[150] M. O. Rabin. A simple method for undecidability proofs and some applications. In Y. Bar-Hillel, editor, *Logic, Methodology and Philosophy of Sciences*, volume 1, pages 58–68. North-Holland, Amsterdam, 1964.

[151] K. Reinhardt. The complexity of translating logic to finite automata. In *Automata, logics, and infinite games*, pages 231–238. Springer, 2002.

[152] N. Robertson and P. D. Seymour. Graph minors. I. Excluding a forest. *Journal on Combinatorial Theory Series B*, 35:39–61, 1983.

[153] N. Robertson and P. D. Seymour. Graph minors II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

[154] N. Robertson and P. D. Seymour. Graph minors X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52:153–190, 1991.

[155] N. Robertson and P. D. Seymour. Graph minors XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63:65–110, 1995.

[156] H. Röhrig. *Tree decomposition: a feasibility study.* Master's thesis, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1998.

[157] D. J. Rose. On simple characterizations of k-trees. *Discrete Mathematics*, 7(3-4):317–322, 1974.

[158] R. M. Sainsbury. *Logical Forms: an introduction to philosophical logic.* John Wiley & Sons, 2nd edition, 2000.

[159] P. Scheffler. *Die Baumweite von Graphen als ein Maß für die Kompliziertheit algorithmischer Probleme.* PhD thesis, Akademie der Wissenschaften der DDR, Berlin, Germany, 1989.

[160] P. Scheffler and D. Seese. A combinatorial and logical approach to linear-time computability. In *EUROCAL'87*, volume 378 of *LNCS*, pages 379–380, 1989.

[161] U. Schöning. *Logik für Informatiker.* Spektrum, Akad. Verlag, Heidelberg, 5 edition, 2000.

[162] D. Seese. Tree-partite graphs and the complexity of algorithms. In *Proceedings of the 5th Conference on Fundamentals of Computation Theory*, volume 199 of *Lecture Notes in Computer Science*, pages 412–421. Springer, 1985.

[163] D. Seese. Linear time computable problems and first-order descriptions. *Mathematical Structures in Computer Science*, 6(6):505–526, 1996.

[164] P. D. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58(1):22–33, 1993.

[165] J. R. Shoenfield. *Mathematical Logic.* A K Peters, 2nd edition, 2001.

[166] C. Sloper. Parameterized complexity and the method of test sets. Master's thesis, University of Bergen, Norway, 2001.

[167] D. Soguet. *Génération automatique d'algorithmes linéaires.* Doctoral dissertation, University Paris-Sud, 2008.

[168] L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[169] L. J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic.* PhD thesis, Massachusetts Institute of Technology, 1974. Available at `http://dspace.mit.edu/handle/1721.1/15540`.

[170] K. Takamizawa, T. Nishizeki, and N. Saito. Combinatorial problems on series-parallel graphs. *Discrete Applied Mathematics*, 3(1):75 – 76, 1981.

[171] K. Takamizawa, T. Nishizeki, and N. Saito. Linear-time computability of combinatorial problems on series-parallel graphs. *J. ACM*, 29(3):623–641, 1982.

[172] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[173] W. Thomas. *Languages, automata, and logic*, pages 389–455. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[174] M. Thorup. All structured programs have small tree-width and good register allocation. *Inf. Comput.*, 142(2):159–181, 1998.

[175] TreewidthLIB. `http://www.cs.uu.nl/research/projects/treewidthlib/`, 2012. Visited 2012-09-16.

[176] L. Trevisan, editor. *Proceedings of the 51st IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 2010.

[177] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press, 1988.

[178] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8:410–421, 1979.

[179] M. van den Nest, A. Miyake, W. Dür, and H. Briegel. Universal resources for measurement-based quantum computation. *Phys. Rev. Lett.*, 97(15):150504, 2006.

[180] J. van Leeuwen. Graph algorithms. In J. van Leeuwen, editor, *Algorithms and Complexity*, volume A of *Handbook of Theoretical Computer Science*, pages 527–631. Elsevier, 1990.

[181] J. M. M. van Rooij, H. L. Bodlaender, and P. Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *ESA*, number 5757 in Lecture Notes in Computer Science, pages 566–577. Springer, 2009.

[182] D. Wagner. Algorithms and models for railway optimization. In *Proceedings of Workshop on Algorithms and Data Structures*, volume 2748 of *Lecture Notes in Computer Science*, pages 198–206. Springer, 2003.

[183] M. Weyer. *Modifizierte parametrische Komplexitätstheorie*. PhD thesis, Universität Freiburg, 2008. In German.

[184] W. M. White. Deciding hamiltonicity in graphs of bounded treewidth. In R. G. Downey, M. R. Fellows, R. Niedermeier, and P. Rossmanith, editors, *Parameterized Complexity - Dagstuhl-Seminar-Report 316*. 2001. Available at `http://www.dagstuhl.de/Reports/01/01311.pdf`.

[185] T. Wimer. *Linear algorithms on k-terminal graphs*. PhD thesis, Clemson University, Clemson, SC, USA, 1987.

[186] T. V. Wimer, S. T. Hedetniemi, and R. Laskar. A methodology for constructing linear graph algorithms. *Congr. Numer.*, 50:43–60, 1985.

[187] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):22–33, 1976.