Evaluation of an MSO-Solver

Alexander Langer, Felix Reidl, Peter Rossmanith, Somnath Sikdar

Dept. of Computer Science, RWTH Aachen University, Germany

Abstract. A fundamental theorem of Courcelle states that every problem definable in Monadic Second-Order Logic (MSO) is solvable in linear time on graphs of bounded treewidth. In this paper, we report on our ongoing effort to develop a general purpose software tool designed to solve MSO-definable optimization and decision problems on graphs of small treewidth. We discuss the theoretical underpinnings of our tool and present experimental results, which indicate that for some natural optimization problems MSO based approaches might be a suitable alternative to ILP solvers.

1 Introduction

Several real-world optimization problems can be modeled by graphs with small treewidth. Interesting examples include optimization problems for train and road networks when the underlying network has low treewidth. For instance, many local railway networks have a generic star-like structure connecting a central station with nearby suburban stations.

A well-known example is the Station Location problem [23, 26, 35]. Here we are given a railway network together with information on the population and their use of the railway infrastructure. The problem is to add new stops in the existing railway network so as to maximize accessibility of the railway infrastructure by the population. A variation is the Bus Stop Location problem [17], where one has to locate the minimum number of bus stops required to ensure that no passenger need walk more than a specified distance from his normal boarding point to reach an express bus stop.

In practice, one strategy to tackle such problems is to artificially transform the problem into easier subproblems on path-like graphs. For instance, one of the approaches Wagner lists in her survey [35] is to decompose the original NP-complete set-cover-type problem into subproblems that are modeled by only a few *line segments*. For such subproblems, the underlying set covering problem has the "consecutive ones property" which ensures that it can be solved in polynomial time by an LP-relaxation [16, 35]. Unfortunately, there might be cases where the consecutive ones property does not hold, or when the given input instance is not splittable into appropriate subproblems. For example, if the task is to find good locations for transmitters to cover an existing railway network with mobile Internet access, interference and obstacles can easily destroy the consecutive ones property. Similarly, such problems often become much harder once we add additional constraints. For example, connectivity is an important aspect for mesh-like wireless networks such as IEEE 802.11s. Finally, we could have multiple optimization criteria like using the minimum number of frequencies to reach the maximum number of customers, or add a minimum amount of new bus stops to benefit a maximum number of customers [29]. Of course, LP or ILP solvers can be used when the problems admit an ILP formulation. A large number of problems of practical interest fall under this category. Another option is to develop tailor-made algorithms that exploit the underlying tree-like structure. It is, however, not clear whether these algorithms will be faster than general ILP solvers. Moreover, they take considerable time and energy to develop. Generic solvers are hence a very useful tool to have, because they alleviate the need to develop customized algorithms, and usually it takes a lot less effort to implement the problem specification than to come up with good algorithms.

In this paper, we report on our generic software tool that solves MSO definable optimization and decision problems for graphs of small treewidth. MSO is a powerful language that has a very rich expressive power and allows to express optimization problems in a natural manner. For example, connectivity constraints can easily be modeled. Many people, however, consider the theorems [1, 5, 10] underlying our approach as purely theoretical. For instance, Niedermeier 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* [28, p. 170f].

Overview. The paper is organized as follows. Firstly, we briefly recap treewidth, MSO and how MSO-definable optimization problems can be solved in theory. We then highlight the technical difficulties in implementing the theoretical algorithms, and how we try to circumvent the problems. We then provide experimental results for four important, natural graph problems, namely MINIMUM VERTEX COVER, MINIMUM DOMINATING SET, 3-COLORABILITY, and MINIMUM CONNECTED DOMINATING SET. The majority of the instances that we deal with are grids and subgraphs of grids. On these instances, MINIMUM VERTEX COVER and 3-COLORABILITY are solvable in polynomial time (because the graphs are bipartite) and the LP-relaxations of the respective ILP formulations already yield optimal results. Hence, our tool has limited practical utility for such problems. However, for the two domination problems, our tool outperforms CPLEX on large instances, even if we allow CPLEX to return non-optimal solutions. Finally, we provide experimental results for a graph obtained from a real world railway network.

2 MSO-definable Optimization Problems

In graph theory, treewidth and pathwidth are two important parameters that describe how close a given graph is to a tree or a path, respectively. Measuring the tree-likeness of a graph is helpful as it not only gives structural insight, but because many NP-complete problems are easy to solve on graphs of small treewidth. Indeed Courcelle's celebrated result states that any problem expressible in Monadic Second Order Logic (MSO) is linear time solvable on graphs of bounded treewidth [5]. As mentioned above, instances of real-world problems often have low treewidth. This fact coupled with Courcelle's Theorem gives hope that there exist algorithms for many NP-complete problems that compute optimal solutions on real-world instances in a reasonable time. However, one cannot blindly implement the approach in [5] as this requires the construction of a tree automaton, for which the power set construction easily consumes too much memory. Hence, it is of interest to develop a generic software tool that can tackle such problems and produce optimal solutions in a reasonable time.

Monadic Second-Order Logic (MSO) extends First-Order Logic by quantification over sets of objects, see, e.g., [13]. MSO is a convenient logic to express properties since it resembles the way we specify properties in a natural language. Therefore, MSO typically allows for a natural expression of constraints. In the LinMSO framework, such MSO-definable constraints can be used to express linear optimization problems: Let $\varphi(U_1, \dots, U_l)$ be an MSO-formula with free set variables $U_1, \dots, U_l$, and let $\alpha_1, \dots, \alpha_l \in \mathbf{Z}$ be integers. For a graph $G = (V, E)$ the problem of computing

$$\min\Big\{ \sum_{k=1}^{l} \alpha_k |U_k| \;\Big|\; U_1, \dots, U_l \subseteq V \text{ and } G \models \varphi(U_1, \dots, U_l) \Big\}$$

is called a *LinMSO-definable optimization problem.* For example, the following well-known graph problems Minimum Vertex Cover and Minimum Dominating Set or the decision problem 3-Colorability can be written as:

- $\min\big\{ |C| \;\big|\; C \subseteq V \text{ and } G \models \mathit{vc}(C) \big\}$,
- $\min\big\{ |D| \;\big|\; D \subseteq V \text{ and } G \models \mathit{ds}(D) \big\}$, and
- $\min\big\{ 0 \;\big|\; G \models \mathit{3col} \big\}$,

where

$$\begin{aligned} \mathit{vc}(C) &= \forall x \forall y (\neg \mathit{adj}(x,y) \vee x \in C \vee y \in C) \\ \mathit{ds}(D) &= \forall x (x \in D \vee \exists y (y \in D \wedge \mathit{adj}(x,y))) \\ \mathit{3col} &= \exists R_1 \exists R_2 \exists R_3 \Big[\forall x \Big(\bigvee_{i=1}^{3} (x \in R_i) \wedge \bigwedge_{i \neq j} (\neg x \in R_i \vee \neg x \in R_j) \Big) \wedge \\ &\qquad \forall x \forall y \Big(\neg \mathit{adj}(x,y) \vee \bigwedge_{i=1}^{3} (\neg x \in R_i \vee \neg y \in R_i) \Big) \Big] \end{aligned}$$

Furthermore, MSO allows to take the transitive closure of the edge relation [4]. For example, connectivity constraints can easily be added by expressing that a set U is connected iff for all non-empty, proper subsets R of U there is an edge from R into $U \setminus R$. In MSO, this translates into

$$\begin{aligned}
\mathit{connected}(U) := \forall R\big((\forall x(x \in R \wedge x \in U)) \wedge &\quad \text{If } R \text{ is a subset of } U,\\
\exists x\,(x \in R) \wedge &\quad R \text{ is not empty, and}\\
\exists x\,(x \notin R \wedge x \in U)\big) &\quad R \text{ is a proper subset of } U,\\
\rightarrow \exists x \exists y(\mathit{adj}(x,y) \wedge x \in R \wedge y \notin R \wedge y \in U) &\quad \text{then we have an edge.}
\end{aligned}$$

The Minimum Connected Dominating Set problem can then conveniently be written as $\min\{\, |D| \mid D \subseteq V \text{ and } G \models \mathit{ds}(D) \wedge \mathit{connected}(D) \,\}$.

The MSO model checking problem is defined as follows: given an MSO-formula φ and a graph G, decide whether G is a model for φ. While this problem is PSPACE-complete [34] for general graphs, it is significantly easier on graphs of bounded treewidth. *Treewidth* is a graph parameter that essentially measures how similar a graph is to a tree. If a graph has small treewidth, it allows for a *tree* or *path decomposition* of small width, which in essence exposes the underlying tree structure of the graph. The definition of treewidth and the details of constructing a tree-decomposition are not required for following this paper and hence are omitted. We refer the reader to surveys such as [2, 3].

Courcelle's Theorem [5] states that any problem definable in MSO can be solved in linear time on graphs of bounded treewidth. This can be generalized to a rich class of counting and optimization problems including the LinMSO-framework [1, 10]. It is well-known [12, 32] that the model checking problem for MSO can be solved by constructing a finite-state bottom-up tree automaton. These methods can be extended to tree automata that recognize a tree decomposition of the input graph if and only if the graph is a model for the MSO formula, see, e.g. [1, 14].

However, it turns out that even for trivial problems like testing connectivity, the straightforward approach of constructing the tree automaton is infeasible in practice [6, 9, 18, 19, 30]. For most formulas, the problem lies in the state explosion in the required power-set construction. This is even the case [30] when optimized software like MONA [20] is used, which has been designed to overcome some of these difficulties [21].

Recently, there have been a couple of approaches to avoid the state explosion problems. In [18, 19], the authors consider Monadic Datalog. In [8, 9], the automata are constructed *on-the-fly* and the power-set construction is avoided by considering only existential formulas without universal quantifiers. To ease the specification of such *fly-automata*, "special treewidth" is introduced in [7].

Here we use a new approach that essentially works as follows (the details are in [22]). Our starting point is a simple algorithm that *evaluates* the formula φ on the input graph $G = (V, E)$ in a recursive manner. If, for example, the formula is $\exists R\psi(R)$ for a set variable R, the algorithm checks whether $G \models \psi(U)$ holds for all sets $U \subseteq V$. On a structure with n elements, this straight-forward recursive model-checking algorithm takes time $O((2^n + n)^q)$ for a formula that has q nested quantifiers, but only requires polynomial space. In particular, one does not need to use the expensive power-set construction which turned out to

Minimum Dominating Set

graph	memory usage	running time
path 1×200	≈ 483 MB	$\approx 3'59''$
grid 2×100	≈ 1354 MB	$\approx 24'42''$

Minimum Vertex Cover

graph	memory usage	running time
path 1×200	≈ 439 MB	$\approx 3'25''$
grid 2×100	≈ 1107 MB	$\approx 17'28''$

Table 1. Running times and memory usage of the first prototype implementation developed in 2008 for two standard graph optimization problems.

cause problems in the practical application of the automata theoretic approach. We then modified this simple algorithm to use dynamic programming on the tree decomposition. The extra information we need to save in the tables for the dynamic programming can be shown to be bounded in terms of the *length* $\|\varphi\|$ of the input formula and the treewidth w only. Therefore, for bounded treewidth and constant φ, the total running time is $O(n)$. In general the constant factors in the $O(n)$ cannot be bounded by an elementary function [15]. For concrete problems we can sometimes give rather precise upper bounds on the size of these tables. For instance, for the Minimum Dominating Set problem, we can show that each table contains at most $O(3^w)$ entries, and each such entry has size at most a polynomial in w. The running time of our generic approach can then be bounded by $O(5^w poly(w)n)$ [22]. We remark that the currently best *specialized* algorithm for this problem [33] requires subset convolution techniques to process the $O(3^w)$ entries in time $O(3^w poly(w))$.

3 Implementation

Our implementation is written in C++. Its development started in 2008, and the first prototype was able to solve Minimum Vertex Cover and Minimum Dominating Set on small grids only. This took a large amount of time and memory on standard computer hardware, as indicated in Table 1. Since then we have introduced several improvements in the algorithm and the implementation. The current version consists of roughly 14,000 lines of code. We use the Google *sparse table* library[1] for efficient *hash sets* and *hash maps*, which are significantly faster than the STL versions provided by `gcc` [2]. The implementation does not utilize multiple threads to benefit from today's multi-core architectures, but in principle the approach is well-suited for parallelization due to the way the dynamic programming works. We plan to add multi-threading support in the future. We note that we are also able to solve problems beyond the LinMSO framework: Courcelle's Theorem has been extended to a much richer class of problems. In particular, using appropriate semiring homomorphisms one can, for example, also enumerate all solutions or count their number [10]. Therefore, even more complicated optimization criteria can be specified. Consider, for example,

[1] http://code.google.com/p/google-sparsehash/
[2] http://google-sparsehash.googlecode.com/svn/trunk/doc/performance.html

the case that one wants to find pareto solutions under two optimization goals, such as minimizing the number of new bus stops to benefit a maximum number of customers. For, one only needs to provide an appropriate homomorphism or "evaluation function." The standard homomorphism for decision and optimizing solutions (find the minimum or maximum size solution or output such a solution) are already available. In the future, we plan to implement a plugin system such that arbitrary homomorphisms can easily be used.

In what follows, we shall briefly describe those improvements to the implementation and the underlying algorithms that had the largest impact on the running times and memory usage.

Implementation Improvements. In the dynamic programming framework, objects are stored in tables and looked up multiple times. As usual we have a table for each node of the tree decomposition, which contains a set of objects that describe *partial solutions.* By the pigeon-hole principle, we know that many of these objects are contained in many different places at the same time, which wastes a lot of memory. Additionally, full comparisons to store or retrieve these objects are very expensive. We now guarantee that each complex object is stored only once. This is possible because they are usually not modified. We implemented a pooling mechanism that, given a complex object, returns a pointer to an equal, existing object, or stores this object for future use. This approach considerably decreased the total memory consumption and allows us to replace deep equality tests by cheap pointer comparisons.

Caching the result of complex operations had another large impact on the running time. Again, the pigeon-hole principle (the number of vertices is assumed to be much larger than the treewidth) tells us that most computations are applied for a large number of times, for instance when we discover a new vertex of the input graph. We now cache the results of these expensive operations: Before we apply an expensive operation, we check in a hash map whether we have computed this operation before, which is much faster than doing the computation itself. In practice we noticed massive speedups. Table 2 compares running times for a few small grid graphs with and without caching. Here we notice a particularly large effect due to the many "self-similarities" in grids.

graph	*no caching*	*caching*
grid 3x1000	8"	350ms
grid 4x1000	33"	1"
grid 5x1000	2'14"	4"

Table 2. The effect of caching complex operations on the running time for the Minimum Dominating Set problem.

Algorithmic Improvements. Besides improving the implementation itself, we also revisited the dynamic programming algorithm. We found that frequently we can decide rather early whether we have a no- or yes-instance. For example, if the subgraph seen so for is not three-colorable, then it is clear that the graph itself is a no-instance and we can immediately discard such colorings. We were able to generalize this concept to arbitrary MSO formulas [22]. We distinguish three cases: "yes", the formula holds on the graph, "no," the formula does not hold, or "unknown," i.e., we have to continue with the dynamic programming approach to find the answer. In general the state is "unknown." However, since we can recursively apply this concept to subformulas, the resulting simplifications let many entries become identical. This saves a lot of time and space, and we consider this the major reason for the large improvements in the running time relative to the first prototype implementation in 2008.

4 Experiments

In this section, we provide experimental results for several graph problems. We selected three standard graph problems that cover the range of packing, covering and coloring problems, namely Minimum Vertex Cover, Minimum Dominating Set and 3-Colorability. Additionally, we consider Minimum Connected Dominating Set, which has applications in (wireless) network design (cf., [11, 24, 31]) and adds a connectivity constraint.

The problem instances we consider are subgraphs of grids, obtained by using a fixed probability to either include an edge or exclude it. Let p denote the probability that an edge is *kept* in the graph, implying that for $p = 1$ the grid remains unchanged. We created grids of small width k, ranging from 1 to 13 in our experiments, whereas the height m of the grid was kept fixed at 1000. Such grids have treewidth k. The edge-probability p was set to values in $[0.05, 1.00]$ with an increment of 0.05 and we run ten tests for each p (but only one for $p = 1$). In total, we therefore created 191 graphs for each $1 \leq k \leq 13$. Out of these, grids of width $1 \leq k \leq 13$ were considered for Minimum Vertex Cover, of width $1 \leq k \leq 8$ for Minimum Dominating Set, of width $1 \leq k \leq 7$ for 3-Colorability, and of width $1 \leq k \leq 5$ for Minimum Connected Dominating Set.

The choice of grids (and subgraphs of grids) stems from two considerations: on the one hand, many optimization problems related to traffic do exhibit a path- or grid-like structure (the latter would be a case where the width of, say, a road cannot be neglected). On the other hand grids offer a readily available bound on the treewidth, namely the width or height (whichever is smaller).

For a second series of tests on the Minimum Connected Dominating Set problem, we created grids with a total number of only about 200 vertices (depending on the width, the height is adjusted accordingly to match the size), and an edge probability between 0.90 and 1.0. We only consider grids with a width between 1 and 6 as this problem is much harder to solve. The best known deterministic algorithm [25] needs time $\Omega(w^w n)$ for treewidth w. For each k, five graphs were considered.

Finally, for a somewhat more realistic scenario we used the data available from OpenStreetMap and created a graph of the Hannover urban railway[3]. The graph obtained after cleaning the raw data from OpenStreetMap had treewidth 2. To this graph, we added possible locations for wireless base stations. For the

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

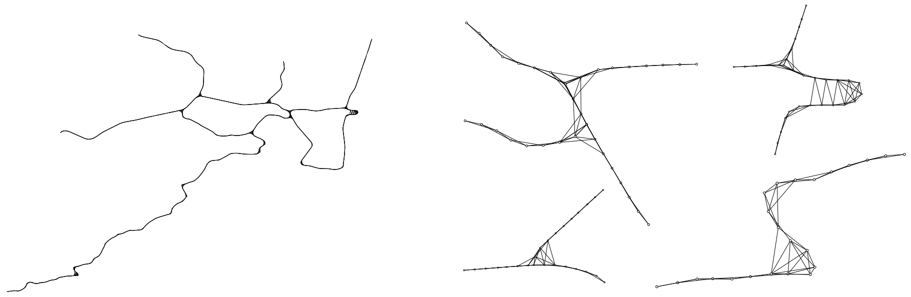


Fig. 1. To cover the Hannover urban rail network with wireless access generates a Minimum Connected Dominating Set problem. On the left side is the whole resulting graph and on the right side you can find some of its parts in detail.

edges, we used a disc graph model, since each base station is assumed to have a bounded maximum range. However, we assume that obstacles might hinder transmission to nearby vertices, so we only include edges between nearby nodes with a probability of 0.9. The resulting graph, depicted in Figure 1, has 673 vertices, 1445 edges, and treewidth bounded by 8. The task now is to select a minimum size connected set of base station locations, i.e., we are to solve an instance of the Minimum Connected Dominating Set problem.

For all the instances mentioned above, we created suitable ILPs that describe these instances. For Minimum Vertex Cover, Minimum Dominating Set, and 3-Colorability, we used the standard formulations. For Minimum Connected Dominating Set, we used the formulation of [27], where the connectivity is guaranteed by requiring a flow between the nodes of the solution.

The test setup is as follows. We focus on multi-purpose frameworks capable of solving a wide range of problems, and therefore did not include any specialized algorithms, which might have advantages in running time but usually take a long time to develop. Naturally, we measured the running time of our tool when asked to solve the problems. For, it was given an MSO specification of the problem and was told to minimize the solution size. Furthermore, we solved these instances by letting CPLEX find optimal or nearly optimal solutions to the ILPs. Since we consider (I)LP solvers the "state of the art" in optimization, we did not include any further frameworks such as SAT-solvers.

For our tool we used a 32bit Linux machine with an Intel Core 2 Quad CPU running at 2.40 GHz and 4 GB RAM. As this solver can only run single-threaded, we let up to four individual test instances be run in parallel and measured the CPU time used. Our tool was compiled with `gcc` version 4.4.5 with the `-O3`-flag. CPLEX Academic Research Edition 12.2.0.0 was used to solve the ILP instances. We let it run on dedicated 32 bit Linux machines all equipped with the Intel Core 2 Duo CPU running at 2.93 GHz (meaning that CPLEX was able to use two dedicated threads) and 4 GB RAM. We let CPLEX stop once an integrality gap of 5% was reached (`set mip tolerances mipgap 0.05`), i.e.,

running time	*error condition*	*best solution found*	*optimal solution*
7545’	out of memory	89	77
13622’	time limit	83	79
15054’	time limit	82	82
6505’	time limit	77	75
6624’	time limit	73	72

Table 3. CPLEX running times for the Minimum Connected Dominating Set problem on subgraphs of a 6×33 grid with an edge probability of 0.95. The time limit was hit after ten times of the CPU usage required by the MSO solver to compute the optimal solution.

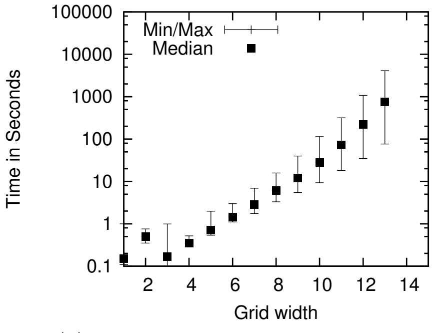
we did not insist on optimal solutions. CPLEX was given a time limit of 10 times the CPU time our tool required to solve these instances to optimality.

Results. CPLEX performs very well for Minimum Vertex Cover and 3-Colorability. Since the graphs are bipartite, the LP relaxation already provides the optimal solution. Our tool is oblivious to this fact and proceeds as it would on any other graph. As the running time of CPLEX is less than one second for these problems we only include the results for the MSO-approach. The running times for Minimum Dominating Set and Minimum Connected Dominating Set, however, show that for certain instances of low treewidth our tool can compete and even outperform CPLEX on the denser graphs. On sparser graphs the LP relaxations again turn out to be optimal or close to optimal.

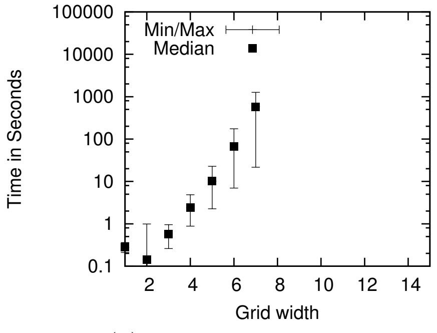
We present the runtime results on the grids of size $k \times 1000$ in the following figures which summarize the running times for each problem by presenting the minimum, median and maximum of the running times for each grid size. In 834 of the runs for Minimum Dominating Set, CPLEX found the optimal solution. In 45 cases, CPLEX hit the time limit. In the remaining cases, it was able to shrink the integrality gap to 5% within the given time limit, but the solution found was not optimal.

When solving Minimum Connected Dominating Set on grids of size $k \times m$, such that $km \approx 200$ and $k > 1$, with one exception CPLEX always hit the time limit, i.e., within 10 times of CPU time of the MSO solver CPLEX was not able to shrink the integrality gap to 5%. The one exception is a case that CPLEX ran out of memory. In a few cases, CPLEX did find the optimal solution, but could not guarantee optimality since the integrality gap was still too large. For the hardest graphs, those of dimension 6×33, the results depicted in Table 3 were obtained.

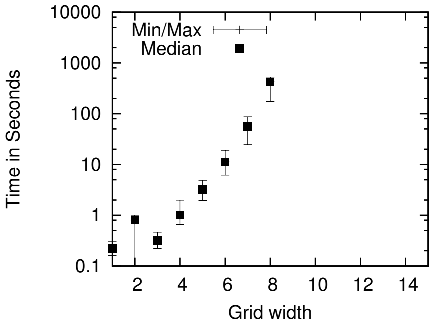
On the large railway network graph depicted in Figure 1, the optimal solution of 130 was found by our MSO solver in about 3761 seconds and with 299 MB of memory usage. On the same instance, we stopped CPLEX after 20945s real time computation. At that point, the best integer feasible solution found so far was 358, with an remaining integrality gap of 52.99%.



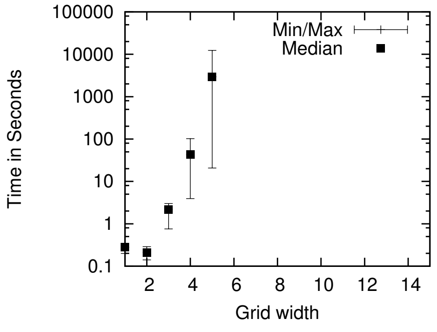
(a) Minimum Vertex Cover



(b) 3-Colorability

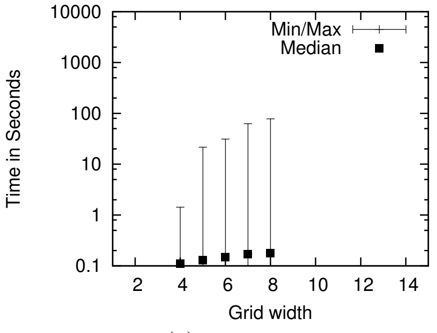


(c) Minimum Dominating Set

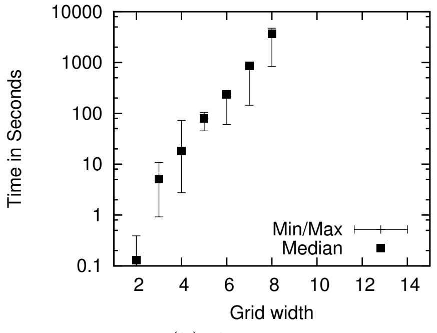


(d) Minimum Connected Dominating Set

Fig. 2. Running times of the MSO-solver for computing optimal solutions



(a) sparse



(b) dense

Fig. 3. Running time of CPLEX for Minimum Dominating Set on dense subgraphs of grids (edge probability $p < 0.9$ and $p \geq 0.9$, respectively). On the majority of the sparse instances the LP relaxation is optimal or close to optimal. The problem becomes significantly harder on denser instances: Even although we allowed to return non-optimal solutions within an integrality gap of 5%, CPLEX took considerably more time on the dense instances than our exact MSO solver.

5 Discussion and Conclusion

As the previous results show, our tool surprisingly performs much better than CPLEX on some instances of small treewidth. Of course, CPLEX does much better on other instances which is to be expected from a highly optimized commercial ILP-solver. The main advantage that our tool possesses is that we allow problems to be specified in a natural logic-based language that is very appropriate for many problems and that for graphs of small enough treewidth, certain problems can be solved much faster than with any other tool. With time, we plan to add in more functionality to make our software a practical tool. The next big challenge will probably be to include some kind of *lazy evaluation* for set variables.

References

1. S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.
2. H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.
3. H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Comput. Sci.*, 209:1–45, 1998.
4. B. Courcelle. Graph rewriting: An algebraic and logic approach. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 193–242. Elsevier, 1990.
5. B. Courcelle. The monadic second order theory of Graphs I: Recognisable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.
6. B. Courcelle. *Graph Structure and Monadic Second-Order Logic, a Language Theoretic Approach.* Cambridge University Press, 2011. Book in preparation.
7. B. Courcelle. On the model-checking of monadic second-order formulas with edge set quantifications. *Discrete Applied Mathematics*, 2011. To appear.
8. 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.
9. 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.
10. B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comput. Sci.*, 109(1-2):49–82, 1993.
11. F. Dai and J. Wu. An extended localized algorithm for connected dominating set formation in ad hoc wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 15(10):908–920, 2004.
12. J. Doner. Tree acceptors and some of their applications. *J. Comput. Syst. Sci.*, 4:406–451, October 1970.
13. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer, 1999.
14. J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer-Verlag, 2006.
15. M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic*, 130(1–3):3–31, 2004.
16. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* Freeman, San Francisco, 1979.

17. J. Gleason. A set covering approach to bus stop location. *International Journal of Management Science*, 3(5), 1975.
18. G. Gottlob, R. Pichler, and F. Wei. Abduction with bounded treewidth: From theoretical tractability to practically efficient computation. In *Proc. of 23rd AAAI*, pages 1541–1546. AAAI Press, 2008.
19. 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.
20. 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/`.
21. N. Klarlund, A. Møller, and M. I. Schwartzbach. MONA Implementation Secrets. In *Proc. of CIAA00*, pages 182–194. Springer-Verlag, 2001.
22. J. Kneis, A. Langer, and P. Rossmanith. Courcelle's Theorem – a game-theoretic approach, 2011. Submitted to Discrete Optimization.
23. 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.
24. W. Liang. Constructing minimum-energy broadcast trees in wireless ad hoc networks. In *Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC)*, pages 112–122, New York, NY, USA, 2002. ACM.
25. D. Lokshtanov, D. Marx, and S. Saurabh. Slightly superexponential parameterized problems. In *Proc. of 22th SODA*, 2011.
26. M. Mammana, S. Mecke, and D. Wagner. The station location problem on two intersecting lines. *Electronic Notes in Theoretical Computer Science*, 92(17):52–64, 2004.
27. 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.
28. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.
29. A. Schöbel. Locating stops along bus or railway linesa bicriteria problem. *Annals of Operations Research*, 136(1):211–227, 2005.
30. D. Soguet. *Génération automatique d'algorithmes linéaires.* Doctoral dissertation, University Paris-Sud, 2008.
31. M. Thai, F. Wang, D. Liu, S. Zhu, and D. Du. Connected dominating sets in wireless networks with different transmission ranges. *IEEE Transactions on Mobile Computing*, 6(7):721–730, 2007.
32. 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.
33. J. M. M. van Rooij, H. L. Bodlaender, and P. Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *ESA*, volume 5757 of *LNCS*, pages 566–577. Springer, 2009.
34. M. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the 14th annual ACM Symposium on Theory of Computing*, STOC '82, pages 137–146. ACM, 1982.
35. 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.