Dynamic Programming on Tree Decompositions using Binary Decision Diagrams: Research Summary

GÜNTHER CHARWAT
TU Wien, Institute of Information Systems
Favoritenstraße 9, 1040 Wien, Austria
(*e-mail:* gcharwat@dbai.tuwien.ac.at)

submitted 29 April 2015; accepted 5 June 2015

Abstract

Dynamic programming (DP) on tree decompositions is a well studied approach for solving hard problems efficiently. State-of-the-art implementations usually rely on tables for storing information, and algorithms specify how the tuples are manipulated during traversal of the decomposition. However, a major bottleneck of such table-based algorithms is relatively high memory consumption. The goal of the doctoral thesis herein discussed is to mitigate performance and memory shortcomings of such algorithms. The idea is to replace tables with an efficient data structure that no longer requires to enumerate intermediate results explicitly during the computation. To this end, Binary Decision Diagrams (BDDs) and related concepts are studied with respect to their applicability in this setting. Besides native support for efficient storage, from a conceptual point of view BDDs give rise to an alternative approach of how DP algorithms are specified. Instead of tuple-based manipulation operations, the algorithms are specified on a logical level, where sets of models can be conjointly updated. The goal of the thesis is to provide a general tool-set for problems that can be solved efficiently via DP on tree decompositions.

KEYWORDS: Logic, Dynamic Programming, Fixed Parameter Tractability, Binary Decision Diagram, Algorithm Design

1 Introduction and Problem Description

For problems that are known to be intractable, one approach is to exploit structural properties of the given input. An important parameter of graph-based instances is "tree-width", which, roughly speaking, measures the tree-likeness of the input. Tree-width is defined on so-called tree decompositions (Robertson and Seymour 1984), where the instance is split into smaller parts, thereby taking into account its structure. The problem at hand can then be solved by dynamic programming (DP). Many problems are fixed-parameter tractable (fpt) with respect to tree-width, that is, they are solvable in time $f(k) \cdot n^{\mathcal{O}(1)}$ where k is the tree-width, n is the input size and f is some computable function. Note that here the explosion in run-time is confined to k instead of the input size. Courcelle showed that every problem that

is definable in monadic second-order logic (MSO) is fixed-parameter tractable with respect to tree-width (Courcelle 1990). There, the problem is solved via translation to a finite tree automaton (FTA). However, the algorithms resulting from such an "MSO-to-FTA" translation are oftentimes impractical due to large constants (Niedermeier 2006). One approach to overcome this problem is to develop dedicated DP algorithms (see, e.g., (Groër et al. 2012; Chimani et al. 2012; Charwat et al. 2015)). Such algorithms typically rely on tables for storing information. However, in practice such implementations usually impose a large memory footprint.

Problem Statement: Although there has been a lot of effort to put Courcelle's Theorem into practice, there is still a gap between problems that are fpt w.r.t. tree-width and implementations that can exploit these results sufficiently in a practical setting. In my doctoral thesis, the challenge is to develop an alternative approach for dynamic programming on tree decompositions. To this end, the idea is to replace tables with a data structure that has undergone decades of research and that was developed in particular to be memory efficient, namely Binary Decision Diagrams (BDDs) (Bryant 1986). A BDD is a data structure where Boolean functions are represented compactly in form of a directed acyclic graph. Using BDDs gives rise to a new way of specifying DP algorithms where it is no longer required to manipulate single tuples (i.e., intermediate solutions) of the tables directly, but, instead, to apply Boolean function manipulations where *sets* of models are changed conjointly. One question that arises is how this can be done efficiently, and another one is whether and how all MSO-definable problems can be specified using this paradigm.

Overall, the challenges to be tackled are to 1) develop a general approach for logic-based formalizations of DP algorithms, and to 2) put these theoretical results into practice by providing a BDD-based DP software framework that mitigates large memory requirements oftentimes observed in state-of-the-art implementations.

2 Background

In this section we introduce tree decompositions, give a short review on dynamic programming (DP) on this data structure and introduce a special format of Binary Decision Diagrams (BDDs), namely Reduced Ordered BDDs (ROBDDs).

2.1 Tree Decompositions

Tree decompositions form the underlying basis for our dynamic programming algorithms.

Definition 1 ((Robertson and Seymour 1984))
A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\mathcal{T}, \mathcal{X})$ where $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$ is a (rooted) tree and $\mathcal{X} : V_\mathcal{T} \to 2^V$ is a function that assigns to each node $t \in V_\mathcal{T}$ a *bag* $X_t \subseteq V$ such that: 1) $\bigcup_{t \in V_\mathcal{T}} X_t = V$; 2) $\{x, y\} \in E \Rightarrow \exists t \in V_\mathcal{T} : \{x, y\} \subseteq X_t$; and 3) $x \in X_{t'} \wedge x \in X_{t''} \wedge t''' \in path(t', t'') \Rightarrow x \in X_{t'''}$.

The *width* w of the decomposition is $max_{t \in V_\mathcal{T}} |X_t| - 1$. The *tree-width* k of a graph is the minimum width over all its tree decompositions.

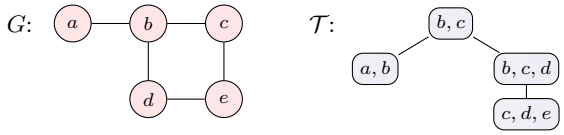

Fig. 1. Example graph G and a possible tree decomposition $\mathcal{T}$ of G.

The first condition states that every vertex of the original graph has to appear in at least one bag of the tree decomposition, the second condition guarantees that adjacent vertices appear together in some bag, and finally nodes whose bags contain the same vertex are connected. Figure 1 depicts an example graph G and a possible tree decomposition $\mathcal{T}$ of width 2. This tree decomposition is optimal w.r.t. width, i.e., the width corresponds to the tree-width of G.

2.2 Dynamic Programming on Tree Decompositions

Dynamic programming (DP) on tree decompositions follows a general pattern of how the solution is constructed:

1. For an input graph G, construct a tree decomposition $\mathcal{T}$. It is well-known that obtaining an optimal decomposition (with respect to width) is NP-hard (Arnborg et al. 1987), but there are heuristics that provide a "good" decomposition in polynomial time (Dermaku et al. 2008; Bodlaender and Koster 2010).
2. Traverse $\mathcal{T}$ in post-order and at each node $t \in V_{\mathcal{T}}$ compute partial solutions to the problem. Here, only information computed in the child node(s) of t and the subgraph of G induced by the vertices contained in bag X_t are to be considered.
3. At the root node of $\mathcal{T}$, due to the properties of tree decompositions, we know that the whole instances was taken into account. Typically, for decision problems (e.g., satisfiability, credulous, or skeptical reasoning) the result is directly available at the root node. For enumeration tasks the tree is traversed a second time (now in pre-order) and the partial solutions are combined in order to obtain the complete solutions.

Usually, intermediate results (obtained in step 2 of the algorithm) are stored in tables where each row *explicitly* represents a single partial solution. In case the problem at hand is fpt w.r.t. tree-width k, the size of each table is bounded by a polynomial of k.

2.3 Binary Decision Diagrams

Herein, we elaborate on an approach where the memory required for storing partial solutions is reduced by using an *implicit* representation. In our approach, a special type of Binary Decision Diagrams (introduced by (Lee 1959) and refined by (Akers 1978)), so called Reduced Ordered Binary Decision Diagrams (ROBDDs) (Bryant 1986), serve as the data structure.

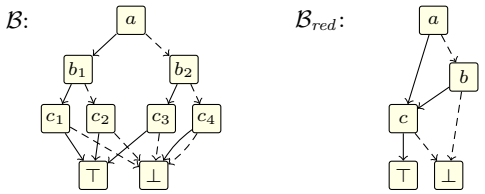

Fig. 2. OBBD and ROBBD of formula $(a \wedge b \wedge c) \vee (a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge c)$.

Definition 2
An *Ordered Binary Decision Diagram* (OBDD) is a rooted, connected, directed acyclic graph $\mathcal{B} = (V_\mathcal{B}, A_\mathcal{B})$ where $V_\mathcal{B} = V_T \cup V_N$ and $A_\mathcal{B} = A_\top \cup A_\bot$. We have:

1. Terminal nodes $\top$ and/or $\bot$ in V_T.
2. Nonterminal nodes $v \in V_N$ where v is a Boolean variable.
3. Each $v \in V_N$ has exactly one outgoing arc in $A_\top$ and one in $A_\bot$.
4. For every path from the root to a terminal node, each variable occurs at most once and in the same order (i.e., we have a strict total order over the variables).

In *Reduced OBDDs* (ROBDDs) the following reduction rules are applied:

- Isomorphic nodes are merged into a single node with several incoming edges.
- Nodes $v \in V_N$ where both outgoing arcs reach the same node $v' \in V_\mathcal{B}$ are removed.

Given an OBDD $\mathcal{B}$, propositional variables V_N and an assignment A to V_N, the *corresponding path* in $\mathcal{B}$ is the unique path from the root node to a terminal node, such that for every $v \in V_N$ it includes the outgoing arc in $A_\top$ ($A_\bot$) iff A gets assigned true (false) for v. A is a satisfying assignment of the function represented by $\mathcal{B}$ iff the path ends in $\top$. In the following we specify BDDs by giving the function in form of a logic formula. Figure 2 shows an OBBD $\mathcal{B}$ and the corresponding ROBBD $\mathcal{B}_{red}$ for the Boolean formula $(a \wedge b \wedge c) \vee (a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge c)$. Nodes c_1, c_2 and c_3 represent the same variable c and have arcs to the same terminal nodes. Hence, these isomorphic nodes are merged to a single node c. Then, both outgoing arcs of b_1 reach c, and b_1 is removed. Furthermore c_4 is removed, resulting in $\mathcal{B}_{red}$.

BDDs support standard logic operators (e.g., $\wedge$, $\vee$, $\neg$, $\leftrightarrow$, ...), existential quantification ($\exists V\mathcal{B}$ for $V \subseteq V_N$) as well as restriction and renaming of variables ($\mathcal{B}[v/\cdot]$ where $\cdot \in \{\top, \bot, v'\}$ for $v \in V_N$). The size of ROBDDs is, in the worst case, exponential, i.e., bounded by $\mathcal{O}(2^{|V_\mathcal{B}|})$ and heavily depends on the variable ordering. It was shown that finding an optimal ordering is NP-complete (Bollig and Wegener 1996). However, there are good heuristics available (e.g., (Rudell 1993)) and, in practice, BDDs are oftentimes exponentially smaller (Friedman and Supowit 1987).

3 Literature Review

Dynamic programming on tree decompositions has been studied in many problem domains such as belief revision (Pichler et al. 2009), answer-set programming (Morak et al. 2011) or abstract argumentation (Dvořák et al. 2012) (for a general overview, see, e.g., (Niedermeier 2006)).

There exist several tools that put these theoretical results into practice. The D-FLAT system (Abseher et al. 2014) combines DP with answer-set programming (ASP). Here, the user specifies the DP algorithm in form of an answer-set program which is executed at each node of the decomposition, thereby defining the DP algorithm explicitly. Furthermore, the tool SEQUOIA (Kneis et al. 2011) implements a game-theoretic approach where the problem is represented as a monadic second-order (MSO) formula. The instance is decomposed and the DP algorithm automatically generated and executed. Additionally, some problem-specific implementations like dynASP (Morak et al. 2011) (for ASP solving) and dynPARTIX (Charwat and Dvořák 2012) (for the area of abstract argumentation) are available. However, these systems are either designed as tools for prototypical DP implementations, are not easily extensible to new application areas or suffer from high memory demands for storing partial solutions during the computation. The latter problem has been addressed, e.g., by proposing heuristics (Betzler et al. 2006) or reducing the number of simultaneously stored tables (Aspvall et al. 2000).

BDDs are a well-established concept used, e.g., in model-checking (Męski et al. 2014), planning (Kissmann and Hoffmann 2014) and software verification (Beyer and Stahlbauer 2014). In recent research the effectiveness of exploiting tree-width by applying decomposition techniques in combination with decision diagrams is studied. In the area of knowledge compilation, so-called "Tree-of-BDDs" (Subbarayan 2005; Fargier and Marquis 2009) are constructed in an offline phase from a given CNF, and queried in the online phase to answer questions on this data structure in linear time. Furthermore, Algebraic Decision Diagrams (ADDs) (Bahar et al. 1997), a concept closely related to BDDs, are used for compiling Bayesian networks in such a way that the structure of the network can be exploited in order to compute inference efficiently (Chavira and Darwiche 2007). Combining DP and decision diagrams has been proven well-suited also for Constraint Optimization Problems (COPs) (Sachenbacher and Williams 2005). The key idea is to employ ADDs to store the set of possible solutions, and the branch-and-bound algorithm is executed on a decomposition of the COP instance. In (Boutaleb et al. 2006), this idea was shown to be superior to earlier approaches by additionally applying (no)good recording during computation.

Furthermore, in (Hooker 2013) "classical" (i.e., without tree decompositions) serial as well as non-serial dynamic programming in combination with decision diagrams is studied for optimization problems.

4 Goal and Current Status of Research

4.1 Goal of the Research

The goal of the thesis is to develop a new methodology for DP on tree decompositions. The following tasks are to be considered:

1. *Sets-of-models based DP algorithms*: One goal is to develop a general approach where DP algorithms on tree decompositions can be specified on a logical level in form of Boolean formula manipulations, such that they can be handled

efficiently and transparently by the underlying BDD data structure. Based on this, alternative algorithm design patterns shall be developed and analyzed.

2. *Study of expressiveness*: An important question to answer is whether and how different types of problems (e.g., decision or optimization problems) can be addressed. Furthermore, it shall be elaborated how problems of different (classical) complexity can be handled, in particular NP-complete problems and problems that are hard for the second level of the polynomial hierarchy. Ultimately, it would be desirable to prove whether all MSO-definable problems can be modeled using our approach such that the resulting algorithms run in fpt time (w.r.t. tree-width).
3. *Implementation*: From a practical perspective, the goal is to develop a system that combines BDDs with DP on tree decompositions. This tool should rely on existing software (HTDECOMP (Dermaku et al. 2008) for obtaining the decomposition, D-FLAT (Abseher et al. 2014) for handling the program flow and CUDD (Somenzi 2012) for BDD management).
4. *Empirical evaluation*: It is expected that the new approach of combining BDDs with DP results in much lower memory requirements than previous attempts. Within the thesis, the memory demands as well as the run-time performance is compared to existing decomposition-based systems, and also to "direct" implementations that are not based on such decompositions. Of particular interest is a comparison on real-world instances that exhibit small tree-width (such as, e.g., provided in (Batagelj and Mrvar 2006; Langer 2012; Bliem 2012)). Furthermore, the influence of the variable ordering within the BDDs on the run-time performance shall be studied.

Overall, the goal is to develop a method that *combines DP on tree decompositions with BDDs* and to *give a clear picture on the applicability of this approach in practice*.

4.2 Status of Research

In recent work (see (Charwat and Woltran 2015)) we accomplished to formalize our approach for NP-complete decision problems. The applied methodology is feasible for problems that are fpt w.r.t. tree-width k, but can also be applied to problems that are not fpt. However, there the run-time complexity is no longer polynomial in the size of the input (for a fixed k). We studied several problems that impose different challenges on our algorithm design, namely 3-Colorability, Boolean Satisfiability, Directed Dominating Set and Hamiltonian Cycle.

Here, we briefly illustrate how the simplest problem, 3-Colorability ("Given a graph G, is G 3-colorable?"), can be solved using our approach. We assume that the tree decomposition $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$ of $G = (V, E)$ is *normalized*, meaning that each node $t \in V_\mathcal{T}$ is of one of the following types: (l) leaf node: t has no children; (i) introduction node: t has exactly one child node t' with $X_{t'} \cup \{u\} = X_t$ where u is the introduced vertex; (r) removal node: t has exactly one child node t' with $X_{t'} = X_t \cup \{u\}$ where u is the removed vertex; and (j) join node: t has exactly two child nodes t' and t'' with $X_t = X_{t'} = X_{t''}$.

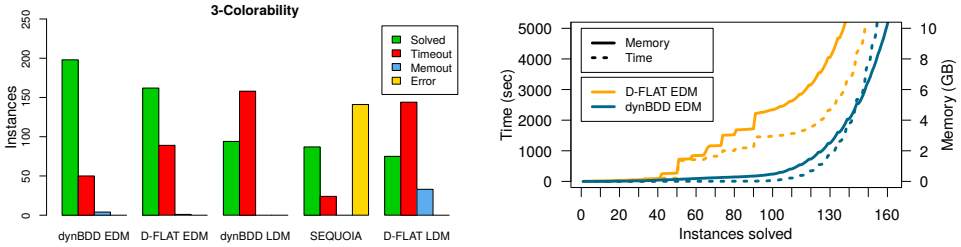

Fig. 3. 3-COLORABILITY: Benchmark comparison with available TD-based systems.

Let the set of colors $C = \{r, g, b\}$. For all $c \in C$ and $x \in V$, the truth value of variable c_x denotes whether vertex x gets assigned color c. The algorithm now works as follows. We traverse $\mathcal{T}$ in post-order. At each node $t \in V_{\mathcal{T}}$ we compute the BDD $\mathcal{B}_t^*$ based on the node type $* \in \{l, i, r, j\}$. Additionally, we take into account the BDDs $\mathcal{B}_{t'}$ and $\mathcal{B}_{t''}$ of the child nodes (if any), and the subgraph of G that is induced by the vertices in X_t. Here, we denote by E_t the edges contained in the induced subgraph. The BDD manipulation operations are given as follows:

$$\begin{aligned}
\mathcal{B}_t^l =& \bigwedge_{c \in C} \bigwedge_{\{x,y\} \in E_t} \neg(c_x \wedge c_y) \wedge \bigwedge_{x \in X_t} (r_x \vee g_x \vee b_x) \wedge \\
& \bigwedge_{x \in X_t} \Big(\neg(r_x \wedge g_x) \wedge \neg(r_x \wedge b_x) \wedge \neg(g_x \wedge b_x) \Big) \\
\mathcal{B}_t^i =& \mathcal{B}_{t'} \wedge \bigwedge_{c \in C} \bigwedge_{\{x,u\} \in E_t} \neg(c_x \wedge c_u) \wedge (r_u \vee g_u \vee b_u) \wedge \\
& \neg(r_u \wedge g_u) \wedge \neg(r_u \wedge b_u) \wedge \neg(g_u \wedge b_u) \\
\mathcal{B}_t^r =& \exists r_u g_u b_u [\mathcal{B}_{t'}] \qquad\qquad \mathcal{B}_t^j = \mathcal{B}_{t'} \wedge \mathcal{B}_{t''}
\end{aligned}$$

For a graph to be 3-colorable we have to guarantee that adjacent vertices do not have the same color and that every vertex gets assigned exactly one color. Intuitively, $\mathcal{B}_t^l$ and $\mathcal{B}_t^i$ are constructed by adding the respective constraints for introduced vertices. In $\mathcal{B}_t^r$, due to the definition of tree decompositions, we know that all constraints related to removed vertex u were already taken into account. Hence, we can abstract away the variables associated with u, thereby keeping the size of the BDD bound by the width of the decomposition. In join nodes, $\mathcal{B}_t^j$ combines the intermediate results obtained in the child nodes of the decomposition. At the root node r of $\mathcal{T}$ (where we impose $X_r = \emptyset$) we have that either $\mathcal{B}_r = \top$ or $\mathcal{B}_r = \bot$, representing the solution to our problem.

We call the approach illustrated above *early decision method* (EDM), meaning that information is incorporated into the BDD as soon as it becomes available (i.e., during introduction of vertices). In (Charwat and Woltran 2015) we developed an alternative algorithm design pattern, called *late decision method* (LDM). In contrast to EDM, here the BDDs are updated just before a vertex is removed from a bag.

All problems mentioned above were implemented in both the EDM and LDM variants, and compared to TD-based systems in a preliminary empirical evaluation.

Results for 3-Colorability are shown in Figure 3 (for details and further results, see (Charwat and Woltran 2015)). The benchmark set contained 252 (randomly generated) instances consisting of 10 up to 1000 vertices. The obtained tree decompositions exhibited a width between 1 and 944. The figure on the left illustrates the number of solved instances, as well as the overall number of observed timeouts (limited to 10 minutes) and memouts (limited to 4GB). SEQUOIA reports "error" whenever the system-internal pre-check determines that the given instance is not solvable. The accumulated runtime and memory requirements over all instances that were solved by the two best-performing systems is given in the right figure. In total, dynBDD (using EDM) required approximately 18% less time and 47% less memory than D-FLAT (EDM). Our experiments suggest that EDM is better-suited for unsatisfiable instances since conflicts can be detected earlier during the computation. However, LDM usually yields smaller BDDs and results in less computational effort. Furthermore, from a developer's perspective, we obtain more concise algorithm specifications. Overall, benchmark results are very promising for all problems considered so far, both with respect to reduced memory requirements and run-time.

5 Open Issues and Expected Achievements

The performance of BDDs heavily depends on the applied variable ordering. An open question is how this ordering can be optimized in our context, e.g., by taking the ordering of the vertices in the tree decomposition into account. Furthermore, from a practical perspective, also debugging and visualization possibilities that may support an algorithm developer are under investigation. It is desirable to develop further algorithm design pattern (besides EDM and LDM) and to study their implications on the run-time performance. Additionally, our approach natively supports parallel problem solving (over decomposition branches), which would be a complementary approach to recent developments on parallel BDD implementations (van Dijk et al. 2013; Lovato et al. 2014). Additionally, it would be interesting to develop problem-specific optimizations for our TD-based algorithms and compare their performance to standard (non-DP) implementations (e.g., for Hamiltonian Cycle (Gebser et al. 2014; Soh et al. 2014)).

Regarding expressiveness, we are interested in optimization problems. Here, alternative types of decision diagrams (DDs), such as Algebraic DDs (Bahar et al. 1997), Edge-valued BDDs (Lai and Sastry 1992) and Multi-valued DDs (Kam et al. 1998) may be appropriate. Most importantly, we want to study our approach in the context of problems that are hard for the second level of the polynomial hierarchy. Here, a challenge to be faced is that each partial solution get associated with a *set* of so-called *counter candidates*, that witness whether a partial solution is valid (see, e.g., (Dvořák et al. 2012)). Here, an open issue is how such sets of counter candidates can be represented in BDDs. Ultimately we want to provide a BDD-based tool-set for various intractable problems, such that they can be solved in fpt time within our approach.

References

Abseher, M., Bliem, B., Charwat, G., Dusberger, F., Hecher, M., and Woltran, S. 2014. The D-FLAT system for dynamic programming on tree decompositions. In *Proc. JELIA*. LNCS, vol. 8761. Springer, 558–572.

Akers, S. B. 1978. Binary decision diagrams. *IEEE Transactions on Computers 100,* 6, 509–516.

Arnborg, S., Corneil, D. G., and Proskurowski, A. 1987. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods 8*, 277–284.

Aspvall, B., Telle, J. A., and Proskurowski, A. 2000. Memory requirements for table computations in partial k-tree algorithms. *Algorithmica 27,* 3, 382–394.

Bahar, R., Frohm, E., Gaona, C., Hachtel, G., Macii, E., Pardo, A., and Somenzi, F. 1997. Algebric decision diagrams and their applications. *Formal Methods in System Design 10,* 2-3, 171–206.

Batagelj, V. and Mrvar, A. 2006. Pajek datasets. `http://vlado.fmf.uni-lj.si/pub/networks/data/`.

Betzler, N., Niedermeier, R., and Uhlmann, J. 2006. Tree decompositions of graphs: Saving memory in dynamic programming. *Discrete Optimization 3,* 3, 220–229.

Beyer, D. and Stahlbauer, A. 2014. BDD-based software verification - Applications to event-condition-action systems. *STTT 16,* 5, 507–518.

Bliem, B. 2012. D-FLAT: Collection of instances with small tree-width. `http://dbai.tuwien.ac.at/proj/dflat/system/examples/instances.tar.gz`.

Bodlaender, H. L. and Koster, A. M. C. A. 2010. Treewidth computations I. Upper bounds. *Inf. Comput. 208,* 3, 259–275.

Bollig, B. and Wegener, I. 1996. Improving the variable ordering of OBDDs is NP-complete. *IEEE Trans. Comp. 45,* 9 (Sep), 993–1002.

Boutaleb, K., Jégou, P., and Terrioux, C. 2006. (No)good recording and ROBDDs for solving structured (V)CSPs. In *Proc. ICTAI*. IEEE Computer Society, 297–304.

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers 100,* 8, 677–691.

Charwat, G. and Dvořák, W. 2012. dynPARTIX 2.0 - Dynamic programming argumentation reasoning tool. In *Proc. COMMA*. FAIA, vol. 245. IOS Press, 507–508.

Charwat, G., Dvořák, W., Gaggl, S. A., Wallner, J. P., and Woltran, S. 2015. Methods for solving reasoning problems in abstract argumentation - A survey. *Artif. Intell. 220*, 28–63.

Charwat, G. and Woltran, S. 2015. Efficient problem solving on tree decompositions using binary decision diagrams. In *Proc. LPNMR*. LNCS. Springer. Accepted for publication.

Chavira, M. and Darwiche, A. 2007. Compiling Bayesian networks using variable elimination. In *Proc. IJCAI*. 2443–2449.

Chimani, M., Mutzel, P., and Zey, B. 2012. Improved Steiner tree algorithms for bounded treewidth. *J. Discrete Algorithms 16*, 67–78.

Courcelle, B. 1990. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput. 85,* 1, 12–75.

Dermaku, A., Ganzow, T., Gottlob, G., McMahan, B. J., Musliu, N., and Samer, M. 2008. Heuristic methods for hypertree decomposition. In *Proc. MICAI*. LNCS, vol. 5317. Springer, 1–11.

Dvořák, W., Pichler, R., and Woltran, S. 2012. Towards fixed-parameter tractable algorithms for abstract argumentation. *Artif. Intell. 186*, 1–37.

Fargier, H. and Marquis, P. 2009. Knowledge compilation properties of Trees-of-BDDs, revisited. In *Proc. IJCAI*. 772–777.

Friedman, S. J. and Supowit, K. J. 1987. Finding the optimal variable ordering for binary decision diagrams. In *Proc. IEEE Design Automation Conf.* ACM, 348–356.

Gebser, M., Janhunen, T., and Rintanen, J. 2014. SAT modulo graphs: Acyclicity. In *Proc. JELIA*. LNCS, vol. 8761. Springer, 137–151.

Groër, C., Sullivan, B. D., and Weerapurage, D. 2012. INDDGO: Integrated network decomposition & dynamic programming for graph optimization. Tech. Rep. ORNL/TM-2012/176.

Hooker, J. N. 2013. Decision diagrams and dynamic programming. In *Proc. CPAIOR*. LNCS, vol. 7874. Springer, 94–110.

Kam, T., Villa, T., Brayton, R., and Sangiovanni-Vincentelli, A. 1998. Multi-valued decision diagrams: Theory and applications. *Multiple-Valued Logic* 4, 9–62.

Kissmann, P. and Hoffmann, J. 2014. BDD ordering heuristics for classical planning. *J. Artif. Intell. Res. (JAIR) 51*, 779–804.

Kneis, J., Langer, A., and Rossmanith, P. 2011. Courcelle's theorem - A game-theoretic approach. *Discrete Optimization 8,* 4, 568–594.

Lai, Y.-T. and Sastry, S. 1992. Edge-valued binary decision diagrams for multi-level hierarchical verification. In *Proc. DAC*. IEEE CSP, 608–613.

Langer, A. 2012. SEQUOIA library and tests. `https://github.com/sequoia-mso`.

Lee, C. 1959. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal 38*, 985–999.

Lovato, A., Macedonio, D., and Spoto, F. 2014. A thread-safe library for binary decision diagrams. In *Proc. SEFM*. LNCS, vol. 8702. Springer, 35–49.

Męski, A., Penczek, W., Szreter, M., Woiźna-Szcześniak, B., and Zbrzezny, A. 2014. BDD-versus SAT-based bounded model checking for the existential fragment of linear temporal logic with knowledge: Algorithms and their performance. *Autonomous Agents and Multi-Agent Systems 28,* 4, 558–604.

Morak, M., Musliu, N., Pichler, R., Rümmele, S., and Woltran, S. 2011. A new tree-decomposition based algorithm for answer set programming. In *ICTAI*. IEEE, 916–918.

Niedermeier, R. 2006. Invitation to fixed-parameter algorithms. Oxford Lecture Series in Mathematics and its Applications, vol. 31. OUP, Oxford.

Pichler, R., Rümmele, S., and Woltran, S. 2009. Belief revision with bounded treewidth. In *Proc. LPNMR*. LNCS, vol. 5753. Springer, 250–263.

Robertson, N. and Seymour, P. D. 1984. Graph minors. III. Planar tree-width. *J. Comb. Theory, Ser. B 36,* 1, 49–64.

Rudell, R. 1993. Dynamic variable ordering for ordered binary decision diagrams. In *Proc. ICCAD*. IEEE CSP, 42–47.

Sachenbacher, M. and Williams, B. C. 2005. Bounded search and symbolic inference for constraint optimization. In *Proc. IJCAI*. PBC, 286–291.

Soh, T., Berre, D. L., Roussel, S., Banbara, M., and Tamura, N. 2014. Incremental sat-based method with native boolean cardinality handling for the Hamiltonian cycle problem. In *Proc. JELIA*. LNCS, vol. 8761. Springer, 684–693.

Somenzi, F. 2012. *CU Decision Diagram package release 2.5.0*. Department of Electrical and Computer Engineering, University of Colorado at Boulder.

Subbarayan, S. 2005. Integrating CSP decomposition techniques and BDDs for compiling configuration problems. In *Proc. CPAIOR*. LNCS, vol. 3524. Springer, 351–365.

van Dijk, T., Laarman, A., and van de Pol, J. 2013. Multi-core BDD operations for symbolic reachability. *Electr. Notes Theor. Comput. Sci. 296*, 127–143.