Journal of Artificial Intelligence Research 51 (2014) 779–804 Submitted 09/14; published 12/14

Research Note

BDD Ordering Heuristics for Classical Planning

Peter Kissmann KISSMANN@CS.UNI-SAARLAND.DE
Jörg Hoffmann HOFFMANN@CS.UNI-SAARLAND.DE
Saarland University, Saarbrücken, Germany

Abstract

Symbolic search using binary decision diagrams (BDDs) can often save large amounts of memory due to its concise representation of state sets. A decisive factor for this method's success is the chosen variable ordering. Generally speaking, it is plausible that dependent variables should be brought close together in order to reduce BDD sizes. In planning, variable dependencies are typically captured by means of causal graphs, and in preceding work these were taken as the basis for finding BDD variable orderings. Starting from the observation that the two concepts of "dependency" are actually quite different, we introduce a framework for assessing the strength of variable ordering heuristics in sub-classes of planning. It turns out that, even for extremely simple planning tasks, causal graph based variable orders may be exponentially worse than optimal.

Experimental results on a wide range of variable ordering variants corroborate our theoretical findings. Furthermore, we show that dynamic reordering is much more effective at reducing BDD size, but it is not cost-effective due to a prohibitive runtime overhead. We exhibit the potential of middle-ground techniques, running dynamic reordering until simple stopping criteria hold.

1. Introduction

Finding good variable orderings is an important task in many areas of Artificial Intelligence, such as constraint satisfaction problems (CSPs), SAT, and planning (for some heuristic search approaches, but especially when applying symbolic search). In many cases, an efficient ordering is determined by evaluating a graphical representation of the underlying problem. For CSPs, for example, the constraint graph can be used to determine a variable ordering for backtracking-based approaches. Typical approaches take the minimum width (Freuder, 1982), maximum degree, or maximum cardinality (Dechter & Meiri, 1989) of nodes in the constraint graph into account. An alternative approach considers the bandwidth of the constraint graph under a given ordering, which is the maximal distance in that ordering of any two nodes that are adjacent in the graph; the idea is to find an ordering that minimizes the bandwidth (Zabih, 1990).

In SAT, a widely used approach to determine the variable order in conflict-driven clause learning (CDCL) is variable state independent decaying sum (VSIDS) (Moskewicz, Madigan, Zhao, Zhang, & Malik, 2001). This is based on the weights of propositional variables, i.e., how often such a variable occurs in the clauses. Recently, Rintanen (2012) noted that for applying SAT solvers to planning tasks, a different ordering might be more efficient, giving better coverage in the typical benchmarks of the international planning competition (IPC). This ordering takes the structure of planning tasks into account, trying to support (sub)goals as early on as possible.

In planning, variable dependencies are typically represented by the causal graph (e.g., Knoblock, 1994; Jonsson & Bäckström, 1995; Brafman & Domshlak, 2003; Helmert, 2006), capturing variable dependencies in terms of co-occurences in action descriptions. This kind of graph has turned out to be useful for a great variety of purposes, including problem decomposition (Knoblock, 1994), system design (Williams & Nayak, 1997), complexity analysis (e.g. Jonsson & Bäckström, 1995; Domshlak & Dinitz, 2001; Brafman & Domshlak, 2003; Katz & Domshlak, 2008; Giménez & Jonsson, 2008; Chen & Giménez, 2010), derivation of heuristic functions (Helmert, 2004, 2006), and search topology analysis (Hoffmann, 2011b, 2011a). For our purposes here, the causal graph's most relevant application is the derivation of variable orderings. That has been done for BDDs, to which we return in detail below, as well as for merge-and-shrink heuristics (Helmert, Haslum, & Hoffmann, 2007; Helmert, Haslum, Hoffmann, & Nissim, 2014). In merge-and-shrink, a complete variable ordering corresponds to a (linear) merging strategy, an order in which variables are merged into a global abstraction. In a recent extension to non-linear merging strategies (Sievers, Wehrle, & Helmert, 2014), the order of the merges is instead given by a tree. Such a merge tree bears some similarity to the concept of vtrees, which are used as a generalization of variable orderings for sentential decision diagram (SDDs) (Darwiche, 2011). Fan, Müller, and Holte (2014) have shown that efficient merge trees can be determined by means of the causal graph. To do so, they use Min-Cuts in the causal graph, putting the two resulting sets of variables into two different branches of the merge tree and recursively continue in the subgraphs.

In this paper, we are concerned with symbolic search based on binary decision diagrams (BDDs) (Bryant, 1986) for optimal planning. A variable ordering here refers to the order in which variables are queried within the BDDs, a key ingredient for the practical efficiency of the approach. In planning, not much work has been invested into finding good variable orderings, but in model checking, where symbolic search originated (McMillan, 1993), many different variable ordering schemes have been proposed in the past (e.g., Malik, Wang, Brayton, & Sangiovanni-Vincentelli, 1988; Minato, Ishiura, & Yajima, 1990). Again, many of those are based on the evaluation of a graphical representation of the problem. Often, bringing “dependent variables” close together results in smaller BDDs. This can be straightforwardly applied to planning, by defining “variable dependencies” via the causal graph. That is exactly how Gamer, a state-of-the-art symbolic search planner, determines its variable ordering (Kissmann & Edelkamp, 2011).

The starting point of our investigation is a feeling of discomfort with the double use of the word “dependency” in the above. In causal graphs, such a dependency means that the corresponding variables appear in at least one common action, so changing the value of one variable may require changing the other variable as well. BDDs, on the other hand, represent Boolean functions φ. If many assignments to a subset P of all variables immediately determine the truth value of φ, independently of the value of the other variables, then the variables in P should be grouped closely together. In planning, φ typically represents a layer of states sharing the same distance to the initial state (forward search) or the goal (backward search). So the concept of “dependence” here relates to determining whether or not a state is a member of such a layer. *What, if anything, does this have to do with causal graph dependencies?*

We do not have a conclusive answer to that question, but we contribute a number of insights suggesting that the two concepts of dependence do not have much in common. We consider the issue from both a theoretical and a practical perspective. On the theoretical side, we introduce a simple formal framework for assessing the strength of variable ordering heuristics in sub-classes of planning. Applying that framework to causal graph based variable orders, we show that these may be exponentially worse than optimal orderings, even for extremely simple planning tasks.

On the practical side, we experiment with a wide range of variable ordering schemes, several ones based on the causal graph, and also a range of techniques adapted from the model checking literature. To get an idea of "how good" these ordering schemes are, on the grand scale of things, we use an upper and a lower "delimiter". For the latter, we use random variable orderings. Not too surpisingly, most ordering schemes are better than random; surprisingly, not all of them are. Indeed, Fast Downward's "level" heuristic (Helmert, 2006) turns out to be much worse than the average random BDD variable ordering.

As the upper delimiter, we employ dynamic reordering techniques that minimize BDD size online, during the construction process. Compared to static up-front variable ordering schemes, such reordering has a much better basis for taking decisions, but is much more time-consuming. It is thus expected for the BDD size results to be much better. The extent to which that happens in our experiments is remarkable, however: Static orderings are hardly ever even a tiny bit better, whereas the advantage of dynamic reordering easily and frequently goes up to three orders of magnitude.

While it has been successfully employed in at least one domain in non-deterministic planning (Cimatti, Pistore, Roveri, & Traverso, 2003), dynamic reordering usually is prohibitively slow and not cost-effective. Still, its prowess at reducing BDD size, combined with our pessimistic outlook on static ordering schemes, suggests that it may be the better alternative. An initial experiment indicates that this could, indeed, be the case: With very simple adaptive stopping criteria, running dynamic reordering only up to a certain point, we obtain better results than with any of the static ordering schemes.

The remainder of the paper is organized as follows. Section 2 gives the necessary background on our planning framework and the use of BDDs. Section 3 introduces our theoretical framework and investigates the properties of causal graph based ordering schemes in a range of well-known planning sub-classes. Section 4 presents our experiments regarding the quality of causal graph based ordering schemes, and Section 5 presents our experiments with adaptive stopping criteria for dynamic reordering. Section 6 concludes the paper with a brief discussion and outlook.

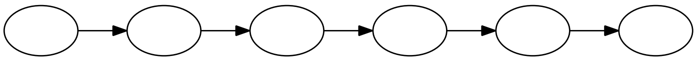
This research note is an extension of the authors' previous short conference paper (Kissmann & Hoffmann, 2013). The present paper contains comprehensive details regarding the technical background and the variable orderings we implemented, and it includes full proofs. The experiments with adaptive stopping criteria for dynamic reordering, Section 5, are new.

2. Background

For BDD-based planning, as argued e.g. by Edelkamp and Helmert (1999), it is important to have a small encoding of the given planning task. So we use a finite-domain variable representation as the basis for our investigation. A **finite-domain representation (FDR)** planning task is a tuple $\Pi = \langle V, A, I, G\rangle$, where V is the set of state variables and each $v \in V$ is associated with its finite domain $\mathcal{D}(v)$. A is a finite set of actions where each $a \in A$ is a pair $\langle pre_a, \mathit{eff}_a\rangle$ of partial assignments to V with pre_a being the precondition and eff_a the effect of action a. The initial state I is a complete assignment to V. The goal G is a partial assignment to V. By $\mathcal{V}(pa)$, for a partial assignment pa, we denote the variables $v \in V$ where $pa(v)$ is defined.

An action $a \in A$ is applicable in a state s iff $pre_a \subseteq s$. For the resulting successor state s' it holds that $s'(v) = \mathit{eff}_a(v)$ for all $v \in \mathcal{V}(\mathit{eff}_a)$ and $s'(v) = s(v)$ for all $v \in V \setminus \mathcal{V}(\mathit{eff}_a)$. A **plan** is a sequence of actions whose successive application starting in the initial state results in a state s_g with $G \subseteq s_g$. A plan is optimal if no plan of shorter length exists.

Binary decision diagrams (BDDs) as introduced by Bryant (1986) represent Boolean functions φ. A BDD β is a directed acyclic graph with one root and two terminal nodes, the 0-sink and the



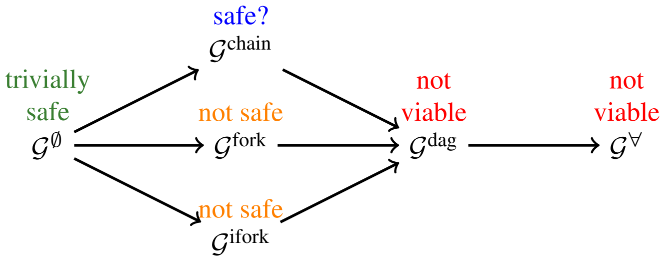


Figure 11: Overview of our classification results. These hold for each of Ω^{ga} and Ω^{cg}, and for each search direction.

variables (see Figure 10 for illustration). Any ordering scheme including only such latter orderings is not viable.

For the forward search direction case, we construct a planning task where all x and y variables are ternary (unknown, true ($\top$), false ($\bot$)), and are unknown initially. The value of x_1 can be set freely; y_i can be set to true or false if x_i is true, and can only be set to true if x_i is false; x_{i+1} can be set freely once y_i has been set to either true or false. In $2n$ steps, we can reach exactly the states characterized by $Q(\wedge, \vee) = \bigwedge_{i=1}^{n}(x_i = \top) \vee (y_i = \top)$. A BDD representing $Q(\wedge, \vee)$ is of exponential size if, e.g., all x variables are placed before all y variables, and any ordering scheme including only such orderings is not viable. $\square$

Note that, for both planning task families $\{\Pi_n\}$ just described, both Ω^{ga} and Ω^{cg} and force the x_i and y_i variable to be ordered in pairs, resulting in BDDs of minimal size (see Figure 10a). In that sense, these two planning task families constitute our only truly positive result: Within them, the ordering information in the causal graph keeps us from making exponentially bad mistakes. That positive message would be much stronger if Ω^{ga} and Ω^{cg} were safe for all families of tasks with chain causal graphs. It remains an open question whether that is so.

Figure 11 gives an overview of our results. The evidence speaks rather clearly against a strong connection between causal graph dependencies and dependencies as relevant for BDD size. Note that the causal graph underlying Theorem 4 – non-viability for $FDR(\mathcal{G}^{\text{dag}})$ – has a very simple form combining a chain with an inverted fork, and that Theorem 2 – non-safety for $FDR(\mathcal{G}^{\text{ifork}})$ – relies on planning tasks which fall into a known syntactically identified tractable class for optimal planning (Katz & Domshlak, 2010). Note also that being "not safe" already is quite bad in practice, incurring an exponential risk unless we have a clever way of choosing an ordering *within* $\Omega(\Pi)$ (which, at the moment, we do not have).

4. What's in a Causal Graph: Practice

While we have shown poor worst-case performance of causal graph based variable ordering schemes in theory, practice might be another matter. To assess the latter, we implemented a comprehensive set of causal graph based variable ordering schemes, comprising 12 such schemes in total, and ran them in comparison to practical "good"/"bad" delimiters. As the "bad" delimiter, we used random orderings. As the "good" delimiter, we used the off-the-shelf dynamic reordering algorithm of Gamer's BDD package CUDD, which is based on sifting (Rudell, 1993).

A few words are in order regarding how sifting works. The variable with the greatest number of nodes in the current BDD is chosen. It is first moved towards the end of the ordering, then towards the beginning of the ordering, by iteratively swapping its position with the next variable in the corresponding direction. Once all positions have been tried, the variable is moved to the position where the BDD size was smallest. This done, the next variable is chosen, until all variables have been processed. For better comparability with our ordering schemes, we restrict the algorithm to keep the variables representing each $\gamma(v)$ together.

As previously indicated, dynamic reordering consumes too much runtime to be cost-effective. In the present experiments, where we are interested only in BDD size, we give dynamic reordering ample runtime. In Section 5, we will identify simple adaptive criteria for stopping dynamic reordering automatically during the search, taking advantage of its size reduction capacity without suffering too much from its runtime consumption.

We ran the benchmarks of the 2011 International Planning Competition (IPC'11), and we used Gamer as the base implementation for all planners, running them on one core of an Intel Xeon X5690 CPU with 3.47 GHz. Unless otherwise stated, we used the IPC'11 settings, namely a timeout of 30 minutes and a memory limit of 6 GB.

4.1 Ordering Schemes

We ran six schemes based directly on the causal graph:

Gamer is Gamer's original ordering scheme, which approximates Ω^{ga}.

GamerPre is like *Gamer* but on a causal graph extended by arcs between pairs of precondition variables. The idea here is to capture not only dependency for forward search, but also for backward search, i.e., when inverting the actions.

WGamer is like *Gamer* but with arcs weighted by the number of relevant actions, i.e., the number of actions inducing the corresponding arcs.

WGamerPre is like *GamerPre* with weighted arcs.

CGLevel is Fast Downward's (Helmert, 2006) level heuristic, which approximates Ω^{cg}. It orders the variables by strongly connected components and, within these components, considers the weighted causal graph and orders variables with smallest incoming weight first. Similar to *WGamer*, the weights correspond to the number of actions that induce an arc.

CGSons is another approximation of Ω^{cg}. It always selects a variable v all of whose parents have already been selected; or at least one of whose parents has already been selected; or an arbitrary variable if no such v exists.

Additionally, we used six ordering schemes we adopted from the model checking literature, based on a structure called the *abstract syntax tree* (AST) (e.g., Maisonneuve, 2009). That is a directed graph containing a root node for the overall task and subtrees for all actions. Each subtree consists of nodes representing the subformulas of the specified action (i.e., subformulas for the action's precondition and its effect). The variables of the task are the leaves of the AST. The leaves are merged, i.e., we have only one node for each variable of the task. Edges point from a node representing some function to all corresponding subtrees.

We construct the AST based on the PDDL input. Consider the following example actions, similar to those in the Floortile domain. We have predicates $at(r,t)$, denoting the tile t robot r is

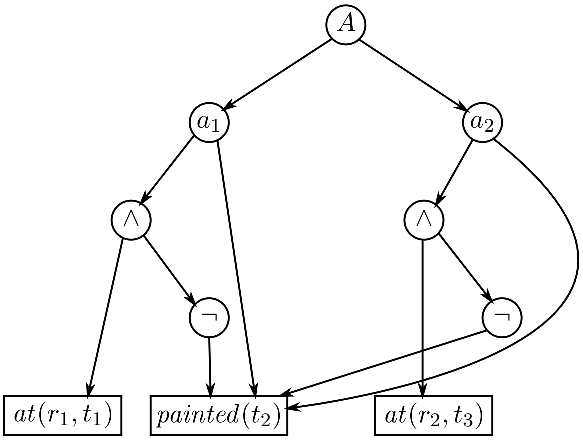


Figure 12: Example AST.

currently on and $painted(t)$ denoting whether tile t has already been painted. We have two actions $a_1 = paint(r_1, t_1, t_2)$ with precondition $(at(r_1, t_1) \wedge \neg painted(t_2))$ and effect $(painted(t_2))$ denoting that a robot r_1 can paint tile t_2 if it currently is on t_1 and t_2 has not been painted. Similarly, action $a_2 = paint(r_2, t_3, t_2)$ with precondition $(at(r_2, t_3) \wedge \neg painted(t_2))$ and effect $(painted(t_2))$ denotes that robot r_2 can paint tile t_2 if it currently is on t_3 and t_2 has not been painted.

Figure 12 illustrates the corresponding AST. We have the root for all actions A, and one subtree for each of the two actions a_1 and a_2. For both actions the preconditions and effects are encoded but we retain only one copy of each variable in the leaves (here relevant only for $painted(t_2)$).

Using their first author's names for reference, the additional ordering schemes are the following.

Butler (Butler, Ross, Kapur, & Mercer, 1991) is an extension of an approach by Fujita, Fujisawa, and Kawato (1988). The latter proposed to perform a depth-first search (DFS) in the AST, starting at the root node, and to order the variables in the order in which they are reached the first time. Butler et al. extended this to a setting with several roots (if we remove the overall root and retain only the subtrees for the various actions we arrive at exactly the same setting). Their approach starts the DFS at the action containing the highest number of variables. Within the tree it advances in a similar manner: It always continues with the subtree that contains the highest number of variables among all subtrees of the current node. The retrieved ordering is then again in the order in which the variables are reached the first time.

Chung1 (Chung, Hajj, & Patel, 1993) is a two-step approach. In the first step it assigns values to all nodes of the AST. Starting at the leaves, assigning them a value of 0, it assigns each inner node the maximum of the values assigned to its successors plus 1. In the second step it performs a DFS starting at the root, which is guided by the values of the nodes, visiting those successors with highest value first. The order in which the variables are reached for the first time is then chosen as the variable ordering.

Chung2 (Chung et al., 1993) determines the shortest distance between each pair of variables, which can be calculated by considering all edges in the AST as undirected. Additionally, the total distances, i.e., the sum of the minimal distances to all other variables, are stored for all variables. A variable with smallest total distance is chosen first. The next one is the one closest to the last variable inserted into the ordering. In case of a tie the distance to the preceding variables is also taken into account.

Maisonneuve (Maisonneuve, 2009) is a greedy approach starting with an empty sequence. In each step it temporarily extends the current sequence by a variable not yet in the sequence. For this variable, a weight is determined, which is the number of variables from this extended sequence that appear in an action, summed over all actions. The variable is then removed from the sequence and the next one added. When all weights are calculated the variable with highest weight is appended to the sequence, and the next iteration starts, calculating the new weights for the remaining variables. In the end, the last sequence contains all variables and thus corresponds to the variable ordering.

Malik (Malik et al., 1988) assigns a level value (the maximal level of all its predecessors plus 1) to each node within the AST. The root is assigned value 0. The variables are then ordered according to their level values, those with highest values coming first.

Minato (Minato et al., 1990) calculates weights for all nodes in the AST. The weight of the root node of each action is set to 1, that of all successors of a node to w/m if w is the node's weight and m the number of successors of that node. One of the variables with highest weight is then chosen first and all its nodes removed (along with all ingoing edges and – recursively – those nodes with no remaining successors). For the reduced graph the weights are recalculated and the procedure continues until finally all variables are in the ordering.

4.2 Bad Delimiter

To get the bad delimiter we ran 5000 "random orderings", where each such ordering corresponds to one run of the IPC'11 benchmark tasks, using a random variable ordering for each instance. To make this feasible we used a time-out of one minute (our backward search implementation is not viable for such a short time-out, so that we use only forward search here). For comparison to this data, the same settings (1 minute time-out, only forward search) was used with the twelve static ordering schemes. Initially we ran all ordering schemes and the random orderings on all tasks; after 200 random runs we removed all tasks from the benchmark set that were not solved at least once during the previous (random or static ordering) runs, retaining 85 tasks. Figure 13 shows coverage, i.e., the number of solved planning tasks, on the x axis, and the fraction of random orderings having that coverage on the y axis. The coverage data for the ordering schemes are shown as vertical lines.

Malik and *CGLevel* lie in and below the middle of the Gaussian distribution, respectively. In other words, *Malik* is as bad as, and *CGLevel* is even worse than, the average random ordering. Matters are not as bleak for the other ten ordering schemes, which are close together and lie clearly above the Gaussian distribution. Compared to a best-of over the random orders, however, all the ordering schemes appear rather humble. Consider Table 1. In particular, consider $n_{+\Omega}^{-r}$, giving the number of instances solved by ordering scheme Ω but not by any random order, and consider $n_{-\Omega}^{+r}$, giving the number of instances not solved by the scheme but solved by some random order. As Table 1 shows, $n_{+\Omega}^{-r}$ is strictly smaller than $n_{-\Omega}^{+r}$ in all but three of the ordering schemes Ω, and is strictly larger (by a single task) only for one of the schemes (namely *Butler*). The average over $n_{+\Omega}^{-r}$ is 2.92 while that over $n_{-\Omega}^{+r}$ is 9.08.

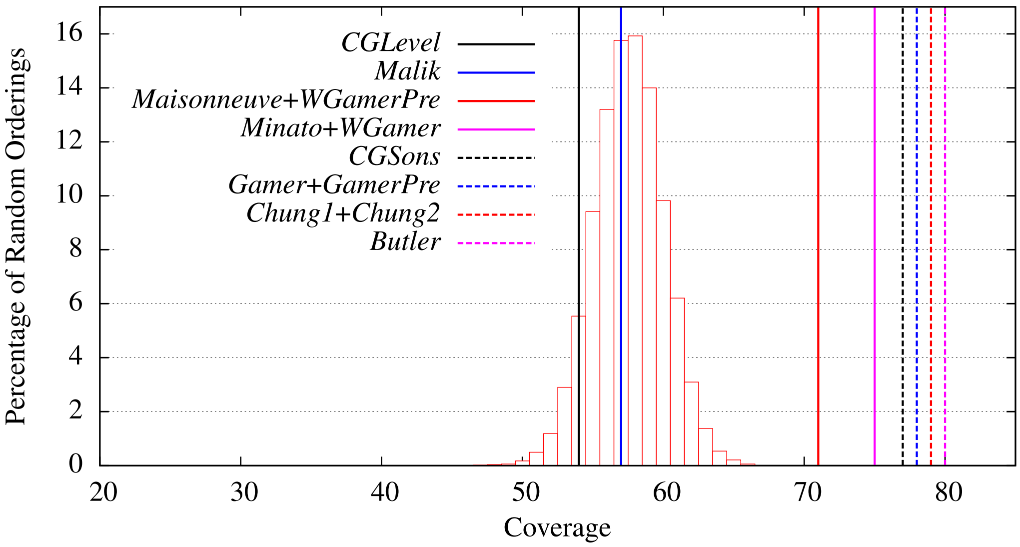


Figure 13: Coverage for random orders vs. ordering schemes. Schemes are ordered top-to-bottom from worst to best coverage. $X+Y$ means that both schemes, X and Y, result in the same coverage.

Type	*Butler*	*CGLevel*	*CGSons*	*Chung1*	*Chung2*	*Gamer*	*GamerPre*	*Maisonneuve*	*Malik*	*Minato*	*WGamer*	*WGamerPre*	Best Scheme
$n^{-r}_{+\Omega}$	3	1	2	4	2	5	5	3	0	4	3	3	6
$n^{+r}_{-\Omega}$	2	26	4	4	2	6	6	11	22	8	7	11	1
$n^{+r}_{+\Omega}$	77	53	75	75	77	73	73	68	57	71	72	68	78
$n^{-r}_{-\Omega}$	3	5	4	2	4	1	1	3	6	2	3	3	0

Table 1: Differences in solved instances for the 85 IPC'11 tasks (1 minute timeout); $-r$ means solved by no random ordering, $+r$ by at least one random ordering, $-\Omega$ not solved by the corresponding ordering scheme, $+\Omega$ solved by the corresponding ordering scheme.

4.3 Good Delimiter

Here we performed bidirectional blind search, i.e., the most competitive setup in general. Figure 14 contains one data point for every pair (I, Ω) of IPC'11 benchmark instance I and ordering scheme Ω that were solved by both (a) Gamer using dynamic reordering starting from an arbitrary variable order (the one returned by Gamer's grounding process), and (b) Gamer using ordering scheme Ω (without dynamic reordering). The time-out is 6 hours for (a), and 30 minutes for (b). The x-value of each data point is the size of the largest BDD constructed for I by (a), the y-value is the size of the largest BDD constructed for I by (b). We allowed a much higher time-out for dynamic reordering because such reordering is not runtime effective: The question we are asking here is merely which of the two methods yields smaller BDDs. Figure 14 shows that dynamic reordering is universally much better at this, giving us sizes that are up to three orders of magnitude smaller than those of the schemes. For a total of 1911 instances (solved by both an ordering scheme and dynamic reordering), in 1431 cases the BDD sizes are smaller by a factor of up to 10 when using dynamic reordering, in

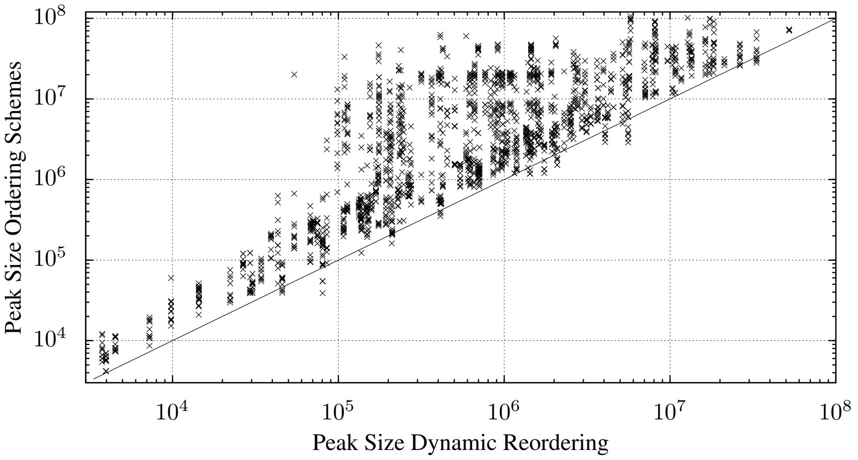


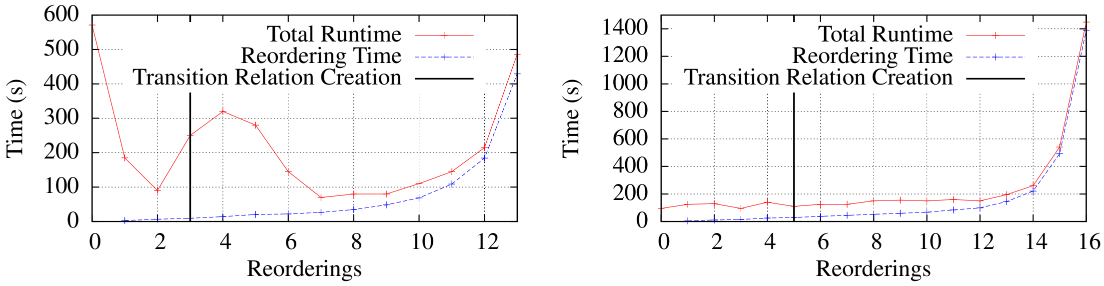
Figure 14: BDD size for dynamic reordering vs. ordering schemes.

Domain	*Butler*	*CGLevel*	*CGSons*	*Chung1*	*Chung2*	*Gamer*	*GamerPre*	*Maisonneuve*	*Malik*	*Minato*	*WGamer*	*WGamerPre*	Dynamic Reordering
Barman	4	4	4	4	8	8	7	4	4	4	5	4	9(5)
Elevators	19	19	19	19	19	19	19	19	19	19	19	19	19(17)
Floortile	8	8	7	8	8	8	8	7	7	8	8	8	7(6)
NoMystery	14	14	14	14	14	13	14	14	15	14	14	14	14(12)
Openstacks	20	18	20	20	20	20	20	19	19	19	20	20	20(20)
PARC-Printer	6	6	5	6	5	6	6	6	5	6	6	7	8(7)
PegSol	17	17	17	17	17	18	18	17	17	17	18	18	18(17)
Scanalyzer	8	7	9	9	8	9	9	9	9	9	9	7	9(9)
Sokoban	17	18	18	17	19	19	19	18	18	18	19	19	19(13)
Tidybot	16	7	14	15	15	9	12	14	9	15	12	8	16(8)
Transport	8	9	9	7	8	8	8	9	7	9	7	7	10(7)
VisitAll	11	9	11	11	11	11	11	11	10	10	11	11	12(11)
Woodworking	16	13	10	14	16	16	16	12	8	16	15	16	19(16)
Total (260)	164	149	157	161	168	164	167	159	147	164	163	158	180(148)

Table 2: Coverage in the IPC'11 tasks. For dynamic reordering, the numbers in parentheses represent the coverage with a 30 minute timeout.

406 cases they are smaller by a factor between 10 and 100, and in 20 cases they are smaller by a factor of more than 100.

Table 2 shows the coverage of the different schemes on the IPC'11 tasks. We can make a similar observation to that of the one minute, only forward search results, namely that *CGLevel* and *Malik* are clearly behind the others. The last column shows the coverage of Gamer using dynamic reordering, and provides two numbers, first the coverage with the 6 hours timeout, second the coverage with the same timeout as the schemes, i.e., 30 minutes. From this it becomes clear that applying dynamic reordering for the entire search time is not feasible in practice when limiting the runtime.



(a) VisitAll, task 011

(b) PegSol, task 015

Figure 15: Total runtime and time spent in reordering for limited number of reordering steps for two example IPC'11 tasks. All reorderings until the vertical line were performed during the transition relation creation.

5. Limited Dynamic Reordering

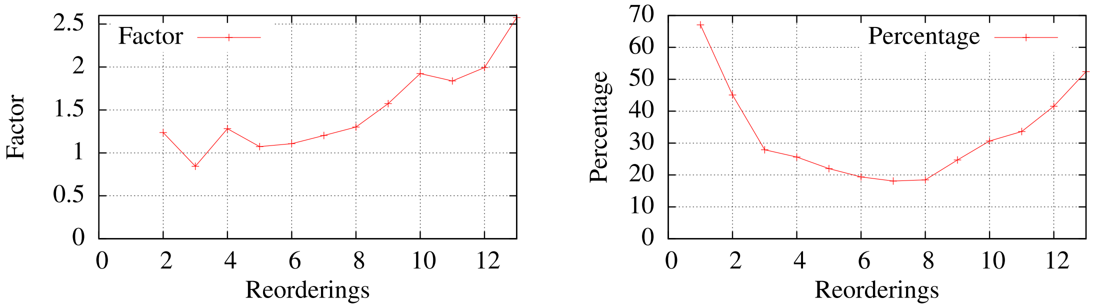
Given the much more memory-efficient behavior of dynamic reordering, a possible approach is to run dynamic reordering for a limited time only, hoping to get an ordering that is good enough for the remainder of the search. Reordering is automatically started when the number of allocated BDD nodes reaches a certain threshold (by default, the first threshold is 4000 nodes), which is dynamically adapted after each reordering (by default, the next threshold is set to 2 times the number of nodes after reordering). A simple way to control dynamic reordering is to limit the number of reordering steps, and to turn dynamic reordering off once the desired number of reorderings has been performed.

For different reordering limits, the total runtime for a task often looks similar to the situation depicted in Figure 15a. With too few reorderings it takes a long time to solve the task due to a bad initial ordering. Also, the first reorderings can sometimes hurt, as they are performed at the very beginning or during construction of the transition relation, before enough information on good orderings is available. However, with too many reorderings solving takes a long time due to an immense overhead in reordering time, which grows exponentially with each step.

An important different behavioral pattern is depicted in Figure 15b: In some domains, such as PegSol or Sokoban, the minimum of the curve is at the very beginning (without any reordering), and the total runtime only increases afterward (mainly based on the increase in reordering time). An explanation for this behavior might be that the initial ordering is already pretty good, so that dynamic reordering cannot improve much and its overhead is incurred in vain. The same also often happens in the easier tasks of a domain, so that learning a good setting based on the simpler tasks seems impossible.

Attempting to exploit these observations to design *adaptive stopping criteria*, geared at finding a good point for stopping dynamic reordering, given only the available observations (e.g., number of BDD nodes before/after reordering, reordering times, current total runtimes), we experimented with the following approaches.

First, we noticed that early on the reordering time increases from step to step by a small factor, but later on that factor increases. In some preliminary runs we saw that often the area of smallest runtime coincides with the situation when the increase in reordering time reaches some threshold, often between 1.25 and 1.75 (see, e.g., Figure 16a and compare it to the runtime minimum in Figure 15a for the same planning task). We call this the *factor* criterion.



(a) Factor of the time for the last reordering and the previous one.

(b) Percentage of the time for the last reordering of the total runtime so far.

Figure 16: Factor and percentage criterion for limited number of reordering steps for task 011 of the IPC'11 VisitAll domain.

Second, another observation from these preliminary runs is that the percentage of the time spent in the last reordering step on the total current runtime often follows a U-like curve, and the minimum of that curve often lies close to the same number of reorderings as the total runtime minimum (see, e.g., Figure 16b and compare to the runtime minimum in Figure 15a for the same task). We employ a *percentage* criterion, which stops reordering after a (possibly local) minimum has been reached, i.e., we compare the percentage of the current step with that of the previous step; when the current one is greater we stop reordering.

Finally, A simple combination of both criteria is to stop reordering as soon as one of them tells us to do so.

To evaluate these adaptive stopping criteria, we ran all tasks of the IPC'11 domains with different limits for the number of reorderings, ranging from 0 to 20.[1] Based on those runs we calculated the results we *would* achieve with the adaptive stopping criteria. See Table 3. The best possible coverage, i.e., the number of tasks solved by at least one setting with limited number of reorderings, is 175, while without any reordering we found 167 solutions. The adaptive stopping criteria yield coverage between 158 and 171. Performance is reasonable for the factor criterion, but is quite bad for the percentage criterion and the combination of both criteria. Recall that the percentage criterion aims at stopping reordering before incurring prohibitive overhead. Indeed, with this criterion, reordering is often stopped earlier than with the factor criterion. In some cases this was detrimental, particularly in the Woodworking domain where this strategy happens to fall into a dramatic local peak of the total-runtime curve, resulting in 9 problem instances no longer being solved.

As, in several cases, dynamic reordering during the transition relation creation was counterproductive, we also ran *delayed reordering*, where dynamic reordering is started only after the BDDs for the transition relation have been created. The results are in Table 4. The case without reordering is unchanged with respect to Table 3. The best possible result is now slightly worse than before, with coverage 174. For the adaptive stopping criteria, the picture changes substantially: In contrast to Table 3, the percentage criterion now excels, delivering coverage just 1 short of the best possible. Regarding the factor criteria, overly small or large factors now are bad and the best behavior (2 short of best possible) is obtained in the middle.

---

[1] In all those cases the highest number of reorderings we could observe was clearly below 20. Either the planner ran out of time or memory, or finished before the last reorderings could be performed. In all cases we used *GamerPre* as the initial ordering, which turned out to be among the best in a preliminary set of experiments.

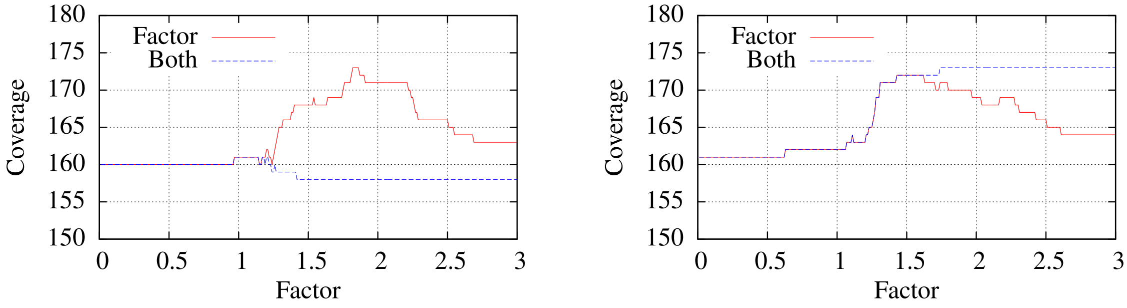
	no	best	factor criterion				percentage	both criteria			
Domain	reord	possible	1.25	1.5	1.75	2.0	criterion	1.25	1.5	1.75	2.0
Barman	7	8	8	8	8	8	8	8	8	8	8
Elevators	19	19	19	19	19	19	19	19	19	19	19
Floortile	8	8	7	8	8	8	8	7	8	8	8
NoMystery	14	16	14	14	14	14	14	14	14	14	14
Openstacks	20	20	20	20	20	20	20	20	20	20	20
PARC-Printer	6	7	6	6	7	7	6	6	6	6	6
PegSol	18	18	17	17	17	18	17	17	17	17	17
Scanalyzer	9	9	9	9	9	9	9	9	9	9	9
Sokoban	19	19	17	17	17	17	17	17	17	17	17
Tidybot	12	14	14	14	14	14	13	13	13	13	13
Transport	8	9	9	9	9	9	9	9	9	9	9
VisitAll	11	12	11	12	12	12	11	11	11	11	11
Woodworking	16	16	10	15	16	16	7	9	7	7	7
Total	167	175	161	168	170	171	158	159	158	158	158

Table 3: Coverage results for different stopping criteria. Immediate reordering.

	no	best	factor criterion				percentage	both criteria			
Domain	reord	possible	1.25	1.5	1.75	2.0	criterion	1.25	1.5	1.75	2.0
Barman	7	8	8	8	8	7	8	8	8	8	8
Elevators	19	19	19	19	19	19	19	19	19	19	19
Floortile	8	8	7	8	8	8	8	7	8	8	8
NoMystery	14	16	16	16	14	14	16	16	16	16	16
Openstacks	20	20	20	20	20	20	20	20	20	20	20
PARC-Printer	6	7	6	7	7	6	7	6	7	7	7
PegSol	18	18	17	17	17	17	17	17	17	17	17
Scanalyzer	9	9	9	9	9	9	9	9	9	9	9
Sokoban	19	19	19	19	19	19	19	19	19	19	19
Tidybot	12	13	13	13	13	13	13	13	13	13	13
Transport	8	9	9	9	9	9	9	9	9	9	9
VisitAll	11	12	12	12	12	12	12	12	12	12	12
Woodworking	16	16	10	15	16	16	16	10	15	16	16
Total	167	174	165	172	171	169	173	165	172	173	173

Table 4: Coverage results for different stopping criteria. Delayed reordering, i.e., reordering started only after creation of the transition relation BDDs.

To shed some light on these observations, Figure 17 shows coverage as a function of more different factor values, for both the case of immediate reordering (Figure 17a) and that of delayed reordering (Figure 17b). In Figure 17a, we see that the percentage criterion stops reordering too early. Without it, the coverage resulting from stopping reordering based solely on the factor criterion can get as high as 173. However, before the ascend in the coverage actually starts the percentage criterion stops reordering, thus cutting off many solutions. In Figure 17b, up to a factor of 1.6 both curves are identical, both mainly increasing with increasing factor. After that, the combination criterion rises another little bit, while the factor criterion alone drops substantially. The combined criterion avoids that drop because, at some point (here, at a factor of roughly 2.0), the percentage criterion stops reordering at least as early as the factor criterion.



(a) Immediate reordering.

(b) Delayed reordering.

Figure 17: Coverage as a function of factor, for the factor criterion alone, and for its combination with the percentage criterion (denoted as "Both").

6. Conclusion

It is tempting to equate the "variable dependencies" in BDD-based symbolic search with those identified in causal graphs, and previous research has done so unquestioningly. Looking a little more closely at this issue, we have shown that causal graph based variable orderings are exponentially bad even in severely restricted sub-classes of planning. Empirically, Fast Downward's level heuristic is worse than random, and all ordering schemes lag far behind off-the-shelf reordering.

One may wonder about the meaning of the theoretical results: How could a static ordering scheme *not* incur exponential overhead in the worst case? We agree with that view in principle, but we did not expect this to happen in planning tasks so restricted as to be tractable for domain-independent optimal planning. It remains to be seen to what extent our classification framework is suitable to characterize the properties of other ordering schemes and/or planning fragments.

Our impression at this point is that static ordering schemes are so limited as to be hopeless. Prior to actually building the BDDs, it appears impossible to extract any reliable information about which form they will take. The way forward, then, is to use dynamic reordering techniques in a more targeted manner. Our initial experiments in that direction did not meet with an immediate breakthrough, but certainly they show promise, especially considering the primitive nature of the method and of the stopping criteria employed. Promising future directions include more flexible on/off strategies for dynamic reordering, machine learning for deciding when to toggle the switch, and planning-specific reordering techniques exploiting the particular structure of the BDDs at hand.

Acknowledgments

We thank the anonymous reviewers of the ICAPS 2013 short version and those of a previous version of this article, whose comments helped tremendously to improve the paper.

References

Brafman, R., & Domshlak, C. (2003). Structure and complexity in planning with unary operators. *Journal of Artificial Intelligence Research*, *18*, 315–349.

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, *35*(8), 677–691.

Burch, J. R., Clarke, E. M., & Long, D. E. (1991). Symbolic model checking with partitioned transition relations. In Halaas, A., & Denyer, P. B. (Eds.), *Proceedings of the International Conference on Very Large Scale Integration (VLSI-91)*, Vol. A-1 of *IFIP Transactions*, pp. 49–58, Edinburgh, Scotland. North-Holland.

Burch, J. R., Clarke, E. M., Long, D. E., McMillan, K. L., & Dill, D. L. (1994). Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *13*(4), 401–424.

Butler, K. M., Ross, D. E., Kapur, R., & Mercer, M. R. (1991). Heuristics to compute variable orderings for efficient manipulation of ordered binary decision diagrams. In *Proceedings of the 28th Conference on Design Automation (DAC-91)*, pp. 417–420, San Francisco, CA, USA. ACM.

Chen, H., & Giménez, O. (2010). Causal graphs and structurally restricted planning. *Journal of Computer and System Sciences*, *76*(7), 579–592.

Chung, P.-Y., Hajj, I. N., & Patel, J. H. (1993). Efficient variable ordering heuristics for shared ROBDD. In *Proceedings of the 1993 IEEE International Symposium on Circuits and Systems (ISCAS-93)*, pp. 1690–1693, Chicago, IL, USA. IEEE.

Cimatti, A., Pistore, M., Roveri, M., & Traverso, P. (2003). Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, *147*(1–2), 35–84.

Darwiche, A. (2011). SDD: A new canonical representation of propositional knowledge bases. In Walsh, T. (Ed.), *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pp. 819–826. AAAI Press/IJCAI.

Dechter, R., & Meiri, I. (1989). Experimental evaluation of preprocessing techniques in constraint satisfaction problems. In Sridharan, N. S. (Ed.), *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, pp. 271–277, Detroit, MI. Morgan Kaufmann.

Domshlak, C., & Dinitz, Y. (2001). Multi-agent offline coordination: Structure and complexity. In Cesta, A., & Borrajo, D. (Eds.), *Recent Advances in AI Planning. 6th European Conference on Planning (ECP-01)*, Lecture Notes in Artificial Intelligence, pp. 34–43, Toledo, Spain. Springer-Verlag.

Edelkamp, S., & Helmert, M. (1999). Exhibiting knowledge in planning problems to minimize state encoding length. In Biundo, S., & Fox, M. (Eds.), *Recent Advances in AI Planning. 5th European Conference on Planning (ECP'99)*, Lecture Notes in Artificial Intelligence, pp. 135–147, Durham, UK. Springer-Verlag.

Fan, G., Müller, M., & Holte, R. (2014). Non-linear merging strategies for merge-and-shrink based on variable interactions. In Edelkamp, S., & Bartak, R. (Eds.), *Proceedings of the 7th Annual Symposium on Combinatorial Search (SOCS'14)*. AAAI Press.

Freuder, E. C. (1982). A sufficient condition for backtrack-free search. *Journal of the Association for Computing Machinery*, *29*(1), 24–32.

Fujita, M., Fujisawa, H., & Kawato, N. (1988). Evaluation and improvements of boolean comparison method based on binary decision diagrams. In *Proceedings of the 1988 International Conference on Computer-Aided Design (ICCAD-98)*, pp. 2–5. IEEE Computer Society Press.

Giménez, O., & Jonsson, A. (2008). The complexity of planning problems with simple causal graphs. *Journal of Artificial Intelligence Research*, *31*, 319–351.

Helmert, M. (2004). A planning heuristic based on causal graph analysis. In Koenig, S., Zilberstein, S., & Koehler, J. (Eds.), *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*, pp. 161–170, Whistler, Canada. Morgan Kaufmann.

Helmert, M. (2006). The Fast Downward planning system. *Journal of Artificial Intelligence Research*, *26*, 191–246.

Helmert, M., Haslum, P., & Hoffmann, J. (2007). Flexible abstraction heuristics for optimal sequential planning. In Boddy, M., Fox, M., & Thiebaux, S. (Eds.), *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, pp. 176–183, Providence, Rhode Island, USA. Morgan Kaufmann.

Helmert, M., Haslum, P., Hoffmann, J., & Nissim, R. (2014). Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the Association for Computing Machinery*, *61*(3).

Hoffmann, J. (2011a). Analyzing search topology without running any search: On the connection between causal graphs and h^+. *Journal of Artificial Intelligence Research*, *41*, 155–229.

Hoffmann, J. (2011b). Where ignoring delete lists works, part II: Causal graphs. In Bacchus, F., Domshlak, C., Edelkamp, S., & Helmert, M. (Eds.), *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*, pp. 98–105. AAAI Press.

Jonsson, P., & Bäckström, C. (1995). Incremental planning. In *European Workshop on Planning*.

Katz, M., & Domshlak, C. (2008). New islands of tractability of cost-optimal planning. *Journal of Artificial Intelligence Research*, *32*, 203–288.

Katz, M., & Domshlak, C. (2010). Implicit abstraction heuristics. *Journal of Artificial Intelligence Research*, *39*, 51–126.

Kissmann, P., & Edelkamp, S. (2011). Improving cost-optimal domain-independent symbolic planning. In Burgard, W., & Roth, D. (Eds.), *Proceedings of the 25th National Conference of the American Association for Artificial Intelligence (AAAI-11)*, pp. 992–997, San Francisco, CA, USA. AAAI Press.

Kissmann, P., & Hoffmann, J. (2013). What's in it for my BDD? On causal graphs and variable orders in planning. In Borrajo, D., Fratini, S., Kambhampati, S., & Oddi, A. (Eds.), *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, pp. 327–331, Rome, Italy. AAAI Press.

Knoblock, C. (1994). Automatically generating abstractions for planning. *Artificial Intelligence*, *68*(2), 243–302.

Maisonneuve, V. (2009). Automatic heuristic-based generation of MTBDD variable orderings for PRISM models. Internship report, Oxford University Computing Laboratory.

Malik, S., Wang, A., Brayton, R., & Sangiovanni-Vincentelli, A. (1988). Logic verification using binary decision diagrams in a logic synthesis environment. In *Proceedings of the 1988 International Conference on Computer-Aided Design (ICCAD-98)*, pp. 6–9. IEEE Computer Society Press.

McMillan, K. L. (1993). *Symbolic Model Checking*. Kluwer Academic Publishers.

Minato, S., Ishiura, N., & Yajima, S. (1990). Shared binary decision diagram with attributed edges for efficient boolean function manipulation. In *Proceedings of the 27th ACM/IEEE Design Automation Conference (DAC-90)*, pp. 52–57, Orlando, FL, USA. IEEE Computer Society Press.

Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., & Malik, S. (2001). Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Conference on Design Automation (DAC-01)*, Las Vegas, Nevada, USA. IEEE Computer Society.

Rintanen, J. (2012). Planning as satisfiability: Heuristics. *Artificial Intelligence*, *193*, 45–86.

Rudell, R. (1993). Dynamic variable ordering for ordered binary decision diagrams. In Lightner, M. R., & Jess, J. A. G. (Eds.), *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design (ICCAD-93)*, pp. 42–47, Santa Clara, CA, USA. IEEE Computer Society.

Sievers, S., Wehrle, M., & Helmert, M. (2014). Generalized label reduction for merge-and-shrink heuristics. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI'14)*, Québec City, Québec, Canada. AAAI Press.

Wegener, I. (2000). *Branching Programs and Binary Decision Diagrams*. SIAM.

Williams, B. C., & Nayak, P. P. (1997). A reactive planner for a model-based executive. In Pollack, M. (Ed.), *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 1178–1185, Nagoya, Japan. Morgan Kaufmann.

Zabih, R. (1990). Some applications of graph bandwidth to constraint satisfaction problems. In *Proceedings of the 8th National Conference of the American Association for Artificial Intelligence (AAAI-90)*, pp. 46–51, Boston, MA. MIT Press.