Constraint and Variable Ordering Heuristics for Compiling Configuration Problems

Nina Narodytska
National ICT Australia
ninan@cse.unsw.edu.au

Toby Walsh
NICTA and UNSW
Sydney, Australia
tw@cse.unsw.edu.au

Abstract

To facilitate interactive design, the solutions to configuration problems can be compiled into a decision diagram. We develop three heuristics for reducing the time and space required to do this. These heuristics are based on the distinctive clustered and hierarchical structure of the constraint graphs of configuration problems. The first heuristic attempts to limit the growth in the size of the decision diagram by providing an order in which constraints are added to the decision diagram. The second heuristic provides an initial order for the variables within the decision diagram. Finally, the third heuristic groups variables together so that they can be reordered by a dynamic variable reordering procedure used during the construction of the decision diagram. These heuristics provide one to two orders magnitude improvement in the time to compile a wide range of configuration.

1 Introduction

Product configuration is often an interactive procedure. The customer chooses a value for a decision variable. They then receive feedback from the configurator about valid values for the remaining decision variables. This continues until a complete and valid configuration is found.Such a scenario requires an efficient mechanism to ensure the current decisions can be consistently extended. Hadzic et al. have proposed a two-phase approach for such interactive configuration [Hadzic *et al.*, 2004]. In the first offline phase, a compact representation is constructed of all product configurations using a decision diagram. This representation is then used by the interactive configurator during the second stage.

In this paper, we focus on optimising the first phase of this configuration process: the compilation of the set of valid product configurations into a decision diagram. This is a computationally hard task that can use a significant amount of CPU time and require large amounts of memory. Although the first phase is performed offline and is not real-time like the second phase, performance is still important. For many large configuration problems, compilation may require more space and time resources than are available. We propose three heuristic techniques for improving the time and space required to construct a decision diagram representing the set of valid solutions to a configuration problems. These heuristics exploit the distinctive clustered and hierarchical structure observed in the constraint graphs of configuration problems. The first heuristic provides an order in which constraints are added to the decision diagram. This limits the growth in the amount of memory used during construction of the diagram. The second heuristic provides an initial order for the variables within the decsion diagram. Finally, the third heuristic groups variables together based on the clustering. These groups are used by a dynamic variable sifting [Rudell, 1993] procedure that reorders variables during the construction of the decision diagram. The combined use of these techniques reduces by one or two orders of magnitude the time to construct decision diagrams for problems from the configuration benchmarks suites [Subbarayan, 2004; Sinz *et al.*, 2003]. Interestingly, the same heuristics perform poorly on satisfiability benchmarks from SATLib suggesting that they are highly tuned to the clustered and hierarchical structure of configuration problems.

2 Compiling configuration problems

A binary decision diagram (BDD) [Bryant, 1986] can be viewed as an acyclic directed graph where edges are labeled by assignments, and a path from the root to the node marked true corresponds to a model. The set of all such paths thus gives the set of all possible models. Efficient procedures exist for constructing and manipulating BDDs. To compile the solutions of a configuration problem into a BDD, we can represent every constraint as a separate BDD and conjoin together these BDDs. As BDDs are added, the size of the resulting BDD grows. For example, Figure 1 shows the relationship between the number of constraints added and the number of nodes in the resulting BDD for the Renault Megane configuration benchmark [Subbarayan, 2004]. The decision diagram grows almost monotonically in size, except at the end where the addition of some critical constraints causes a dramatic drop in size. We observe similar behavior with other configuration benchmarks.

Such growth is surprsing as quite different behavior is typically observed with combinatorial problems. For example,

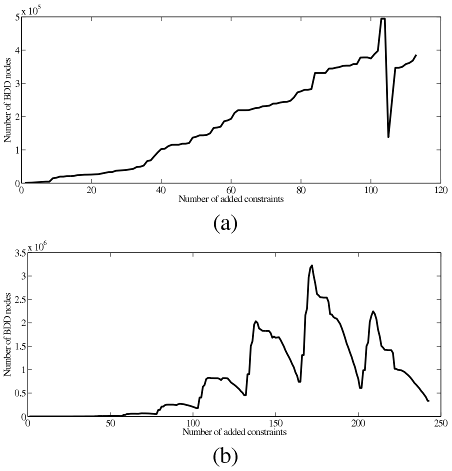

Figure 1: Dynamics of BDD growth for (a) the Renault Megane configuration benchmark and (b) Langford(11,2) problem

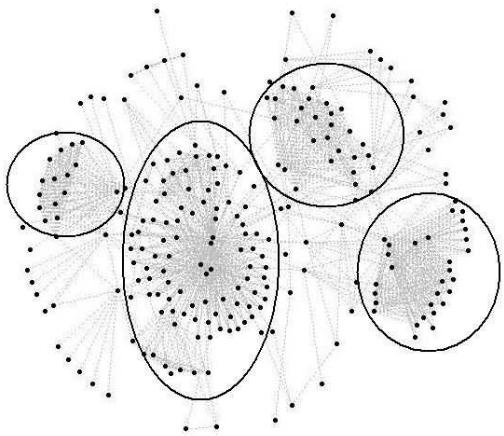

Figure 2: Constraint graph of a Mercedes configuration benchmark (C211_FS).The circles indicate major clusters identified by Markov CLuster Algorithm(MCL)

Figure 1b shows the growth in the size of the BDD for Langford's number problem [Gent and Walsh, 1999]. The decision diagram can be orders of magnitude bigger at an intermediate point than at the end. The amount of memory required to represent the intermediate BDD may therefore be more than is available, even though the final BDD representing all problem solutions may be comparatively small. In addition, even if there is adequate memory to represent the intermediate BDD, the time to construct the final BDD is adversely affected since the time to add a new constraint to a BDD depends linearly on the size of the BDD.

Monotonic growth in the size of a decision diagram is highly desirable from the perspective of memory consumption and speed. We therefore tried to identify the structure of configuration problems monotonic behavior. Our goal is to use these properties in a more directed manner when constructing a BDD. We begin our investigations with the weighted primal constraint graphs. A primal constraint graph of a problem is an undirected graph where nodes correspond to problem variables and edges describe constraints. Two variables are connected by an edge iff they participate in at least one common constraint. The weight of the edge is equal to the number of common constraints involving these two variables. Figure 2 shows the constraint graph of a typical configuration benchmark.[1] The constraint graph has a distinct tree-like skeleton. It contains a tree of clusters, most of which have a star-like structure with a few central nodes connected to a large number of peripheral nodes. In contrast, the constraint graph of a combinatorial problem like Langford's numbers problem is more clique-like.

The tree-like structure of constraint graphs can help explain the monotonic behavior of BDD. Consider an idealised binary constraint problem whose constraint graph is a tree. By ordering the constraints from the root to the leaves of the tree, we can achieve a monotonic growth in the size of the BDD. However, adding constraints in a random order can lead to non-monotonic growth.

Configuration problems also often have strong custered structures. For example, the circles on Figure 2 indicate major clusters identified by the Markov CLuster Algorithm(MCL) [van Dongen, 2000a] for the C211 FS benchmark. A cluster typically corresponds to a group of variables which are very tightly connected. For example, they might describe a single component of the product, e.g., the engine of the car. Adding to the decision diagram all the constraints within a cluster typically reduces the number of valid combinations of the clustered variables and thus the size of the BDD. Our hypothesis thus is that configuration problems have a distinctive hierarchical and clustered structured which often permits monotonic growth in the size of the BDD.

3 Constraint ordering heuristics

Based on these observations, we propose a heuristic for adding constraints that attempts to ensure monotonic growth in the size of the BDD by keeping its size as small as possible on every step. The heuristic attempts:

- **To respect the tree-like structure of many configuration problems**.In particular, we want the heuristic to guarantee monotonic growth in the extreme case when the constraint graph is a tree.
- **To respect the clustered structure of many configuration problems**. In particular, we want the heuristic to add constraints from one cluster at a time.
- **To keep the number of variables small**. Typically, a BDD grows in size with the number of variables added.

[1]Most configuration benchmarks contain a large number of unary constraints. In order to obtain more meaningful results, we eliminated these constraints by performing pure literal deletion and unit propagation for all problems in our experiments.

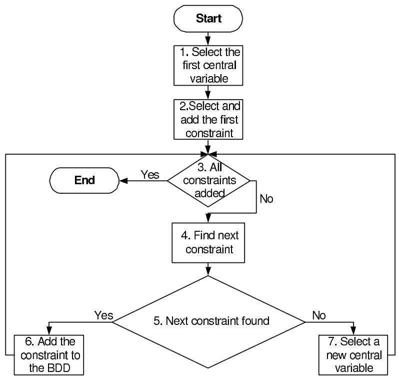

Figure 3: Flowchart of Algorithm 1

Therefore, we want the heuristic to add as few new variables as possible at each step.

Among the many heuristic algorithms that we implemented and evaluated, the following algorithm, referred to as Algorithm 1, produced the best results for the majority of the benchmarks. Figure 3 shows a flowchart of the algorithm. The internal state of the algorithm consists of a list of constraints already added to the BDD, a list of remaining constraints, a list of variables already added to the BDD, a list of remaining variables and a stack of variables that have been used as central variables (the current central variable is located at the top of the stack). A central variable is one of the most constrained variables of the problem, usually the center of one of the clusters.

Step 1. Selection of the first central variable. Among all problem variables, a variable whose adjacent edges have the largest total weight is selected to be the first central variable. This variable is stored at the top of the stack.

Step 2. Selection of the first constraint. Among all constraints that include the central variable, a constraint with the biggest number of variables in its scope is selected.

Step 4. Selection of the next constraint. The next constraint to add to the BDD is selected from the set of remaining constraints.

4.1 All constraints that contain the current central variable are selected from among the remaining constraints. If no such constraints exist, then Step 4 terminates without selecting a candidate constraint.

4.2 From the obtained set of constraints, all constraints that contain the smallest number of variables not yet added to the BDD are selected.

4.3 From the obtained set of constraints, constraints with the smallest number of variables are selected.

4.4 For each selected constraint, the sum of weights of adjacent edges of all its variables is computed and those constraints with the largest sum are selected. The first such constraint becomes the next candidate for being added to the BDD.

Step 7. Selection of the next central variable.

7.1 The set of all neighbours of the current central variable is computed (the current central variable is the variable located at the top of the stack of central variables).

7.2 From the obtained set of variables, all variables that do not participate in scopes of any of the remaining constraints are eliminated.

7.3 If the obtained set of variables is empty, the current central variable is popped from the stack of central variables and the algorithm returns to step 7.1.

7.4 From the obtained set of variables, a variable whose adjacent edges have the largest total weight is selected to be the next central variable. This variable is stored at the top of the stack of central variables. If there are several such variables, the first in the used variable ordering is selected.

This algorithm selects a central variable and adds all constraints involving this variable, trying to add as few new variables as possible on every step. Among all constraints that pass 4.1 to 4.3, a constraint containing the most influential variables is selected, since such constraint is more likely to reduce BDD size. Then the next central variable is selected from the neighbours of the current central variable. We select the heaviest variable hoping that it will be the center of the next cluster (which is usually the case).

Figure 4 shows the results of applying this algorithm to two configuration benchmarks. The graphs compare adding constraints in a random order, in the original order specified in the benchmark description, and in the order produced by Algorithm 1. In both cases, the order produced by Algorithm 1 results in almost monotonic growth of the BDD and kept the BDD size smaller than the other two orderings on all steps. As a result, this algorithm significantly reduced the time to construct the BDD. We obtained similar results for other problems from the configuration benchmarks suite, which we were able to solve without the dynamic variable reordering optimisation presented in the following section.

Another interesting feature of the graphs is that the original constraint ordering was much more efficient than the random ordering. We conjecture that the original ordering usually reflects the natural structure of the problem. For example, it typically groups together constraints describing a single component of the product. Our constraint ordering heuristic was, however, able to make further improvements to this order.

4 Variable ordering

In the previous section, we showed that in configuration problems it is often possible to achieve monotonic growth in the size of the BDD using constraint ordering heuristics. For large configuration problems, this may still not be enough. In order to reduce the space and time requirements of these problems, we would like to find ways to reduce further the size of the BDD. By reordering variables, it is often possible to reduce dramatically the size of a BDD [Bryant, 1986].

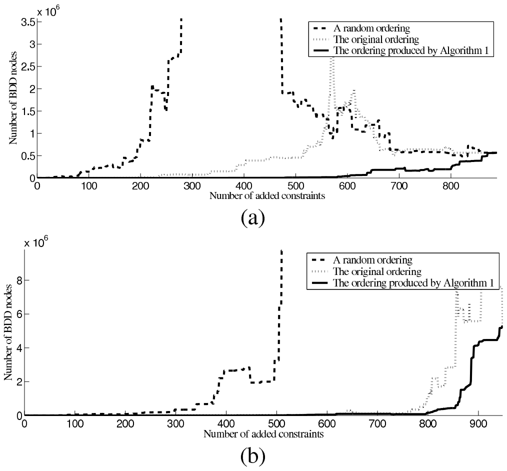

Figure 4: Dynamics of BDD growth when adding constraints in a random order, the original order, and in the ordering produced by Algorithm 1 for (a) the C211_FS configuration benchmark and (b) the C638_FVK configuration benchmark

Unfortunately, determining the optimal variable ordering is NP-hard [Bollig and Wegener, 1996]. Heuristics are therefore used in practice. Variable ordering techniques can be divided into two groups: static and dynamic variable ordering algorithms. Static algorithms compute an ordering of variables before BDD construction. Dynamic algorithms attempt to minimise BDD size by improving variable ordering after the BDD has been partially or completely constructed.

The next sections describe static and dynamic variable ordering heuristics that we developed for configuration problems. These heuristics are based on the following simple observations. First, we observe that locating strongly dependent variables close to each other typically reduces the BDD size. Second, if in a group of variables, one or several variables strongly influence the assignments of all other variables, then these variables should be placed higher in variable ordering.

4.1 Static variable ordering heuristics

The proposed static variable ordering algorithm, referred to as Algorithm 2, consists of three steps. Step 1 finds and groups strongly dependent variables. Step 2 orders variable within groups. Step 3 orders groups relative to each other.

Step 1. First, we identify and group together strongly dependent variables. We employ the fact that configuration problems have clustered structure and assume that variables within a cluster are strongly related to each other, while variables in different clusters are weakly related. We use the MCL algorithm to decompose problem constraint graphs into clusters and group variables belonging to a cluster together in the static variable ordering.

Step 2. Second, we try to order variables inside clusters so that variables that influence the assignments of other variables are placed higher in the ordering. Most clusters found in configuration problems have a small number of central variables connected to a large number of peripheral nodes (see for example Figure 2). Typically, the central variables determine the values of the peripheral variables. Therefore, we put them first in the variable ordering and sort the rest of the variables by their proximity to the center. The specific algorithm for ordering variables within clusters is as follows:

2.1. For each variable in the cluster, compute the total weight of all its adjacent edges. We will refer to this value as the weight of the variable.

2.2. Among all variables in the cluster, select a variable with the biggest weight. This variable is considered the center of the cluster and is placed in the beginning of the cluster in the variable ordering.

2.3. Variables in the cluster are sorted by the weight of edges connecting them to the central variable (variables that are not directly connected to the center are pushed to the back of the cluster).

2.4. Variables that are not sorted by the previous step are sorted by their weights (the heaviest variables are put first). Variables with equal weights are sorted by the total weight of their neighbours.

Step 3. We establish an ordering among clusters: we place clusters that are weakly connected to the rest of the constraint graph in front of other clusters. The assumption is that relatively independent clusters do not increase the size of the BDD greatly. In the extreme case, a completely isolated cluster can be placed in the variable ordering without increasing the size of the rest of the BDD. The degree of isolation of a cluster is determined based on its projection. According to [van Dongen, 2000a], the projection of a node in a cluster is "the total amount of edge weights for that node in that cluster (corresponding to neighbours of the node in the cluster) relative to the overall amount of edge weights for that node (corresponding to all its neighbours)". The projection of a cluster is "the average of all projection values taken over all nodes in the cluster". Clusters are sorted in descending order of their projections.

4.2 Dynamic variable grouping heuristics

Dynamic variable ordering algorithms try to minimise the size of an existing BDD by reordering its variables. Rudell [Rudell, 1993] has proposed a sifting algorithm for dynamic variable reordering and demonstrated that it achieves significant reduction of BDD size for some types of constraint satisfaction problems. The idea of the sifting algorithm is to move each variable up and down in the order to find a position that provides a local minimum in the BDD size. This procedure is applied to every problem variable sequentially. We applied the sifting algorithm provided by the CUDD package [GLU, 2002] to configuration benchmarks. We used an adaptive threshold to trigger variable reordering. Whenever the BDD size reached the threshold, we performed variable reordering and, if the reduced BDD was bigger than 60% of the current threshold, the threshold was increased by

50%. The initial threshold was equal to 100000 BDD nodes. These parameter values were selected empirically.

Panda and Somenzi [Panda and Somenzi, 1995] noticed that dependent variables tend to attract each other during variable sifting, which results in groups of dependent variables being placed in suboptimal positions. To avoid this effect, dependent variables should be kept in contiguous groups that are moved as one during variable sifting. They developed the group sifting algorithm, which automatically finds and groups dependent variables. In our experiments (not cited here) this algorithm slightly improved performance of variable sifting; however much better performance gain can be obtained by taking into account problem structure. As described in Section 4.1, in configuration problems, groups of dependent variables can be identified based on the cluster decomposition of the constraint graph. We modified the variable sifting algorithm to partition problem variables into contiguous groups corresponding to clusters identified by MCL. Grouped variables are kept contiguous by the reordering procedure. In addition, we allow variables within the group to be reordered before performing the group sifting.

When performing variable grouping, it is important to put only strongly connected variables in the same group and avoid grouping weakly connected variables. Therefore, among the clusters found by the MCL algorithm, we only group clusters that have projections bigger than 0.35.

5 Experimental results

We evaluated the three heuristics on problems from the configuration benchmarks suites [Subbarayan, 2004] and [Sinz *et al.*, 2003]. The algorithms were implemented in C++ using the CUDD 2.3.2 BDD package from the GLU 2.0 library [GLU, 2002] and the implementation of the MCL algorithm obtained from [van Dongen, 2000b]. In all experiments, MCL was used with the inflation parameter set to 5. This parameter affects cluster granularity; we select it to be equal to 5 to get fine-grained clusterings. Experiments were run on a 3.2GHz Pentium 4 machine with 1GB of RAM.

In most cases, MCL produced a satisfactory clustering of the constraint graph. However, it failed on several large problems: C209 FA, C210 FVF, and C211 FW. In these problems, the majority of variables are grouped into a single cluster as there are several variables connected to virtually every problem variable. Therefore whenever MCL encounters a cluster that contains more than half of problem variables, we removed its central variables, which are the heaviest variables in the cluster, and repeated the the clustering algorithm. The removed central variables are placed at the top of the initial variable ordering as they are likely to be influential.

Table 1 gives results of our experiments. As can be seen from these results, the constraint ordering heuristics in Algorithm 1 (column 3) reduce the time to construct BDDs compared to random (column 1) and, in most cases, the original (column 2) constraint ordering. We note that the original constraint ordering typically produces quite good results too. As observed before, we conjecture that the original ordering follows the problem structure very closely. For example, all constraints describing a single component of the product are typically placed contiguously in the original ordering. On the other hand, Algorithm 1 is able to find a good constraint ordering even if the constraints are randomly shuffled.

Column 6 shows the effect of the static variable ordering algorithm (Algorithm 2) on BDD construction speed. It produced comparable results to the original variable ordering (column 3) and performed an order of magnitude better than the random variable ordering (not given here), which means that we correctly identified the structure of the problem.

Comparing columns 4 and 5, 7 and 8 we can see that variable ordering heuristic based on grouping clustered variables reduces BDD construction time compared to pure variable sifting [Rudell, 1993] for the majority of benchmarks.

Interestingly, we also tried these three heuristics on a wide range of satisfiability benchmarks from SATLib. We observed uniformly poor performance. We conjecture therefore that configuration problems have an unusual hierarchical and clustered structure which we can exploit when compiling solutions into a decision diagram.

6 Related work

Hadzic et. al. proposed using BDDs to represent the solutions of configuration problems [Hadzic *et al.*, 2004]. However, they were mainly concerned with reducing the size of the final BDD in order to improve the responsiveness of the configurator and not with the efficiency of the BDD construction. In contrast we focus on reducing time and memory requirements for BDD construction. For example, our first heuristic attempts to optimize the order in which constraints are added to the BDD. This does not affect the size of the final BDD, just the size of intermediate BDDs.

Sinz [Sinz, 2002] has proposed an alternative approach to the precompilation of the solutions of configuration problems based on construction of the optimal set of prime implicates.

Static variable ordering techniques have been extensively studied for verification problems. Such problems can be described by a model connectivity graph, an analogue of the constraint graph. A number of variable ordering heuristics have been developed based on the topology of model connectivity graph. These heuristics follow the same guidelines as the ones we used in the Algorithm 2. Namely, they keep strongly connected variables close in variable ordering and put the most influential variables on top [Chung *et al.*, 1994]. Jain et. al. [Jain *et al.*, 1998] proposed a different approach based on construction of variable orderings for a series of BDDs that partially capture the functionality of the circuit. This approach was further developed in [Lu *et al.*, 2000]. While these methods are specialized to verification, it would be interesting nevertheless to adapt them to configuration and compare with the heuristics described here.

7 Conclusions

We have proposed three heuristics for reducing the time and space required to compile the solutions to a configuration problem into a decision diagram. We first showed that the growth in the size of the decision diagram depends strongly on the order in which constraints are added, and proposed a constraint ordering heuristic based on the hierarchical and

Constraint ordering			Random	Original	Alg 1	Alg 1	Alg 1	Alg 1	Alg 1	Alg 1
Variable ordering			Original	Original	Original	Original + sifting	Original + sifting + grouping	Alg 2	Alg 2 + sifting	Alg 2 + sifting + grouping
Benchmark[2]	#Vars	#Cons	1	2	3	4	5	6	7	8
Renault	99	112	54	30	**25**	27	27	26	27	27
C169 FV	39	76	0.02	0.02	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**
D1119 M23	47	178	0.14	0.07	0.07	0.07	0.05	**0.04**	**0.04**	**0.04**
C250 FW	123	321	0.3	0.15	0.08	0.08	**0.07**	0.1	0.1	0.1
C211 FS	238	889	1432	231	48	19	14	30.5	45	**4.5**
C638 FVK	426	948	–	407	259	52	23	**8**	**8**	**8**
C638 FKB	495	1572	–	–	–	522	135	–	3377	**115**
C171 FR	441	1775	–	–	–	—	**1602**	–	9697	1626
C210 FVF	489	1849	–	–	–	–	**1261**	–	–	1604
C209 FA	492	1939	–	–	–	3858	**1097**	–	3189	1228
C211 FW	337	3188	–	–	–	**1380**	3813	–	1771	6924
C638 FKA	517	5272	–	–	–	3761	**390**	–	4300	459

Table 1: Comparison of different BDD construction algorithms. Columns 1 to 8 show average CPU time spent on BDD construction in seconds across 5 runs. The run-time of MCL algorithm is included. "–" denotes that the given problem could not be solved by the corresponding algorithm either because the BDD size exceeded 15,000,000 nodes or because it was interrupted after 10,000 secs.

clustered structure of the primal constraint graph of many configuration problems. We further exploited these properties of configuration problems to develop heuristics for static and dynamic variable ordering. The net effect of the proposed heuristics is one to two orders of magnitude improvement in the time to compile the solutions of benchmark configuration problems. In addition, we were able to solve some problems that could not be solved on our hardware without the heuristics due to memory limitations.

References

[Bollig and Wegener, 1996] Beate Bollig and Ingo Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9):993–1002, 1996.

[Bryant, 1986] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[Chung *et al.*, 1994] Pi-Yu Chung, Ibrahim Hajj, and Janak Patel. Efficient Variable Ordering Heuristics for Shared ROBDD. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 1690–1693, 1994.

[Gent and Walsh, 1999] Ian P. Gent and Toby Walsh. Langford's number problem in CSPLib. `http://csplib.org/prob/prob024/index.html`, 1999.

[GLU, 2002] GLU BDD packages. `ftp://vlsi.colorado.edu/pub/vis/`, 2002.

[Hadzic *et al.*, 2004] Tarik Hadzic, Sathiamoorthy Subbarayan, Rune Møller Jensen, Henrik Reif Andersen, Henrik Hulgaard, and Jesper Møller. Fast backtrack-free product configuration using a precompiled solution space representation. In *Proceedings of the International Conference on Economic, Technical and Organizational aspects of Product Configuration Systems*, pages 131–138, 2004.

[2]after performing pure literal deletion and unit propagation

[Jain *et al.*, 1998] Jawahar Jain, William Adams, and Masahiro Fujita. Sampling schemes for computing OBDD variable orderings. In *Proceedings of the 1998 IEEE/ACM International Conference on Computer-Aided Design*, pages 631–638, 1998.

[Lu *et al.*, 2000] Yuan Lu, Jawahar Jain, Edmund M. Clarke, and Masahiro Fujita. Efficient variable ordering using aBDD based sampling. In *Proceedings of the 37th Conference on Design Automation*, pages 687–692, 2000.

[Panda and Somenzi, 1995] Shipra Panda and Fabio Somenzi. Who are the variables in your neighborhood. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design*, pages 74–77, 1995.

[Rudell, 1993] Richard Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design*, pages 42–47, 1993.

[Sinz *et al.*, 2003] Carsten Sinz, Andreas Kaiser, and Wolfgang Küchlin. Formal methods for the validation of automotive product configuration data. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 17(1):75–97, 2003.

[Sinz, 2002] Carsten Sinz. Knowledge compilation for product configuration. In *Proceedings of the Configuration Workshop, 15th European Conference on Artificial Intelligence*, pages 23–26, 2002.

[Subbarayan, 2004] Sathiamoorthy Subbarayan. CLib: configuration benchmarks library. `http://www.itu.dk/research/cla/externals/clib`, 2004.

[van Dongen, 2000a] Stijn van Dongen. A cluster algorithm for graphs. Technical Report INS-R001, CWI, Netherlands, Amsterdam, 2000.

[van Dongen, 2000b] Stijn van Dongen. MCL implementation. `http://micans.org/mcl/`, 2000.