ECAI 2023
K. Gal et al. (Eds.)

doi:10.3233/FAIA230268

Partial Compilation of SAT Using Selective Backbones

Andrea Balogh[ab], **Guillaume Escamocher**[a] **and Barry O'Sullivan**[ab]

[a]Insight SFI Research Centre for Data Analytics
School of Computer Science & IT, University College Cork, Ireland
[b]Confirm Centre for Smart Manufacturing
School of Computer Science & IT, University College Cork, Ireland

Abstract. Our goal in this paper is to significantly decrease the compiled size of a given Boolean instance with a large representation, while preserving as much information about the instance as possible. We achieve this by assigning values to a subset of the variables in such a way that the resulting instance has a much smaller representation than the original one, and its number of solutions is almost as high as the starting one. We call the set of variable instantiations that we make the selective backbone of the solutions that we keep. Large selective backbones allow for smaller representations, but also eliminate more solutions. We compare different methods of computing the selective backbone that offer the best compromise.

1 Introduction

Diagnosis, planning, and product configuration problems [26] are often processed in an online setting, where the same problem instance needs to be solved many times answering queries, such as "How many possible configurations exist for a product?" or "Is this configuration valid?". Compilation methods were developed to deal with the complexity of solving combinatorial problems offline by creating a representation that is able to answer queries in time that is polynomial in the size of the representation [2]. The knowledge compilation map [10] serves as a guide between the different representations and their capabilities. Knowledge compilation is the field that explores various representations, analysing their size and time complexity with respect to different queries and transformations. Due to time and memory demand, compilation may fail. Finding compact representations is often the bottleneck of compilation methods.

One approach that is sometimes used is *approximate* or *partial* compilation whereby a subset of the solutions of a set of constraints are compiled, e.g. those that are considered most important or most likely to be useful [21]. Others have considered partial compiled representations that are generated from a search-tree over which a variable ordering is defined, and removing domain values that would lead to dead-ends, thereby also removing solutions [5]. There are many use-cases where some solution loss is acceptable. For example, one might wish to compile a large subset of solutions to an embedded device where space is at a premium [21]. Or in a diagnosis setting, storing the most likely solutions (diagnoses) might be sufficient.

In this paper we explore the possibility of obtaining a partial compilation by compiling the set of solutions corresponding to a subset that we obtain by fixing the values of a set of variables. Preferably the selected variable instantiations partition the initial set of solutions such that most solutions are represented in a smaller compilation. In order to obtain these variable instantiations we look at the backbone of the instance. The backbone is the set of variable instantiations that appear in all solutions for this instance. A common application for backbones is to characterize the difficulty of a problem [1, 15, 18, 24], but they have also been previously used to reduce instance size [23]. Any variable that is represented in the backbone will take up little space in a compilation of the instance. Indeed, the only information that we need to keep about this variable is the one value that it takes in all solutions. Since all variables represented in a backbone can be completely documented within a space of linear size in their number, large backbones are highly desirable in the context of instance compilation. This is a notion that has some similarities with the notion of streamlining constraints for search [13] where constraints are added to characterise a subset of solutions that can be found efficiently, but which limits the solutions that can be found. Specifically, those solutions that are inconsistent with the streamlining constraints are no longer available.

Because of their restrictive definition, backbones of non-trivial instances are in general extremely small, and often even empty. For this reason, we instead study in this paper *selective backbones*. These are also defined as the set of variable instantiations that appear in some set of solutions. However, unlike the all-or-nothing approach taken by standard backbones, selective backbones can be defined with regard to a non-exhaustive set of solutions. If this set includes a large and representative enough subset of solutions, we can then discard the solutions it does not contain and obtain, with minimal loss of information, compact backbone compilations of instances with initially small or no backbones.

Of course, there exists a trade-off between the size of a set of solutions and the number of variables covered by its selective backbone. The more solutions a set contains, the smaller its selective backbone is likely to be. Informally, what we want is to find the sweet spot where the set of solutions starts to be comprehensive enough to meaningfully capture the overall structure and characteristics of an instance, in order to lose as little information as possible during the compilation, and the selective backbone remains large enough to provide a significant reduction in the size of the compiled instance. How to quantify whether this reduction is worth more than the solutions that were removed from the set, and how to efficiently determine the location of the ideal compromise, are the subjects of this paper.

Methods to reduce compilation size have been explored through the context of modifying compilation languages to represent sym-

* Corresponding Author. Email guillaume.escamocher@insight-centre.org

metries more efficiently. In [4] dynamic symmetry breaking is explored by extending the languages FBDD and DDG to Sym-FBDD and Sym-DDG respectively. The Variable Shift SDD (VS-SDD) [19] was introduced to exploit variable substitution by merging subtrees which represent the same formula but with different literals. In [3] they explored the possibility of eliminating the symmetrical solutions before compilation. In [12] they introduce an extension to the OMDD representation with the aim to reduce the size of the representation. They extract some information about some of the variables and store this in a separate formula, these formulas forming a satellite structure around the main formula, the nucleus. The satellites are connected to the nucleus by a unique variable instantiation.

In this paper we focus on Binary Decision Diagrams (BDDs) as the compiled representation and a set of generated instances, the ISCAS suite and some planning benchmarks for experiments. In Section 2 we define the notion of selective backbone. In Section 3 we discuss the seven greedy heuristics we implemented and in Section 4 we evaluate them on three problem sets.

2 Definitions

We focus on instances of the Boolean Satisfiability problem (SAT). This problem was the first to be proven NP-Complete [9], and has since been one of the most popular constraint problems to be studied. Before compilation, SAT instances are normally specified in Conjunctive Normal Form (CNF).

Definition 1 (CNF Representation)**.** *A SAT instance in CNF is composed of:*

- *A set of n Boolean variables.*
- *A set of m clauses. Each clause is in turn composed of a disjunction of literals, where a literal can either be a variable or its negation.*

To obtain partial compilations we will be employing selective backbones. Every selective backbone is a partial assignment to some instance.

Definition 2 (Partial Assignment, Support)**.** *Let I be a SAT instance and let B be a set of p variables of I. A set of p variable instantiations, one to each variable of B, is a* partial assignment*. If P is a partial assignment on a set B of variables, then we say that B is the* support *of P.*

Some partial assignments are special, in that they are considered to solve the instance.

Definition 3 (Solution)**.** *If C is a clause, then we say that a partial assignment P* satisfies *C if a variable that is assigned 1 in P is present in C, or if the negation of a variable that is assigned 0 in P is present in C. If I is an instance, then a partial assignment to all variables of I that satisfies all clauses of I is a* solution*, or* model*, for I.*

If the number of solutions (sometimes called *model count*) for an instance I with n variables is s, then compiling all solutions for I can be trivially done by listing $s \times n$ variable instantiations. However, it is sometimes possible to represent the same information in a more compact way, for example when some variables are assigned the same value in all solutions.

Definition 4 (Backbone)**.** *Let I be a constraint instance. The* backbone *of I is the intersection of all solutions for I.*

Example 1. *Let RE be an instance with 9 Boolean variables such that the solutions of RE are the ten solutions listed in Table 1. There is exactly one variable, x_7, which takes the same value in all solutions for RE, therefore the backbone of RE is the set $\{x_7 = 1\}$.*

Table 1. All solutions for the instance RE.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
S_1	1	1	1	0	1	0	1	0	1
S_2	1	1	0	0	0	0	1	1	1
S_3	0	1	1	0	1	0	1	0	0
S_4	1	0	0	0	1	0	1	0	0
S_5	0	0	1	1	0	0	1	1	0
S_6	0	0	0	0	1	0	1	0	0
S_7	1	0	1	0	1	1	1	0	1
S_8	0	0	0	1	0	1	1	0	1
S_9	1	0	1	1	0	1	1	0	1
S_{10}	1	0	1	1	0	1	1	1	1

Because backbones are defined so strictly, instances with more than a few solutions will usually have small or non-existent backbones. To be able to use the general idea of a backbone in instances with many solutions, we instead apply the notion to a subset of the solutions for an instance.

Definition 5 (Selective Backbone)**.** *Let I be a constraint instance and let S be a set of solutions (not necessarily exhaustive) for I. The* selective backbone *of S is the intersection of all solutions of S.*

Example 2. *Let RE be the instance from Example 1. Consider the set $S = \{S_4, S_5, S_6\}$ of some solutions for RE. There are exactly four variables, x_2, x_6, x_7 and x_9, that take the same value in all solutions of S. Therefore the selective backbone of S is the set $\{x_2 = 0, x_6 = 0, x_7 = 1, x_9 = 0\}$.*

In the same manner that every set of solutions uniquely defines a selective backbone, every partial assignment uniquely defines a particular solution set.

Definition 6 (Selected Solution Set)**.** *Let P be a partial assignment for a constraint instance I. The* selected solution set *of P is the set of all solutions for I that are supersets of P.*

Remark 1. *If P is a partial assignment with a non-empty selected solution set, then the selective backbone of the selected solution set of P is a superset (not always strict) of P.*

Example 3. *Let RE be the running example instance. Consider the partial assignments $P_1 = \{x_2 = 0, x_6 = 0, x_7 = 1, x_9 = 0\}$, $P_2 = \{x_2 = 0, x_6 = 1, x_7 = 1, x_9 = 1\}$ and $P_3 = \{x_2 = 1, x_6 = 0, x_7 = 1, x_9 = 1\}$, all with the same support $\{x_2, x_6, x_7, x_9\}$. The selective solution set of P_1 is $\{S_4, S_5, S_6\}$, the selective solution set of P_2 is $\{S_7, S_8, S_9, S_{10}\}$ and the selective solution set of P_3 is $\{S_1, S_2\}$. Note that P_3 is not the selective backbone of $\{S_1, S_2\}$, which is $\{x_1 = 1, x_2 = 1, x_4 = 0, x_6 = 0, x_7 = 1, x_9 = 1\}$, but it is a subset of it.*

For a given number p of variables, we look at two ways to define what is the best selective backbone of size p.

- p-SB: For a given instance I, the p-Selective Backbone problem is to find the partial assignment on p variables with the largest selected solution set.

- p-rSB: For a given instance I, the p-ratio Selective Backbone problem is to find the partial assignment P on p variables with the highest ratio of the size of the selected solution set of P divided by the size of the compiled representation of the instance obtained from I after making all variable instantiations in P.

Example 4. *Let RE be the running example instance. We consider the p-SB problem for $p = 2$. We want a set of two variable instantiations with the largest possible selected solution set. No pair of distinct variable instantiation are present in all solutions for RE, nor even in all but one or all but two solutions, but there are multiple pairs of variable instantiations that are present in seven out of the ten solutions for RE. One such pair of variable instantiations is $\{x_2 = 0, x_7 = 1\}$, which is present in the solutions $\{S_4, S_5, S_6, S_7, S_8, S_9, S_{10}\}$. Therefore the set $\{x_2 = 0, x_7 = 1\}$ is an answer to the 2-SB problem for the instance RE.*

On one hand, the p-rSB problem captures both sides of the compromise between minimal loss of solutions and compiled size, while the p-SB problem is only concerned with the number of solutions. On the other hand, the p-SB problem is independent of the form in which an instance is compiled, while the p-rSB problem depends on the representation chosen.

In our experiments we compile instances into Binary Decision Diagrams (BDD). While the initial construction of a BDD is expensive, once it has been built it allows for quick answers to many otherwise computationally demanding questions, such as model counting.

Definition 7 (BDD Representation)**.** *A Binary Decision Diagram [6] is a directed acyclic graph that represents a set of solutions. Edges are labeled by variable instantiations and paths from the root node to the terminal true node represent the solutions.*

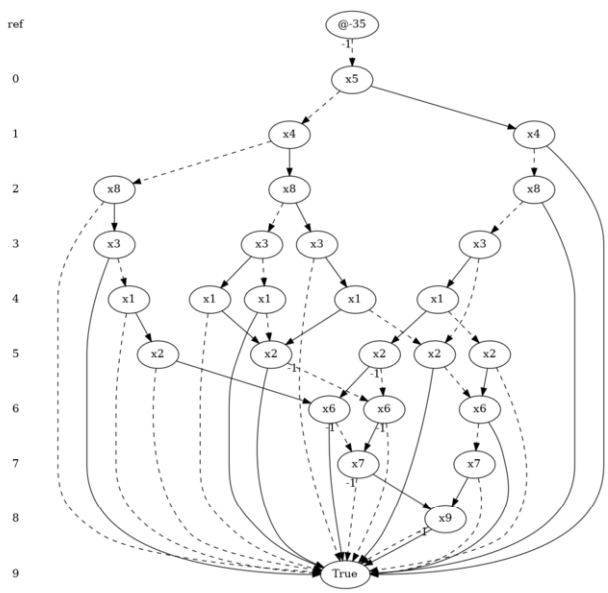

Figure 1. An initial compilation of RE. Nodes represent variables, low edges are dashed (variable is assigned 0), high edges solid (variable is assigned 1).

Example 5. *A BDD representation of the running example is given in Figure 1. A "-1" label signifies a complemented edge, which is defined as follows [25]: a complemented edge to true (respectively false) is interpreted as false (respectively true), and a complemented edge to an internal node is interpreted by toggling the complement bit on both successors.*

Compilation methods describe top-down [17] and bottom-up [8] techniques to compile a set of clauses. Top-down compilation recursively compiles the clauses using conditioning, for example the trace of DPLL using a SAT solver. Bottom-up compilation takes CNF clauses and compiles these using the `apply` [6] operation which joins the representations together efficiently. The variable ordering used for the BDD significantly impacts its size but it is not feasible to find the best ordering as this is an NP-hard problem. [20]

3 Description of Heuristics

For an instance with n variables the number of potential selective backbones of size p is $\binom{n}{p} \times 2^p$, so examining all possible selective backbones to find the optimal one is infeasible for large instances. For this reason we instead consider greedy heuristics.

We are going to compare four methods of solving the p-SB and p-rSB problems. All four methods are inductive, they start from an answer to the problem for $p - 1$, then add a variable instantiation to create a selective backbone of size p which is an answer to the p-SB or p-rSB problem. Since updating an already existing BDD with one variable instantiation does not require recomputing the whole BDD from scratch, building the answer for p upon the answer for $p - 1$ allows for finding the answers for each selective backbone size from 0 to the total number n of variables in the instance with only one expensive BDD construction, the initial one.

The first method is **random**. It simply selects p random variables and one value for each one, and does not differentiate between the two distinct problems p-SB and p-rSB. Because it is fully random, it can select, sometimes early on, a set of variable instantiations that cannot be extended to a solution for the whole instance. The inductive nature of the method will then keep the size of the selected solution set at 0 until the end. To avoid being stuck with an empty selected solution set, we refine the selection of the new variable instantiation by giving a score to all possible candidates.

Definition 8 (Selective Backbone Score)**.** *Let I be an instance and let P be the selective backbone of some subset of the solutions for I. Let I' be the instance obtained from I after making all variable instantiations in P.*

- *For the p-SB problem, the* score *of P is the size of its selected solution set.*
- *For the p-rSB problem, the* score *of P is the size of its selected solution set divided by the size of I' in the chosen compiled representation for the problem.*

For the p-SB problem, the score of a selective backbone composed of only one variable instantiation can be seen as an exact measure of the variable bias [14] of its singleton support.

If the representation for the p-rSB problem is a BDD, we define its size by the number of nodes reachable from the root of the tree.

Now that we are able to directly compare different selective backbones, we can determine for a given one which the next variable instantiation would lead to the highest score.

Definition 9 (Winning Variable Instantiation)**.** *Let P be a selective backbone and let Q be a set of variable instantiations such that every variable present in Q is absent from P (but a variable can appear*

more than once in Q). We say that the winning variable instantiation *of P and Q is the instantiation $q \in Q$ such that for every variable instantiation $q' \in Q$ with $q' \neq q$, the score of $P \cup q$ is higher than the score of $P \cup q'$.*

Example 6. *Consider the running example instance RE. Let $P = \{x_8 = 0, x_9 = 1\}$ and let $Q = \{x_1 = 0, x_6 = 0, x_6 = 1, x_7 = 0\}$.*

- *The selected solution set of $P \cup \{x_1 = 0\}$ is the set of solutions where x_8 is assigned 0, x_9 is assigned 1, and x_1 is assigned 0. This is the singleton set $\{S_8\}$.*
- *The selected solution set of $P \cup \{x_6 = 0\}$ is $\{S_1\}$.*
- *The selected solution set of $P \cup \{x_6 = 1\}$ is $\{S_7, S_8, S_9\}$.*
- *Since no solution for RE assigns 0 to x_7, the selected solution set of $P \cup \{x_7 = 0\}$ is empty.*

The winning variable instantiation of P and Q is therefore $x_6 = 1$, because it leads to the largest selected solution set when being added to P.

We present the remaining three methods in order from fastest to slowest. As we show in the next section, the fastest method (**random_selection**) gives the worst overall results of the three, while the slowest (**dynamic**) usually finds higher scoring selective backbones than the other two.

The second method is **random_selection**. It is described in Algorithm 1. Just like **random**, it picks the next variable randomly. However, the value chosen to assign to the variable is the one that achieves the highest score. This ensures that the selected solution set of the answer will never be empty, as long as the original instance has at least one solution.

Algorithm 1 Solving the p-SB or p-rSB problem with the method **random_selection**.

```
1: if p == 0 then
2:     return ∅
3: end if
4: P ← answer to the (p − 1)-SB problem
5: x ← random variable not present in P
6: a ← winning_variable_instantiation(P, {x = 0, x = 1})
7: return P ∪ {a}
```

Example 7. *Let us apply **random_selection** to the p-SB problem on the running example instance RE. Suppose that the first variable randomly chosen is x_2. This variable is assigned 0 more often than it is assigned 1, so the answer for $p = 1$ is $\{x_2 = 0\}$.*

If the second variable randomly chosen is x_4, we then need to compare the scores of the two selective backbones $P = \{x_2 = 0, x_4 = 0\}$ and $P' = \{x_2 = 0, x_4 = 1\}$. The selected solution set of P is $T = \{S_4, S_6, S_7\}$, and the selected solution set of P' is $T' = \{S_5, S_8, S_9, S_{10}\}$. Since T' is larger than T, the answer for $p = 2$ is P'.

The third method is **static**. It is described in Algorithm 2. It starts by ranking all $2n$ variable instantiations, with n being the total number of variables in the instance, in decreasing order of the score that they would get if they were composing a selective backbone on their own. It then returns the first p variable instantiations, avoiding repeated variables.

Example 8. *Let us apply **static** to the p-SB problem on the running example instance RE. If each of the $\times 9$ variable instantiations was a selective backbone on its own, the three with the highest score would be:*

Algorithm 2 Solving the p-SB or p-rSB problem with the method **static**.

```
1: A ← all 2n variable instantiations {a_1, ..., a_2n} in decreasing
   order of score({a_i})
2: for i ← 1 to p do
3:     a ← first variable instantiation in A
4:     P ← P ∪ {a}
5:     x ← variable assigned by a
6:     A ← A \ {x = 0, x = 1}
7: end for
8: return P
```

1. *$\{x_7 = 1\}$ with all 10 solutions for RE in the selected solution set.*
2.-3. *$\{x_2 = 0\}$ and $\{x_8 = 0\}$, each with a selected solution set of size 7.*
4.-8. *$\{x_1 = 1\}$, $\{x_3 = 1\}$, $\{x_4 = 0\}$, $\{x_6 = 0\}$, and $\{x_9 = 1\}$, each with a selected solution set of size 6.*

So the answer for $p = 1$ would be $\{x_7 = 1\}$, the answer for $p = 2$ would be either $\{x_7 = 1, x_2 = 0\}$ or $\{x_7 = 1, x_8 = 0\}$, and the answer for $p = 3$ would be $\{x_7 = 1, x_2 = 0, x_8 = 0\}$. The answers for $p = 4$ to $p = 8$ would add variable instantiations on variables x_1, x_3, x_4, and x_6, so the instantiation on variable x_5 would not be picked until $p = 9$.

The method **static** only computes the score associated with each variable instantiation once, at the beginning. This saves time, but it means that once a variable instantiation is picked the scores of the remaining instantiations are still based on solutions that might have been already eliminated. The fourth and last method, **dynamic**, takes into account the variable instantiations previously chosen when computing the winning variable instantiation, so all scores are up to date.

Algorithm 3 Solving the p-SB or p-rSB problem with the method **dynamic**.

```
1: if p == 0 then
2:     return ∅
3: end if
4: P ← answer to the (p − 1)-SB problem
5: A ← the 2(n − (p − 1)) variable instantiations on variables not
   present in P
6: a ← winning_variabe_instantiation(P, A)
7: return P ∪ {a}
```

Example 9. *Let us apply **dynamic** to the p-SB problem on the running example instance RE. For $p = 1$, the best variable instantiation to pick is $\{x_7 = 1\}$, which appears in all solutions for the original instance.*

For the second variable instantiation, $\{x_2 = 0\}$ and $\{x_8 = 0\}$ are tied winners. Let us pick the latter, giving us an answer of $P_2 = \{x_7 = 1, x_8 = 0\}$ for $p = 2$.

*For $p = 3$, there are three variable instantiations that get the highest selective backbone score (5 models in the selected solution set) when combined with P_3: $\{x_2 = 0\}$, $\{x_4 = 0\}$, and $\{x_5 = 1\}$. Let us pick $\{x_5 = 1\}$, giving us an answer of $P_3 = \{x_7 = 1, x_8 = 0, x_5 = 1\}$ with a selected solution set $T_3 = \{S_1, S_3, S_4, S_6, S_7\}$. Note that $\{x_5 = 1\}$ appears in a **dynamic** answer for $p = 3$, while from Example 8 we know it cannot appear in a **static** answer until $p = 9$.*

The single best candidate to extend P_3 is $x_4 = 0$, which is the only variable instantiation on a remaining variable to be present in all five solutions left. This gives us an answer of $\{x_7 = 1, x_8 = 0, x_5 = 1, x_4 = 0\}$ for $p = 4$.

4 Results

Experiments were run on a machine with an Intel(R) Xeon(R) CPU E5620 @ 2.40GHz running Ubuntu 22.04.2. Three types of benchmarks were tested: a generated dataset[1] from a construction used to produce challenging model counting instances [11], the ISCAS circuit suites[2] and some instances from the planning benchmark.[2] The ISCAS circuit design and the planning benchmarks have been extensively used in the field of knowledge compilation and model counting as well [7, 16, 22].

Table 2. Best ratios achieved for each type of instances. For each dataset, Column 2 (Nb) is the number of instances in the set, and Column 3 (Nb with backbone) is the number of instances in the set with a non-empty backbone.

Instance type	Nb	Nb with backbone	Best adjusted MC/BDD size ratio			
			min	max	average	median
Dataset A	10	0	1.776	3.662	2.54	2.66
Dataset B	10	0	5.383	13.94	8.68	9.01
iscas89	5	1	1.264	4.944	2.47	1.94
iscas93	2	0	11.501	26.701	19.10	19.10
iscas99	7	0	1.151	4.053	2.64	2.93
blocks	8	7	1.064	5.049	2.65	2.54
bomb	8	8	1.016	2.32	1.60	1.60
coins	7	9	1.2	1.851	1.39	1.34
comm	4	4	1	15.863	5.02	1.60
emptyroom	21	0	1	7.469	1.66	1.00
flip	20	0	1	1.095	1.00	1.00
ring	16	0	1	3.2	1.64	1.45
safe	26	0	1	1.873	1.35	1.42
sort	11	0	1.852	5.244	3.83	3.89
uts	13	12	1.005	5.51	3.49	3.94

The generated set contain 10 instances with 15 variables and 45 clauses (Dataset A) and 10 instances with 30 variables and 90 clauses (Dataset B). Within the ISCAS benchmark, we looked at instances from iscas89, iscas93 and iscas99 containing problems with 26-252 variables and of 66-639 clauses. Within planning we looked at the following instance types: blocks, bomb, coins, comm, emptyroom, flip, ring, safe, sort and uts. Instances in here contain 5-600 variables and 10-1901 clauses. For all cases we only considered CNFs with at most 600 variables. We ran all experiments with 1 hour timeout. All code necessary to reproduce the experiments is available online[1].

We implemented the previously mentioned heuristics in Python 3 and used CUDD[3] for BDD compilations. More specifically we used part of Temporal Logic Planning (TuLiP) toolbox[4] that serves as a Python and Cython wrapper of CUDD. During compilation after each *apply* operation we apply group sifting in order to find good variable ordering for the new BDD.

For our experiments we looked at three greedy heuristics, **random_selection**, **static**, **dynamic** solving the p-SB problem and **random_selection_ratio**, **static_ratio**, **dynamic_ratio** to solve the p-rSB problem. We also compared these with a completely random set of variable instantiations, denoted as **random**. As mentioned above, all heuristics are incremental, the selection of p variable instantiation contains the selection of $p-1$ variable instantiations extended with a new variable instantiation. With each variable instantiation the size of the solution set is monotonically decreasing.

The goal is to find the best variable instantiation set that covers most of the solutions but also has a smaller compiled size. To be able to compare the heuristics across instances with different initial values of model count and BDD compilation size, we define the adjusted ratio AR:

$$AR = \frac{|S|/|BDD|}{|S_{ini}|/|BDD_{ini}|}$$

with

- $|S|$ the size of the current solution set
- $|BDD|$ the size of the current BDD compilation
- $|S_{ini}|$ the size of the initial solution set
- $|BDD_{ini}|$ the size of the initial BDD compilation

This ratio represents how much better (if greater than 1) or worse (if smaller than 1) is the compilation obtained from the current selective backbone compared to the initial compilation. We focus on answering two questions:

- **Q1**: Is there a compilation such that the adjusted ratio is higher than 1?
- **Q2**: Is there a compilation that is significantly more compact if we want to keep at least $x\%$ of the initial solution set?

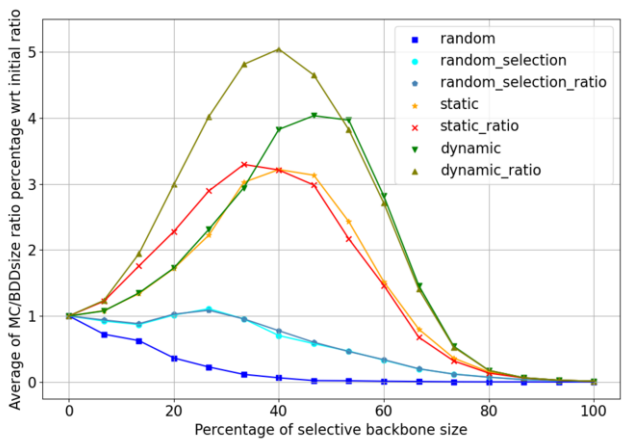

Figure 2. Average Adjusted Ratio over Dataset A and Dataset B.

In some cases multiple variable variable instantiations of size p exist that cover the same number of solutions but a different solution set. For the p-SB problems, we break these ties arbitrarily. The introduction of p-rSB changes this to favouring the p variable instantiations that have a smaller compilation. This approach allows us to first select the variable instantiations that impact the BDD size but with minimal solution loss. This is visible from the fact that **dynamic_ratio** achieves the best trade-offs. Also in case the problem has proper backbones (Definition 4) **dynamic_ratio** will not necessarily select them at first as they do not change the size of the BDD. Therefore problems with large backbones would have fewer opportunities for a good selective backbone. In our experiments only 41

[1] https://github.com//baloghAndi//partialKC_using_selective_backbones
[2] https://www.cril.univ-artois.fr/KC/benchmarks.html
[3] https://web.archive.org/web/20150215010018/ http://vlsi.colorado.edu/ fabio/CUDD/cuddIntro.html
[4] https://github.com/tulip-control/dd

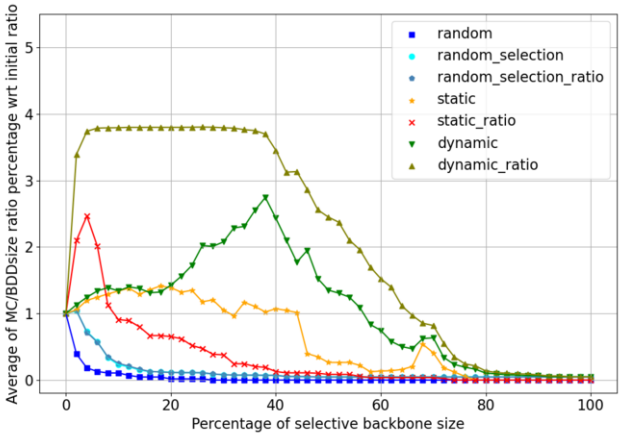

Figure 3. Average Adjusted Ratio over ISCAS instances.

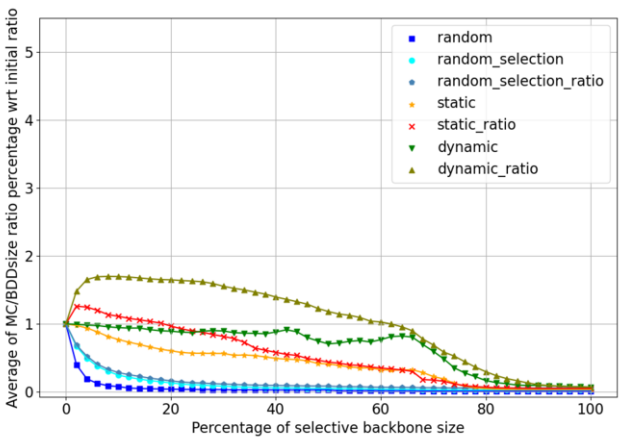

Figure 4. Average Adjusted Ratio over planning instances.

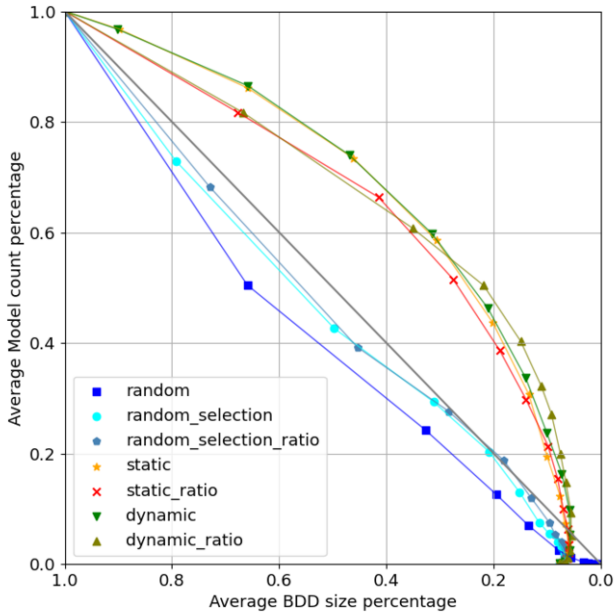

Figure 5. Average efficiency over dataset A and B.

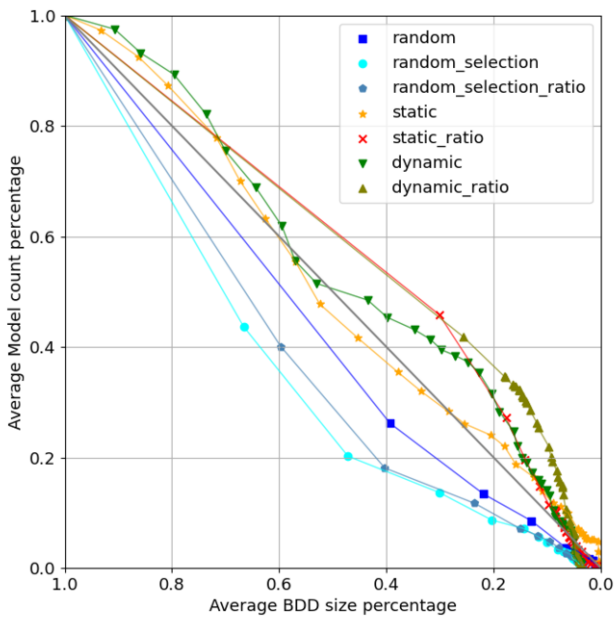

Figure 6. Average efficiency over ISCAS instances.

out of the total 168 instances have proper backbones, with one from ISCAS and the rest from the planning benchmarks as seen in Table 2. The sizes of the proper backbones range between 0.48% to 30% of the number of variables. Table 2 contains more information based on instance types, about the number of instances for each type (Column 2) and how many of them have a non-empty backbone (Column 3). To answer **Q1** the last columns of Table 2 highlight information about the best adjusted ratio for each instance type. There are 36 instances where there is no improvement for the adjusted ratio, some are due to the initially small number of solutions.

A visual representation of the adjusted ratio can be found in Figure 2, Figure 3 and Figure 4, showing the average adjusted ratio for each benchmark respectively. The first data point denotes the $p = 0$ problem for which the adjusted ratio is 1. The higher the adjusted ratio is, the better the selection is. Since each instance has a different number of variables the maximum value of p is also different, so we averaged the adjusted ratio according to the percentage of the selective backbone size divided by the total number of variables. The most gain can be seen in the generated instances, as the ratio curve reaches higher values. In each figure **dynamic_ratio** performs best, followed by **static_ratio** for the first few selections, which is then dominated by **dynamic**. In Figure 2 we observe a clear peak for **dynamic_ratio** at 40% of the variables assigned. In Figure 3 and Figure 4 there is more like a plateau where the ratio is roughly the same for a few variable instantiations for **dynamic_ratio**. That is because the ISCAS and planning instances have variables that split the solution space less uniformly than in the generated dataset. For all

Table 3. Average compilation times in seconds for instance types.

Instance type	Init compilation	random	random_selection	random_selection_ratio	static	static_ratio	dynamic	dynamic_ratio
Dataset A	0.217	0.309	0.313	0.43	0.362	0.34	0.333	0.326
Dataset B	2.613	2.946	3.076	3.24	3.277	3.287	3.421	3.243
iscas89	1290.631	216.865	221.961	295.039	239.691	295.71	374.492	317.405
iscas93	295.331	90.589	96.605	104.362	138.229	135.261	180.098	159.95
iscas99	156.584	45.901	47.435	50.009	51.184	47.731	78.151	63.092
blocks	1119.291	1051.07	1047.16	1025.909	1085.312	1058.3	1230.959	1160.956
bomb	386.295	232.778	265.839	265.896	295.138	302.942	1470.723	1265.708
coins	2333.587	194.944	214.181	554.477	254.396	560.743	556.6	441.289
comm	316.963	360.663	386.863	999.783	436.769	450.071	866.599	635.998
emptyroom	2999.494	321.221	336.584	334.526	365.968	360.129	640.034	616.192
flip	1.691	0.638	0.745	0.711	0.88	0.885	0.958	1.001
ring	1060.66	305.258	312.894	323.228	327.277	322.515	408.546	348.009
safe	18.332	194.905	220.732	226.521	293.156	248.284	483.846	462.226
sort	337.266	419.213	439.866	415.04	497.05	425.432	689.139	541.916
uts	205.234	145.488	152.462	152.579	189.794	161.606	357.711	248.549

experiments **random** performed worse, reaching a set of variable instantiations without any solutions quite early. Due to the incremental property the selection never improves.

To answer **Q2** we look at Figure 5, Figure 6 and Figure 7 which shows the averaged efficiency for each heuristic for each benchmark set. The 45° grey line denotes the area where the solution set size is directly proportional to the BDD size, that is there is no significant benefit. By representing $x\%$ of the solutions the BDD compilation will be $x\%$ of its original size too. Therefore we want to be on the right of this diagonal, as close as possible to the top of the plot. For this metric the generated instance results show the most significant benefits as seen in Figure 5, the efficiency is quite far from the diagonal. In each Figure 5, Figure 6 and Figure 7 all three random based heuristics perform poorly, as they are all under the grey diagonal. Using these figures one can determine that for representing $x\%$ of the solution set what is the smallest representation any of the heuristics can find.

Table 3 summarizes the average time per instance spent on the initial BDD compilation and for all the heuristics to explore n selective backbones. Most of the runtime is spent on the initial compilation, this is a required first step for all the heuristics. The next seven columns show the time spent on finding n selective backbones for each heuristic. More time is taken by **dynamic** and **dynamic_ratio** since both heuristics explore $n - p$ possible variable instantiations for the p selective backbone problem. For each possible instantiation **dynamic** and **dynamic_ratio** call the *apply* function in order to calculate next BDD. The ratio figures offer a detailed view of the progression of the adjusted ratio with respect to the increasing selective backbone. The efficiency plots highlight the existence of compilation that represents $x\%$ of all solutions but the compilation size is less than $x\%$ of the initial compilation size. Note that we did not exhaustively optimize the BDDs, so our results do not state that there is no smaller BDD with the same solution set. We prove that even with minimal effort such compilations exist.

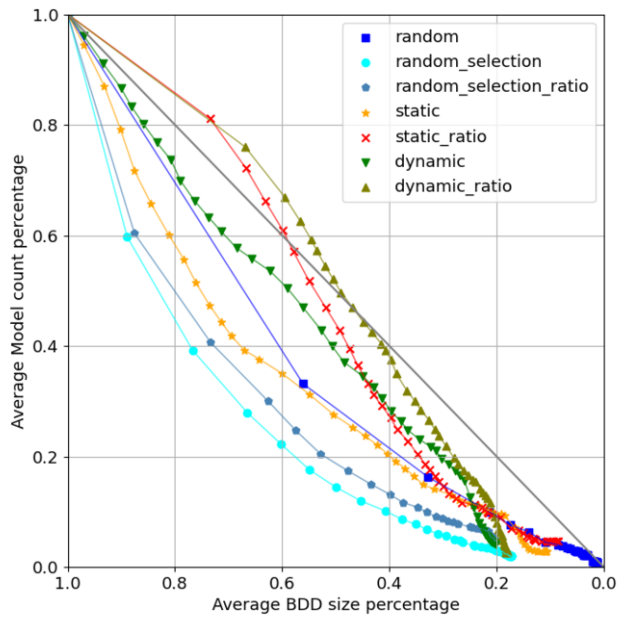

Figure 7. Average efficiency over planning instances.

5 Conclusion

We have empirically proven that for many Boolean instances, belonging to various problem types, it is possible to obtain partial compilations that are far more space-efficient than the full one in terms of the number of solutions conserved. Our approach to find such compilations centers around the concept of selective backbone, a notion that we have introduced and is in essence a partial variable instantiation that is present in a given subset of the solutions to an instance.

Future work could focus on seeking even more efficient ways to find a large selective backbone that covers many solutions, for example by bypassing the initial compilation, which is expensive in both time and space. This could potentially be done by only approximating the number of solutions containing a given partial assignment, instead of aiming for the exact count. Another interesting research direction may be to look at representations other than BDDs.

Acknowledgments

This paper is based upon research supported by the Science Foundation Ireland under Grants 12/RC/2289-P2 and 16/RC/3918 which are co-funded under the European Regional Development Fund.

References

[1] Dimitris Achlioptas, Carla P. Gomes, Henry A. Kautz, and Bart Selman, 'Generating satisfiable problem instances', in *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA*, eds., Henry A. Kautz and Bruce W. Porter, pp. 256–261. AAAI Press / The MIT Press, (2000).

[2] Jérôme Amilhastre, Hélène Fargier, and Pierre Marquis, 'Consistency restoration and explanations in dynamic CSPs application to configuration', *Artif. Intell.*, **135**(1-2), 199–234, (2002).

[3] Andrea Balogh and Barry O'Sullivan, 'Breaking symmetry for knowledge compilation', in *SAC '23: The 38th ACM/SIGAPP Symposium on Applied Computing, Tallinn, Estonia, March 27 - 31, 2023*. ACM, (2023).

[4] Anicet Bart, Frédéric Koriche, Jean-Marie Lagniez, and Pierre Marquis, 'Symmetry-driven decision diagrams for knowledge compilation', in *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, eds., Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pp. 51–56. IOS Press, (2014).

[5] J. Christopher Beck, Tom Carchrae, Eugene C. Freuder, and Georg Ringwelski, 'A space-efficient backtrack-free representation for constraint satisfaction problems', *Int. J. Artif. Intell. Tools*, **17**(4), 703–730, (2008).

[6] Bryant, 'Graph-based algorithms for boolean function manipulation', *IEEE Transactions on Computers*, **C-35**(8), 677–691, (1986).

[7] Florent Capelli, Jean-Marie Lagniez, and Pierre Marquis, 'Certifying top-down Decision-DNNF compilers', in *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pp. 6244–6253. AAAI Press, (2021).

[8] Arthur Choi and Adnan Darwiche, 'Dynamic minimization of sentential decision diagrams', in *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*, eds., Marie desJardins and Michael L. Littman. AAAI Press, (2013).

[9] Stephen A. Cook, 'The complexity of theorem-proving procedures', in *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, eds., Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, pp. 151–158. ACM, (1971).

[10] Adnan Darwiche and Pierre Marquis, 'A knowledge compilation map', *J. Artif. Intell. Res.*, (2002).

[11] Guillaume Escamocher and Barry O'Sullivan, 'Generation and prediction of difficult model counting instances', *CoRR*, **abs/2212.02893**, (2022).

[12] Hélène Fargier, Jérôme Mengin, and Nicolas Schmidt, 'Nucleus-satellites systems of omdds for reducing the size of compiled forms', in *28th International Conference on Principles and Practice of Constraint Programming, CP 2022, July 31 to August 8, 2022, Haifa, Israel*, ed., Christine Solnon, volume 235 of *LIPIcs*, pp. 23:1–23:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, (2022).

[13] Carla P. Gomes and Meinolf Sellmann, 'Streamlined constraint reasoning', in *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, ed., Mark Wallace, volume 3258 of *Lecture Notes in Computer Science*, pp. 274–289. Springer, (2004).

[14] Eric I. Hsu, Christian J. Muise, J. Christopher Beck, and Sheila A. McIlraith, 'Probabilistically estimating backbones and variable bias: Experimental overview', in *Principles and Practice of Constraint Programming, 14th International Conference, CP 2008, Sydney, Australia, September 14-18, 2008. Proceedings*, ed., Peter J. Stuckey, volume 5202 of *Lecture Notes in Computer Science*, pp. 613–617. Springer, (2008).

[15] Philip Kilby, John K. Slaney, Sylvie Thiébaux, and Toby Walsh, 'Backbones and backdoors in satisfiability', in *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, eds., Manuela M. Veloso and Subbarao Kambhampati, pp. 1368–1373. AAAI Press / The MIT Press, (2005).

[16] Jean Marie Lagniez, Emmanuel Lonca, and Pierre Marquis, 'Definability for model counting', *Artificial Intelligence*, **281**, 103229, (2020).

[17] Jean-Marie Lagniez and Pierre Marquis, 'An improved Decision-DNNF compiler', in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, ed., Carles Sierra, pp. 667–673. ijcai.org, (2017).

[18] Rémi Monasson, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman, and Lidror Troyansky, 'Determining computational complexity from characteristic 'phase transitions'', *Nature*, **400**(6740), 133–137, (1999).

[19] Kengo Nakamura, Shuhei Denzumi, and Masaaki Nishino, 'Variable shift SDD: A more succinct sentential decision diagram', in *18th International Symposium on Experimental Algorithms, SEA 2020, June 16-18, 2020, Catania, Italy*, eds., Simone Faro and Domenico Cantone, volume 160 of *LIPIcs*, pp. 22:1–22:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, (2020).

[20] Nina Narodytska and Toby Walsh, 'Constraint and variable ordering heuristics for compiling configuration problems', *IJCAI International Joint Conference on Artificial Intelligence*, 149–154, (2007).

[21] Barry O'Sullivan and Gregory M. Provan, 'Approximate compilation for embedded model-based reasoning', in *Proceedings of AAAI-2006*, (2006).

[22] Umut Oztok and Adnan Darwiche, 'A top-down compiler for sentential decision diagrams', in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, eds., Qiang Yang and Michael J. Wooldridge, pp. 3141–3148. AAAI Press, (2015).

[23] Johannes Josef Schneider, Christine Froschhammer, Ingo Morgenstern, Thomas Husslein, and J. M. Singer, 'Searching for backbones — an efficient parallel algorithm for the traveling salesman problem', *Computer Physics Communications*, **96**, 173–188, (1996).

[24] John K. Slaney and Toby Walsh, 'Backbones in optimization and approximation', in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, ed., Bernhard Nebel, pp. 254–259. Morgan Kaufmann, (2001).

[25] Tom van Dijk, Robert Wille, and Robert Meolic, 'Tagged BDDs: Combining reduction rules from different decision diagram types', in *2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017*, eds., Daryl Stewart and Georg Weissenbacher, pp. 108–115. IEEE, (2017).

[26] Hao Xu, Souheib Baarir, Tewfik Ziadi, Lom-Messan Hillah, Siham Essodaigui, and Yves Bossu, 'Optimisation for the product configuration system of Renault: towards an integration of symmetries', in *SPLC '21: 25th ACM International Systems and Software Product Line Conference, United Kindom, 2021, Volume B*. ACM, (2021).