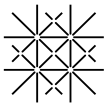


University of Basel

Analysis of Variable Orders for Symbolic Search

Bachelor thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Artificial Intelligence
https://ai.dmi.unibas.ch

Examiner: Prof. Dr. Malte Helmert
Supervisor: Dr. David Jakob Speck

Mátyás Bartha
m.bartha@stud.unibas.ch
2021-050-075

03.10.2024

Acknowledgments

I want to thank Prof. Dr. Malte Helmert for giving me the opportunity to write my Bachelor's thesis in the Artificial Intelligence Research Group.
A very special thank you also goes to my supervisor, Dr. David Jakob Speck, who was always available to answer any questions I had and supported me throughout my work.

Abstract

This thesis investigates the impact of variable ordering on the efficiency of symbolic search in classical planning, with focus on Binary Decision Diagrams (BDDs). The efficiency of symbolic search is significantly affected by the structure of BDDs, which is determined by the order of variables. The study evaluates the efficiency of existing variable orders, including Forward Ordering and Gamer Ordering, across a range of planning domains. A new variable ordering, the *Blocks Ordering* is introduced and specifically designed to optimize the performance of the symbolic search in the Blocks World domain. The experimental results show that the Blocks Ordering noticeably reduces the number of BDD nodes generated for the Blocks World problem, leading to improved performance compared to the other variable orders. It can be concluded that while no single variable ordering is universally optimal for all domains, domain-specific orderings like Blocks Ordering can provide performance benefits in symbolic search.

Table of Contents

1 Introduction

The field of classical planning represents a central area of research within the domain of artificial intelligence (AI). The objective is to develop algorithms that can identify a sequence of actions that will effectively transform an initial state into a goal state. Planning is a fundamental component of many AI applications, such as robotics, automated scheduling, game playing, and logistics.

Symbolic search has become a well-established methodology for solving planning tasks. The most appropriate data structures are selected, for example, binary decision diagrams (BDDs)[3], which directly represent entire sets of states. The appropriate data structure permits the manipulation of the entire set of states in an efficient manner throughout the search process. However, the efficiency of the process is dependent upon the internal representation of the problem. Binary decision diagrams are composed of binary variables. The order of the variables in a BDD is of great consequence with regard to the size and complexity of the BDD. An appropriate variable order results in a compact BDD, which enables the symbolic search algorithm to operate more efficiently.

The objective of this thesis is to investigate the effectiveness of different variable orders in the symbolic search based on analysis for selected domains[1] A new ordering, the Blocks Ordering is introduced with the aim of enhancing the efficiency of the symbolic search for the Blocks World problem.

[1] The domains that are used are part of the unofficial collection of (sequential) IPC benchmark instances. https://github.com/aibasel/downward-benchmarks

2
Background

2.1 Classical Planning

Classical planning domains and tasks in this thesis are defined by the SAS+ formalism[1]:

2.1.1 Planning Task

A planning task is a 4-tuple $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, where:

- $\mathcal{V}$ is a finite set of state variables v, each with a finite-domain $\text{dom}(\mathcal{V})$. A fact is an assignment of a state variable $({}_v i \rightarrow d_i)$ with $d_i \in \text{dom}(v_i)$. A partial state is a consistent set of facts. If every state variable $v_i \in \mathcal{V}$ is assigned a value of its domain $\text{dom}(v_i)$ we call it a state. We denote the set of all possible states $\mathcal{S}$.
- $\mathcal{A}$ is a finite set of actions a, each with a precondition $\text{pre}(a)$ and effect $\text{eff}(a)$, where both are partial states, and a cost value $\text{cost}(a) \in \mathbb{R}_0^+$. We say that action a is applicable in state s if $s \subseteq \text{pre}(a)$.
- $\mathcal{I}$ is a state called the initial state.
- $\mathcal{G}$ is a partial state called the goal.

2.1.2 Plan

A Plan $\pi = \langle a_1, ..., a_n \rangle$ for a planning task Π is a sequence of actions which takes us from the initial state $\mathcal{I}$ to the goal state $\mathcal{G}$. The cost of π is defined as $\text{cost}(\pi) = \sum_{i=1}^{n} cost(a_i)$.

The search for a plan with minimal cost is called **optimal planning**.

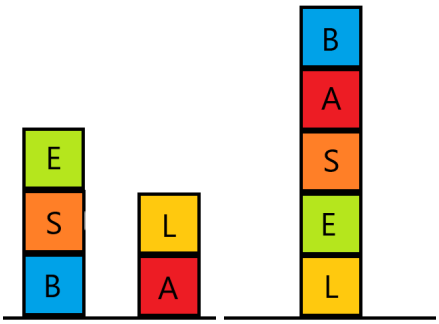
2.1.3 Blocks World

Blocks world is a classical problem in AI. The following rules apply[4]:

- There is a finite number of blocks
- Each block is either on another block or on the table

- A block can be clear or another block is on top of it
- The actions are:
  - pick-up: Pick up a block from the table if it's clear
  - put-down: Put down the block on the table
  - stack: Put the block on top of a clear block
  - unstack: Pick up the block from another block if it's clear

The Blocks World planning problem is to get from a start configuration, the initial state 2.1(a) to the goal configuration, the goal state 2.1(b) by applying the actions one after each other. The optimal Blocks World planning problem is to do so with a minimal number of moves[11]. In this thesis we consider the case where every action has the same cost and we are looking for an optimal plan.



(a) Initial State. (b) Goal State.

Figure 2.1: Example of an initial and goal state of the Blocks World problem.

2.2 Symbolic Search

Symbolic search is a state space exploration technique that was initially proposed in the field of model checking[9]. In contrast to explicit state search, the symbolic search performs operations on the whole set of states instead of individual states. A set of states S is defined by its characteristic function f_S, that maps a state $s \in \mathcal{S}$ to Boolean values $\top$, denoting $True$ if a given state s belongs to S, and $\bot$, denoting $False$ if a given state s does not belong to S. Using the characteristic function f_S it is possible to directly perform set operations, such as the union ($\cup$), intersection ($\cap$) and complement ($\overline{S}$).
Using Transition Relations (TRs), a set of operators $\mathcal{O}$ can be represented by a Boolean function f_t, that maps every pair $\langle s_1, s_2 \rangle$ with $s_1, s_2 \in \mathcal{S}$ to $\top$ if s_2 can be reached from s_1 by applying an operator $o \in \mathcal{O}$. In other words the function returns $True$ if s_2 is a successor of s_1 and $False$ if not. Given a set of states S and a Transition Relation f_t, the set of all successor and predecessor states can be computed using the image and pre-image operator, respectively[13].

2.3 Binary Decision Diagrams

A Binary Decision Diagram (BDD)[2] is a data structure used for the representation of a Boolean function.
Such a Boolean function is represented as a rooted, directed, acyclic graph with two terminal nodes. The terminal nodes have the Boolean values 0 (False) or 1 (True). The inner nodes correspond to binary variables with two successors, where one edge represents that the variable is $False$ and the other is $True$. Following a path from the root to a terminal node, the value of the terminal node labels the result of the corresponding variable assignment.

2.3.1 Ordered Binary Decision Diagrams

Ordered Binary Decision Diagrams are just like BDDs but with a defined variable ordering. In an Ordered Binary Decision Diagram (OBDD), the variables are arranged in a specific fixed order, and on all paths from the root to a terminal node the variables appear in that same order.

2.3.2 Reduced Order Binary Decision Diagrams

A Reduced Order Binary Decision Diagram (ROBDD) takes an OBDD and applies two additional optimizations:

- Eliminate Redundant Nodes: If two nodes in the OBDD have the same children, they are merged into one node.
- Eliminate Redundant Edges: If both edges from a node point to the same child, that node is removed and all incoming edges are redirected to its child.

Since from now on we will only consider ROBDDs, we will use BDD as a synonym for it.

2.3.3 Variable Ordering

As can be seen in 2.2, the variable ordering completely determines the size of a BDD that represents a specific function[13]. In the first BDD (2.2 (a)), the variables are ordered as x_1,y_1,x_2,y_2,x_3,y_3, resulting in a compact, polynomial-size diagram. This ordering keeps the conjunctions $(x_1 \wedge y_1)$, $(x_2 \wedge y_2)$, $(x_3 \wedge y_3)$ together, which simplifies the decision structure. Each path through the diagram evaluates one conjunction at a time, allowing the diagram to share nodes efficiently and reduce the total number of nodes and edges significantly. In contrast, the second BDD (2.2 (b)) orders the variables as x_1,y_1,x_2,y_2,x_3,y_3, which results in an exponentially larger diagram. In this ordering, variables from different conjunction terms $(x_1 \wedge y_1)$, $(x_2 \wedge y_2)$, $(x_3 \wedge y_3)$ are mixed in the variable ordering. As a result decisions about the y-variables are deferred until all decisions about the x-variables have been made. Assuming that the number of generated BDD nodes during Symbolic Search correlates with the time the algorithm takes to find a plan, the variable order seems to play a major role in

the duration of the search. It is therefore a central task to optimise the variable order[2] in order to reduce the BDD size. However, the problem of finding an optimal variable order for Binary Decision Diagrams (BDDs) has been proven to be NP-hard.[6].

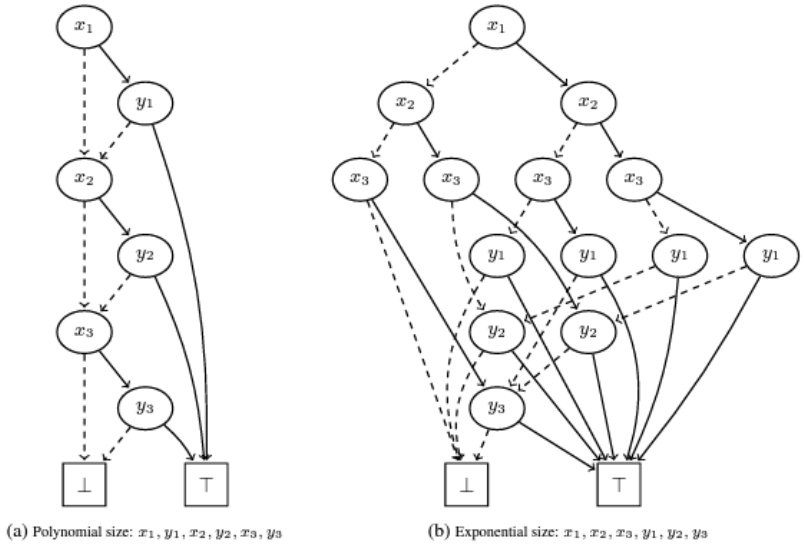


Figure 2.2: Two BDDs for the function $f = (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee (x_3 \wedge y_3)$ with different variable orderings.[13]

[2] Each finite domain variable $v \in \mathcal{V}$ can be represented by $\lceil log_2 |D_v| \rceil$ binary variables. As a consequence of this transformation, all variables in this thesis can be regarded as binary variables.[13]

3
Blocks Ordering

In this chapter, the Blocks Ordering is introduced, a BDD variable ordering specifically created to optimize the performance of symbolic search for the Blocks World domain. The aim is to outperform the existing the existing variable orders of the SymK Planner[12] for this specific domain.

3.1 Existing Variable Orders

- **Forward Ordering:** The Forward Ordering describes a total variable order that is defined by the causal graph. The variable order is defined in such a way that the removal of all edges $v \to v'$ with $v > v'$ induces an acyclic subgraph of the causal graph. This is consistent with the pruning process in The Fast Downward Planning System paper[5].
- **Gamer Ordering:** The Gamer Ordering describes a BDD variable order, where the variables are selected through domain analysis. Optimization is achieved by minimizing the distance of variable dependencies in the causal graph[6, 8].

3.2 Idea

Kissmann and Hoffmann[8] examined the potential for deriving optimal variable orderings on the basis of the causal graph. The findings indicated that this approach is not feasible in almost all cases. In practice, however, the Gamer Ordering, which is based on the causal graph, has proved to be highly effective, despite the results of the theoretical study [7, 8]. Nevertheless, as Kissmann and Hoffmann[8] demonstrated, there is potential for improvement over the gamer ordering through empirical evaluation of different ordering strategies, including random ones. Here, we employ a modern symbolic search planner to conduct a similar analysis and examine selected domains in greater detail with targeted orderings.

The Blocks Ordering is a domain-specific variable ordering, that follows the strategy to create a variable ordering based on the goal state of the Blocks World planning problem. The objective is to identifying an ordering that groups variables, that are clearly related to

each other in the goal configuration, close to each other in the variable ordering.

3.3 Computing the Blocks Ordering

The structure for the variables is the same for all instances:

- For every block, a variable represents, whether the block is clear or not.
- For every block, a variable represents, whether it is currently being held, which other block is on it and whether the block is on the table.
- One variable representing whether the hand is empty or not.

Following this structure, every block can directly be associated with two variables. There is one variable that does not belong to any block.

The Blocks Ordering computes the variable ordering based on the goal configuration:
By mapping each block to the one that is on top of it in the goal configuration, a simple vector with the order of the blocks can be computed. That vector stores the order of each stack from bottom to top.
Now every element of this ordered vector, containing the names of the blocks, is replaced with the two corresponding variables (as described above). This results in an order, where variables that are closely linked to each other in the goal configuration are also closely linked in the variable order. As the variable representing whether the hand is empty or not is not directly belonging to any other variable, it is by choice inserted first in the order.

4

Experimental Results

In this chapter the results gathered are presented. For the experiments the following tools were used:

- An unofficial collection of (sequential) IPC benchmark instances[3].
- SymK Planner[12], a classical optimal and top-k planner based on symbolic search that extends Fast Downward[5].
- Downward Lab[10], a Python package for running the experiments for the Fast Downward planning system.
- sciCore's INFAI environment on the partition infai2. The limit was set to 5 minutes for time, 6144MB for memory.

To get insight into the domain sizes, an experiment was run for all domains in the benchmark dataset.
As can be seen in table 4, three domains: blocks, hiking-opt14-strips and transport-opt14-strips are highlighted. The following experiments are run on these domains as the number of variables is smaller by a lot compared to the other domains. As a result, many runs can be made, and a solution can be found for most of the instances in the given time.

	# Instances	Min. # Variables	Max. # Variables	Avg. # Variables	Median # Variables
airport	31	27	948	266	136
assembly	30	107	490	286	297
blocks	35	9	35	20	19
hiking-opt14-strips	20	7	14	11	12
satellite	30	6	173	70	59
transport-opt14-strips	20	8	13	10	11

Table 4.1: Selection of domains with the corresponding statistics on the number of instances and number of variables per instance. The yellow marking indicates the domains on which the experiments will be run.

[3] Artificial Intelligence Group - University of Basel, https://github.com/aibasel/downward-benchmarks

For all experiments, the number of BDD nodes is used as a measure for the search time for a planning problem.
The correlation plot shown in figure 4.1 provides a clear illustration of the relationship between the number of BDD nodes and the search time. For this experiment, the data from 1,000 searches for all instances of the domains blocks, hiking-opt14-strips and transport-opt14-strips were used.
With a correlation coefficient of 0.92, the relationship is both strong and positive, indicating that as the number of BDD nodes increases, the search time increases in a corresponding manner. The result confirms the hypothesis and allows for the use of the number of BDD nodes as a measure for the search time.

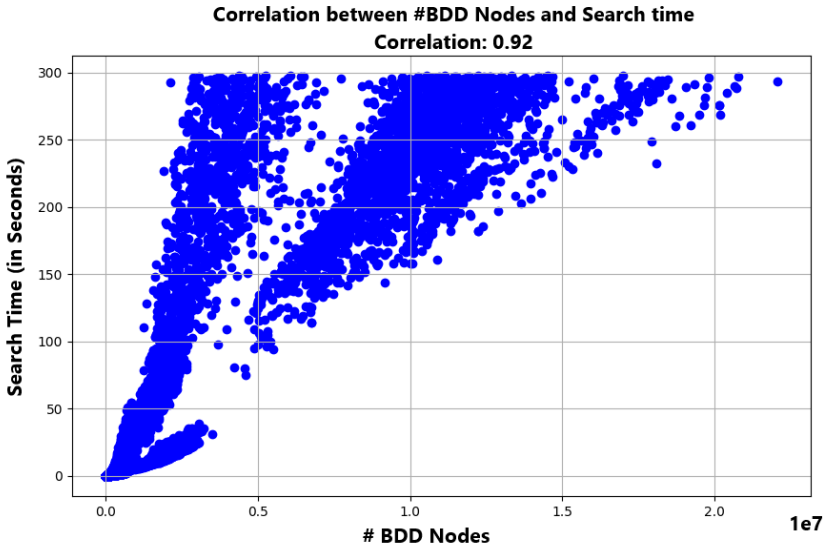
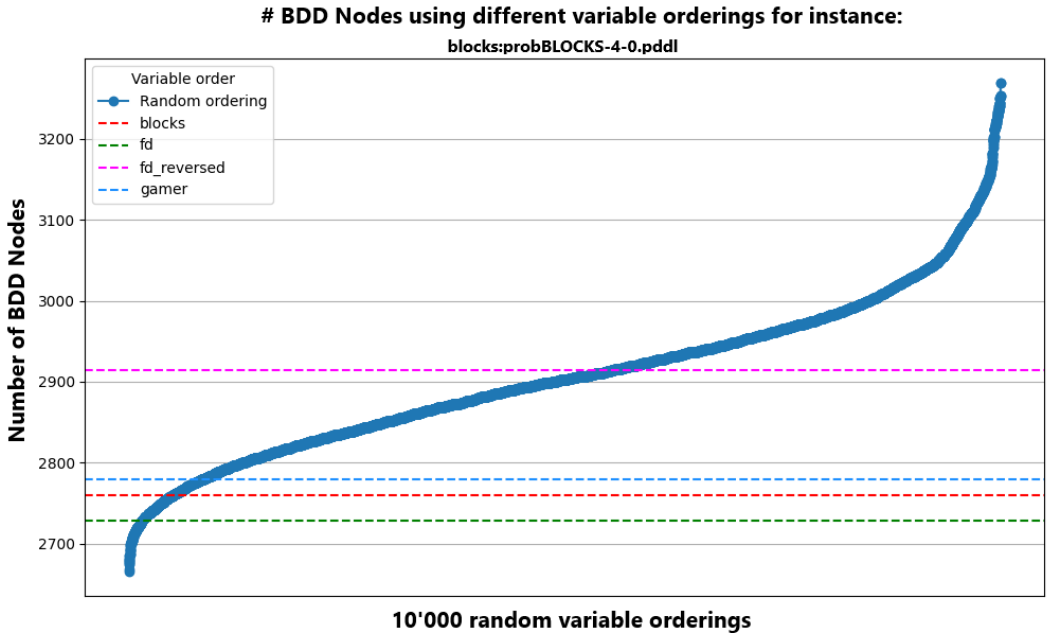
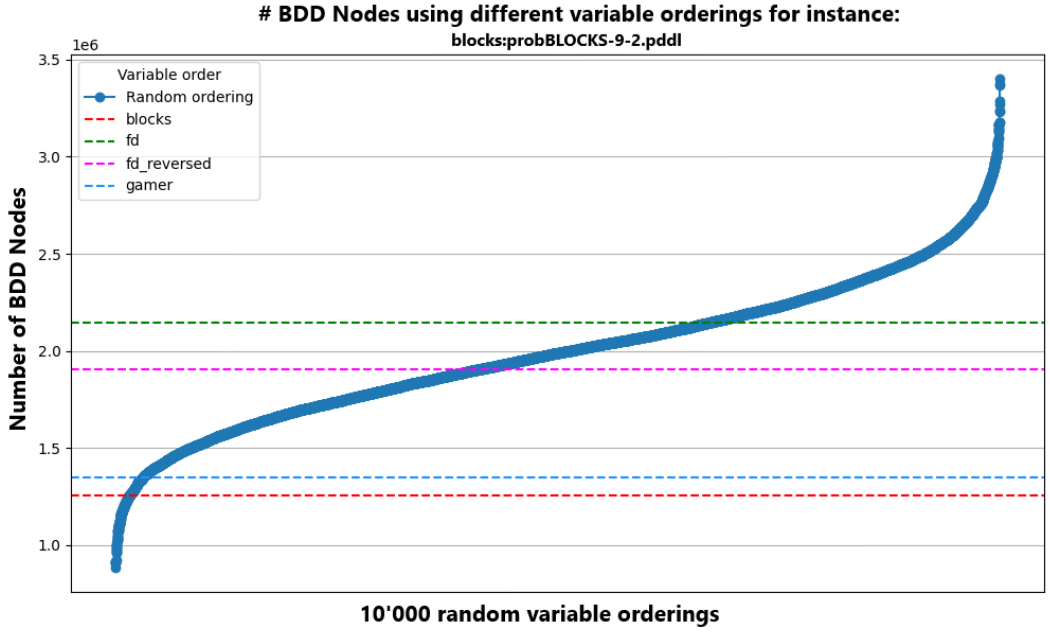


Figure 4.1: Correlation between the number of BDD nodes and the actual search time from 1'000 random runs for all instances of the domains: blocks, hiking-opt14-strips, transport-opt14-strips with time limit of 5 minutes.

Figure 4.2 shows the number of generated BDD nodes for two instances of the Blocks World domain. The thick blue graphs show the number of generated BDD nodes for 10'000 random orderings, with the x-axis being sorted by ascending values y values. The constant graphs show the number of BDD nodes for the four orderings: Blocks Ordering, Forward Ordering, Forward Ordering reversed and Gamer Ordering.
In figure 4.2(a), the results for the smallest instance of the Blocks World domain, probBLOCKS-4-0.pddl with 9 variables, can be seen. It can be noticed that all orderings are close to each other, except the Forward Ordering reversed. While the Forward Ordering reversed performs approximately like an average random ordering, the other three, including Blocks Ordering, perform much better than average.
Figure 4.2(b) shows the results for a bigger instance, probBLOCKS-9-2.pddl with 19 variables. It becomes evident that the Gamer and Blocks Ordering outperform the other two for this instance. The Forward ordering performing even worse than average random ordering. It can also be noticed that the Blocks Ordering is the best possible ordering for this instance.



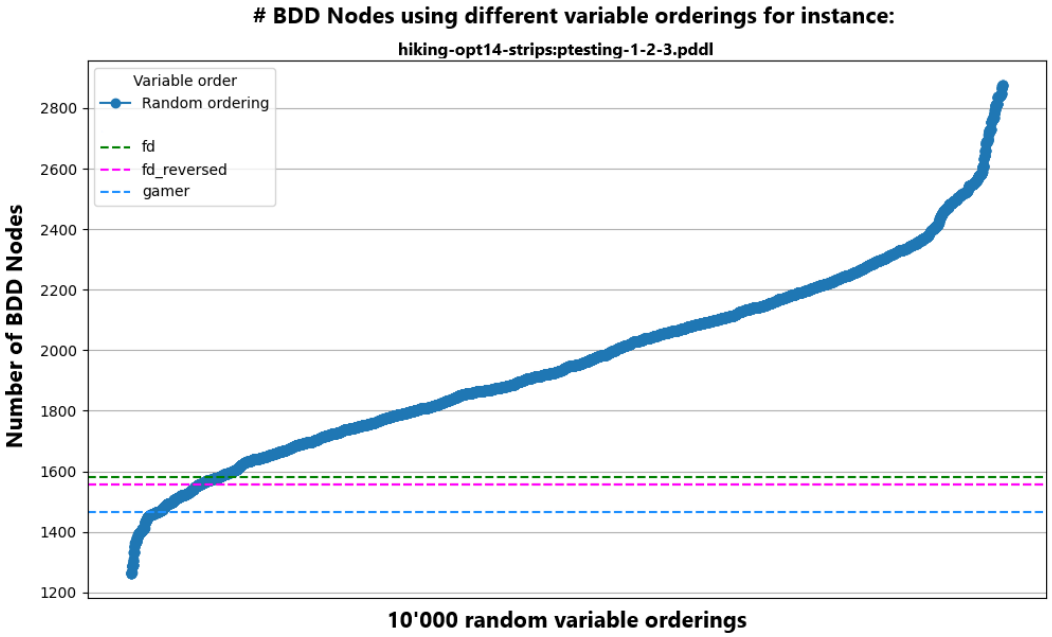
(a) blocks:probBLOCKS-4-0.pddl



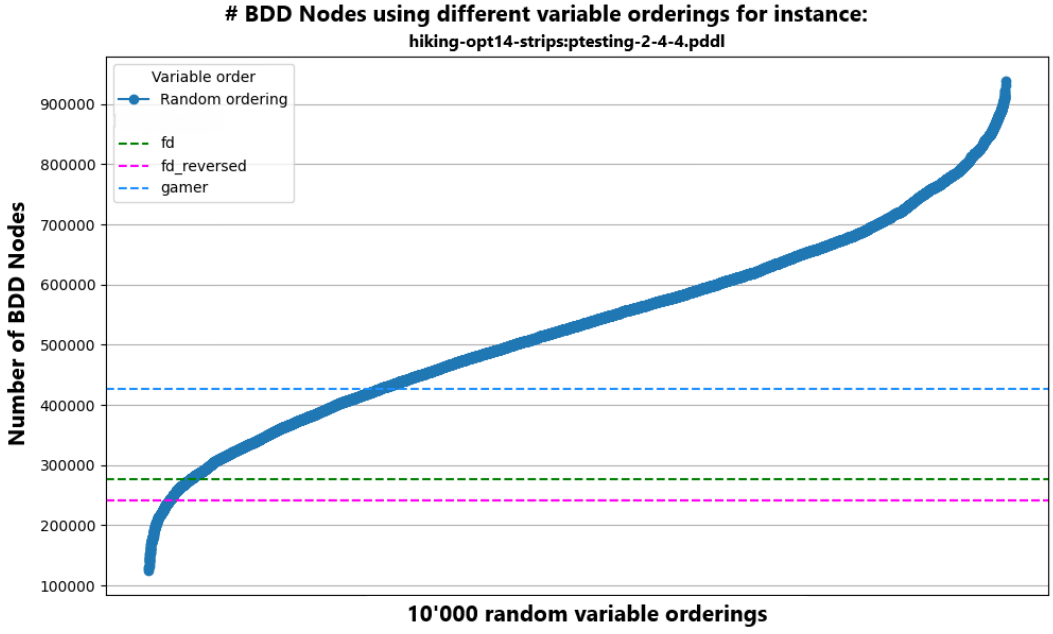
(b) blocks:probBLOCKS-9-2.pddl

Figure 4.2: Comparison between Blocks, Gamer, Forward and Reversed Forward ordering in terms of the number of BDD nodes for two instances of the Blocks World problem. The thick blue line shows in ascending order on the x-axis the number of BDD nodes for 10'000 random variable orderings as a comparative value.

In figure 4.3 the number of generated BDD nodes for two instances of the domain hiking-opt-14 can be seen. As the Blocks Ordering is really domain specific for the Blocks World problem, it is not considered in the following figures.
A direct observation of the figures allows the conclusion to be drawn that all three orders are superior to the average random variable order. As an exemplar, consider the ptesting-1-2-3.pddl instance with seven variables (Figure 4.3(a)). In this case, the number of generated BDD nodes is approximately the same, with the Gamer Ordering resulting in the fewest.
In contrast, the larger instance with 14 variables demonstrates a lower performance for the gamer ordering, exhibiting only a slight improvement over the average random ordering. The search with the other two orderings result in a comparable number of generated nodes, with the Reversed Forward Ordering showing a slight advantage. (Figure 4.3(b)



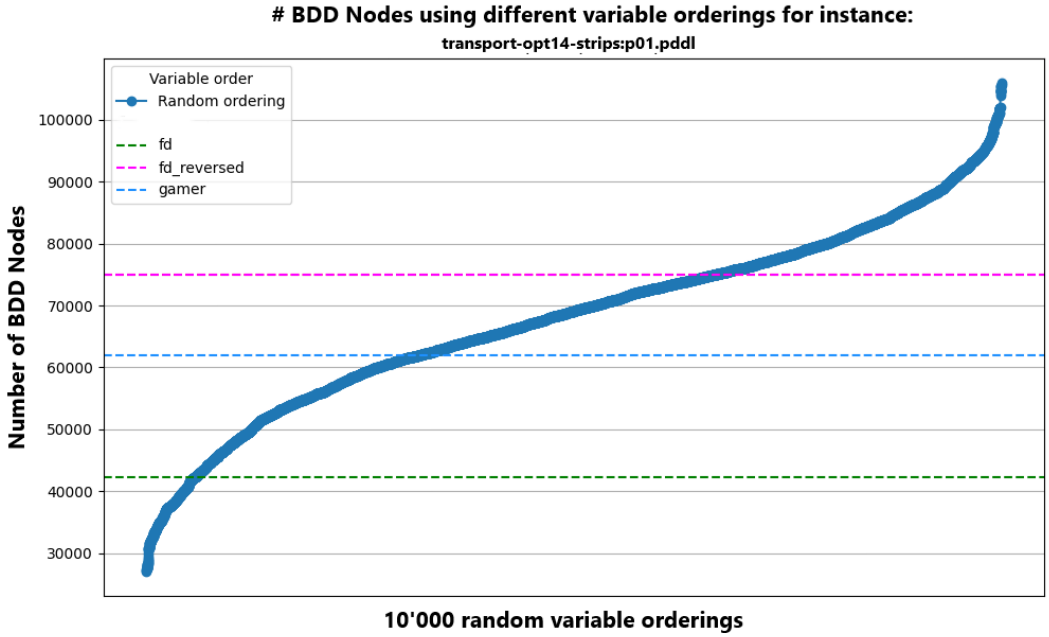
(a) hiking-opt14-strips:ptesting-1-2-3.pddl



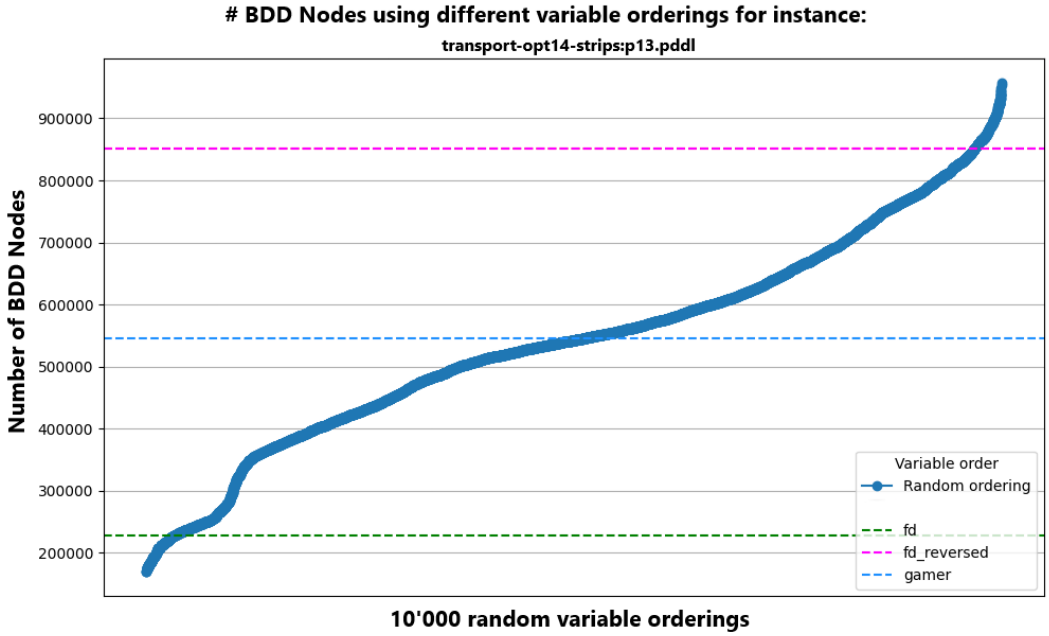
(b) hiking-opt14-strips:ptesting-2-4-4.pddl

Figure 4.3: Comparison between Blocks, Gamer, Forward and Reversed Forward ordering in terms of the number of BDD nodes for two instances of the hiking-opt14-strips problem. The thick blue line shows in ascending order on the x-axis the number of BDD nodes for 10'000 random variable orderings as a comparative value.

Different from the other domains, for the transport-opt14-strips problem there are significant differences in performance when different orderings are used for the search, as can be seen in figure 4.4.
Both instances show the same pattern. Forward Ordering leads to the least number of generated BDD nodes, the Reversed Forward Ordering to the most and the Gamer ordering is around average, compared to the 10'000 random orderings. Although this result is already pronounced in the small instance with eight variables (Figure 4.4(a)), the discrepancy is even more significant when considering the larger instance(Figure 4.4(b)). In this case, Forward Ordering leads close to the optimal, while the Reversed Forward Ordering leads close to the least optimal random ordering result.



(a) transport-opt14-strips:p01.pddl



(b) transport-opt14-strips:p13.pddl

Figure 4.4: Comparison between Blocks, Gamer, Forward and Reversed Forward Ordering in terms of the number of BDD nodes for two instances of the transport-opt14-strips problem. The thick blue line shows in ascending order on the x-axis the number of BDD nodes for 10'000 random variable orderings as a comparative value.

Table 4 shows the number of BDD nodes for all instances of the Blocks World domain. The orderings: Forward (fd), Reversed Forward (fd_reversed), Gamer, Blocks, Random Min., Random Avg. and Random Max. For all instances 1'000 Random Orderings were run. It can be seen that the best out of the 1'000 random orderings is better than all the other orderings for almost every instance. However, that there is one exception. Namely for the instance probBlocks-6-0.pddl, where the search with the Blocks Ordering terminates with even less BDD nodes. In general, the Blocks Ordering outperforms the Gamer Ordering for most of the instances. It is also of interest to note that the search for the instances probBlocks-10-0.pddl, probBlocks-10-1.pddl and probBlocks-10-1.pddl with Forward Ordering does not terminate within the specified time, whereas all other orderings terminate. For all the larger instances, none of the tested orderings are able to find a plan within the given time. Therefore, they are not included in the table.

No. generated BDD nodes for each ordering and domain: blocks							
probBLOCKS-	fd	fd_reversed	gamer	blocks	random_min	random_avg	random_max
4-0.pddl	2'729	2'914	2'779	2'760	**2'687**	2'906	3'269
4-1.pddl	2'744	2'759	2'706	2'698	**2'622**	2'828	3'110
4-2.pddl	2'688	2'752	2'637	2'705	**2'584**	2'795	3'085
5-0.pddl	7'740	7'766	7'462	6'906	**6'825**	7'625	8'640
5-1.pddl	7'525	7'505	7'438	6'888	**6'685**	7'852	8'584
5-2.pddl	7'903	7'956	7'762	7'402	**7'197**	8'070	9'168
6-0.pddl	21'663	20'098	19'318	**16'419**	16'928	19'795	22'470
6-1.pddl	20'477	21'442	20'625	19'326	**18'602**	21'959	25'567
6-2.pddl	25'019	23'861	24'035	20'999	**20'709**	24'746	29'137
7-0.pddl	78'372	77'591	79'703	62'214	**57'173**	76'685	96'278
7-1.pddl	96'693	89'485	96'170	76'346	**69'098**	92'903	116'492
7-2.pddl	96'493	90'65	94'383	71'321	**66'755**	91'521	114'297
8-0.pddl	376'149	513'936	410'304	378'555	**337'136**	503'215	713'504
8-1.pddl	447'400	495'100	429'105	386'315	**342'432**	507'102	709'573
8-2.pddl	314'670	412'661	366'677	328'139	**269'284**	403'238	600'806
9-0.pddl	1'761'462	1'633'487	1'398'548	1'576'599	**1'049'560**	2'056'195	3'204'175
9-1.pddl	1'7670044	1'941'773	2'002'606	1'593'867	**958'791**	1'998'799	3'498'922
9-2.pddl	2'147'168	1'906'061	1'349'475	1'254'512	**975'943**	2'001'918	3'014'648
10-0.pddl		7'559'152	10'142'085	5'515'395	**4'575'285**	9'838'566	14'650'466
10-1.pddl		8'930'919	11'690'140	10'449'805	**7'412'743**	11'955'211	16'984'492
10-2.pddl		6'727'287	7'084'823	7'768'799	**4'211'680**	9'855'783	14'523'085
11-0.pddl							
11-1.pddl							
11-2.pddl							

Table 4.2: Number of BDD nodes for all instances of the Blocks World domain for Forward, Reversed Forward, Gamer, Blocks and the minimum/maximum/average of 1'000 random variable orderings. The best performing ordering, with the smallest number of BDD nodes, is highlighted in green.

The table 4 presents a comparison of all orderings for the domain "hiking." It should be noted that the blocks ordering is not included, as it was created to optimize search for the Blocks World problem. As already seen in table 4, the best results are obtained when by using the best out of the 1'00 random orderings. It is clearly noticeable that the Reversed Forward Ordering is the optimal ordering for this domain, even solving an instance within the given time that none of the other orders can solve. The Gamer Ordering" has the poorest results. For some instances, it is even less effective than the average Random Ordering.

No. generated BDD nodes for each ordering and domain: hiking-opt14-strips						
ptesting-	fd	fd_reversed	gamer	random_min	random_avg	random_max
1-2-3.pddl	1'581	1'557	1'467	**1'288**	1'966	2'873
1-2-4.pddl	3'025	2'924	2'891	**2'302**	3'950	6'151
1-2-5.pddl	8'989	8'611	8'348	**6'061**	11'924	19'471
1-2-7.pddl	23'080	23'134	22'215	**15'142**	32'359	56'832
1-2-8.pddl	34'004	34'336	33'016	**21'933**	48'549	87'847
2-2-3.pddl	16'556	12'396	22'027	**10'786**	24'506	37'732
2-2-4.pddl	125'673	69'066	154'471	**65'339**	191'205	318'853
2-2-5.pddl	1'132'249	498'681	1'279'251	**439'233**	1'594'378	2'916'351
2-2-6.pddl		1'894'049	5'657'965	**1'765'435**	4'513'811	7'800'984
2-2-7.pddl		**5'504'408**				
2-2-8.pddl						
2-3-4.pddl	267'919	242'045	376'394	**99'734**	362'712	625'982
2-3-5.pddl	2'930'678	2'319'895	4'250'647	**791'328**	3'210'558	5'831'557
2-3-6.pddl						
2-3-7.pddl						
2-4-3.pddl	27'202	27'324	38'176	**18'171**	46'953	74'204
2-4-4.pddl	276'540	241'256	426'525	**134'612**	536'626	895'814
2-4-5.pddl	3'352'070	2'453'229		**1'239'217**	2'957'221	5'033'305
2-4-6.pddl						
2-4-7.pddl						

Table 4.3: Number of BDD nodes for all instances of the hiking-opt14-strips domain for Forward, Reversed Forward, Gamer and the minimum/maximum/average of 1'000 random variable orderings. The best performing ordering, with the smallest number of BDD nodes, is highlighted in green.

The results for the domain transport-opt14-strips are shown in table 4. The "Gamer Ordering" was excluded once more as it is only meaningful when considered in the context of the Blocks World domain. It is apparent that only a small number of instances are solved at all. Also in this case, the best random ordering is the most effective ordering, solving the search with the fewest BDD nodes. The results demonstrate that the Gamer Ordering is approximately average, while the Reversed Forward Ordering is notably inferior. In comparison, the Forward Ordering performs remarkably well, solving instances that the other orderings don't.

No. generated BDD nodes for each ordering and domain: transport-opt14-strips						
	fd	fd_reversed	gamer	random_min	random_avg	random_max
p01.pddl	42'252	74'698	61'976	**27'282**	68'140	104'870
p02.pddl	770'582	1'545'442	1'025'546	**500'843**	1'135'348	1'894'820
p03.pddl	10'057'689			**8'132'250**	13'847'062	22'065'092
p04.pddl						
p05.pddl						
p06.pddl						
p07.pddl	222'552	649'062	407'951	**201'023**	543'245	998'361
p08.pddl						
p09.pddl						
p10.pddl						
p11.pddl						
p12.pddl						
p13.pddl	228'002	851'982	546'626	**174'286**	541'351	939'113
p14.pddl	7'691'581			**4'695'803**	9'053'570	12'223'319
p15.pddl						
p16.pddl						
p17.pddl						
p18.pddl						
p19.pddl						
p20.pddl						

Table 4.4: Number of BDD nodes for all instances of the transport-opt14-strips domain for Forward, Reversed Forward, Gamer and the minimum/maximum/average of 1'000 random variable orderings. The best performing ordering, with the smallest number of BDD nodes, is highlighted in green.

5
Discussion

In this section the results from chapter 4 are discussed regarding the impact of different variable orderings on the performance of symbolic search in classical planning, particularly focusing on the new Blocks Ordering for the Blocks World domain.

The Blocks Ordering demonstrated superior performance compared to other existing orderings, like Gamer, Forward and Reversed Forward Ordering in the Blocks World problem, where the Blocks Ordering resulted in a notable reduction in the number of BDD nodes generated. This reduction in the number of BDD nodes correlates with a reduction in search time and therefore in an increase of computational efficiency.

This improvement can be explained by the fact that the other orderings are designed to be used for general problems, even though the Gamer Ordering works in a comparable way by analyzing the domain. In contrast, the Blocks Ordering is designed to focus only on the Blocks World domain and optimize the performance of Symbolic Search only for that problem. The Blocks Ordering method optimises the sequence of variables in a way that is closely aligned with the structure of the problem. Variables that are related are placed as close as possible together. As the experiments have shown, this results in a more compact BDD and a more efficient search.

The comparison of the other variable orderings such as the Gamer, Forward and the Reversed Forward ordering across different planning domains does not lead to a clear result as to which is best. The Gamer Ordering, which aims to group variables that are closely related together based on an analysis of the domain's causal graph is, did not outperform the other orderings for all domains.

These results demonstrate that there is no general solution for the variable ordering in symbolic search which always provides the best results. Each domain has unique characteristics that influence which ordering will be most effective.

6
Conclusion

The experimental analysis presented in this thesis shows that the variable ordering plays an important role in the performance of symbolic search in classical planning. The new Blocks Ordering has shown to be effective in the Blocks World domain, outperforming other variable orders like Forward, Reversed Forward and Gamer Ordering in terms of the number of BDD nodes generated and consequently in search time. The results suggest that domain-specific optimizations can lead to improvements in search performance. However, the performance improvement by the Blocks Ordering is achieved by focusing on a single domain. No ordering strategy was optimal across all tested domains.

Bibliography

[1] Christer Bäckström and Bernhard Nebel. Complexity results for SAS^+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[2] Randal E. Bryant. Symbolic manipulation of Boolean functions using a graphical representation. In Hillel Ofek and Lawrence A. O'Neill, editors, *Proceedings of the 22nd ACM/IEEE Conference on Design Automation (DAC 1985)*, pages 688–694, 1985.

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

[4] Naresh Gupta and Dana S. Nau. On the complexity of blocks-world planning. *Artificial Intelligence*, 56(2–3):223–254, 1992.

[5] Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[6] Peter Kissmann and Stefan Edelkamp. Improving cost-optimal domain-independent symbolic planning. In Wolfram Burgard and Dan Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)*, pages 992–997. AAAI Press, 2011.

[7] Peter Kissmann and Jörg Hoffmann. What's in it for my BDD? on causal graphs and variable orders in planning. In Daniel Borrajo, Subbarao Kambhampati, Angelo Oddi, and Simone Fratini, editors, *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, pages 327–331. AAAI Press, 2013.

[8] Peter Kissmann and Jörg Hoffmann. BDD ordering heuristics for classical planning. *Journal of Artificial Intelligence Research*, 51:779–8046, 2014.

[9] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[10] Jendrik Seipp, Florian Pommerening, Silvan Sievers, and Malte Helmert. Downward Lab. https://doi.org/10.5281/zenodo.790461, 2017.

[11] John Slaney and Sylvie Thiébaux. Blocks World revisited. *Artificial Intelligence*, 125 (1–2):119–153, 2001.

[12] David Speck, Robert Mattmüller, and Bernhard Nebel. Symbolic top-k planning. In Vincent Conitzer and Fei Sha, editors, *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, pages 9967–9974. AAAI Press, 2020.

[13] Álvaro Torralba. *Symbolic Search and Abstraction Heuristics for Cost-Optimal Planning*. PhD thesis, Universidad Carlos III de Madrid, 2015.