Heuristics for Multivalued Decision Diagrams in Branch & Bound

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Jonathan Tjong

Algorithmics Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Heuristics for Multivalued Decision Diagrams in Branch & Bound

Author: Jonathan Tjong
Student id: 5232597

Abstract

Decision diagrams have steadily become more prominent in the field of combinatorial optimization, being able to outperform the state-of-the-art in e.g. scheduling problems [13]. They have proven even more capable with the introduction of methods such as decision diagram-based Branch & Bound. Often, layer-based binary decision diagram (BDD) encodings are used to encode the problem domain, where a layered structure determines the decisions and decision variables. Recently, state-based multivalued decision diagram (MDD) encodings have also been considered. These do not rely on a layered structure, as instead, each state makes its decisions on the variables independently of other states. This allows for more flexible decision-making as states do not have to compromise with other states on the decisions they make.

This thesis compares these newer state-based MDDs with the commonly used layer-based BDDs, while also introducing and evaluating heuristics for the state-based MDDs. These heuristics include a new beam restriction heuristic that limits the branching factor of the MDDs, a dynamic variable ordering strategy adapted for the new state-based context, and a new local bound for the maximum independent set problem (MISP).

The experiments of this thesis show that the state-based MDDs generally outperform the layer-based BDDs in both runtime and search tree size. Only for some instances with low graph densities, the state-based MDDs were slower. This is resolved with the newly introduced beam restrictions, which can significantly lower the runtime. This speedup is also shown for the graph coloring problem, although the application of the beam restrictions is not as straightforward there as with the MISP. Both the dynamic variable ordering and the new local bound for the MISP show great promise in increasing the efficiency of the search, but are both held back by the additional overhead they introduce. Fortunately, these two techniques can share the added overhead while gaining the combined benefits, resulting in great performance for the MISP when both methods are used together.

Thesis Committee:

University supervisor:	Prof. Dr. M.M. de Weerdt, Faculty EEMCS, TU Delft
Weekly supervisor:	Prof.Dr. W.-J. van Hoeve, Carnegie Mellon University
Committee Member:	Prof. Dr. Y. Murakami, Faculty EEMCS, TU Delft

Preface

With this thesis, I am nearing the end of my journey at the TU Delft. These past 5 years flew by, but also felt twice as long. I am very happy with how these 5 years have gone, and I feel both lucky and proud that things ended up the way they did.

I would like to thank my two supervisors, Mathijs de Weerdt and Willem-Jan van Hoeve. I must admit, when Mathijs de Weerdt proposed the idea of doing my thesis with him and Willem-Jan van Hoeve, it seemed a bit daunting at first. But in the end, I am very grateful for this (lucky) opportunity to have happened. Mathijs de Weerdt guided me in the right direction and made sure the thesis ended up in the right place. Giving good, constructive feedback and always encouraging throughout. Willem-Jan van Hoeve was very helpful and knowledgeable in his field. But above all, he was always enthusiastic and optimistic whenever we discussed ideas and results.

I would also like to thank my family and friends, who have always encouraged and supported me. No matter what choices I would make, I knew they would be with me cheering me on.

Jonathan Tjong
Delft, the Netherlands
June 26, 2025

Contents

List of Figures

Chapter 1

Introduction

Decision diagram-based optimization (DDO) [7] is the field that uses decision diagrams to represent and solve (combinatorial) optimization problems. While decision diagrams have been around for a long time, mainly being used to represent problem structures, they increasingly have been used in optimization by creating bounds. With the introduction of decision diagram-based Branch & Bound, it has become possible to actually solve NP-hard problems and compete with other state-of-the-art solvers. This method works similarly to the Branch & Bound algorithm used in e.g. mixed integer linear programming [28], using upper and lower bounds from decision diagrams to search and prune the state space.

However, while the DD-based solvers have improved significantly over time, most applications still use the same layer-based structure for their decision diagrams. This is often in the form of a layer-based decision diagram encoding, in the case of this thesis, a layer-based binary decision diagram (BDD). Here, the search is structured in layers, and each layer considers a specific decision variable of the optimization problem. However, this means that decisions are restricted to each layer, and all states in the layer have to decide on the same variable, which may not always be optimal. Sometimes the chosen variable is not even relevant to a state in the layer, leading to redundant transitions being made.

Recently, state-based decision diagram solvers have been explored, no longer relying on the commonly used layered architecture. Now, with state-based DDs, decisions are made per state and are no longer restricted to layers, making decisions more flexible and eliminating the need for pointless transitions. In the context of this thesis, we will specifically look at a state-based multivalued decision diagram (MDD).

As such, the main question this thesis will try to answer is the following:

> *Can a state-based MDD encoding surpass the commonly used layer-based BDD encoding? What heuristics can further improve this MDD encoding?*

This can be broken down into several subquestions:

1. What does the state-based MDD encoding do differently compared to the layer-based BDD encoding? And do these differences positively impact the performance?

2. Each state does not necessarily have to expand with all available decision options. A "beam search"-like restriction could be applied to manipulate the rate at which the decision diagram expands. What kind of effect would different beam sizes have on the performance of the state-based MDD encoding?

3. Could a dynamic variable ordering strategy exploit the flexible state-based structure of the MDD encoding? How would this compare to various static variable ordering strategies?

4. What would the effect be on the state-based MDD encoding if we improve local bound pruning? What if it was combined with the new dynamic variable ordering strategy?

The maximum independent set problem (MISP) will be specifically considered and used as the main example throughout this thesis. Some of the heuristics will also be applied to the graph coloring problem to explore their behaviour on a different problem domain.

The contributions of this thesis are as follows. We compare the state-based MDD encoding with the layer-based BDD encoding. Furthermore, we introduce multiple heuristics to improve the MDD encoding: a "beam restriction" heuristic and a dynamic variable ordering strategy. Finally, we present an improved local bound for the MISP.

We show that the MDD encoding outperforms the BDD encoding for most instances of the MISP in runtime and search tree size. The instances where the state-based MDDs are slower, are those with a low input graph density. The new beam restriction heuristic can speed up the search, such that the state-based MDDs end up faster than the layer-based BDDs. The speedup that the beam restrictions bring is also seen with the graph coloring problem, after some modelling modifications. The performance with the MISP can be further improved with the new local bound for the MISP, especially when used alongside the new dynamic variable ordering strategy.

The next chapter, Chapter 2, explores the related advancements and implementations in literature. Afterwards, Chapter 3 explains the necessary background information for the rest of this thesis. Chapter 4 covers the main contributions of this thesis. Chapter 5 presents the results of the performed experiments, which are then further discussed in Chapter 6. Finally, the main conclusions and future work can be found in Chapter 7.

Chapter 2

Related Work

With the introduction of approximate decision diagrams (DDs), the use of decision diagrams in optimization has become more promising as more improvements and applications are discovered. This chapter discusses the main advancements and relevant applications of these approximate DDs in literature.
The questions that this chapter aims to answer are:

- How did approximate decision diagrams develop over time?
- How did this result in a DD-based Branch & Bound (B&B) algorithm?
- What heuristics and improvements have been presented and what are their benefits and drawbacks?
- What problems were these methods used for and how well did they perform compared to other methods?
- How much have state-based decision diagrams been explored, as opposed to the regular layer-based decision diagrams?
- Is there still a gap in research which this thesis can explore and possibly answer?

To find the relevant literature, a combination of Google Scholar and secondary sources were used. The main secondary sources were surveys by Castro et al. [12] and van Hoeve et al. [42]. For Google Scholar, examples of queries that were used (in various combinations with each other) are: "relaxed decision diagrams", "heuristics", "decision diagram-based branch and bound", "variable ordering". To determine the relevance of a paper, the abstract was read, followed by the introduction and conclusion. If the paper was indeed relevant to the questions listed before, other chapters (such as the method and experimental results) were scanned or read.

Section 2.1 covers the conception and iterative improvement of relaxed and restricted DDs and the bounds they provide. Afterwards, Section 2.2 goes over how these approximate DDs are used in a DD-based Branch & Bound (B&B) algorithm, which has been modified and improved over time.

Similarly to the works covered in this chapter, this thesis aims to improve the bounds and performance of approximate DDs (within a DD-based B&B algorithm). However, most of the works mentioned in this chapter rely on layer-based DD encodings of optimization

problems (where decision variables are chosen per layer), while this thesis focuses on optimization problems represented by a state-based DD encoding instead (where decision variables are chosen per state).

2.1 Approximate Multivalued Decision Diagrams

In 2007, Andersen et al. [2] proposed the idea of relaxed decision diagrams for constraint programming, whereafter Bergman et al. [3] applied them on optimization problems. Bergman et al. discovered that these relaxed DDs can be used to find a dual bound on the solution. And unlike exact DDs, relaxed DDs do not grow exponentially large. An exact DD has to represent the entire solution space, which can become exponentially large in the worst case when dealing with NP-hard problems. Relaxed DDs can limit the size of the diagram by relaxing the problem, making for a reliable way to obtain a dual bound. Bergman et al. also introduce a top-down construction method for the construction of the relaxed MDDs, where layer after layer is constructed, ensuring each layer stays within the maximum width. They show that the dual bounds from relaxed MDDs are promising in the context of optimization and that they can outperform the commonly used linear programming (LP) bounds.

Cire and van Hoeve [13] utilized the bounds from relaxed MDDs for scheduling problems, such as constraint-based scheduling. They show that these relaxed MDDs can significantly improve state-of-the-art solvers while keeping their generality.

Furthermore, they presented a method to construct relaxed DDs incrementally, instead of the top-down construction from Bergman et al. [3]. This method was adapted from Hadzic et al. [23] and Hoda et al. [24], who used approximate MDDs for constraint programming. The method starts from a 1-width MDD, whereafter nodes are split until each layer is exact or has reached its maximum width. Infeasible and suboptimal edges are filtered out in the process.

Shortly after the introduction of relaxed DDs, Bergman et al. [6] introduced their counterpart: restricted DDs. The restricted DDs under-approximate the set of feasible solutions and give a primal bound on the solution. These bounds converge to the optimal solution, while the DD itself is easy to construct. They applied the restricted DDs to set covering and set packing problems and compared their performance to that of a state-of-the-art integer programming (IP) solver. The restricted DDs outperformed the latter in certain cases with the set covering problem and worked especially well on the set packing problem.

Kinable et al. [31] combined the use of discrete relaxations from MDDs with continuous relaxations from LP, to be used within a constraint programming (CP) framework. Specifically, they use dual information from the LP relaxations to strengthen the MDD relaxations. The MDDs are integrated into the CP model to incorporate more information about the sequencing problems. This is done by making a global constraint, which is called for each search node to construct a relaxed MDD for a dual bound. The domains of the variables in the CP model are given to the MDD to remove unnecessary edges pre-emptively. The aim

is to be able to factor in several complex side constraints, such as time windows and precedence constraints. The method was tested on various time-dependent traveling salesman problems (TDTSP) and improved upon regular mixed-integer linear programming (MILP) and CP models.

Cappart et al. [11] used deep reinforcement learning to find variable orderings that can improve the bounds on relaxed and restricted DDs. The order in which variables are explored can significantly impact the efficiency of decision diagrams [5]. Deep learning is used, as the problem is too complex and the search space is too large for regular reinforcement learning. Deep reinforcement learning is able to generalize what it has learned to states that it has not had the chance to visit yet. The method is compared to other established variable orderings and seems to outperform them in certain contexts. However, it needs an estimate of the density of the input graphs during training, which makes it less general for practical use.

Maschler and Raidl [33] compared the regular top-down construction of the relaxed DDs [3] with constructing the relaxed DDs with incremental refinement [13]. For this, the approximate DDs computed bounds for a prize-collecting sequencing problem, the PC-JSOCMSR from Horn et al. [26] (the prize-collecting variant of the job sequencing with one common and multiple secondary resources problem). Both construction methods seem to perform similarly, except for instances with skewed distributions, where the incremental method works best.

Horn et al. [27] continued with constructing approximate MDDs for the PC-JSOCMSR. Similarly to this thesis, they move away from the usual layer-based approach and opt for a state-based approach instead. For the construction of the relaxed MDDs, the nodes are put in a priority queue with a dual bound heuristic, inspired by the A* algorithm. Similar nodes are merged when this queue gets too large, allowing for merging between layers (if the MDD were to have layers). Moreover, they speed up the construction of the restricted DDs by reusing the already constructed relaxed DDs. From their experiments, the new construction method generally outperforms the other common construction methods (top-down and iterative refinement) for the approximate DDs.

Overall, the approximate DDs have been studied and improved steadily over the years. They have been used in a good number of applications, as they are general techniques that can provide strong bounds on various problems. Future research could be done on combining these approximate DDs with other methods such as LP, CP and deep reinforcement learning. Furthermore, research on state-based decision diagrams (e.g. the A* approach by Horn et al. [27]) is still limited, since most applications of approximate DDs use the standard layer-based architecture. This further motivates the need for exploration around these state-based DDs, which this thesis aims to do.

2.2 Decision Diagram-Based Branch & Bound

After the introduction of relaxed DDs, Bergman et al. [7] proposed a decision diagram-based Branch & Bound (B&B) algorithm in 2016. It adapts the well-known Branch & Bound algorithm, used in e.g. mixed integer linear programming (MILP) [28], by using relaxed and restricted DDs (from Section 2.1) to provide the bounds instead of LP relaxations. They found their method to be competitive with a state-of-the-art IP solver. Furthermore, they showcase their method to be parallelizable, suitable for solving in a distributed setting. The algorithm is further explained in Section 3.7, as this thesis will explore the use of this DD-based B&B scheme and the approximate DDs used within.

Gillard et al. [20] improved this DD-based B&B algorithm by introducing two new techniques to accommodate better pruning. First, local bound pruning (LocB) creates local upper bounds for each node in the exact cutset of the relaxed DD. This can then be used to prevent nodes from being put in the priority queue and to immediately prune nodes coming out of the priority queue when possible.

Additionally, rough upper bound pruning (RUB) uses upper bounds to discard nodes *while compiling* the relaxed and restricted DDs. Thus, ruling out nodes before they are created and expanded while also allowing more room for nodes to fit in the maximum width. Though, the computational cost of the RUB needs to be very small, as it is computed for every single node of the approximate DDs. It also requires a problem-specific implementation. The two techniques show promising results individually, and even better results when both are used simultaneously. A similar method to RUB is present in the CODD framework, which is used in the experiments of this thesis.

Rudich et al. [39] proposed an alternative to B&B, peel-and-bound, by peeling off subgraphs of already constructed relaxed DDs and reusing them as starting points for constructing new relaxed DDs. Instead of a regular top-down construction of relaxed DDs [3], the incremental construction method from Cire and van Hoeve [13] is used instead. With peel-and-bound, Rudich et al. aim to reduce the repetition of filtering the same edges after splitting a node in the incremental construction method. Subgraphs are peeled off and stored, while new relaxed DDs are created from these subgraphs instead of a 1-width MDD. They show that peel-and-bound outperforms B&B on the sequence ordering problem (SOP) and allows for larger scale relaxed DDs by lowering the cost of generating the diagram iteratively.

A year later Rudich et al. [40] revisit peel-and-bound, generalizing the method, while introducing new (general) heuristics for peel-and-bound. In addition to the previously explored SOP, they show it performing well on the traveling salesman problem with time windows (TSPTW).

González et al. [22] combined the DD-based B&B algorithm and integer linear programming (ILP) to solve the maximum independent set problem (MISP) and other problems. In this work, instead of solely relying on DDs to solve the complete problem, it uses DDs to explore and extract subproblems. Subproblems of the DD vertices are either solved

with B&B (as with regular DD strategies) or ILP. Supervised machine learning is used to detect which of the two methods is better suited to expand certain vertices.

Parjadis et al. [38] continued the work by Cappart et al. [11] (Section 2.1), using deep reinforcement learning to find variable orderings for the approximate DDs. In this work, deep reinforcement learning is integrated into the B&B search. They show great improvements in efficiency (as the B&B search tree is smaller, consisting of fewer nodes) when solving instances of the MISP. However, the execution time is significantly increased, due to the overhead from multiple calls to the deep reinforcement learning network. An attempt at reducing the overhead is made by caching and reusing data, but it seems to not be enough to overcome this overhead.

Nafar & Römer [36] presented two new heuristics to improve the bounds of the relaxed DDs used in the decision diagram-based B&B algorithm. Firstly, they introduce a dynamic variable ordering called Current Degree Sum (CDS). The method looks at the remaining subgraphs of each state in the decision diagram. It uses a new concept called the "current degree" of the eligible vertices i.e. their local degree in the subgraph of a state. Across a whole layer, the "current degree sum" of a vertex is the total sum of its current degrees in that layer. With this, the variable ordering strategy they propose is to select the vertex with the minimum current degree sum per layer.

Secondly, they present a merge heuristic called Border Tie (BT) merging. When a relaxed DD has to merge nodes, instead of merging all the worst nodes falling outside of the maximum width, it first finds and merges nodes with a similar objective value to the nodes on the cutoff point induced by the width. As such, information from still viable nodes is preserved instead of being merged with potentially useless nodes.

Their experiments, using MISP instances, show improved bounds when either method is used and even better performance when both are used simultaneously.

This thesis will also look at dynamic orderings similar to CDS, but with a state-based MDD encoding for the MISP instead of a layer-based BDD encoding. With a state-based MDD, a dynamic ordering may have even more potential, as variable decisions are made for each state rather than each layer. Though, more reordering takes place, as the vertices need to be reordered per state instead of per layer.

DD-based B&B has proven itself to be a viable method to solve discrete optimization problems, especially with the introduction of various improvements and heuristics. Hybrid approaches with ILP and deep reinforcement learning are also promising, which may lead to further research in the future. Similarly to the previous Section 2.1 though, state-based decision diagrams have not been fully explored within a DD-based B&B algorithm. Fortunately, further research is made simpler with the newly introduced CODD framework [34], which is made with state-based DD-based B&B as its focus. This makes it more accessible to implement state-based B&B models. Thus, the aim of this thesis is to use this opportunity to further explore state-based B&B. This is partly done by learning from these existing advancements to layer-based B&B, modifying and applying some of them to this new state-based architecture.

Chapter 3

Preliminaries

This chapter covers several related concepts that serve as the foundation for the rest of the paper. The concept of decision diagrams is explained in the next section, Section 3.1. Following this, the maximum independent set problem is defined in Section 3.2, with its Binary Decision Diagram (BDD) and Multi-Valued Decision Diagram (MDD) encoding covered in Section 3.3 and 3.4 respectively. The graph coloring problem is also briefly explained along with its MDD encoding in Section 3.5. Afterwards, Section 3.6 discusses approximate decision diagrams, relaxations of the regular decision diagrams. These are used in the decision diagram-based Branch & Bound algorithm, further explored in Section 3.7. Heuristics for the Branch & Bound method, variable ordering and local bound pruning, are covered in Section 3.8 and 3.9. Finally, a somewhat different method, the beam search algorithm, is explained in Section 3.10. This algorithm served as inspiration for a new method introduced in this thesis.

For further insight on the topic of decision diagrams, a recent survey by van Hoeve [42] is recommended. Castro et al. [12] also published a survey on recent advances surrounding this topic. Lastly, the paper by Ow and Morton [37] covers the foundations of beam search, while introducing filtered beam search.

3.1 Decision Diagrams

Decision diagrams (DDs) started as simple graphical representations, mainly used for Boolean functions (for example, by Akers in 1978 [1]). However, more recently, they have also been used in optimization [12], e.g. scheduling [13], routing [31], and regionalization [21]. Furthermore, with the introduction of relaxed decision diagrams in 2007 by Andersen et al. [2], a decision diagram-based Branch & Bound algorithm was presented by Bergman et al. [7] in 2016. This, similarly to existing Branch & Bound algorithms, utilizes upper and lower bounds to solve problems efficiently (further explained in Section 3.7).

A decision diagram (as described by Castro et al. [12]) is a (weighted) directed acyclic graph or DAG $D = (N, A)$, with a node set N and an arc set A. Unlike search trees, a DAG allows nodes to have more than one parent node. Each arc $a \in A$ represents a transition between nodes in consecutive layers. The root node $r \in N$ is the initial node (situated in the

first layer), and the leaf nodes $t \in N$ are the terminal nodes (all nodes in the last layer). Each arc in the graph has a label (indicating the decision made) and an arc length. These decisions form the graph into a layered structure, as for each decision stage, there is a corresponding layer of nodes in the graph. A path p from the root r to any terminal node t represents a solution to the problem (consisting of the decisions made by the transitions in the path), along with its total length corresponding to its objective value. Furthermore, a distinction between Binary Decision Diagrams (BDDs) and Multi-Valued Decision Diagrams (MDDs) is made, where at each node a binary or multi-valued decision is made, resulting in each node having two or any number of outgoing arcs, respectively. Following this, each node can be expanded starting from the root node, resulting in the whole state-space being represented in an exact decision diagram. From this, the optimal solution (for a maximization problem) can be found by taking the longest path p.

The manner in which decision diagrams operate is reminiscent of dynamic programming (DP), where states and state transitions are traversed to efficiently solve the problem. Hooker [25] even showed how to connect the theory of weighted decision diagrams with that of DP transition graphs.

The next three sections (Section 3.2, 3.3 and 3.4) define the maximum independent set problem and show how decision diagrams could be used to encode this problem in different ways, either with a BDD or an MDD.

3.2 The Maximum Independent Set Problem

The maximum independent set problem (MISP) [19], a well-known NP-Hard problem, is the main example problem considered in this thesis.

Given a graph $G = (V, E)$, with a set of vertices V and a set of edges E, the aim of the MISP is to find the maximal subset of vertices $Z \subseteq V$, such that no vertices in Z are directly connected to each other by an edge in E.

Figure 3.1 shows an example of a graph G with five vertices. The optimal solution for this instance of the MISP is the following set of vertices: $\{1, 4, 5\}$. These vertices are not neighbours of each other. Including either vertex 2 or 3 would not be allowed, as they are directly connected to some of the vertices that are already in the set. Any other set of vertices would either violate the rules of the problem or be smaller than the optimal solution.

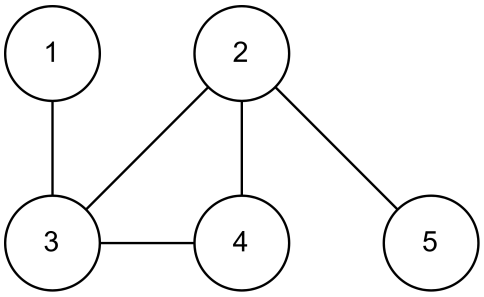


Figure 3.1: Example of an input graph for the maximum independent set problem (MISP). Each vertex is labelled with a number shown in each centre.

3.3 A BDD Encoding For the MISP

The MISP is commonly encoded with a Binary Decision Diagram (BDD) [4], where each state contains a set of eligible vertices $S \subseteq V$. A BDD can be seen in Fig. 3.2, encoding the example problem of the MISP in Fig. 3.1. For the ith layer L_i, a vertex v_i is chosen according to the variable ordering strategy assigned at the start. Then, all nodes (i.e. states) in layer L_i make the decision to include or exclude v_i in the solution. This is done by each state transitioning to two other states (one transition for including v_i, one transition for excluding v_i). These transitions follow the state transition function $f(S, v_i)$ seen in Equation (3.1).

$$f(S, v_i) = \begin{cases} S \setminus \{N(v_i), v_i\} & \text{if include } v_i \text{ in solution} \\ S \setminus \{v_i\} & \text{if exclude } v_i \text{ from solution} \end{cases} \tag{3.1}$$

If v_i is included in the solution, v_i and all adjacent vertices are excluded from the set of eligible vertices S (since no adjacent vertices are allowed in the independent set of the MISP). If instead, v_i is chosen not to be included in the solution, it transitions to the state where only v_i is excluded from S. This is repeated until no further decisions can be made (when the set of eligible vertices S is empty). A more detailed DP model can be found in Appendix A.1.2.

Note that it is possible that a state in L_i does not contain v_i. In these cases, the state does not make a transition to include v_i, only making the transition where it excludes v_i (to an identical state, as no changes are made to the set of eligible vertices).

There are some BDD variants with arcs that traverse multiple layers (by skipping unnecessary layers). One such example is Minato's Zero-Suppressed BDD [35], which eliminates nodes that point to the terminal node. These decision diagrams can become more compact, though they still follow the same decision-making as regular BDDs.

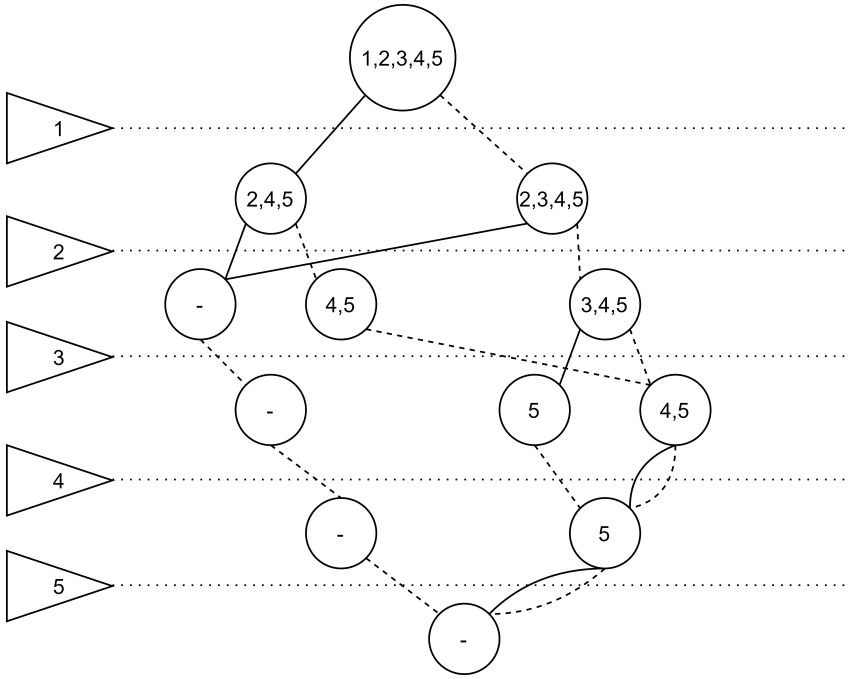


Figure 3.2: Example of an exact BDD for MISP example in Fig. 3.1. Each state in the BDD shows the eligible vertices available at that particular state. The variable decisions are represented on the left, indicating which vertex is chosen for each layer. The solid lines represent the choice to include the chosen vertex of that layer, while the dashed line represents the choice of not including it.

3.4 An MDD Encoding For the MISP

While the MISP is commonly encoded with a BDD as described in Section 3.3, another way to encode the MISP is with a Multi-Valued Decision Diagram (MDD), introduced by Curry et al. [15]. An example of an MDD for the example MISP instance from Fig. 3.1 can be seen in Fig. 3.3. In a BDD, for each state S in a layer, the decision is made to include or exclude a single vertex in the independent set. However, with an MDD, a state S makes $|S|$ transitions to other states, where each transition represents the decision to include a different eligible vertex in the independent set. The transitions follow the state transition function $f(S, v_i)$ seen in Equation (3.2).

$$f(S, v_i) = S \setminus (N(v_i) \cup \{v_1, ..., v_i\}) \qquad \forall v_i \in S \tag{3.2}$$

where $N(v_i)$ indicates the set of neighbors of vertex v_i. Again, each transition represents including a vertex v_i from state S in the solution. Including a vertex $v_i \in S$ in the solution excludes v_i and the neighboring vertices $N(v_i)$ from the next state according to the rules of the MISP. Additionally, the state S is sorted, meaning vertices with a lower numbered label $v_1 ... v_{i-1}$ can be excluded for symmetry breaking (hence why the set of vertices $\{v_1, ..., v_i\}$ is removed from S in the transfer function). The full DP model can be found in Appendix A.1.2.

With this encoding, decisions are not confined to the layered structure like with a regular BDD. Decisions depend entirely on each individual state, rather than being the same for a whole layer. As such, more informed decisions can be made locally. This also avoids the cases in a BDD, where a decision is made for a certain variable v_i across a whole layer, while some states in that layer may not contain v_i. For these states, only the decision to exclude v_i is possible and as such, a redundant transition is made (generating an identical state).

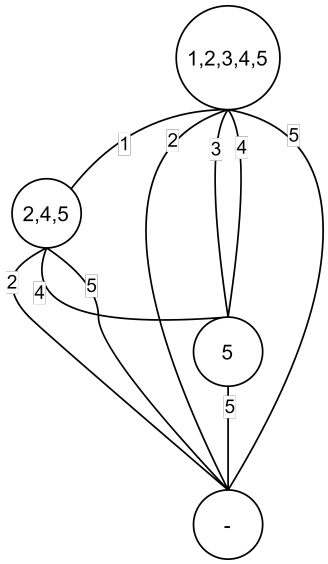


Figure 3.3: Example of an exact MDD for MISP example in Fig. 3.1. Each state in the MDD shows the eligible vertices available at that particular state. The variable decisions are shown with numbers in the transition lines, indicating which vertex is chosen to be included in the independent set.

3.5 The Graph Coloring Problem

Another NP-hard problem that is used in some of the experiments in this thesis is the graph coloring problem [30]. Given a graph $G = (V,E)$, with the sets of vertices V and edges E, each vertex can be assigned a color. An assignment of colors where no adjacent vertices share the same color is called a "vertex coloring". The aim of the problem is to find the chromatic number $\chi(G)$, i.e. the minimum number of colors needed to create a valid vertex coloring for the graph G.

This thesis adapts the MDD encoding used in the CODD framework [34]. Each state S in the MDD tracks the color assignment for the already covered vertices. For each vertex $v_i \in V$, its set of available colors C_i is considered. C_i consists of all k previously assigned colors plus a new color $k+1$, minus the colors of neighbouring vertices. The current state S_i makes a transition for each available color in C_i, where each transition assigns a different color to v_i in the next state S_{i+1}. A more detailed DP model for the graph coloring problem can be seen in Appendix A.2

Figure 3.4 shows an example of a state transitioning into other states when following this MDD encoding. At the top of this example is a partially colored graph where vertex v_5 still needs to be assigned a color. The state makes a transition for each color in $C_5 = \{0,1,3\}$. The previously assigned colors are $\{0,1,2\}$ and the next color 3 is also considered. However, the neighbour v_2 already has color 2, which invalidates that color for v_5.

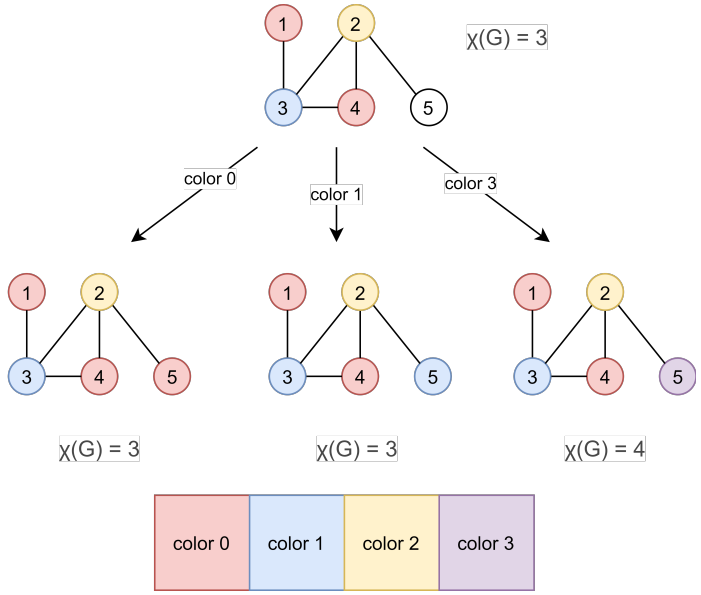


Figure 3.4: Example of the behaviour of the MDD encoding for the graph coloring problem. A partially colored graph is considered, where v_5 needs to be assigned a color. Each vertex is labelled with a number shown in each centre. The available colors are shown in the table below the graph. The coloring number $\chi(G)$ is shown for each graph.

3.6 Approximate Decision Diagrams

Often it is infeasible to use only exact decision diagrams to represent the state space to solve a particular problem. The state space is usually exponential in size, so the diagram will grow accordingly. Therefore, approximate diagrams are used to make traversing the state space more feasible.

Approximate decision diagrams can be either a *relaxed* decision diagram or a *restricted* decision diagram. Both types restrict each layer of the decision diagram with a maximum allowed width, a maximum number of nodes per layer.

A relaxed decision diagram does this by merging nodes in a layer until the maximum width is achieved. The merge is allowed to introduce infeasible solutions, as long as no feasible solutions are lost and the objective function is dual-approximated (i.e. the approximate solution or path from root to terminal is not worse than the exact solution or path). Therefore, the relaxed solution space is a superset of the original solution space, resulting in a dual bound on the solution.

Alternatively, a restricted decision diagram adheres to the maximum width by removing the least promising nodes, losing feasible solutions while still guaranteeing that the remaining solutions remain feasible. This results in a subset of the original solution space and a primal bound on the solution.

It is worth noting that merging or removing exact nodes does not always have to result in adverse effects. This is the case with node domination. Node n_1 dominates node n_2 iff any partial solution after n_2 can also be applied after n_1, while n_1 has the same or a better objective value than n_2 [14]. In the special case that a node dominates another node, the dominated node can be removed without preventing the optimal solution from being found.

Furthermore, the MDD encoding of the MISP explained in Section 3.4, does not actually use the concept of layers. As such, to still group states similar enough to be merged together, Curry et al. [15] propose a merge condition where states are only merged if they share the same last selected vertex. For this thesis however, we define the merge condition to allow states to only merge if they have made the same number of transitions from the root state. This is functionally the same for the regular MDD encoding. However, the latter works better for one of the new methods introduced in this thesis (beam restrictions).

3.7 Branch & Bound

Bergman et al. [7] introduced a decision diagram-based Branch & Bound algorithm, which can be used to compute the exact solution using these approximate decision diagrams. A general outline of the algorithm can be seen in Alg. 1. First, the initial state of the problem is represented in the root state (line 2). This state is put into the priority queue Q, which keeps track of the unexplored states.

For each iteration of the loop, the next state S is popped from Q according to the objective function (line 4). A restricted and relaxed DD is constructed from S to get a primal and dual bound respectively (lines 5 & 6). Additionally, an exact cutset C is extracted from the relaxed DD. This means that all nodes in C are exact (not merged) and that every path in the relaxed DD from the root vertex to a leaf vertex has to pass through one of these nodes in C (i.e. to get any possible solution, the path has to pass through a node in C). Because of this, and the fact that the nodes in C are independent of each other, the completeness of the algorithm is ensured [7]. The CODD framework (a layerless framework used in this thesis) uses a method similar to the frontier cutset [7], where the cutset C consists of all exact nodes where after the next decision step, any of the nodes are merged if needed [34].

If the dual bound is smaller than the primal bound, the state S is non-optimal and can be pruned (line 7). Otherwise, all states S' in the cutset C are added to Q (line 8), serving as new starting points to explore further. Then, the next iteration begins, popping a new state from Q. Once Q is empty, the entire search space has been sufficiently explored and the problem has been solved.

Algorithm 1 Outline of DD-based B&B

Input: w = maximum width
1: Initialize empty priority queue Q
2: Add root state to Q
3: **while** $|Q| > 0$ **do**
4: Pop state S from Q (according to objective function)
5: Create restricted DD to get primal bound
6: Create relaxed DD to get dual bound and exact cutset C
7: **if** dual bound $>$ primal bound **then**
8: **for each** state S′ in C **do**
9: Add S′ to Q
10: **end for**
11: **end if**
12: **end while**

3.8 Variable ordering heuristics

The order in which the vertices are selected is one of the most important factors in determining the efficiency of decision diagrams. This is already well-known for BDDs, as it has a big influence on the final width (and therefore size) of the BDD, as well as the strength of the bounds of relaxed BDDs [4, 5]. However, finding the optimal ordering is also known to be an NP-complete problem [8], hence why variable ordering heuristics are often used.

There are two main variants of variable orderings: static and dynamic. Static variable orderings precompute a variable ordering before any search is done. This means that it only has to compute an ordering once, making it relatively cheap to compute in the long run. However, as a result, it can only rely on the limited information available from the initial problem state, which may be less relevant to a state encountered later on during search. On the other hand, the dynamic variable orderings change the order of the variables while exploring the search space. It has information on the current state during search, which allows for better decision-making on the go. Though, keeping track of this information dynamically introduces more complexity and overhead, which may be more time-consuming than the efficiency gained with the improved decision-making.

This thesis explores the effect of various variable ordering heuristics on the performance of state-based MDD encodings. This is done for the MISP as our main running example. The following existing static variable orderings for the MISP are considered in the experiments of this thesis:

Maximum Degree

This is a straightforward variable ordering, where the vertex with the maximum degree is repeatedly selected. After a vertex is selected, it is removed from the graph and the degrees of the neighboring vertices are updated accordingly.

Min Width

Remove the vertex with the minimum degree from the graph, removing any edges it may share with other vertices [17]. Then add it in front of any previously selected vertices in the ordering (s.t. the first vertex removed from the graph will be considered last in the ordering). Repeat until the graph is empty.

Maximum Connectivity

Repeatedly select an unselected vertex that shares the most edges with other previously selected vertices [41]. To break ties, select the vertex with the maximum degree among the tied vertices.

Maximal Paths

Create a maximal path by selecting a vertex in the graph [4]. Then extend the tail of the path by repeatedly including an unselected neighbouring vertex (sharing an edge with the tail) to the path, attaching it to the tail, and becoming the new tail. Repeat until this is no longer possible, after which the head of the path is extended in a similar matter. Whenever any vertex (including the first one) is added to the path, it is also added to the ordering in the same order. When both the head and the tail of the path cannot be extended further, it has become a maximal path. The maximal path is removed from the graph. Create new maximal paths in the same way until the graph is empty.

Maximal Cliques

Start creating a clique by selecting the vertex with the maximum degree [29]. Add vertices that can maintain this clique, until this is no longer possible. Create new cliques in a similar manner by selecting the next unselected vertex with the maximum degree, eventually creating a maximal clique decomposition of the graph. Finally, add the largest clique to the ordering, then iteratively add the clique that shares the most edges with the previously added clique.

3.8.1 Graph Coloring

Aside from the MISP, where different variable orderings are compared in this thesis' experiments, the graph coloring problem is also considered in other experiments of this thesis. For the graph coloring problem, the experiments use the DSATUR variable ordering strategy by Brélaz [10], which is specifically made for the graph coloring problem. It uses the "saturation degree" of the vertices to order them, hence the name DSATUR.

DSATUR

Take the vertex with the maximum degree and give it color 1. Then, for each iteration, select an uncolored vertex with a maximal saturation degree, i.e. the vertex adjacent to the largest number of different colors. To break ties, select the vertex with the maximum

degree. Give the selected vertex the lowest possible color (such that it shares no color with the neighbouring vertices). Repeat until all vertices are colored. The order in which the vertices were selected determines the variable ordering used in the B&B algorithm.

3.9 Local Bound Pruning

Another effective heuristic for B&B search is local bound pruning, which allows for early pruning and reducing the solution space during search. This technique is introduced by Gillard et al. [20] under the name "rough upper bound pruning", but since this thesis mainly uses the CODD framework [34], we use their terminology of "local bound pruning" instead. With an already computed lower bound (from a restricted DD), an approximate upper bound can be computed for the next state to determine if they can be pruned before being created. This also means that nodes pruned by this method do not count towards the maximum width of the approximate DDs, leaving more room for more promising states. The approximate upper bound is problem-specific though. It also needs to be fast to compute, as it is computed for every node in the approximate DDs.

As an example, the local bound Gillard et al. [20] give for the MISP is shown in Theorem 1. The theorem is modified by removing the weight of vertices, as this thesis focuses on an unweighted version of the MISP.

Theorem 1. *Given a graph $G = (V, E)$, the size of any maximum independent set in G, the independence number $\alpha(G)$, is upper bounded by:*

$$\alpha(G) \leq |V|$$

This upper bound follows from the fact that the total set of vertices V of graph G is the superset of the optimal solution for graph G. The upper bound can be applied to any subgraph of the original problem instance, potentially allowing for early pruning during search if the subgraph is proven to be suboptimal.

3.10 Beam Search

The following section goes over the beam search algorithm. While not often associated with B&B search, it served as inspiration for certain modifications made to the method of this thesis. Beam search is a simple method that restricts the number of paths that are explored in parallel. This way, it tries to approximately solve the problem while greatly reducing the memory needed to do so.

Beam search, first introduced in 1976 by Lowerre [32], is a heuristic search algorithm. It tries to find an approximate solution by only expanding the most promising paths of the search tree and pruning the rest. By only exploring a fixed number of paths, it becomes a polynomial-time algorithm instead of an exponential-time one. This is done in a breadth-first search manner, as it expands nodes per layer, without backtracking. Notably, the concept of only exploring promising paths (by keeping a fixed width) while discarding the rest is similar to that of restricted decision diagrams (explained in Section 3.6).

Fig. 3.5 shows an example of a search tree T. With a beam width k, for each layer of the tree, only the best k nodes are expanded according to the evaluation function. This is done layer-by-layer (similar to breadth-first search). The other nodes in the layer are discarded. As such, only k paths or "beams" are explored at a time and the beam search algorithm becomes polynomial in the size of the problem. However, it remains an approximate method, so it relies on the accuracy of its evaluation function. With a larger k, the risk of accidentally discarding an optimal solution is reduced, with the downside of increasing the computation cost.

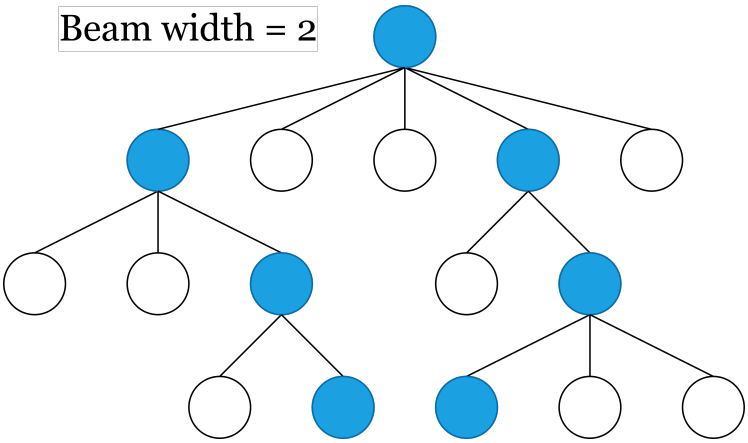


Figure 3.5: Example of a search tree traversed with beam search. Colored nodes are selected by the beam search algorithm, uncolored nodes are discarded

Generally, two types of evaluation functions are used to determine which nodes to include in the beam. "One-step priority evaluation functions" concern themselves only with the next decision to be made, looking at a local scale. Methods that look at a more global view, are called "total cost evaluation functions". These are generally more accurate, but also more expensive to compute.

A trade-off can be made by introducing a "filter width", first introduced by Ow and Morton [37]. All nodes of a layer are evaluated by a cheap local function (one-step priority evaluation). Then, given a filter width f, only the f best nodes are evaluated by a more expensive global function (total cost evaluation). Finally, the best k nodes are selected from these filtered nodes to form the beam.

The method introduced by this thesis, which is inspired by beam search, follows the notion of prioritizing the most promising paths. However, since we would like to keep the method exact, we introduce an additional path that preserves the dismissed options. This makes the new method more of a heuristic to guide the search, rather than an actual approximation of the solution. The approximation is already covered by the restricted DDs of the B&B algorithm, which provide a primal bound on the solution.

Chapter 4

New Heuristics for Multivalued Decision Diagrams

Since state-based multivalued decision diagrams (MDDs) have seen limited exploration, there is still plenty of opportunity for improvement. This chapter introduces several new heuristics specifically made for these state-based MDDs with the aim of improving their performance. Chapter 4.1 covers the new "beam restriction" technique, which prioritizes the most promising paths and makes the most out of the transitions that are made. Next, a dynamic variable ordering strategy is adapted to fit into this new state-based context in Chapter 4.2. Finally, Chapter 4.3 presents a new local bound for the maximum independent set problem (MISP) that benefits from being used alongside the new dynamic variable ordering.

4.1 Restricting the MDD Expansion

A Multi-Valued Decision Diagram (MDD) encoding, such as the one for the MISP covered by Section 3.4, may come with potential benefits over its BDD counterpart. Smarter decisions due to being state-based instead of layer-based can prevent suboptimal transitions. This also potentially enhances the effect of heuristics such as variable orderings, since they can now be applied on a state-by-state basis and do not have to compromise between all states in a layer.

However, the state-based MDD encoding can run into issues with the imposed maximum width of the approximate DDs used by B&B. This maximum width follows the merge condition where states are only merged if they have made the same number of transitions from the root state. With certain problem domains such as the MISP, the rate at which states expand does not interact well with the maximum width. If the root state S_0 expands, it will generate $|V|$ states (with V being the total number of vertices in the case of the MISP). In the worst case, each of these states will generate again close to $|V|$ states, etc. The imposed maximum width of the approximate DDs used by B&B will quickly be exceeded, leading to an excess amount of state generation and merging. For the MDD encoding to work on problems like the MISP, there needs to be a way to prevent the branching factor from becoming

too large to handle.

Other search algorithms with similar issues have tried to limit the exploration rate before, using techniques such as the beam search algorithm (explained in Section 3.10). Beam search restricts the number of paths that are explored in parallel, by limiting the number of states that expand for each layer. Since the MDD encoding is more state-based rather than layer-based, the restriction would apply to the number of transitions made per state instead of the number of states per layer. Applying the beam restriction to the MISP encoding gives the following transition function, described by Equation (4.1). For each state S, instead of exploring all vertices in S, only the "beam" B, i.e. the best b vertices $v_i \in B$, make a transition to a new state. Additionally, an extra transition, the "no-beam" transition, with label $i = -1$ is created to cover the unexplored vertices.

$$f(S,i) = \begin{cases} S \setminus (M(v_i) \cup \{v_1, ..., v_i\}) & \text{if } v_i \in B \\ S \setminus B & \text{if i = -1} \end{cases} \tag{4.1}$$

The first line of f denotes that for each vertex in the beam $v_i \in B$, a transition is made that includes v_i in the solution. Which vertices are put in the beam B is determined by the variable ordering function applied to the algorithm. Following the MISP, choosing to include v_i in the solution means excluding v_i and its neighbors $M(v_i)$ from the eligible set of vertices in future states. And similarly to the old transition function from Equation (3.2), along with v_i, the lower numbered vertices $v_1, ..., v_{i-1}$ are excluded for symmetry breaking. These are already covered by previous paths since the order of elements in the independent set does not matter (e.g. the independent set $\{v_1, v_2, v_3\}$ is the same as the set $\{v_3, v_2, v_1\}$).

However, while regular beam search is an approximate algorithm, the aim of our method is to compute the exact solution. Therefore, each state needs to make an extra transition, where no vertices are selected to be included in the solution. The extra no-beam transition with label $i = -1$ is represented in the second line of f from Equation (4.1). This no-beam transition only removes the vertices $v_i \in B$ from future states. As such, it represents all possible solutions that include none of the vertices $v_i \in B$. The exploration of the leftover vertices is saved for later, in case the exploration of the promising vertices $v_i \in B$ fails. With this, all possible solutions are still preserved and encoded by the decision diagram. The weight of this transition is set to 0 (as opposed to a weight of 1 with the other transitions) to not interfere with the objective value of the solution. A weight of 0 also makes this a less desirable option for the solver, ensuring that other transitions are considered first.

Unfortunately, the new no-beam transition does not work as well with the original merge condition proposed by Curry et al. [15], where states are only merged if they share the same last selected vertex. If a state followed any no-beam transitions, it would fall behind and merge suboptimally since its "last selected vertex" would be from a few decisions ago. This is why for this thesis, the merge condition is changed to only merge states with the same number of transitions from the root state. As such, the maximum width of the approximate DDs applies to each series of states that have the same number of transitions from the root state. This still groups relevant states together for merging while including the states following the no-beam transitions. Though it makes the method more of a hybrid between state-based and layer-based compilation.

How these changes alter the existing construction of relaxed and restricted DDs can be seen in Alg. 2. The best b vertices are selected with a variable ordering (line 4), after which (line 5) $b+1$ states are created (b decisions to include a particular vertex and one decision to not include any of them). How these changes affect the DP model (mainly the label generation function and the state transition function) can be seen in Appendix A.1.3.

Algorithm 2 Outline of relaxed/restricted DD construction with beam restrictions

Input: Root state R, beam width b, maximum width w
Definition: L_i = Set of states at decision stage i, i transitions from the root
1: $i \leftarrow 0$, Add Root state R to L_0
2: **while** $|L_i| > 0$ **do**
3: **for each** state S in L_i **do**
4: Select b vertices according to the variable ordering
5: Create $b+1$ states for each decision + the extra no-beam transition
6: Add these states to L_{i+1}
7: **end for**
8: **if** $|L_{i+1}| > w$ **then**
9: Merge or prune states in L_{i+1} until $|L_{i+1}| \leq w$
(depending on if it is a relaxed or restricted DD)
10: **end if**
11: $i \leftarrow i+1$
12: **end while**
13: **return** Completed relaxed/restricted DD (with bounds)

A graphical example of the beam restriction can be seen in Fig. 4.1, where the MISP example MDD from Section 3.4 is restricted with a beam width $b = 2$. Notably, at the expansion of the root node, the number of explored paths is restricted from five to three. In practice, this reduction would be far greater, as problem sizes are much larger than this example problem of five vertices, going from $|V|$ paths to $b+1$ paths.

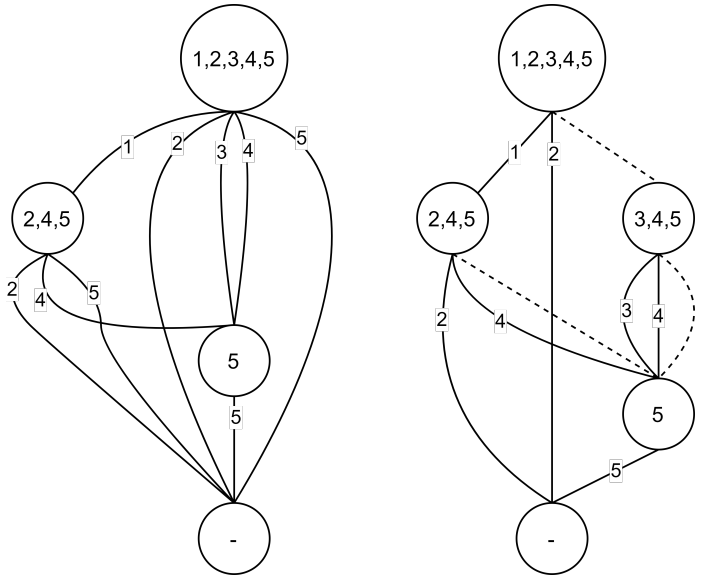


Figure 4.1: (a) The original MDD, which can be seen in Fig. 3.3. It is an MDD for the MISP example in Fig. 3.1. (b) Example of the MDD restricted with a beam width $b = 2$. Each state in the MDD shows the eligible vertices available at that particular state. The variable decisions are shown with numbers in the transition lines, indicating which vertex is chosen to be included in the independent set. A dashed line represents the extra no-beam transition where no vertices are chosen, which has an objective value of 0 instead of 1.

4.1.1 The No-Beam Transition

Previously, the no-beam transition was defined as saving the leftover decisions that were not chosen by the beam for later. However, what does this actually mean and how does this apply to other problem domains?
The main way the no-beam transitions are interpreted in this thesis is as follows:

1. We defer the decisions that the beam did not cover.

The beam restrictions method was modelled after interpretation 1 for the MISP. No vertices are included in the independent set, and instead, the options covered by the beam are simply removed from the eligible set. As a result, the weight of the transition is 0 instead of 1. However, there is an alternative way to interpret the no-beam transitions:

2. The leftover options are pre-emptively merged together into one state.

For the MISP, the no-beam transition also acts like interpretation 2, as the leftover options are "merged" into one new state. The only difference with an actual merged state is that the weight of the no-beam transition is 0, not 1. This is not an issue, since the relaxed decision diagram will prioritize the options from the beam and the no-beam state can act as a "pre-emptively merged" state that would result from the excess state creation and merging whenever the maximum width is reached. Even if the no-beam state is further explored by

the relaxed DD, the state is still exact and a weight of 0 would still be correct since no vertex was included in the independent set.

These two interpretations are more distinct when looking at another problem domain, the graph coloring problem. Here, the two interpretations lead to different models, as one single model does not reflect both at once like with the MISP. This may result in different behaviour when applying the beam restrictions on the MISP compared to on the graph coloring problem, depending on which model is used.

Each state in the graph coloring problem is defined as $S = (A, v, k, C)$, where A is the color assignments of the already colored vertices, v is the current vertex that needs to be colored, k is the highest numbered color used so far, and C is the set of eligible colors for vertex v. First, with interpretation 1, if we simply defer the decisions outside of the beam, the following transition function applies, as seen in Equation (4.2). Here, c is the label of the chosen color to assign to vertex v_i, or the label for the no-beam transition if $c = -1$. The set of colors in the beam is denoted by B. A' is the resulting color assignment list where vertex v_i is colored with color c.

$$f(S,c) = \begin{cases} (A', v_{i+1}, k, C_{i+1}) & \text{if } 0 \geq c \geq k \\ (A', v_{i+1}, k+1, C_{i+1}) & \text{if } c = k+1 \\ (A, v_i, k, C_i \setminus B) & \text{if } c = -1 \end{cases} \tag{4.2}$$

Here, vertex v_i is colored with color c for all colors in the beam B. For the no-beam transition (where $c = -1$), the new state stays on the same vertex v_i, while the eligible colors C_i shrink by removing the beam B.

Interpretation 2 on the other hand, follows the transition function from Equation (4.3).

$$f(S,i) = \begin{cases} (A', v_{i+1}, k, C_{i+1}) & \text{if } 0 \geq c \geq k \\ (A', v_{i+1}, k+1, C_{i+1}) & \text{if } c = k+1 \\ (A, v_{i+1}, k, C_{i+1}) & \text{if } c = -1 \end{cases} \tag{4.3}$$

Now, the no-beam transition moves on to the next vertex v_{i+1} to mimic a regular merged state. Since this is a pre-emptively merged state, A remains unchanged, as when merging states with different color assignments for a vertex, the vertex becomes uncolored as a relaxation. This results in a very optimistic relaxed state, where v_i remains uncolored.

A graphical example of the two interpretations can be seen in Figure 4.2, where a beam width of 1 is used (with an extra transition for the special case of the new color $k+1$). Vertex v_i can be colored with colors 1, 2, 3, or a new color $k+1$. Interpretation 1 defers the decisions that are not covered by the beam of 1. Interpretation 2 merges the remaining decisions into a relaxed state. For the graph coloring problem, an option could be to treat the new color $k+1$ as a special case and keep it separate from the relaxed state.

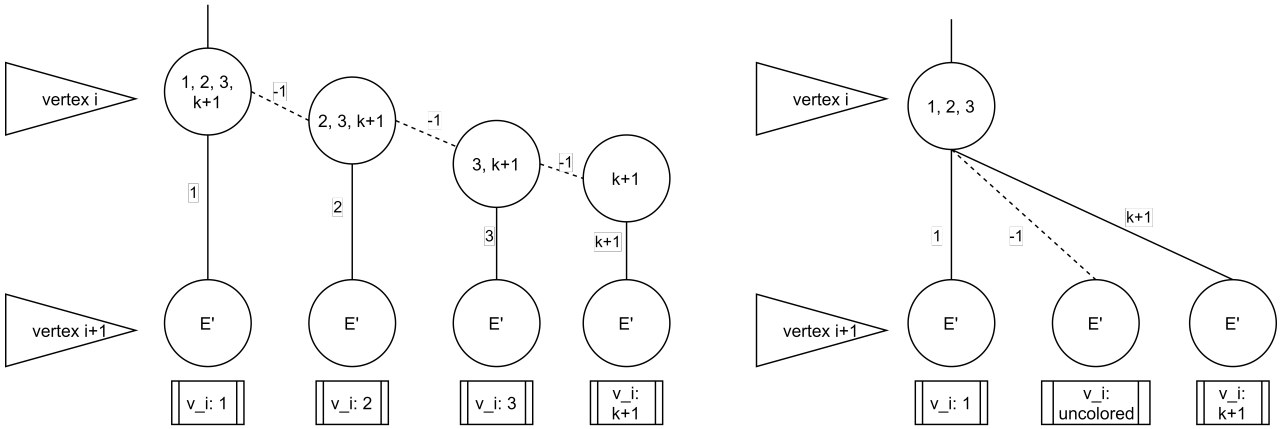


Figure 4.2: Example of the two interpretations for the no-beam transition in the graph coloring problem. Here, vertex v_i can be colored with colors 1, 2, 3, or k+1. A beam width of 1 is used here. a) Interpretation 1, where the decision is deferred. b) Interpretation 2, where the leftover options are merged (resulting in vertex v_i remaining uncolored). Each state shows the eligible colors at that particular state for the current vertex. A dashed line represents the no-beam transition. The variable decisions are shown with numbers in the transition lines, indicating the color to assign to the current vertex (or -1 for the no-beam transition). The color assignment of v_i (normally tracked in A) is indicated in the boxes below the graphs.

Both of these interpretations are natural ways to interpret the behaviour of the no-beam transitions. Interpretation 1 is more common from an AI heuristic search perspective, where custom labels are introduced to denote specific actions. An example of this is the giant-tour representation of vehicle routing problems by Funke et al. [18], which uses special labels for partial moves such as opening and closing trucks. Interpretation 2 leans more towards a decision diagram perspective, as it essentially follows the main idea of a relaxed decision diagram and relaxes the problem even further.

It is more likely for interpretation 2 to prevent excess state creation and merging, as it allows for far fewer states to be created by making this pre-emptively merged state. However, one critical problem with interpretation 2 lies with the implementation of the method. As this thesis implements its models in the CODD framework [34], CODD treats the no-beam state as an exact state instead of a relaxed state. This can be problematic, as it interferes with the frontier cutset of the Branch & Bound algorithm. This cutset would be able to include these relaxed no-beam states, while it should only contain exact states. Currently, it is not possible to define these no-beam states as a relaxed state at the modelling level. It would require changing the internal design of CODD to directly implement the beam restrictions to follow interpretation 2. Since we focus on the modelling level in this thesis, we cannot currently implement interpretation 2. Therefore, we will focus on interpretation 1 for the graph coloring problem in this thesis and recommend the developers of CODD to consider interpretation 2 in the future as well.

4.1.2 A Layerless BDD

Interestingly, setting the beam width to $b = 1$ results in a "layerless" version of the regular BDD encoding of the MISP. A "layered" BDD is one where each layer is associated with a single decision, where all nodes in that layer make a decision on the same decision variable (e.g. to include or exclude a vertex for the MISP). This layerless version is more state-based, as decisions are made based on each individual state, rather than across a whole layer. This is the only major difference; the models are mostly similar otherwise. The extra no-beam transition where none of the vertices from the beam are included in the solution acts like the "do not include" transition from the BDD encoding, resulting in two transitions per state like a BDD. The new merge condition also mimics the behavior of a layered BDD, since all states in a layer have the same number of transitions from the root state. The BDD restricts merging to be within a layer, which is the same as restricting the merging to states that have the same transition count.

In theory, the layerless BDD should be strictly better than the layer-based BDD. Decisions are now made per state instead of per layer. Because of this, each state can choose the most optimal vertex or variable to make a decision on instead of having to go through all vertices or variables for each layer. This would result in certain paths finishing earlier, as irrelevant decisions are skipped.

As for the difference between a beam size of 1 as opposed to 2 or more, the beam=1 version can only explore one decision per expansion; it can only select one vertex to include in the solution. For each decision, a no-beam transition is created. On the other hand, the "multiple beams" version can explore several paths at once (and potentially prune some of these paths), while creating only one extra no-beam transition. Though, it should be noted that the better the variable ordering strategy, the smaller the beam width can be. With a better ordering, the first few vertices chosen have more potential, while more transitions for further exploration are unnecessary.

Furthermore, the "multiple beams" version can be slowed down by the maximum width of the decision diagram, imposed by the approximate decision diagrams used by the Branch & Bound algorithm. The larger the beam width, the more states are created and the more the maximum width is exceeded. As such, more states need to be merged at once, which requires more computation. This additional cost means that smaller beam widths are likely to be more preferable.

4.1.3 Minimization Problems

Minimization problems, unlike maximization problems such as the MISP, have another aspect to consider before applying the beam restrictions to their MDD encoding. With the MISP, the weight of the no-beam transition can be set to 0 to ensure other transitions (with higher weights) get prioritized. However, with a weight of 0, the no-beam transition is prioritized in minimization problems. Rather than deferring the decisions of the no-beam transitions, they are exhaustively explored instead, which defeats their purpose. Thus, a different weight needs to be considered for minimization problems.

The graph coloring problem (explained in Section 3.5) is such a minimization problem.

To deprioritize the no-beam state, the weight of the no-beam transition needs to be higher than 0, which is the weight of the other transitions where one of the k previously assigned colors is selected. Though this new custom weight cannot be too high, as multiple no-beam transitions can then cost more than introducing a new color $k+1$ (where taking a new color has a weight of 1), resulting in the solver possibly ruling out solutions where introducing a new color could have been prevented. As such, we decide to set the custom weight to a small value, e.g. $1/(|V|+1)$. With this, the accumulated weight of taking multiple no-beam transitions will not exceed the weight 1 of taking a new color.

To correct the artificially added weights of the no-beam transitions, an extra step needs to be added to the MDD. After the terminal state T, it now always goes to a new terminal state T', where the weight of the transition $T \to T'$ subtracts the accumulated weights of the traversed no-beam transitions. The number of no-beam transitions can be tracked for each state, and can thus be corrected at the end of the diagram. This is quite a general solution that can be applied to any minimization problem. However, the specific weight of the no-beam transitions ($1/(|V|+1)$ for the graph coloring problem) may not universally work on any problem and should be specially considered for each problem. The new DP model for the graph coloring problem can be seen in Appendix A.2.2.

Additionally, when applying the beam restrictions to the graph coloring problem, the order in which the colors are considered for each vertex becomes relevant. Before, without any beam restrictions, each state simply made a transition for each of the available colors (following the MDD encoding from Section 3.5). Fortunately, the DSATUR variable ordering strategy [10] (as explained in Section 3.8.1), which is used for the graph coloring experiments in this thesis, suggests using the lowest numbered colors first. Therefore, we make the beam (of beam size b) consist of the b lowest numbered available colors.

4.2 Dynamic Variable Ordering

With the use of an MDD encoding and its state-based nature, dynamic variable orderings behave differently than in a layer-based context. In a layer-based scheme, a dynamic variable ordering has information on the states in the current layer and chooses a single decision variable to decide on for all those states in that layer. On the other hand, in a state-based context the ordering only has information on the current state and, as such, can make different decisions for each state. This allows for better decision-making, since each decision is specially made for each individual state, with the most relevant information available. However, this may introduce even more complexity and overhead, as each state now has to track relevant information dynamically and reorder the decision variables.

This thesis explores a dynamic variable ordering for the state-based MDD encoding of the MISP, inspired by the variable ordering strategy from Nafar and Römer [36] (discussed in Section 2.2). The ordering relies on the remaining subgraphs in each state, i.e. what remains of the initial problem graph in the current state after previous decisions have been made. From the subgraph, the local degree of each vertex (i.e. the number of neighbouring vertices in the subgraph) can be determined. In practice, instead of keeping track of subgraphs for each state, it is sufficient to only track the local degrees, which reduces com-

plexity and memory usage. Furthermore, the information about the local degrees is passed from state to state and updated dynamically rather than being recomputed from scratch to avoid redundant computation.

Unlike the strategy of Nafar and Römer, which prioritizes the vertex with the minimum local degree, we propose selecting the vertex with the maximum local degree. In our experience, maximizing pruning potential and decreasing the search space (as discussed in Section 3.8) is more important than aiming for the optimal solution from the start. Especially in a state-based context, where more pruning allows the improved decision-making (decisions made per individual state) to make more diverse decisions, while eventually reaching the optimal solution without specific guidance from the variable ordering.

Normally, the complexity of such a dynamic variable ordering would be increased with an MDD encoding. With a BDD encoding, only a single vertex is chosen, which requires a single pass over the vertices to find the best vertex. With a regular MDD encoding, the worst case is when $|V|$ vertices are chosen at a time, which would require fully sorting the vertices with a worst-case time complexity of $O(|V|\log(|V|))$. However, this can be prevented with the use of the beam restrictions from Section 4.1. When restricted, only b vertices are chosen, where b is the beam width. As a result, only b passes through all vertices are required to find the b best vertices with a time complexity of $O(|V| \cdot b)$. This is preferable when the beam width b is relatively small compared to $|V|$.

An example of an MDD that applies this new dynamic variable ordering strategy can be seen in Figure 4.3. The beam restriction method from Section 4.1 is also applied here. The subgraphs are shown in the states, from which the local degrees of the vertices can be deduced (for illustration purposes only, as the method only keeps track of the local degrees). Although the local degree is not used as much in this example, it is due to the fact that the vertices with the maximum local degree are chosen at the root node (vertex 2 and 3) that a lot of neighbouring vertices are pruned in the subgraphs. On top of the pruned neighbours, symmetry breaking also shrinks the subgraphs further (removing vertices with a number lower than the chosen vertex from the subgraph).

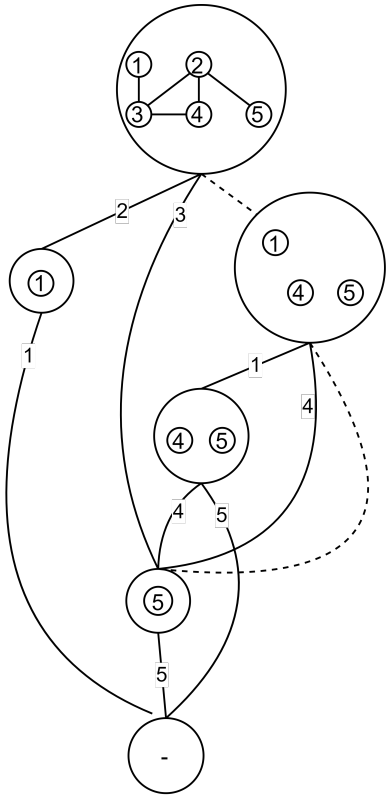


Figure 4.3: Example of an MDD (for the MISP example in Fig. 3.1) restricted with a beam width of 2, following the new dynamic variable ordering. Each state in the MDD shows the remaining subgraph at that particular state. The variable decisions are shown with numbers in the transition lines, indicating which vertex is chosen to be included in the independent set. A dashed line represents the extra no-beam transition where no vertices are chosen, which has an objective value of 0 instead of 1.

4.3 Local Bound for the MISP

As explained in Section 3.9, a local bound can significantly improve the efficiency of the DD-based B&B algorithm, as it allows for early pruning during B&B search. In this section, we introduce a new local bound on the MISP and discuss how this local bound interacts with various variable ordering strategies.

The new local bound is defined in Theorem 2, followed by its proof, which combines known graph-theoretic results. The local bound is only applicable if the maximum degree $\Delta(G) > 0$. In the case where $\Delta(G) = 0$ (i.e. when the graph G has no edges), a local bound of $\alpha(G) \leq |V|$ is used instead.

Theorem 2. *Given a graph $G = (V,E)$ and its maximum degree $\Delta(G) > 0$, the size of any maximum independent set in G, i.e. $\alpha(G)$, is upper bounded by:*

$$\alpha(G) \leq |V| - \lceil \frac{|E|}{\Delta(G)} \rceil$$

Proof. Let $M \subseteq V$ be a maximum independent set of G, with $\alpha(G) = |M|$. The complement of M gives a minimal vertex cover $C = V \setminus M$ (i.e. a set of vertices that covers each edge in G, with no proper subset of C also being a vertex cover). Each vertex in C can only cover ΔG edges in E. Therefore, the size of C has the following lower bound:

$$|C| \geq \lceil \frac{|E|}{\Delta(G)} \rceil$$

with a ceiling on the fraction, since $|C| \in \mathbb{N}$ and there need to be sufficient vertices to cover all edges in E. Given that $|V| = |M| + |C|$ (Corollary 7.1 in Graph Theory with Applications by Bondy and Murty [9]), we can derive the following:

$$\begin{aligned} |M| &= |V| - |C| \\ \alpha(G) &= |V| - |C| \\ \alpha(G) &\leq |V| - \lceil \frac{|E|}{\Delta(G)} \rceil \end{aligned}$$

□

This local bound is almost always better than the original local bound for the MISP (Theorem 1) proposed by Gillard et al. [20]. Since $|E| \geq 0$ and $\Delta(G) \geq 0$, the local bound is almost always strictly lower than $|V|$. The only exception to this is whenever there are no edges in the remaining subgraph ($|E| = 0$). Then, this local bound will default to the original one by Gillard et al. ($\alpha(G) \leq |V|$), which means the new local bound cannot be worse than the original one.

The new local bound works particularly well with variable ordering strategies that prioritize vertices with the maximum degree. Since these vertices are chosen and removed first, the maximum degree of the remaining subgraphs decreases, which consequently lowers the local bound from Theorem 2 (by lowering the denominator $\Delta(G)$). This brings the bound closer to the optimal solution and allows for more pruning. While the nominator $|E|$ also decreases, this decrease is usually less impactful than that of the denominator, as often $|E| \gg \Delta(G)$.

However, as noted by Gillard et al. [20], the local bound needs to be very inexpensive to compute, since it is computed for each state in the approximate DDs. For the bound presented here, the local degrees of the vertices in the subgraph of each state need to be tracked. Fortunately, this is already tracked by the new dynamic variable ordering strategy of Section 4.2, which means the bound can be computed with minimal overhead when combining the two methods.

Furthermore, since the new dynamic variable ordering is essentially a maximum degree variable ordering strategy, it should be able to strengthen the bound even faster than its

static counterpart. A static variable ordering only has access to the initial problem graph G when it fixes the ordering at the start. On the other hand, the dynamic ordering tracks the local degrees of the vertices during search, which means it knows the vertex with the maximum degree of the actual remaining subgraph of the current state. This allows for smarter decision-making by targeting the local maximum degree vertex to lower the local bound even faster.

Chapter 5

Experimental Results

This chapter covers the setup and results of the experiments performed for this thesis. The main goals of the experiments are to compare the state-based MDD encoding with the layer-based BDD encoding and to evaluate the impact of various heuristics, which were either adapted or made for the state-based MDD encoding. More specifically, the following questions are considered:

- How does a state-based MDD encoding perform compared to a layer-based BDD encoding?
- How do different beam restriction widths affect the performance?
- How do various static variable ordering strategies perform with a state-based MDD encoding?
- How well does the new dynamic variable ordering for the state-based MDD encoding work compared to its static counterpart?
- What impact does the new local bound for the MISP have on the performance and how does this differ with various variable ordering heuristics?

Since the maximum independent set problem (MISP) is the main example problem considered in this thesis, the experiments are performed on (randomly generated) instances of the MISP with its respective encodings. The beam restrictions are also applied to the graph coloring problem to also experiment with a minimization problem (as the MISP is a maximization problem).

The experimental setup is discussed in Section 5.1, while the metrics are further explained in Section 5.2. As for the experiments, the static variable ordering strategies are compared in Section 5.3 to determine which strategy to use in the following experiments. Next, Section 5.4 compares the state-based MDD with the layer-based BDD, while also incorporating different beam widths into the comparison. The beam restrictions are also applied to the graph coloring problem in Section 5.5. The experiment involving the new dynamic variable ordering strategy is covered in Section 5.6. Lastly, the new local bound for the MISP is tested with various variable orderings in Section 5.7.

5.1 Experimental Setup

All solvers were implemented and run within the CODD [34] framework, a DD-based solver for combinatorial optimization. Important to note is that CODD is a state-based solver, which allows for the implementation of the new techniques for state-based MDD encodings presented in this thesis. The regular BDD encoding of the MISP was also implemented using CODD, where the layer-based behavior of a regular BDD encoding is mimicked by traversing the vertices one by one (with the state variable n seen in Appendix A.1.1). By default, the MISP uses the existing local bound from Gillard et al. [20].

The instances consist of randomly generated graphs with predetermined sizes and densities. For each unique combination of size and density, 25 instances were created. The instances in the experiments use graphs of size 100 for the MISP and size 50 for the graph coloring problem, with densities ranging from 0.1 through 0.9. These sizes were chosen, as bigger sizes would result in the solver consistently reaching the maximum timeout of one hour with the harder instances.

The experiments were run on the Delftblue [16] supercomputer. Each instance was run on a "standard compute node", i.e. an Intel Xeon E5-6248R 24C 3.0GHz with 3.9GB of memory. The time limit was one hour per instance.

The code used for the experiments in this thesis can be found on the following Github page: https://github.com/JonathanTjong/Thesis_DD

5.2 Metrics

Various metrics are used to measure the performance of the methods in the experiments. First there is execution time (in seconds), which measures how fast a method is in practice. However, the execution time can vary each time the program is run due to extraneous factors. Therefore, the required number of B&B nodes is also measured. The number of nodes indicates the size of the B&B search tree used in CODD when solving a particular problem. This gives a good measure of the efficiency of the program, as it is not as prone to the interference of these extraneous factors.

Each value of the final results is the median of the 25 iterations on a unique instance type, i.e. random instances with the same size and density. The median is used to account for outliers in the results, as with combinatorial optimization problems, each implementation has different instances that happen to be disproportionately more difficult than the other instances (due to factors such as the initialization or variable ordering). The variability of the results is measured with the median absolute deviation (MAD), which is shown as transparent regions around the median lines in the experiment plots.

5.3 Static Variable Ordering Strategies

To compare the various beam widths for the MDD encoding, we first have to decide on which variable ordering would be most suitable. Therefore, the first experiment compares various existing static variable ordering strategies used with the BDD and MDD encodings.

The new dynamic variable ordering strategy is covered separately later on (in Section 5.6). The following static variable orderings are considered in this experiment:

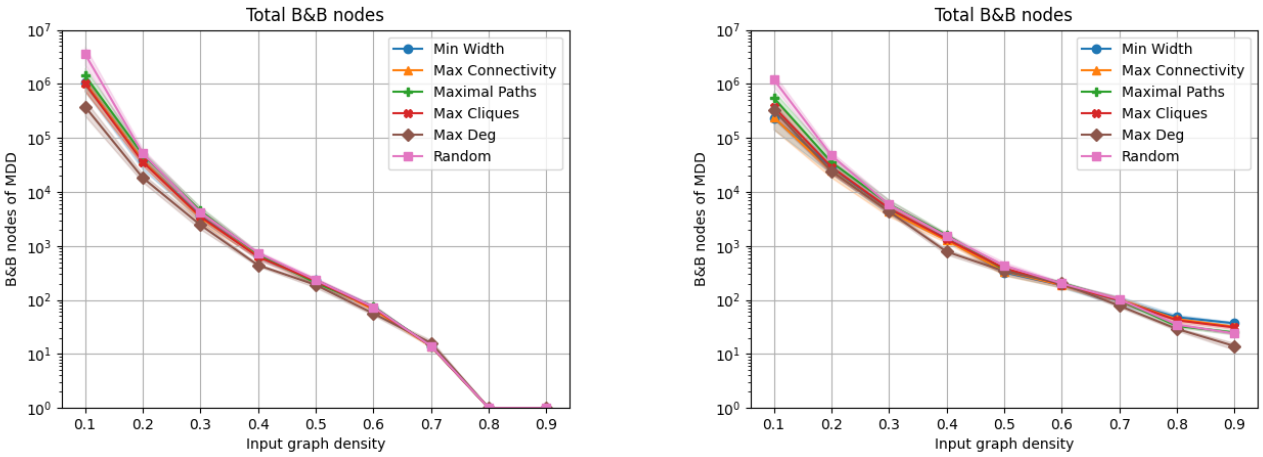
- Random
- Max Degree
- Min Width
- Max Connectivity
- Maximal Paths
- Max Cliques

More detailed explanations on these variable orderings can be found in Section 3.8. For this experiment, the MDD uses a beam width of 4, which appeared to be a good general beam width in early experiments. Further exploration of various beam widths will be covered in other experiments of this thesis.

Expectations

These variable ordering strategies (excluding random ordering) have proven to be effective in the literature. As such, they should perform quite well (at least better than random), especially with the state-based MDDs, which allow for more flexible decision-making while following these variable orderings.

Results



(a) MDD encoding (with beam=4)

(b) BDD encoding

Figure 5.1: Comparison of static variable orderings using the: a) MDD encoding with a beam=4, b) BDD encoding. The total number of B&B nodes is measured across different input graph densities from 0.1 to 0.9, with an initial graph size of 100 and a timeout of 1 hour.

Figure 5.1 shows that for both the MDD and BDD encoding, the different static variable orderings perform well overall, outperforming the random ordering in harder instances (in-

stances with lower densities). Only the graphs measuring B&B nodes are shown here, as the runtimes also show a similar relative performance between the methods. What is remarkable, is that the max degree ordering seems to outperform the others when using the MDD encoding (by at least one order of magnitude) with the harder instances. The main thing that differentiates it from the other methods is the fact that it is quite a greedy approach, whereas other strategies try to also pick subsequent vertices that are close to each other in the graph. Perhaps a greedy approach works well with the flexible decision-making of the state-based MDD encoding. Since the max degree variable ordering works well for both the BDD and MDD encoding, it will be used in the other experiments as the main variable ordering.

5.4 Restricted MDD with Beams

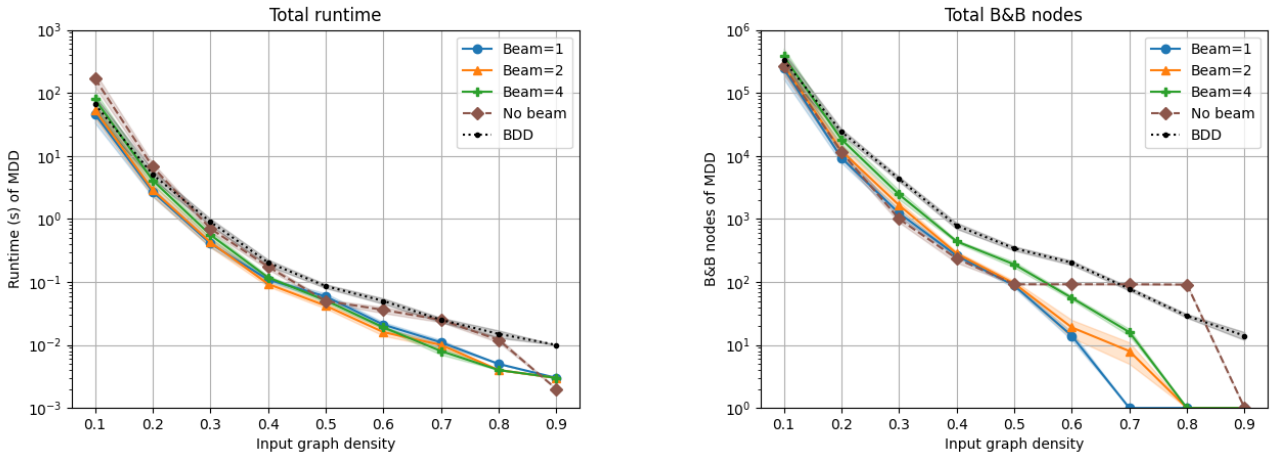
This section compares the performance of the state-based MDD encoding with the layer-based BDD encoding. Additionally, the impact of different beam restriction widths will be analysed. The variable ordering strategy used in this experiment (for both the MDD and BDD encoding) is the max degree ordering, as this ordering performed well for both the MDD and BDD encoding in the previous experiment (Section 5.3).

Expectations

Given the differences between these encodings, the expectation is that the MDD encoding outperforms the BDD encoding, due to its state-based nature. Decisions can be made per state instead of per layer, with the aim of reducing the number of transitions and nodes needed to solve the problem.

As for the beam widths, we expect lower beams to be faster than higher beams. A high beam could exceed the maximum width of the relaxed DDs more quickly, spending more time creating and merging nodes. However, a lower beam would need to create more no-beam transitions (where no vertices from the beam are chosen), possibly being less efficient in B&B nodes compared to higher beams. Though, the better the variable ordering strategy is, the lower the beam width can afford to be, as fewer options need to be explored at a time. Important to note, using no beam restrictions is essentially the same as having the highest possible beam width, since it allows all possible transitions to be explored.

Results



(a) (b)

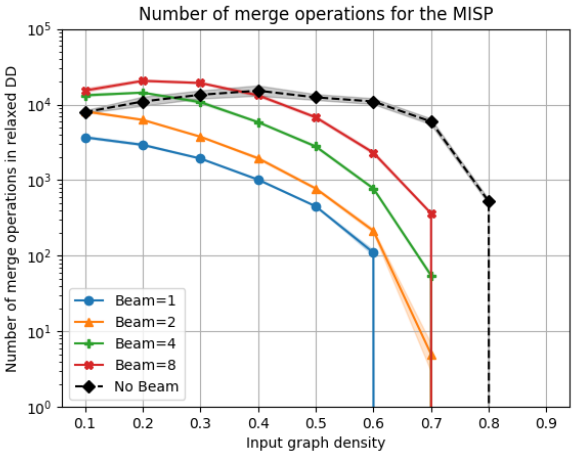
Figure 5.2: Comparison between the BDD encoding and the MDD encoding with various beam widths. Measured in a) execution time in seconds, b) total B&B nodes used. Both plots are measured across different input graph densities from 0.1 to 0.9, with an initial graph size of 100 and a timeout of 1 hour.

Figure 5.2 shows that the MDD encoding outperforms the BDD encoding for instances of the MISP, with the exception of the MDD with higher beams (and no beams) being slower for lower-density instances. This shortcoming of the MDD encoding is fixed with a sufficiently low beam restriction. However, it should be noted that both methods are closer in runtime and use a similar amount of B&B nodes for a density of 0.1. This might be due to the fact that with a low density, pruning is limited, as there are inherently fewer edges and neighbours in the graph. With the MISP, pruning takes place when a vertex is selected and the neighbouring vertices are dismissed, which is why the input graph density affects the pruning potential. The strength of the state-based MDD encoding is its more flexible decision-making, aiming to prune more of the search space. With these low density instances, this strength is diminished, bringing the performance of the BDD and MDD encoding closer together.

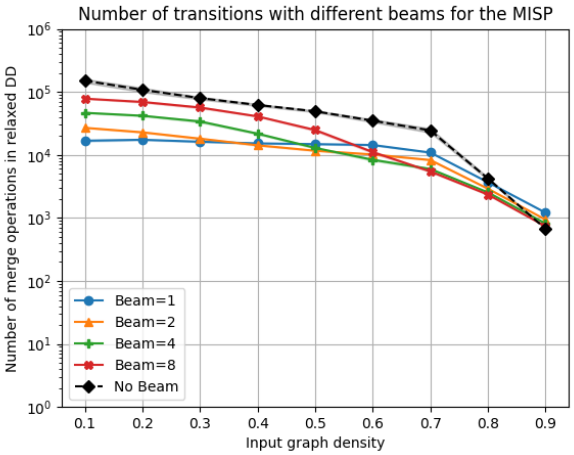
As for the beam widths, it seems that a very high beam (or no beam restrictions) can be very efficient in B&B nodes, but has the downside of being slower than the BDD encoding. This is likely due to the excess node generation and merging that comes with larger beam widths, running into problems with the maximum width of the relaxed DD. Applying (smaller width) beam restrictions can significantly improve the runtime of the MDD encoding, while still being competitive in terms of B&B nodes.

5.4.1 Effects on Relaxed DDs

The main goal of the beam restrictions are to prevent the excess state creation and merging whenever the maximum width is reached when constructing the relaxed DDs. To see if this indeed is the case, two experiments were conducted to measure the number of merge operations and the number of state transitions performed during the construction of a relaxed decision diagram for the MISP, the results of which are shown in Figures 5.3a and 5.3b respectively.



(a) Number of merge operations performed.



(b) Number of state transitions performed, i.e. the number of arcs in the relaxed decision diagram.

Figure 5.3: Visualization of the effects of the beam restrictions on the MISP. Two aspects of constructing a relaxed decision diagram for the MISP are shown: the number of merge operations and the number of state transitions. Various beam restriction widths are compared, alongside the original method without any beam restrictions.

Figure 5.3a shows that with a sufficiently low beam restriction width, the construction always requires fewer merge operations than when no beam restrictions are used. This difference is several orders of magnitude for most densities higher than 0.2. (The vertical drops indicate that no merges take place at higher densities.)

To confirm that the decrease in merge operations comes from the prevention of excess state creation and merging, we can look at the number of state transitions in the DD construction. Looking at the final number of states would not be as useful, since the excess states are merged until the maximum width is adhered to. However, the state transitions (i.e. the arcs in the decision diagram) from when the excess states were created are still present, now pointing to the newly merged state. As such, the number of state transitions can be seen as a direct representation of the number of created states throughout the DD construction process.

Indeed, as seen in Figure 5.3b, the number of state transitions decreased significantly, by several orders of magnitude. This indicates that the beam restrictions do have a major impact on the reduction of the excess state creation and merging. The decrease in merge

operations with higher input graph densities seems to be more due to the decreased difficulty of the instances, as the number of state transitions does not drop as fast.

Only towards the higher input graph densities (higher than 0.7) does the original method without beam restrictions catch up. This is likely due to these instances being too simple, as indicated by the previous experiments in Figure 5.2. Also, because no merge operations are performed in the instances with a density of 0.9, the number of transitions actually becomes lower when using no beam restrictions. This is because with no merge operations, the no-beam transitions do not prevent any excess state creation and merging. Though, this is not that concerning, as this only happens with easy instances to begin with.

5.5 Beam Restrictions in Graph Coloring

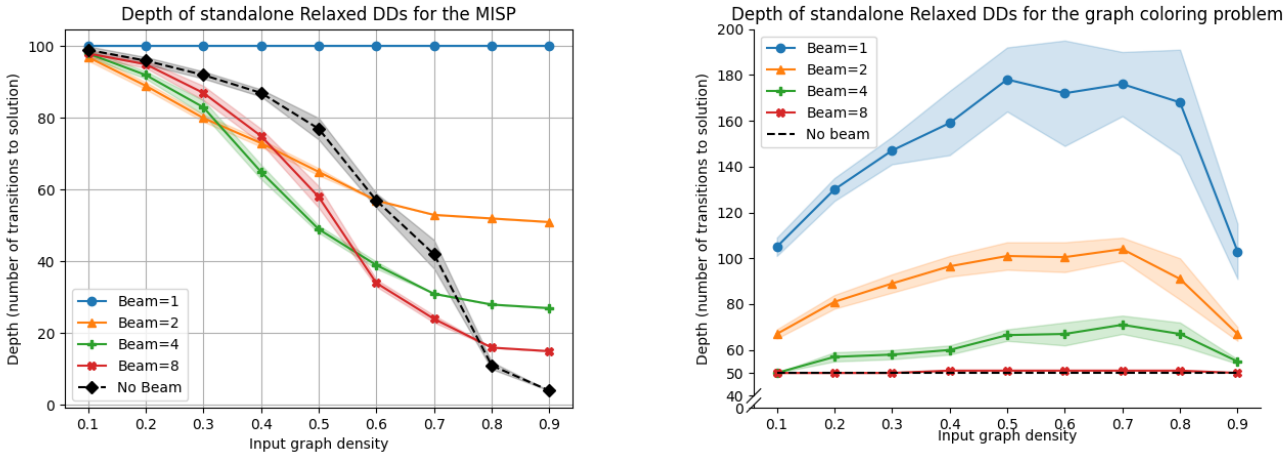
We would like to see whether or not the beam restrictions and our understanding of it applies more generally. Therefore, we experiment on another problem domain aside from the MISP, the graph coloring problem (explained in Section 3.5). This problem domain has a different structure compared to the MISP, which may lead to different behaviour of the model with the beam restrictions heuristic. Instead of a maximization problem, the graph coloring is a minization problem, which leads to extra measures that need to be taken (as discussed in Section 4.1.3). Additionally, the branching factor (which the beam restrictions are supposed to restrict) grows significantly the further the solution space is explored as more colors are assigned and the number of possible colors grows, whereas with the MISP, the branching factor is at its largest at the start.

5.5.1 Decision Diagram Depths

Another important aspect to keep in mind is that these two problem domains behave differently when beam restrictions are applied. Specifically, the beam restrictions have a different impact on the depths of the decision diagrams (and consequently the overall size of the diagrams, since the width is restricted for approximate decision diagrams). This is mainly due to the different problem encoding structures and how the beam restrictions are applied. As discussed in Section 4.1.1, a model following interpretation 1 is used for the graph coloring problem.

Figure 5.4 shows this behaviour for both problem domains. For the MISP, the depth of the diagram depends on how fast the eligible set of vertices shrinks with each decision (with the MDD encoding). Though, for very low beam widths the depth is prevented from shrinking, as beam 1 has to consider every vertex in V for some solutions, resulting in a depth of $|V| = 100$. Beam 2 converges to 50 transitions, but here it is already apparent that for most instances in this example this is not an issue, as most diagram depths are above that limit.

On the other hand, Figure 5.4b shows that for graph coloring (given the MDD encoding that is used here) the depth of the decision diagrams can only increase with lower beams. This is because each vertex in the input graph has to be assigned a color, making the minimum depth $|V|$ (in Figure 5.4b $|V| = 50$). With no beam restrictions, all colors are explored



(a) MISP

(b) Graph coloring

Figure 5.4: Comparison between the behaviour of various beam widths on the depths of decision diagrams for a) the MISP, b) the graph coloring problem. This is measured across different input graph densities, with an initial graph size of a) 100, b) 50 and a timeout of 1 hour.

at once, but with lower beam widths, the number of transitions made for each vertex increases. As a result, lower beam widths should be used with caution on the graph coloring problem, whereas with the MISP this is not necessarily the case. It should be noted that for the graph coloring problem, if one were to follow interpretation 2 instead, the depth would stay at $|V|$ and not increase due to the introduction of a custom relaxed state.

5.5.2 Custom Weights for Minimization Problems

As discussed in Section 4.1.3, the beam restriction heuristic needs to be slightly altered for minimization problems such as the graph coloring problem.

Figure 5.5 shows what happens if beam restrictions are applied to the graph coloring problem without any changes for minimization problems. Unlike with the MISP, the beam restrictions are not able to improve the runtime and only increase it with lower beam widths. This is because the no-beam transitions are prioritized, which is the opposite of the intention of the heuristic.

After applying the method discussed in Section 4.1.3 (adding custom weights to no-beam transitions), we get the results shown in Figure 5.6. These results are more similar to those of the MISP. Lower beam widths can reduce the runtime, while needing slightly more B&B nodes. This is most apparent with the harder instances around input graph densities 0.5 and 0.6. Now, it properly disincentivises taking the no-beam transition and applies the minimum color ordering of the DSATUR variable ordering.

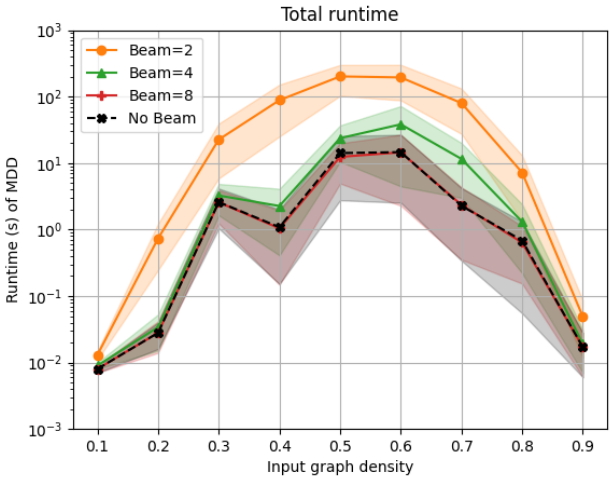
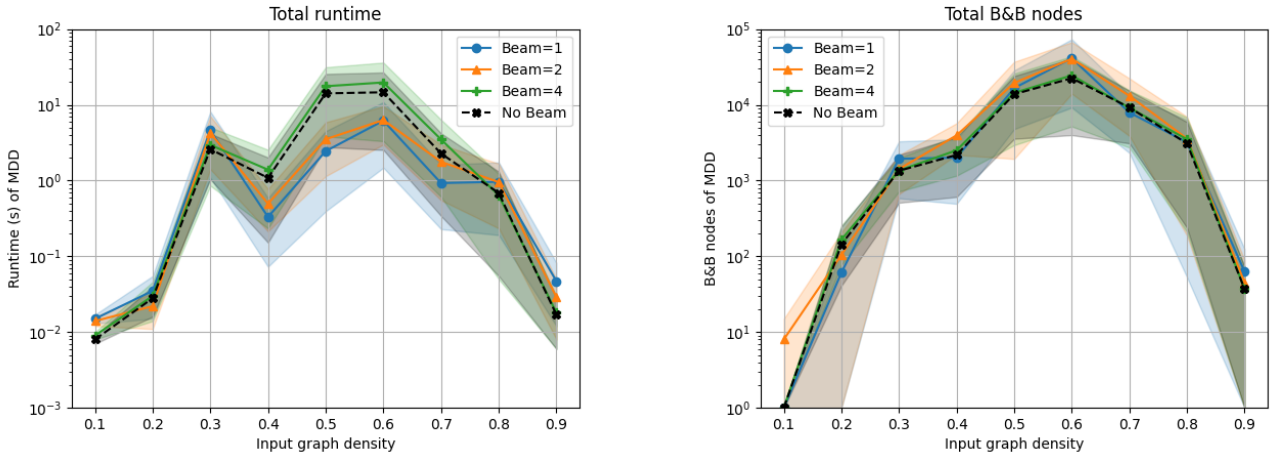


Figure 5.5: Runtime of various beam widths for the graph coloring problem **without special consideration for it being a minimization problem** (i.e. without custom weights for no-beam transitions). This is measured across different input graph densities, with an initial graph size of 50. Beam=1 is left out of the plot because of many instances reaching the maximum time limit of 1 hour.

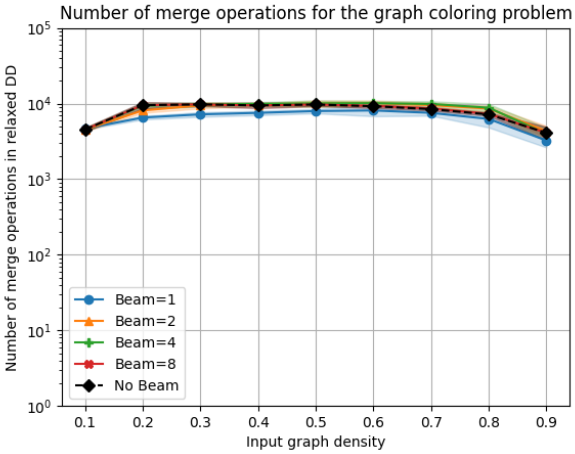


(a) (b)

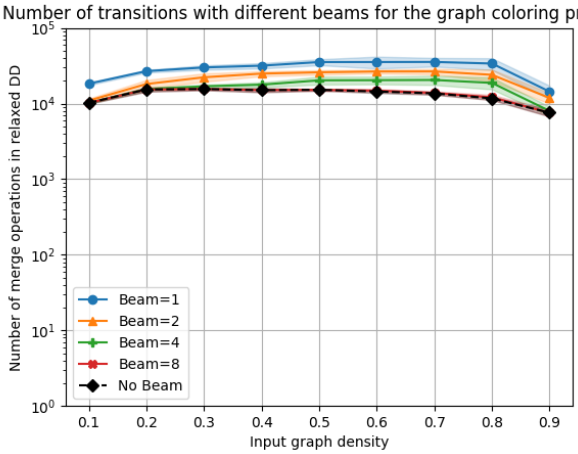
Figure 5.6: Comparison between various beam widths applied to the graph coloring problem with extra measures for minimization problems. Measured in a) execution time in seconds, b) total B&B nodes used. Both plots are measured across different input graph densities from 0.1 to 0.9, with an initial graph size of 50 and a timeout of 1 hour.

5.5.3 Effects on Relaxed DDs

Similarly to the experiments for the MISP (Section 5.4.1), we can observe the number of merge operations and state transitions for the construction of the relaxed DDs to determine if the beam restrictions worked well.



(a) Number of merge operations performed.



(b) Number of state transitions performed, i.e. the number of arcs in the relaxed decision diagram.

Figure 5.7: Visualization of the effects of the beam restrictions on the graph coloring problem. Two aspects of constructing a relaxed decision diagram for the graph coloring problem are shown: the number of merge operations and the number of state transitions. Various beam restriction widths are compared, alongside the original method without any beam restrictions.

Figure 5.7a shows that the merging behaviour does not change that much, as opposed to the MISP. Furthermore, Figure 5.7b shows that the number of transitions actually grows instead of shrinks. This indicates that the improved runtime (and slightly improved merging behaviour) likely comes from the imposed ordering of the color choices. Because with beam restrictions the lowest numbered colors are prioritized, which is in line with the DSATUR ordering. With no beam restrictions there is no such order, as all colors are considered at once.

The increase in state transitions is related to the problem pointed out in Section 5.5.1. Due to the encoding of the graph coloring problem, the structure of the decision diagram is a lot more rigid than with the MISP, as each vertex needs to be assigned a color. This makes it difficult for the beam restrictions to prevent excess state creation and merging, as the options that fall outside of the beam still pertain to the same decision, not the next one. Important to note is that this would not be a problem for interpretation 2 of the beam restrictions method, discussed in Section 4.1.1. There, the leftover options are directly merged together into a single relaxed state, decreasing the number of transitions instead of increasing it.

5.6 Dynamic Variable Ordering Strategies

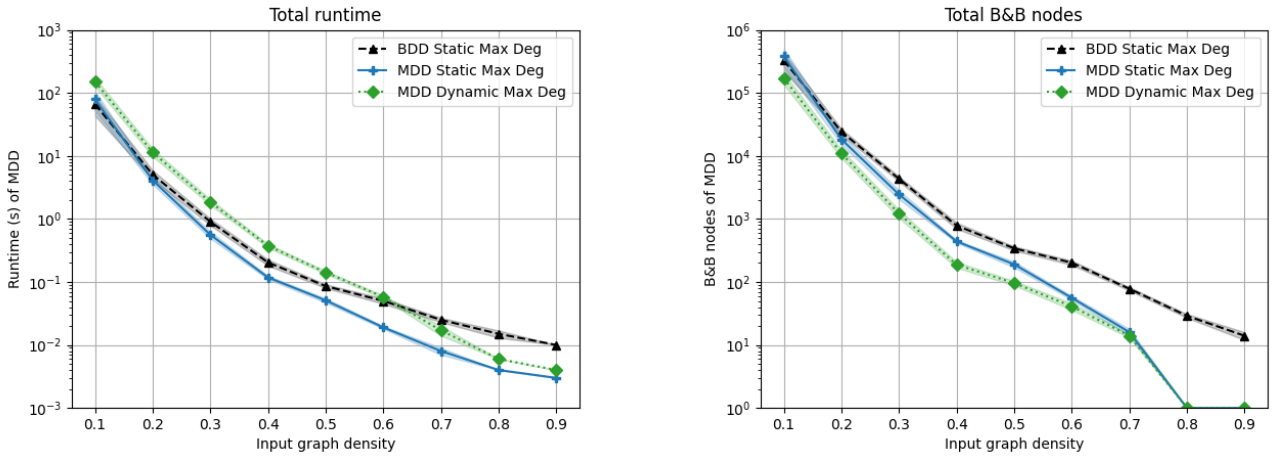
This experiment compares the new dynamic variable ordering strategy (proposed in Section 4.2) with its static counterpart, the static max degree ordering. Instead of looking at the degrees of the vertices in the initial graph, the local degrees (tracked during search) are used to order the vertices. The dynamic and static orderings will both be applied to a state-based MDD encoding with a beam width of 4. Additionally, the BDD is shown with the same static variable ordering (max degree) for comparison. Unfortunately, no dynamic ordering for the BDD encoding was implemented, as this would be difficult in the CODD framework due to the lack of layers and layer information in CODD.

Expectations

The main purpose of the dynamic variable ordering strategy is that it is able to make smarter decisions compared to its static counterpart. Rather than being fixed at the start, decisions are made at each state depending on the current situation at that particular state. Therefore, we expect it to be more efficient with fewer B&B nodes. However, a static variable ordering in the state-based MDD is already quite dynamic, as decisions are different for each state (compared to layer-based BDDs, where decisions are restricted to each layer). This might mean that the potential gain from a dynamic variable ordering is lower with state-based MDDs.

Furthermore, the static variable ordering is likely to end up being faster than the dynamic ordering. A static ordering only has to compute an ordering of the vertices once, while the dynamic ordering does so at each state. Additionally, information needs to be tracked from state to state, leading to even more overhead.

Results



(a) (b)

Figure 5.8: Comparison between the static and dynamic max degree variable ordering for the state-based MDD encoding. Additionally, the BDD encoding is shown using the static max degree ordering. Measured in a) execution time in seconds, b) total B&B nodes used. Both plots are measured across different input graph densities from 0.1 to 0.9, with an initial graph size of 100 and a timeout of 1 hour.

As expected, Figure 5.8a shows that the dynamic variable ordering is unfortunately slower than its static counterpart. It is even slower than the BDD, which shows that the gain in decision-making is not enough to make up for the additional overhead and complexity. Fortunately, the smarter decision-making was able to improve efficiency with fewer B&B nodes compared to the static variable ordering in both the MDD and BDD. It is still able to improve with an order of magnitude, even with the static variable ordering MDD already being more flexible with its decision-making than the BDD.

5.7 Local Bound Pruning

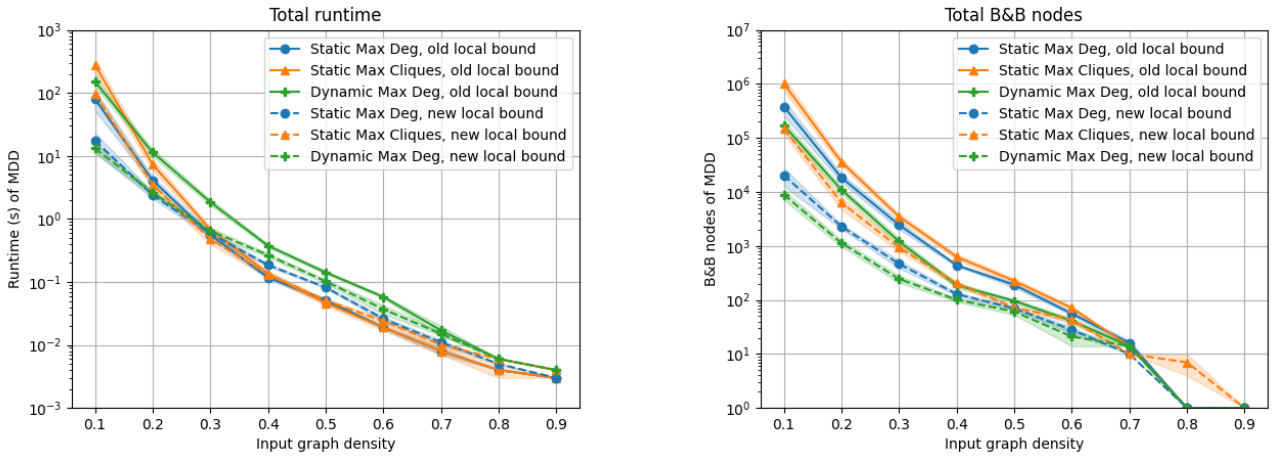
The experiment in this section tests the performance of the new local bound proposed in Section 4.3 to the existing one from Gillard et al. [20] using various variable ordering strategies. Specifically, the static max cliques and the static and dynamic max degree variable orderings are used. The max cliques ordering is used to measure how local bound pruning performs with an ordering that does not necessarily prioritize vertices with the maximum degree. The variable orderings are applied to the MDD encoding with a beam width of 4. Furthermore, the default implementation (without the new local bound) actually uses the local bound from Gillard et al. [20] (Theorem 1) by default. Important to note, when the local bound is used with a static variable ordering in this experiment, similar code is used to that of the dynamic variable ordering in order to track the necessary local information

for the local bound. This introduces the same overhead of tracking additional information, but does not include the overhead of the changing ordering from state to state (since a static ordering is used instead).

Expectations

As the new local bound should be strictly better than the default local bound (except for trivial cases), we expect an increase in performance compared to the default implementation. Furthermore, it is expected to work better with the max degree variable orderings, as these prioritize reducing the maximum degree of the subgraphs, tightening the upper bound further. Lastly, the local bound does introduce extra overhead, since accurate information is needed on the current state. However, this is mitigated with the dynamic variable ordering, which needs to track the exact same information, resulting in no additional overhead when adding local bound pruning. The static variable orderings are likely to be slowed down by the overhead though, the experiment will show if this is mitigated by the gain in pruning potential.

Results



(a) (b)

Figure 5.9: Performance of state-based B&B with and without the new local bound. When the new local bound is not applied, the default local bound by Gillard et al. [20] is applied. Static max cliques and the static and dynamic max degree variable orderings are used. Measured in a) execution time in seconds, b) total B&B nodes used. Both plots are measured across different input graph densities from 0.1 to 0.9, with an initial graph size of 100 and a timeout of 1 hour.

Figure 5.9b shows that indeed, the new local bound improves the efficiency of the B&B search significantly, by several orders of magnitude. This improvement is slightly more with the max degree variable orderings, as opposed to the max cliques ordering (for the lower density instances). This difference between the variable orderings is not as big as

expected; it is likely that the max cliques ordering also sufficiently brings down the overall maximum degree of the subgraphs for the lower bound to work well.

For the runtime (Figure 5.9a), the new local bound significantly improves the dynamic variable ordering, even outperforming the static ordering strategies with the harder (low density) instances. The two methods can share the overhead, while combining their strengths. Furthermore, the dynamic ordering is able to maximize the potential of the new local bound by being able to make the optimal decision per state to lower the maximum degree. Interestingly, for the static variable orderings, the new local bound only improves the runtime with the harder instances. For easier instances, adding the new local bound makes them slower, as the overhead becomes a relatively larger detriment compared to the diminished benefits gained.

Chapter 6

Discussion

This chapter discusses the results from Chapter 5 and the overall insights gained from this thesis. We go over the research questions from Section 1 and see if we can answer them with our obtained results.

1. What does the state-based MDD encoding do differently compared to the layer-based BDD encoding? And do these differences positively impact the performance?

The state-based MDDs are more flexible and efficient than the layer-based BDDs, due to the fact that states can make their own decisions instead of having to decide on the same decision variable as the other states in a layer. The experiments show that even with no additional heuristics, the state-based MDD generally outperforms the layer-based BDD, both in runtime and B&B nodes. The only notable exception to this is with very low density instances, where the state-based MDD is likely slowed down due to fewer neighbouring vertices being pruned when a vertex is selected (as there are fewer neighbours in general with low-density instances). This is a bigger problem for the state-based MDDs, as their branching factor can be quite large at the start, which does not shrink fast enough if few vertices are pruned. This is further supported by the fact that the new beam restrictions heuristic improves the runtime significantly, mostly by restricting the branching factor of the MDDs. Still, even without any beam restrictions, we encourage further exploration of state-based encodings for other problem domains because of their overall efficiency.

2. Each state does not necessarily have to expand with all available decision options. A "beam search"-like restriction could be applied to manipulate the rate at which the decision diagram expands. What kind of effect would different beam sizes have on the performance of the state-based MDD encoding?

The new beam restrictions method significantly speeds up the performance of the state-based MDD encoding for the MISP. This can be explained by the other experiments, which show that the beams have a major impact on the construction of the relaxed DDs. Both the number of merge operations and created nodes are significantly reduced. With this, the construction of the relaxed DDs requires way fewer operations and is faster as a result.

When applying the beam restrictions on another problem domain, the graph coloring problem, it does not necessarily result in the same behaviour. While the runtime does improve, the construction of the relaxed DDs becomes worse, requiring the creation of more B&B nodes. This indicates that the beam restrictions do help in other areas, likely with the imposed ordering of the color choices, but fail to replicate the positive impact on the construction of the relaxed DDs. To properly generalize the beam restrictions method, we would need to integrate it into the solver instead of only changing the model, as we identified in Section 4.1.1. We suspect that this would improve the relaxed DD construction rather than worsening it. The benefit of the current version with the imposed ordering would also apply, leading to an overall improved and general heuristic.

Important to note, while we do suggest integrating the beam restrictions into the solver, we also recommend still applying problem-specific design choices when modelling. In some cases there are special situations that may warrant special treatment, such as the new color $k+1$ in the graph coloring problem. This can help structure the model and further improve the effect of the beam restrictions.

3. Could a dynamic variable ordering strategy exploit the flexible state-based structure of the MDD encoding? How would this compare to various static variable ordering strategies?

The dynamic variable ordering strategy showed that it was capable of making smarter decisions compared to its static variant. The number of B&B nodes was decreased (or at least similar) across all densities. However, this improvement did not translate into faster runtimes because of the additional overhead introduced by the more complex strategy. It seems that with the current implementation, the static variable ordering is "efficient enough", while being very lightweight with comparatively little computational cost. Another reason why the dynamic variable ordering may not perform very well is that, in a state-based context, the vertices are reordered per state instead of per layer. This multiplies the additional work that comes with a dynamic ordering, resulting in relatively more overhead compared to a dynamic ordering in a layer-based context. It should also be noted that for the static ordering, the state-based MDD encoding is inherently more "dynamic" compared to the layer-based BDD encoding. Instead of adhering to the decisions of the layers, each node in the state-based encoding can make different decisions compared to other nodes in the same layer. This could also be a reason why going from a static ordering to a dynamic one may not result in as much of a benefit as is normally seen with layer-based encodings.

For this thesis, the dynamic variable ordering was only applied to the MISP (within the context of a state-based DD encoding). If it were to be applied to other problem domains in a state-based context, we would similarly expect improved efficiency in terms of B&B nodes since the point of a dynamic ordering is the ability to make smarter decisions on the fly. Whether the runtime would also be affected similarly is less clear, as this is more problem-specific. On the one hand, the increase in speed depends on the difference between the static and dynamic ordering. The dynamic ordering benefits more if the states or nodes change a lot from the root node. If this is not the case, a precomputed static order would suffice, since the root node would contain enough information for a decent variable ordering.

On the other hand, whether the dynamic ordering is slowed down too much is dependent on the overhead of tracking extra local information and reordering the vertices. Still, we encourage further exploration of dynamic variable orderings for state-based DDs, as with more complex problems, they may be able to leverage the complexity of the problem to make decisions that improve upon the precomputed decisions even further.

4. What would the effect be on the state-based MDD encoding if we improve local bound pruning? What if it was combined with the new dynamic variable ordering strategy?

The new local bound for the MISP resulted in a significant increase in efficiency, indicated by the major decrease in created B&B nodes. As for the runtime, only for the harder instances does it improve the speed of the static variable ordering. But for the other instances, the improved efficiency seems to not outweigh the extra overhead. The fact that the overhead can be shared with the dynamic variable ordering does make the combination of the two a viable option (especially for the harder instances), unlike each of the two approaches on their own. This combination is more efficient in B&B nodes, while being similar in runtime to the static variable ordering. We suspect that this combination would perform even better with more complex problem instances, as it performed well for harder instances and with increased complexity, the two methods have more opportunity to make use of the increase in efficiency.

Chapter 7

Conclusions and Future Work

This thesis explores the recently introduced state-based multivalued decision diagram (MDD) encodings, which depart from the commonly used layer-based binary decision diagram (BDD) encodings by making decisions per state instead of per layer. Aside from showing that the former generally outperforms the latter for the maximum independent set problem (MISP), this thesis also introduces new heuristics on graph problems for these new state-based MDDs, which can significantly improve both the runtime and search tree size for the MISP. Some of these heuristics may not be able to improve the method on their own, but show major improvements when combined.

We show that the state-based MDD encoding outperforms the layer-based BDD encoding with the MISP. Though, this is only when applying the new beam restrictions on the state-based MDDs, as without these, the MDDs can become slower than the regular layer-based BDDs for harder MISP instances, likely due to the large branching factor of the unrestricted MDDs. This speedup with the beam restrictions is also seen in the graph coloring problem, after applying some modifications to mitigate modelling issues that arise with minimization problems. Additionally, we show that the decision-making of the new dynamic variable ordering strategy can improve upon the decision-making of its static counterpart, but the additional overhead still makes it slower overall with the current implementation. Furthermore, we present a new local bound for the maximum independent set problem (MISP), which can significantly reduce the search tree needed to solve instances of the MISP. The extra overhead does cancel out most of the performance gain in runtime, except for low-density problem instances. It also greatly improves the new dynamic variable ordering, with which it can share the same overhead. This makes it possible for this combination of techniques to outperform the regular static variable ordering heuristics in B&B nodes, while having a similar runtime to its static counterpart. With larger density instances, the instances used in our experiments are not difficult enough for the efficiency gain to outweigh the additional overhead. We suspect that for harder instances (that are perhaps larger) this performance difference is more prominent.

For future work, the state-based MDD encodings are promising, and with the introduction of state-based decision diagram solvers such as CODD [34] there is great potential for further exploration. This thesis only covers the MISP and the graph coloring problem, so other problem domains are still left unexplored.

As for the heuristics, the beam restrictions method could be further generalized by fully integrating them into the solver instead of applying them when modelling. This would make them more straightforward to apply to other problem domains. It would also become more user-friendly by preventing any issues that may arise when manually changing the weights of transitions for each different encoding. However, there is still merit in including problem-specific design choices at the modelling level (such as taking a new color $k+1$ in the graph coloring problem). Lastly, local bounds can be very effective at increasing the efficiency of the model, as some of our experiments show. Even though they need to be custom-made per problem domain, it can be worth to look into small manners to tighten the local bound to allow for early pruning, by utilizing more information on the current state or perhaps adding even more information per state. This additional information could then also be used to strengthen the dynamic variable ordering strategy.

Each of these avenues for future work has the potential to further improve the performance of the DD-based solvers and to help them solve more challenging combinatorial optimization problems. Especially shifting from the regular layer-based encodings to state-based ones opens up many possibilities, introducing a different perspective on modelling these optimization problems.

Bibliography

[1] Akers. Binary decision diagrams. *IEEE Transactions on computers*, 100(6):509–516, 1978.

[2] Henrik Reif Andersen, Tarik Hadzic, John N Hooker, and Peter Tiedemann. A constraint store based on multivalued decision diagrams. In *Principles and Practice of Constraint Programming–CP 2007: 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007. Proceedings 13*, pages 118–132. Springer, 2007.

[3] David Bergman, Willem-Jan Van Hoeve, and John N Hooker. Manipulating mdd relaxations for combinatorial optimization. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 20–35. Springer, 2011.

[4] David Bergman, Andre A Cire, Willem-Jan Van Hoeve, and John N Hooker. Variable ordering for the application of bdds to the maximum independent set problem. In *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems: 9th International Conference, CPAIOR 2012, Nantes, France, May 28–June1, 2012. Proceedings 9*, pages 34–49. Springer, 2012.

[5] David Bergman, Andre A Cire, Willem-Jan van Hoeve, and John N Hooker. Optimization bounds from binary decision diagrams. *INFORMS Journal on Computing*, 26(2):253–268, 2014.

[6] David Bergman, Andre A Cire, Willem-Jan van Hoeve, and Tallys Yunes. Bdd-based heuristics for binary optimization. *Journal of Heuristics*, 20:211–234, 2014.

[7] David Bergman, Andre A Cire, Willem-Jan Van Hoeve, and John Hooker. *Decision diagrams for optimization*, volume 1. Springer, 2016.

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

[9] John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. *Graph theory with applications*, volume 290. Macmillan London, 1976.

[10] Daniel Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[11] Quentin Cappart, Emmanuel Goutierre, David Bergman, and Louis-Martin Rousseau. Improving optimization bounds using machine learning: Decision diagrams meet deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1443–1451, 2019.

[12] Margarita P Castro, Andre A Cire, and J Christopher Beck. Decision diagrams for discrete optimization: A survey of recent advances. *INFORMS Journal on Computing*, 34(4):2271–2295, 2022.

[13] Andre A Cire and Willem-Jan Van Hoeve. Multivalued decision diagrams for sequencing problems. *Operations Research*, 61(6):1411–1428, 2013.

[14] Vianney Coppé, Xavier Gillard, and Pierre Schaus. Modeling and exploiting dominance rules for discrete optimization with decision diagrams. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 226–242. Springer, 2024.

[15] Timothy Curry, Laurent Michel, and Willem-Jan van Hoeve. Pyddopt: A modeling interface for decision diagrambased optimization. Presented at DPSOLVE 2023, 2023.

[16] Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 2). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2, 2024.

[17] Eugene C Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM (JACM)*, 29(1):24–32, 1982.

[18] Birger Funke, Tore Grünert, and Stefan Irnich. Local search for vehicle routing and scheduling problems: Review and conceptual integration. *Journal of heuristics*, 11: 267–306, 2005.

[19] Michael R Garey and David S Johnson. "strong"np-completeness results: Motivation, examples, and implications. *Journal of the ACM (JACM)*, 25(3):499–508, 1978.

[20] Xavier Gillard, Vianney Coppé, Pierre Schaus, and André Augusto Cire. Improving the filtering of branch-and-bound mdd solver. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 231–247. Springer, 2021.

[21] Nicolas Golenvaux, Xavier Gillard, Siegfried Nijssen, and Pierre Schaus. Partitioning a map into homogeneous contiguous regions: A branch-and-bound approach using decision diagrams (short paper). In *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, pages 45–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2023.

[22] Jaime E Gonzalez, Andre A Cire, Andrea Lodi, and Louis-Martin Rousseau. Integrated integer programming and decision diagram search tree with an application to the maximum independent set problem. *Constraints*, 25(1):23–46, 2020.

[23] Tarik Hadzic, John N Hooker, Barry O'Sullivan, and Peter Tiedemann. Approximate compilation of constraints into multivalued decision diagrams. In *International Conference on Principles and Practice of Constraint Programming*, pages 448–462. Springer, 2008.

[24] Samid Hoda, Willem-Jan Van Hoeve, and John N Hooker. A systematic approach to mdd-based constraint programming. In *Principles and Practice of Constraint Programming–CP 2010: 16th International Conference, CP 2010, St. Andrews, Scotland, September 6-10, 2010. Proceedings 16*, pages 266–280. Springer, 2010.

[25] John N Hooker. Decision diagrams and dynamic programming. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 10th International Conference, CPAIOR 2013, Yorktown Heights, NY, USA, May 18-22, 2013. Proceedings 10*, pages 94–110. Springer, 2013.

[26] Matthias Horn, Günther R Raidl, and Elina Rönnberg. An a* algorithm for solving a prize-collecting sequencing problem with one common and multiple secondary resources and time windows. In *Proceedings of PATAT*, pages 235–256, 2018.

[27] Matthias Horn, Johannes Maschler, Günther R Raidl, and Elina Rönnberg. A-based construction of decision diagrams for a prize-collecting scheduling problem. *Computers & Operations Research*, 126:105125, 2021.

[28] Lingying Huang, Xiaomeng Chen, Wei Huo, Jiazheng Wang, Fan Zhang, Bo Bai, and Ling Shi. Branch and bound in mixed integer linear programming problems: A survey of techniques and trends. *arXiv preprint arXiv:2111.06257*, 2021.

[29] Anthony Karahalios and Willem-Jan van Hoeve. Variable ordering for decision diagrams: A portfolio approach. *Constraints*, 27(1):116–133, 2022.

[30] Richard M Karp. On the computational complexity of combinatorial problems. *Networks*, 5(1):45–68, 1975.

[31] Joris Kinable, Andre A Cire, and Willem-Jan van Hoeve. Hybrid optimization methods for time-dependent sequencing problems. *European Journal of Operational Research*, 259(3):887–897, 2017.

[32] B.T. Lowerre. The HARPY speech recognition system. Ph.D. Thesis. 1976.

[33] Johannes Maschler and Günther R Raidl. Multivalued decision diagrams for prize-collecting job sequencing with one common and multiple secondary resources. *Annals of Operations Research*, 302:507–531, 2021.

[34] Laurent Michel and Willem-Jan van Hoeve. Codd: A decision diagram-based solver for combinatorial optimization. In *ECAI 2024*, pages 4240–4247. IOS Press, 2024.

[35] Shin-ichi Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In *Proceedings of the 30th International Design Automation Conference*, pages 272–277, 1993.

[36] Mohsen Nafar and Michael Römer. Strengthening relaxed decision diagrams for maximum independent set problem: Novel variable ordering and merge heuristics. In *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

[37] Peng Si Ow and Thomas E Morton. Filtered beam search in scheduling. *The International Journal Of Production Research*, 26(1):35–62, 1988.

[38] Augustin Parjadis, Quentin Cappart, Louis-Martin Rousseau, and David Bergman. Improving branch-and-bound using decision diagrams and reinforcement learning. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 18th International Conference, CPAIOR 2021, Vienna, Austria, July 5–8, 2021, Proceedings 18*, pages 446–455. Springer, 2021.

[39] Isaac Rudich, Quentin Cappart, and Louis-Martin Rousseau. Peel-and-bound: Generating stronger relaxed bounds with multivalued decision diagrams. *arXiv preprint arXiv:2205.05216*, 2022.

[40] Isaac Rudich, Quentin Cappart, and Louis-Martin Rousseau. Improved peel-and-bound: Methods for generating dual bounds with multivalued decision diagrams. *Journal of Artificial Intelligence Research*, 77:1489–1538, 2023.

[41] Willem-Jan van Hoeve. Graph coloring lower bounds from decision diagrams. In *Integer Programming and Combinatorial Optimization: 21st International Conference, IPCO 2020, London, UK, June 8–10, 2020, Proceedings*, pages 405–418. Springer, 2020.

[42] Willem-Jan van Hoeve. An introduction to decision diagrams for optimization. In *Tutorials in Operations Research: Smarter Decisions for a Better World*, pages 117–145. INFORMS, 2024.

Appendix A

DP Models

This appendix chapter describes the dynamic programming (DP) models of the methods used in this thesis. A DP model consists of the following:

- A set of states, including the initial and terminal states.
- A label generation function $\lambda : \mathcal{S} \rightarrow \mathcal{U}$ which generates the different decisions that can be made when a state is expanded. The label that it produces is denoted by l in the DP models.
- A state transition function $\tau : \mathcal{S} \times \mathcal{U} \rightarrow \mathcal{S}$ which transitions a state to another state based on a given label (corresponding to a decision).
- A state cost function $c : \mathcal{S} \times \mathcal{U} \rightarrow \mathbb{R}$ which gives the cost or weight of a transition.
- A merge operator $\oplus : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}$ which merges two states together.

A.1 MISP

The DP model uses an input graph $G = (V, E)$. The aim is to find an independent set of vertices such that no two vertices in the set are neigbours in G. $N(v)$ is the set of neighbouring vertices of vertex v. The vertices V are ordered beforehand when using a static variable ordering (or ordered dynamically with a dynamic ordering).

A.1.1 BDD encoding

- State definition: $< S, n >$ where $S \subseteq V$ is the set of eligible vertices still available to be included in the independent set and n is the current vertex for which a decision is made.
  - Initial state: $< V, 0 >$
  - Terminal state: $< \emptyset, |V| >$
- Label generation function: $\lambda(< S, n >) = \{0, 1\}$

- State transition function: $\tau(< S,n >,l) = \begin{cases} < \emptyset, |V| > & \text{if } |S| = 0 \\ < S \setminus (N(v_n) \cup \{v_n\}), n+1 > & \text{if } l = 1 \\ < S \setminus \{v_n\}, n+1 > & \text{if } l = 0 \end{cases}$
- State cost function: $c(< S,n >,l) = 1$
- Merge operator: $\oplus(< S_1,n_1 >, < S_2,n_2 >) =< (S_1 \cup S_2), n_1 >$ if $n_1 = n_2$

A.1.2 MDD encoding

Important to note, for the dynamic variable ordering, the states are defined by $< S,t,D >$ instead. Here, D tracks the local degrees of each vertex in S. The label generation function uses D to determine the label l dynamically. D is updated in the state transition function and by the merge operator to accurately reflect the local degrees.

- State definition: $< S,t >$ where $S \subseteq V$ is the set of eligible vertices still available to be included in the independent set and t is the number of transitions from the root node
  - Initial state: $< V,0 >$
  - Terminal state: $< \emptyset, |V| >$
- Label generation function: $\lambda(< S,t >) = S$
- State transition function: $\tau(< S,t >,l) = \begin{cases} < \emptyset, |V| > & \text{if } |S| = 0 \\ < S \setminus (N(v_l) \cup \{v_1, ..., v_l\}), t+1 > & \text{else} \end{cases}$
- State cost function: $c(< S,t >,l) = 1$
- Merge operator: $\oplus(< S_1,t_1 >, < S_2,t_2 >) =< (S_1 \cup S_2), t_1 >$ if $t_1 = t_2$

A.1.3 MDD encoding with Beam Restrictions

The DP model now also uses a beam width b. The beam B is the set of vertices chosen to be included in the beam in the label generation function.

- State definition: $< S,t >$ where $S \subseteq V$ is the set of eligible vertices still available to be included in the independent set and t is the number of transitions from the root node
  - Initial state: $< V,0 >$
  - Terminal state: $< \emptyset, |V| >$
- Label generation function: $\lambda(< S,t >) = \begin{cases} S & \text{if } |S| \leq b \\ B \cup \{-1\} & \text{else} \end{cases}$
- State transition function: $\tau(< S,t >,l) = \begin{cases} < \emptyset, |V| > & \text{if } |S| = 0 \\ < S \setminus (N(v_l) \cup \{v_1, ..., v_l\}), t+1 > & \text{if } v_l \in B \\ < S \setminus B, t+1 > & \text{if } l = -1 \end{cases}$

- State cost function: $c(< S,t >,l) = \begin{cases} 0 & \text{if } l = -1 \\ 1 & \text{else} \end{cases}$
- Merge operator: $\oplus(< S_1,t_1 >,< S_2,t_2 >) =< (S_1 \cup S_2),t_1 > \text{if } t_1 = t_2$

A.1.4 Local Bounds

The DP model for the MISP also uses a local bound, either the old bound by Gillard et al. [20] or the new local bound introduced in this thesis. The new local bound requires the same structure as that of the dynamic variable ordering, where the states are defined as $< S,t,D >$. The additional state parameter D tracks the local degrees in S.

- Old local bound: $f(< S,t >) = |V|$
- New local bound: $f(< S,t,D >) = |V| - \lceil \frac{|E|}{\Delta(G)} \rceil$

A.2 Graph Coloring

The DP model uses an input graph $G = (V,E)$. The aim is to find a color assignment A such that no neighbouring vertices share the same color. A in the states represents the color assignments of the vertices. In the state transition function, A' is used to indicate a new color assignment, where the current vertex v is colored according to the chosen color. A' in the merging function represents a merged version of the color assignments, where the vertices are uncolored if they differed between A_1 and A_2, otherwise keeping their color. The vertices V are ordered beforehand when using a static variable ordering (or ordered dynamically with a dynamic ordering).

A.2.1 MDD encoding

- State definition: $< A,v,k,C >$ where A represents the color assignments of the vertices, v is the current vertex that needs to be colored, k is the highest numbered color used so far, and C is the set of eligible vertices for v.
  - Initial state: $< A,0,0,\{0,1\} >$
  - Terminal state: $< A,0,|V|,C >$
- Label generation function: $\lambda(< A,v,k,C >) = C$
- State transition function: $\tau(< A,v,k,C >,l) = \begin{cases} < A,0,|V|,C > & \text{if } v = |V| \\ < A',v+1,k,\{0,1,...,k+1\} > & \text{if } 0 \leq l \leq k \\ < A',v+1,k+1,\{0,1,...,k+2\} > & \text{if } l = k+1 \end{cases}$
- State cost function: $c(< A,v,k,C >,l) = \begin{cases} 0 & \text{if } 0 \leq l \leq k \\ 1 & \text{if } l = k+1 \end{cases}$

- Merge operator: $\oplus(< A_1, v_1, k_1, C_1 >, < A_2, v_2, k_2, C_2 >) =< A', v_1, k_1, C_1 \cup C_2 >$ if $v_1 = v_2 \wedge k_1 = k_2$

A.2.2 MDD encoding with Beam Restrictions

Here, the DP model uses a beam width b. The beam B is the set of vertices chosen to be included in the beam in the label generation function.

- State definition: $< A, v, k, C, e >$ where A represents the color assignments of the vertices, v is the current vertex that needs to be colored, k is the highest numbered color used so far, C is the set of eligible vertices for v, and e is the number of no-beam transitions taken so far.
  - Initial state: $< A, 0, 0, \{0, 1\}, 0 >$
  - Terminal state: $< A, 0, |V|, C, 0 >$
- Label generation function: $\lambda(< A, v, k, C, e >) = \begin{cases} C & \text{if } |C| \leq b \\ B \cup \{-1\} & \text{else} \end{cases}$
- State transition function: $\tau(< A, v, k, C, e >, l) = \begin{cases} < A, 0, |V|, C, 0 > & \text{if } v = |V| \\ < A', v+1, k, \{0, 1, ..., k+1\}, e > & \text{if } 0 \leq l \leq k \\ < A', v+1, k+1, \{0, 1, ..., k+2\}, e > & \text{if } l = k+1 \\ < A, v, k, C \setminus B, e+1 > & \text{if } l = -1 \end{cases}$
- State cost function: $c(< A, v, k, C, e >, l) = \begin{cases} 0 & \text{if } 0 \leq l \leq k \\ 1 & \text{if } l = k+1 \\ \frac{1}{|V|+1} & \text{if } l = -1 \\ \frac{1}{|V|+1} \cdot e & \text{if } v = |V| \end{cases}$
- Merge operator: $\oplus(< A_1, v_1, k_1, C_1, e_1 >, < A_2, v_2, k_2, C_2, e_2 >) =< A', v_1, k_1, C_1 \cup C_2, max(e_1, e_2) >$ if $v_1 = v_2 \wedge k_1 = k_2$