UNIVERSITY OF CALIFORNIA,
IRVINE

Methods for advancing combinatorial optimization over graphical models

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Natalia Flerova

Dissertation Committee:
Rina Dechter, Chair
Alexander Ihler
Richard Lathrop

2015

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

Page

List of Algorithms

ACKNOWLEDGMENTS

This thesis would not be possible without help and contributions of many people: my advisor, committee members, colleagues, friends and family.

I owe a great debt of gratitude to my advisor, Professor Rina Dechter, who guided and supported me throughout grad school. Rina always provided encouragement and motivation when I needed it. I thank her for constructive criticism, advice and her patience. It has been a pleasure and a privilege to work with her.

My warmest gratitude goes to my committee members, Professors Alexander Ihler and Richard Lathrop for their time, comments and suggestions. I would like to thank Professor Ihler for many fruitful research discussions. I would also like to thank Professor Lathrop for teaching me how to deal with a classroom of undergrads in my first ever quarter as a TA.

I thank the members of my research group, past and present: William Lam, Junkyuu Lee, Silu Yang, Filjor Broka, Lars Otten, Vibhav Gogate and Kalev Kask, for many interesting and insightful conversations. I am especially grateful to Radu Marinescu and Emma Rollon, with whom I had a pleasure to collaborate on several papers.

Last but not least, I could not have done this without the incredible support of my family. My parents and my husband Sidharth have always provided me with confidence and cheered me up, especially in the time of deadlines.

I am grateful for the assistance I have received for my graduate studies from NSF grants IIS-1065618 and IIS-1254071, from the United States Air Force under Contract No. FA8750-14-C-0011 under the DARPA PPAML program and by the Donald Bren School of Information and Computer Science at University of California, Irvine.

CURRICULUM VITAE

Natalia Flerova

EDUCATION

Doctor of Philosophy in Computer Science **2015**
University of California, Irvine *Irvine, California*

Master of Science in Computer Science **2012**
University of California, Irvine *Irvine, California*

Specialist in Automated Control Systems **2005**
Baltic Technical State University *St. Petersburg, Russia*

RESEARCH EXPERIENCE

Graduate Research Assistant **2010–2015**
University of California, Irvine *Irvine, California*

Visiting Researcher **2007–2008**
University of California, Irvine *Irvine, California*

TEACHING EXPERIENCE

Teaching Assistant **2009–2010**
University of California, Irvine *Irvine, California*

Lecturer **2005–2006**
Baltic Technical State University *St. Petersburg, Russia*

REFEREED CONFERENCE PUBLICATIONS

Weighted Best-First Search for W-Optimal Solutions over Graphical Models **2015**
AAAI workshop on Planning, Optimization and Search

Evaluating Weighted DFS Branch and Bound over Graphical Models **2014**
Symposium on Combinatorial Search

Weighted Best First Search for MAP **2014**
International Symposium on Artificial Intelligence and Mathematics

Anytime AND/OR Best-First Search for Optimization in Graphical Models **2013**
ICML workshop on Interaction Between Inference and Learning

Preliminary Empirical Evaluation of Anytime Weighted AND/OR Best-First Search for MAP **2012**
NIPS workshop on Discrete and Combinatorial Problems in Machine Learning

Join-graph based cost-shifting schemes **2012**
Conference on Uncertainty in Artificial Intelligence

Search Algorithms for m Best Solutions for Graphical Models **2012**
AAAI Conference on Artificial Intelligence

Mini-bucket Elimination with Moment Matching **2011**
NIPS workshop on Discrete and Combinatorial Problems in Machine Learning

Inference Schemes for M Best Solutions for Soft CSPs **2011**
CP workshop on Preferences and Soft Constraints

Heuristic Search for M Best Solutions with Applications to Graphical Models **2011**
CP workshop on Preferences and Soft Constraints

Bucket and mini-bucket Schemes for M Best Solutions over Graphical Models **2011**
Special Issue of Lecture Notes in Computer Science: Graph structures for knowledge representation and reasoning

M best solutions over Graphical Models **2010**
CP workshop on Constraint Reasoning and Graphical Structures

SOFTWARE

MBE-M-OPT `http://graphmod.ics.uci.edu/group/mbemopt`
Mini-bucket elimination for finding the m best solutions to MPE/MAP problem in Bayesian/Markov networks.

M-BEST-SEARCH `http://graphmod.ics.uci.edu/group/mbestsearch`
OR and AND/OR best-first and depth-first branch and bound search for finding the m best solutions to MPE/MAP problem in Bayesian/Markov networks.

W-SEARCH `http://graphmod.ics.uci.edu/group/wsearch`
Anytime best-first and depth-first branch and bound search that uses weighted heuristic to find approximate solutions to MPE/MAP task.

ABSTRACT OF THE DISSERTATION

Methods for advancing combinatorial optimization over graphical models

By

Natalia Flerova

Doctor of Philosophy in Computer Science

University of California, Irvine, 2015

Rina Dechter, Chair

Graphical models are a well-known convenient tool to describe complex interactions between variables. A graphical model defines a function over many variables that factors over an underlying graph structure. One of the popular tasks over graphical models is that of combinatorial optimization. Although many algorithms have been developed with this task in mind, the vast majority are designed to find an optimal solution, minimum or maximum, of an objective function. In many applications, however, it is desirable to obtain not just a single optimal solution, but a set of the first m best solutions for some integer m.

The main part of this dissertation focuses on this problem, which we call the m-best optimization task. We show that the m-best task can be expressed within the unifying framework of semirings, making known inference algorithms defined, and their correctness and completeness for the m-best task immediately implied. We subsequently describe elim-m-opt, a new bucket elimination algorithm for solving the m-best task, provide algorithms for its defining combination and marginalization operators and analyze its worst-case performance. An extension of the algorithm to the mini-bucket framework provides bounds for each of the m best solutions.

Subsequently, we extend existing search algorithms to the m-best task. We present a new algorithm m-A* and prove that all A*'s desirable properties, including soundness, completeness and optimal efficiency, are maintained. Since best-first algorithms require extensive memory, we also extend the memory-efficient depth-first branch and bound to the m-best task. We adapt both algorithms to optimization tasks over graphical models (e.g., Weighted Constraint Satisfaction Problems and Most Probable Explanation in Bayesian networks), and provide complexity analysis and an empirical evaluation. Our experiments with 5 variants of best-first search and depth-first branch and bound search confirm that the best-first approach is largely superior when memory is available, but branch and bound is more robust. We also demonstrate that both styles of search benefit greatly from the heuristic evaluation function with increased accuracy.

Unlike the leading previously developed m-best schemes that utilize LP-relaxation techniques, e.g., algorithms by Fromer and Globerson (2009) and Batra (2012), our algorithms always guarantee solution optimality. We will show that, when the number of required solutions is small, our m-best search schemes are quite competitive with these related algorithms in terms of runtime, while for a larger number of required solutions our methods are by far superior.

The second part of this thesis focuses on finding approximate solutions to optimization problems. Unfortunately solving exactly optimization problems over complex models, that represent intricate dependencies occurring in real life domains, can often be infeasible within practical time and space limits. Many approximation schemes exist, but most of them do not come with any solution sub-optimality guarantees. We apply the ideas of weighted heuristic search, popular in path-finding, to graphical models, yielding new search algorithms that not only provide sub-optimality bounds, but also utilize extra available time and space to improve the accuracy of the solution in an anytime manner and, if resources are available, eventually terminate with an optimal solution. We report on a significant empirical evaluation, demonstrating the usefulness of weighted best-first search as approximation anytime schemes (that have sub-optimality bounds) and compare against one of the best depth-first branch and bound solvers to date. We also investigate the impact of different heuristic functions on the behavior of the algorithms.

Additionally, we explore several algorithms taking advantage of two common approaches for bounding MPE queries in graphical models: mini-bucket elimination and message-passing updates for linear programming relaxations. Each method offers a useful perspective for the other; our hybrid approaches attempt to balance the advantage of each. We demonstrate the power of our hybrid algorithms through extensive empirical evaluation. Most notably, a branch and bound search guided by the heuristic functions calculated by our new scheme won the first place in the 2011 Pascal2 inference challenge.

Chapter 1

Introduction

Graphical models, e.g., Bayesian, Markov or constraint networks, are a well-known framework for representation of probabilistic and deterministic information. They allow to model complicated interactions between variables in an intuitive way, using directed or undirected graphs that capture problem structure.

Some of the most popular tasks over graphical models, arising in many practical applications, are optimization queries, the goal of which is to find a variable assignment that either maximizes or minimizes the objective function. In particular, the common tasks include finding the most likely state of a belief network, known as Most Probable Explanation (MPE) or as Maximum A Posteriori (MAP) problem. Another is finding a solution that violates the least number of constraints in a constraint network, i.e., constraint satisfaction (CSP) problem. These tasks are NP-hard.

Many effective optimization algorithms that exploit the underlying graph structure of the problems have been developed over the years. Conceptually these algorithms typically fall into either inference or search category.

The research presented in this dissertation is concerned with the advancement of graphical model algorithms along three different dimensions:

- extending the existing optimization schemes to finding not a single, but a set of multiple solutions ordered by their costs
- extending to graphical models the ideas of anytime search that uses non-admissible inflated heuristic (known as weighted heuristic search)
- improving the existing heuristics used by the graphical model-oriented search algorithms

The following section describes the outline of the dissertation and the contributions. The rest of this chapter contains preliminary definitions and gives examples of graphical models and optimization algorithms that are relevant to our entire work. In each of Chapters 2, 4 and 5 the second section contains additional background relevant to this particular chapter. Other sections of these chapters and the entire Chapter 3 present our original contributions.

■ 1.1 Thesis Outline and Contributions

■ 1.1.1 Bucket Elimination for M-best Optimization Task

Chapter 2 focuses on the task of generating the first m best solutions for a combinatorial optimization problem defined over a graphical model (e.g., the m most probable explanations for a Bayesian network), for some integer m. Such tasks often arise in practice, for example, in situations where there exist multiple solutions to the problems with almost identical probabilities and it is desirable to identify them all and present to an expert for further analysis (as it happens in the area of genetic linkage analysis) or when some vague constraints of a preference nature are hard to formally incorporate in the problems (e.g., in economics).

Though a number of algorithms solving the m-best task are available, many of them have very obvious practical deficiencies. They either use a brute force approach for generating additional solutions, resulting in a large overhead compared to finding a single solutions (e.g., [61]), or assume availability of an entire search graph in memory (e.g., most path-finding algorithms, see [30] for an overview), which is infeasible for most problems over graphical models.

Contributions

- We show that the m-best task can be expressed within the unifying framework of semirings, defining the corresponding combination and marginalization operators. Such formulation makes known inference algorithms defined and their correctness and completeness for the m-best task immediately implied.
- We extend a well-known inference algorithm bucket elimination [19] to the m-best task, yielding a new algorithm elim-m-opt, and provide algorithms for its defining combination and marginalization operators.
- We analyze the worst-case performance of elim-m-opt, contrasting it with the previously developed related schemes.
- We propose an extension of the algorithm to the mini-bucket framework [25], generating bounds for each of the m best solutions and provide an empirical demonstration of this algorithm.

■ 1.1.2 Heuristic Search for M-best Task

Chapter 3 is devoted to finding the m best solutions using best-first or depth-first branch and bound search. While most exact schemes developed for solving the m-best optimization problems in graphical models are inference based, they frequently suffer from large memory and time requirements, including our elim-m-opt algorithm described above. At the same time, for the regular optimization task of finding the single best solution there exists a class of algorithms known to be efficient: AND/OR search schemes, including AOBD/OR Best First search [69], AND/OR Branch and Bound [68] and Breadth-Rotating AND/OR Branch and Bound [76]. These algorithms are guided by heuristics, typically generated using mini-bucket elimination algorithm [50, 51], and take advantage of problem decomposition by exploring an AND/OR search space [23]. Note that throughout we use "efficient" in the intuitive meaning of the word, rather than as the technical term used to imply polynomial-time algorithms in computer science. Clearly, the optimization schemes over graphical models never have polynomial complexity, since the tasks are NP-hard.

Contributions

- We present a new algorithm m-A*, extending the well-known A* to the m-best task, and proving that all its desirable properties are maintained. In particular, we show that m-A* is sound and complete, that it is optimally efficient in terms of expanded nodes and optimally efficient in terms of number of times each node is expanded when the heuristic is consistent.

- Since best-first algorithms require extensive memory, we also extend the memory-efficient depth-first branch and bound to the m-best task, and analyze the search space explored by this new algorithm, which we call m-BB.

- We adapt both algorithms to optimization tasks over graphical models (e.g., Weighted CSP and MPE in Bayesian networks) and provide an analysis of their asymptotic worst case time and space complexity.

- We present the first, to our best knowledge, systematic empirical evaluation of these m-best search algorithms over graphical models. We evaluated 5 variants of m-best search schemes on various real-world and simulated benchmarks, comparing and contrasting the performance of m-best best-first and depth-first branch and bound search when exploring various underlying search spaces: an OR tree, an AND/OR tree or, in case of m-best best-first search, additionally an AND/OR graph.

- Our empirical comparison of m-best search algorithms with some of the most efficient to-date m-best schemes based on the Linear Programming relaxation [37, 5] showed that our algorithms are often more efficient for large values of m, while guaranteeing solution optimality.

■ 1.1.3 Anytime Weighted Heuristic Search for Graphical Models

Chapter 4 discusses the applicability of the well-known notion of weighted heuristic search, i.e., a search that uses a heuristic (originally admissible) multiplied by a positive weight $w > 1$ to make it inadmissible. The idea of weighted heuristic search was proposed for best-first search and, in particular, A*, in the context of path-finding [79] and proved to often facilitate faster results and smaller search space, while yielding a solution, whose cost is guaranteed to be within the factor of w from the optimal. A variety of anytime versions of the weighted best-first search have been subsequently proposed [43, 64, 97, 80, 96, 80]. However, their potential for graphical models was largely ignored, possibly because of their memory costs and because the alternative depth-first branch and bound seemed very appropriate for bounded depth problems. The weighted depth-first search has not been studied for graphical models.

Contributions

- We import the ideas of anytime weighted best-first search from the path-finding domain, investigating the impact of weighted heuristic on the solution accuracy, runtime and size of the explored search space for AOBF algorithm over graphical models.

- We present two anytime weighted heuristic search schemes: wAOBF and wR-AOBF.

Both run the AOBF with a weighted heuristic iteratively, while decreasing the weight at each iteration. Algorithm wAOBF runs each iteration from scratch, while wR-AOBF exploits previous computations in the style of ARA* [64].

- We also suggest anytime weighted depth-first branch and bound algorithms based on AOBB and BRAOBB schemes.
- We conduct an extensive empirical evaluation of the proposed schemes on a large variety of benchmarks, demonstrating the potential of both weighted best-first search and weighted depth-first branch and bound algorithms as anytime schemes (that have sub-optimality bounds) and compare against one of the best depth-first branch and bound solvers to date (BRAOBB).
- We also formulate the notion of focused search space, a space often yielding fast exploration. We analyze its properties and derive the value of the weight that, when used by weighted best-first search, guarantees the search to be focused. Moreover, this weight value can be used to bound the optimal solution cost based on just a single arbitrary solution to the problem.

■ 1.1.4 Cost-Shifting Schemes for Better Approximation

Chapter 5 is focused on the ways to improve the accuracy of the bounds produced by the mini-bucket elimination. This algorithm is valuable not only as an approximate optimization scheme, but also as a heuristic generator for the AND/OR search schemes. The main source of the error in mini-bucket elimination arises from the necessity to split a set of functions containing a particular variable in their scope into smaller groups and process these groups independently. Intuitively such operation is equivalent to making several copies of the variable in the graph. If these copies take the same assignment, then the optimal solution to the modified problem is also an optimal solution to the original optimization problem. However, in practice it is most often not the case, and thus the solution obtained by mini-bucket elimination is not exact.

An alternative common approximate approach to optimization in graphical models is based on Linear Programming relaxation [99]. This relaxation technique transforms an NP-hard optimization problem into a related problem, that is solvable in polynomial time and whose solution is a guaranteed to be a bound on the optimal solution to the original problem.

Contributions

- We take advantage of both of the methods for bounding MPE queries in graphical models: mini-bucket elimination and message-passing updates for linear programming relaxations, developing two hybrid algorithms, which attempt to balance the advantages of each approach.
- We demonstrate the power of our hybrid algorithms through empirical evaluation, assessing the schemes' performance both as bounding schemes and as heuristic generators for the search algorithms. Most notably, a branch and bound search guided by the heuristic function calculated by one of our new algorithms has won the first place in the 2011 PASCAL2 inference challenge.

■ 1.2 Preliminaries and Background

The remainder of this chapter is devoted to the background and concepts upon which this work builds. Additional definitions, pertaining to individual tasks solved, are given at the beginning of corresponding chapters. We start by introducing the notion of graphical models.

■ 1.2.1 Graphical Models

Many real life applications, for example, in the domains of machine vision [31, 93], genetic linkage analysis [3, 32, 33] or natural language processing [18, 60], involve a large group of random variables that interact with each other. The explicit representation of the joint distribution over the entire set is exponential in the number of variables and thus is often infeasible to specify explicitly. Graphical models constitute a formal framework that allows to represent the joint distribution compactly in a factored form using a graph-based representation that captures the problem's structure. For more details see [78, 20, 21, 46].

■ 1.2.1.1 Notation

We denote variables or sets of variables by capital letters (e.g., X, Y, Z, S) and values of variables by lower case letters (e.g., x, y, z, s). An assignment $(X_1 = x_1, ..., X_n = x_n)$ can be abbreviated as $\mathbf{x} = (x_1, ..., x_n)$. The domain of variable X_j is denoted D_{X_j} or $D(X_j)$. For a set of variables $\mathbf{S}$, $\mathbf{D_S}$ denotes the Cartesian product of the domains of variables in $\mathbf{S}$. If $\mathbf{X} = \{X_1, ..., X_n\}$ and $\mathbf{S} \subseteq \mathbf{X}$, $\mathbf{x_S}$ denotes the restriction of $\mathbf{x} = (x_1, ..., x_n)$ to variables in $\mathbf{S}$ (also known as the projection of $\mathbf{x}$ over $\mathbf{S}$), namely $\forall x_j \in \mathbf{x}$, x_j is in $\mathbf{x_S}$ if and only if the variable X_j belongs to the set $\mathbf{S}$. We denote functions by letters f, g, h, etc., and the scope (set of arguments) of a function f by $Scope(f)$. The projection of a tuple $\mathbf{x}$ on the scope of a function f can also be denoted by $\mathbf{x}_{Scope(f)}$ or, for brevity, by $\mathbf{x}_f$. We also denote a function f over a subset of variables $\mathbf{S}_j = \{X_1, \ldots, X_r\}$ as $f_{\mathbf{S}_j}$ or, when the scope is clear, by f_j. Abusing notation, we sometime write $f_{\mathbf{S}_j}(\mathbf{x})$ to mean $f_{\mathbf{S}_j}(\mathbf{x}_{\mathbf{S}_j})$. We will sometimes denote the scope variables as arguments, writing, e.g., $f(\mathbf{S})$ or $f(X_1, X_2, X_3)$. We will also use the terms "instantiation" and "assignment" interchangeably.

Definition 1.1 (**Elimination operators**, [21]). *Given a function f_j over a scope $\boldsymbol{S}_j$, the functions $(\min_{\boldsymbol{X}} f_j)$, $(\max_{\boldsymbol{X}} f_j)$, and $(\sum_{\boldsymbol{X}} f_j)$, where $\boldsymbol{X} \subseteq \boldsymbol{S}_j$, are defined over $\boldsymbol{U} = \boldsymbol{S}_j - \boldsymbol{X}$ as follows: for every $\boldsymbol{U} = \boldsymbol{u}$, denoting by $(\boldsymbol{u}, \boldsymbol{x})$ the extension of tuple $\boldsymbol{u}$ by the tuple $\boldsymbol{X} = \boldsymbol{x}$,*

$(\min_{\boldsymbol{X}} f_j)(\boldsymbol{u}) = \min_{\boldsymbol{x}} f_j(\boldsymbol{u}, \boldsymbol{x})$*,* $(\max_{\boldsymbol{X}} f_j)(\boldsymbol{u}) = \max_{\boldsymbol{x}} f_j(\boldsymbol{u}, \boldsymbol{x})$*, and* $(\sum_{\boldsymbol{X}} f_j)(\boldsymbol{u}) = \sum_{\boldsymbol{x}} f_j(\boldsymbol{u}, \boldsymbol{x})$*. Given a set of functions* $f_1, ..., f_k$ *defined over the scopes* $\boldsymbol{S}_1, ..., \boldsymbol{S}_k$*, the product function* $\prod_j f_j$ *and the sum function* $\sum_j f_j$ *are defined over* $\boldsymbol{U} = \cup_j \boldsymbol{S}_j$ *such that for every* $\boldsymbol{u} \in D_{\boldsymbol{U}}$*,* $(\prod_j f_j)(\boldsymbol{u}) = \prod_j f_j(\boldsymbol{u}_{\boldsymbol{S}_j})$ *and* $(\sum_j h_j)(\boldsymbol{u}) = \sum_j h_j(\boldsymbol{u}_{\boldsymbol{S}_j})$*. The operator* $\arg\max_{\boldsymbol{X}} f_j$ *(or* $\arg\min_{\boldsymbol{X}} f_j$*) returns a tuple* $\boldsymbol{x}$ *for which function* f_j *attains its maximum (minimum) value.*

Note that we use terms *elimination* and *marginalization* interchangeably. For convenience when we denote a function obtained by an eliminating operator (e.g., $\sum$) over a function f defined over a set of variables $\mathbf{Y}$, where $\mathbf{X} \subseteq \mathbf{Y}$, we will use interchangeably $\sum_{\mathbf{X}} f$, $\sum_{\mathbf{x}} f(y)$ and $\sum_{\mathbf{X}} f(\mathbf{Y})$, all defining a function over the scope $\mathbf{Y} - \mathbf{X}$, with the meaning made clear by the context.

■ 1.2.1.2 Graphical models

A graphical model is a collection of local functions over subsets of variables that conveys probabilistic, deterministic, or preferential information, and whose structure is described by a graph. The graph captures independencies or irrelevance information inherent in the model, that can be useful for interpreting the modeled data and, most significantly, can be exploited by reasoning algorithms. The set of local functions can be combined in a variety of ways to generate a global function, whose scope is the set of all variables.

Definition 1.2 (Graphical model). *A* graphical model $\mathcal{M}$ *is a 4-tuple* $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \bigotimes \rangle$*:*

1. $\mathbf{X} = \{X_1, \ldots, X_n\}$ *is a finite set of variables;*

2. $\mathbf{D} = \{D_1, \ldots, D_n\}$ *is the set of their respective finite domains of values;*

3. $\mathbf{F} = \{f_1, \ldots, f_r\}$ *is a set of non-negative real-valued discrete functions, defined over scopes of variables* $\boldsymbol{S}_i \subseteq \boldsymbol{X}$*. They are called* local *functions.*

4. $\bigotimes$ *is a combination operator, e.g.,* $\bigotimes \in \{\prod, \sum\}$ *(product, sum)*

The graphical model represents a global function*, whose scope is* $\boldsymbol{X}$ *and which is the combination of all the local functions:* $\bigotimes_{j=1}^{r} f_j$.

The choice of the variables, functions and the concrete combination operator defines a particular kind of graphical model. We focus on Bayesian networks, Markov networks and Weighted Constraint Satisfaction Problems (WCSPs).

Definition 1.3 (**Bayesian network**, [78])**.** *A* Bayesian *(also known as* belief*)* network *is a graphical model* $\mathcal{B} = \langle \mathbf{X}, \mathbf{D}, \mathbf{P}, \prod \rangle$*, where* $\mathbf{X}$ *is the set of discrete random variables with domains* $\mathbf{D}$ *and where functions* $\mathbf{P} = \{P_j(X_j|\boldsymbol{pa}_j)\}$ *are conditional probability tables defined relative to a directed acyclic graph* G *over* $\mathbf{X}$*, where for every* X_j*,* $\boldsymbol{pa}_j = \{X_{j_1}, \ldots, X_{j_k}\}$ *are the parents of* X_j*, i.e., for each* X_{j_k} *there is an edge pointing from* X_{j_k} *to* X_j*. A Bayesian network represents the joint probability distribution given by:* $P(\mathbf{X}) = \prod_{j=1}^{n} P(X_j|\boldsymbol{pa}_j)$*.*

For example, consider a Bayesian network with 5 variables, whose directed acyclic graph (DAG) is given in Figure 1.1(a).

Given a Bayesian network $\mathcal{B} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{P}, \prod \rangle$ the most common optimization task is to find the Most Probable Explanation (MPE), also known as Maximum A Posteriori hypothesis (MAP). Its aim is to compute the maximum probability $P^* = \max_{\mathbf{X}} \prod_{f \in \mathbf{P}} f$ and the corresponding assignment $\mathbf{x}^* = \arg\max_{\mathbf{X}} \prod_{f \in \mathbf{P}} f$.

Definition 1.4 (**Markov network**, [78],[21])**.** *A* Markov network *is a graphical model* $\mathcal{T} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \prod \rangle$*, where* $\boldsymbol{F} = \{f_{\boldsymbol{S}_1}, \ldots, f_{\boldsymbol{S}_r}\}$ *is a set of functions, often referred to as* potentials*, where each potential* $f_{\boldsymbol{S}_j}$ *is a non-negative real-valued function defined on scopes* $\boldsymbol{S}_j \subseteq \mathbf{X}$*. A Markov network represents the joint probability distribution given by:* $P(\boldsymbol{X}) = \frac{1}{Z} \prod_{j=1}^{r} f_j$*, where* $Z = \sum_{\boldsymbol{X}} \prod_{j=1}^{r} f_j$*. The normalization constant* Z *is called the partition function, computing it is one of the main tasks over Markov networks.*

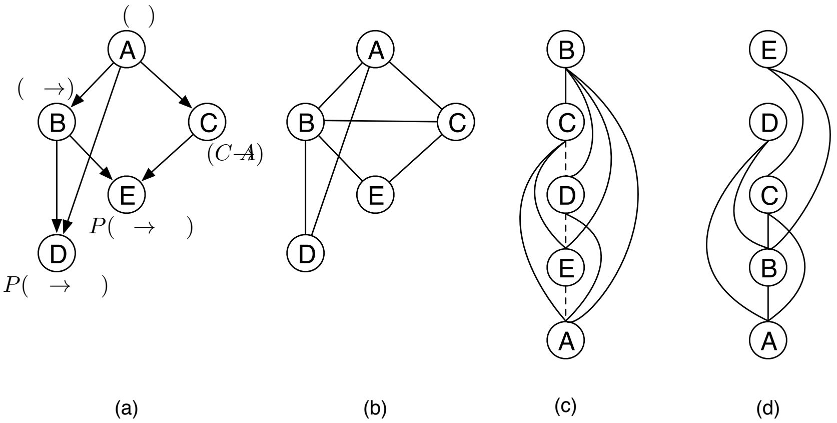


Figure 1.1: (a) A DAG of a Bayesian network, (b) its primal graph (also called moral graph), (c) its induced graph along $o = (A, E, D, C, B)$, and (d) its induced graph along $o = (A, B, C, D, E)$ (after [41]).

Definition 1.5 (**Weighted Constraint Satisfaction Problem, WCSP**, [21]). WCSP *is a graphical model* $\mathcal{C} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$*, where* $\mathbf{F} = \{f_{\boldsymbol{S}_1}, \ldots, f_{\boldsymbol{S}_r}\}$ *is a set of real-valued non-negative functions. Each function* $f_{\boldsymbol{S}_j}$*, also called cost-component, has a scope* $\boldsymbol{S}_j \subseteq \mathbf{X}$ *and assigns "0" (no penalty) to allowed tuples and a positive integer penalty cost to the forbidden tuples.*

The primary optimization task over WCSP is finding a minimal cost assignment (min-sum): $C^* = \min_{\mathbf{x}} \sum_j f_j(\mathbf{x})$ and the optimizing configuration $\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_j f_j(\mathbf{x})$. Historically this task is also sometimes referred to as *energy minimization*. It is equivalent to an MPE/MAP task in the following sense: if $C^*_{max} = \max_{\mathbf{x}} \prod_j f_j(\mathbf{x})$ is a solution to an MPE problem, then $C^*_{max} = \exp(-C^*_{min})$, where C^*_{min} is a solution to a min-sum problem $C^*_{min} = \min_{\mathbf{x}} \sum_j g_j(\mathbf{x})$ and $\forall j, g_j(\mathbf{x}) = -\log(f_j(\mathbf{x}))$.

A graphical model defines a primal graph capturing dependencies between the variables.

Definition 1.6 (**Primal graph**). *The* primal graph *of a graphical model is an undirected graph that has variables as its vertices. An edge connects any two variables that appear in the scope of the same function.*

For a Bayesian network, the primal graph is also called a moral graph.

Definition 1.7 (**Moral graph**). *The* moral graph *of a directed graph is an undirected graph obtained by connecting all parents of a node to each other and removing direction.*

Figure 1.1(b) depicts the primal graph of the Bayesian network in Figure 1.1(a). Note that parents of variable E (variables B and C) are connected in the primal graph.

An important feature of a graphical model, that characterizes the complexity of its reasoning tasks, is the induced width.

Definition 1.8 (**Ordered graph, induced width** ([21])). *An* ordered graph *is a pair* (G, o) *where* G *is an undirected graph, and* $o = (X_1, \ldots, X_n)$ *is an ordering of nodes. The* width of a node *is the number of the node's neighbors that precede it in the ordering. The* width of a graph along an ordering o *is the maximum width over all nodes. An* induced ordered graph *is obtained from an ordered graph as follows: nodes are processed from last to first based on* o*; when node* X_j *is processed, all its preceding neighbors are connected. The width of an ordered induced graph along the ordering* o *is called* induced width along o *and is denoted by* $w^*(o)$*. The* induced width *of a graph, denoted by* w^**, is the minimal induced width over all its orderings. Abusing notation we sometimes use* w^* *to denote the induced width along a particular ordering, when the meaning is clear from the context.*

Figures 1.1(c) and 1.1(d) depict the induced graphs of the example primal graph in Figure 1.1(b) along the orderings $o = (A, E, D, C, B)$ and $o' = (A, B, C, D, E)$, respectively. The dashed lines correspond to the induced edges, namely edges that are absent from the moral graph, but were introduced in the induced graph. The induced width along ordering o is $w^*(o) = 4$ and the one along ordering o' is $w^*(o') = 2$.

■ 1.2.2 Variable Elimination for Inference in Graphical Models

Common approaches to solving optimization tasks over graphical models include variable elimination and conditioning algorithms, such as search (for details see [21]). In this section we present two schemes based on the variable elimination idea, leaving the discussion of search algorithms to Section 1.2.4.

■ 1.2.2.1 Exact Inference: Bucket Elimination

Bucket elimination (BE) [19] is a framework that provides a unifying view of variable elimination algorithms for a variety of reasoning tasks.

Algorithm 1 presents the bucket elimination algorithm for the MPE task. As an input it accepts a graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \prod \rangle$, a variable ordering o and a marginalization operator max. For WCSP the combination operator is $\sum$ and the marginalization operator is min. Given a variable ordering, each variable is associated with a bucket constructed as follows: all the functions defined on variable X_j, but not on variables appearing later in the ordering, are placed into the bucket of X_j. We denote the bucket of variable X_j as $bucket_{X_j}$ or $\mathbf{B}_{X_j}$. By $Scope(\mathbf{B}_{X_j})$ we denote the variables appearing in the functions in $bucket_{X_j}$. Namely, if $\mathbf{B}_{X_j}$ contains functions $f_1, \ldots, f_k$ with scopes $\mathbf{S}_1, \ldots, \mathbf{S}_k$ respectively, then $Scope(\mathbf{B}_{X_j}) = \cup_{p=1}^{k} \mathbf{S}_p$. We say that the variable that appears later in the ordering o is a higher-index variable than the ones that appear sooner.

Once the buckets are created, BE processes buckets from last to first. It computes new functions by combining all the functions in the bucket and then applying to the resulting function (known as the “bucket function”) the elimination operator (e.g., max). The newly computed functions, also called messages, are placed in lower buckets using the following rule. A function generated in $\mathbf{B}_{X_j}$ is placed in $\mathbf{B}_{X_k}$, where X_k is the latest variable in $Scope(\mathbf{B}_{X_j})$ relative to ordering o, excluding X_j. We denote this function by $h_{X_j \to X_k}$. This phase of

Algorithm 1: Bucket Elimination [19]

```
Input: Graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \prod \rangle$, variable ordering $o$, marginalization operator
       max
Output: An optimal solution to the MPE task over $\mathcal{M}$ and the optimal assignment
//Initialize
Partition the functions in $\mathbf{F}$ into $bucket_{X_1}, \dots, bucket_{X_n}$, where $bucket_{X_p}$ contains all
functions whose highest-index variable according to the ordering $o$ is $X_p$;
//Backward pass
for $p \leftarrow n$ down to 1 do
    Let $g_1, \dots, g_r$ be the functions in $bucket_{X_p}$ (including both original functions and
    previously generated messages) having scopes $\mathbf{S}_1, \dots, \mathbf{S}_r$, respectively;
    if $X_p$ is instantiated $(X_p = x_p)$ then
        Assign $X_p = x_p$ to each $g_j$ and put each resulting function into its appropriate
        bucket;
    else
        Generate the message function $h_{X_p \to X_k}$: $h_{X_p \to X_k} = \max_{X_p} \prod_{j=1}^{r} g_j$, where $X_k$ is the
        highest-index variable in $Scope(h_{X_p \to X_k}) = \cup_{j=1}^{r} \mathbf{S}_j - X_p$;
        place $h_{X_p \to X_k}$ in $bucket_{X_k}$;
//Forward pass
Assign a value to each variable along ordering $o$ which optimizes the combination of the
functions currently in the bucket ;
return The function computed in the bucket of the first variable and the corresponding
assignment;
```

the algorithm is known as “the backward pass”. Subsequently, during the “forward pass”, the algorithm constructs a solution by assigning a value to each variable along the ordering, consulting the functions created during the backward phase. Note that the forward pass is relevant only to optimization tasks. For a summation task, such as finding the partition function, the algorithm bucket elimination would have only a backward pass.

As an illustration we apply the bucket elimination algorithm to the network in Figure 1.1(a) along $o = (A, E, D, C, B)$, solving an MPE problem. We compute P^* as

$$\begin{aligned} P^* &= \max_{a,b,c,d,e} P(a,b,c,d,e) && (1.1) \\ &= \max_{a,b,c,d,e} P(a)P(c|a)P(e|b,c)P(d|a,b)P(b|a) && (1.2) \\ &= \max_a P(a) \max_e \max_d \max_c P(c|a) \max_b P(e|b,c)P(d|a,b)P(b|a). && (1.3) \end{aligned}$$

Bucket elimination computes this expression from right to left using the buckets, as shown:

1. $bucket_B$: $h_{B\rightarrow C}(a,d,c,e) = \max_b P(e|b,c)P(d|a,b)P(b|a)$
2. $bucket_C$: $h_{C\rightarrow D}(a,d,e) = \max_c P(c|a)h_{B\rightarrow C}(a,d,c,e)$
3. $bucket_D$: $h_{D\rightarrow E}(a,e) = \max_d h_{C\rightarrow D}(a,d,e)$
4. $bucket_E$: $h_{E\rightarrow A}(a) = max_e h_{D\rightarrow E}(a,e)$
5. $bucket_A$: $P^* = \max_a P(a)h_{E\rightarrow A}(a)$,

A schematic trace of the algorithm is shown in Figure 1.2.

Bucket elimination can be viewed as message passing from leaves to root along a so-called bucket tree, whose nodes are the buckets and $bucket_X$ is a child of $bucket_Y$, if there is a function $h_{X\rightarrow Y}$ which is generated in $bucket_X$ and placed in $bucket_Y$ during BE. The root bucket is often the first bucket. For example, the bucket tree of the problem in Figure 1.2 is a chain, since each bucket receives a message from only one other bucket. It was shown that,

Theorem 1.1. *[19, 21] Given a graphical model with variable ordering o having induced width $w^*(o)$, the time and space complexity of the bucket elimination scheme is $O(r \cdot k^{w^*(o)+1})$ and $O(n \cdot k^{w^*(o)})$ respectively, where r is the number of functions, n is the number of problem variables and k is the maximum domain size.*

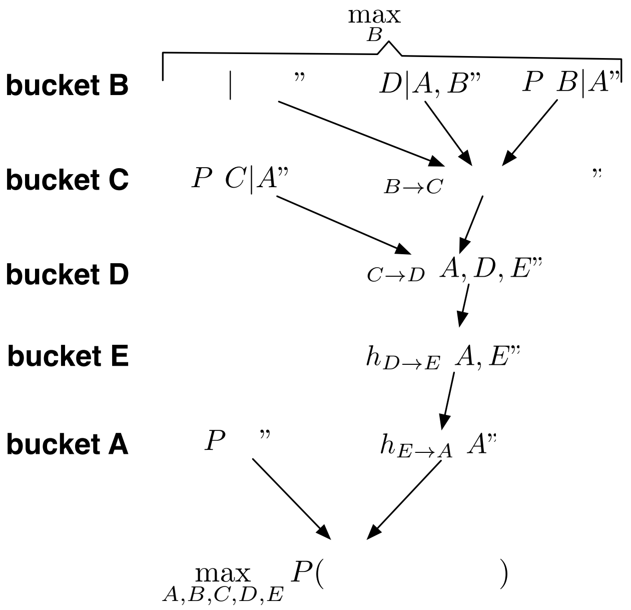


Figure 1.2: A trace of bucket elimination algorithm

Since the complexity of bucket elimination algorithm is exponential in induced width along the ordering, ideally we want to find a variable ordering that has the smallest induced width. Although this problem has been shown to be NP-hard [4], there are a few greedy heuristic algorithms that provide good orderings [20, 21].

■ 1.2.2.2 Approximate Inference: Mini-Bucket Elimination

Bucket elimination is infeasible for many practical problems having large induced width. Thus an approximate version of the algorithm, called mini-bucket elimination (MBE) was proposed [25]. MBE (Algorithm 2) bounds the space and time complexity of the full bucket elimination. Given a variable ordering, the algorithm associates each variable X_k with a bucket, which contains all functions defined on this variable, but not on higher index variables, as bucket elimination does. Subsequently, when processing buckets, large buckets are partitioned into smaller subsets, called *mini-buckets*, each containing at most $i + 1$ distinct variables. The parameter i is called the i-bound. In the following we often use “i-bound”

Algorithm 2: Mini-Bucket Elimination [25]

```
Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \prod \rangle$, variable ordering $o$, marginalization operator max, parameter $i$
Output: An approximate solution to the MPE task over $\mathcal{M}$ and an assignment to all variables
//Initialize
Partition the functions in $\mathbf{F}$ into $bucket_{X_1}, \dots, bucket_{X_n}$, where $bucket_{X_p}$ contains all functions whose highest-index variable is $X_p$.
//Backward pass
for $p \leftarrow n$ down to 1 do
    Let $g_1, \dots, g_r$ be the functions in $bucket_{X_p}$ (including both original functions and previously generated messages); let $\mathbf{S}_1, \dots, \mathbf{S}_r$ be the scopes of functions $g_1, \dots, g_r$;
    if $X_p$ is instantiated $(X_p = x_p)$ then
        Assign $X_p = x_p$ to each $g_j$ and put each resulting function into its appropriate bucket;
    else
        Partition functions in $\mathbf{B}_{X_p}$ into mini-buckets, generating the partitioning $Q_{X_p} = \{q_p^1, \dots, q_p^l\}$, where each $q_p^t \in Q_{X_p}$ has no more than $i + 1$ variables;
        foreach $q_p^t \in Q_{X_p}$ do
            Generate the message function $h^t_{X_p \to X_k} = \max X_p \prod_j g^t_j$, where $g^t_j \in q^t_p$ and $X_k$ is the highest-index variable in $Scope(h^t_{X_p \to X_k}) = \cup_j Scope(g^t_j) - X_p$;
            Add $h^t_{X_p \to X_k}$ to $bucket_{X_k}$;
//Forward pass
Assign a value to each variable in the ordering $o$ so that the combination of the functions in each bucket is optimal, according to the marginalization operator max;
return The function computed in the bucket of the first variable and the corresponding assignment;
```

and "i" interchangeably. We denote the mini-buckets obtained by partitioning the bucket $\mathbf{B}_{X_p}$ by $Q_{X_p} = \{q_p^1, \dots, q_p^n\}$, where q_p^j is the j^{th} mini-bucket of variable X_p. MBE generates an upper bound on the optimal MPE/MAP value, $\hat{P} \geq P^*$ (and lower bound on the optimal WCSP value).

To demonstrate the execution of MBE we again turn to the network in Figure 1.1(a), with the ordering $o = (A, E, D, C, B)$. Let us set $i = 2$, i.e., restrict each mini-bucket to contain no more than 3 variables. Since the scope of $bucket_B$ is greater than 3, it is necessary to split $bucket_B$ into two separate mini-buckets, which will then be processed independently. Let us assume that one of them contains function $P(E|B, C)$ and the other functions $P(D|A, B)$

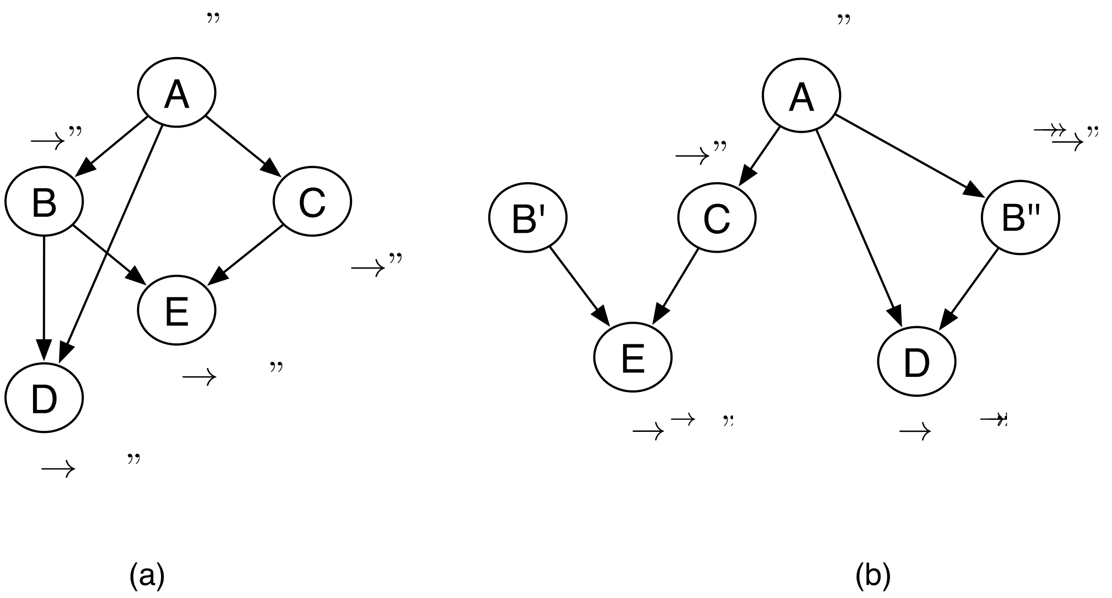


Figure 1.3: (a) The original Bayesian network, (b) The network with duplicated variable B

and $P(B|A)$. The question of how to best distribute the functions between mini-buckets is not trivial and is usually solved heuristically. More information can be found in [81].

Splitting $bucket_B$ into two mini-buckets can be viewed as replacing variable B by two duplicate variables: B' and B''. We denote the corresponding mini-buckets as $bucket_{B'}$ and $bucket_{B''}$. Figure 1.3(a) shows the original network, and 1.3(b) presents the network with duplicated variables. Note that in Figure 1.3(b) variable A is no longer connected to B', since no function has both these variables in its scope. An execution of MBE is equivalent to running an exact bucket elimination algorithm on the resulting modified problem[1]:

1. $bucket_{B'}$: $h_{B' \to C}(c, e) = \max_b P(e|b, c)$

2. $bucket_{B''}$: $h_{B'' \to D}(a, d) = \max_b P(d|a, b)P(b|a)$

3. $bucket_C$: $h_{C \to E}(a, e) = \max_c P(c|a)h_{B' \to C}(c, e)$

[1] Notice also that the network in 1.3(b) is not fully legitimate as a Bayesian network since B' has no function. This however has no real consequence and will not be further discussed.

4. $bucket_D$: $h_{D\to A}(a) = \max_d h_{B''\to D}(a,d)$

5. $bucket_E$: $h_{E\to A}(a) = \max_e h_{C\to E}(a,e)$

6. $bucket_A$: $\hat{P} = \max_a P(a) h_{E\to A}(a) h_{D\to A}(a)$

Figure 1.4 shows the trace of the mini-bucket elimination algorithm.

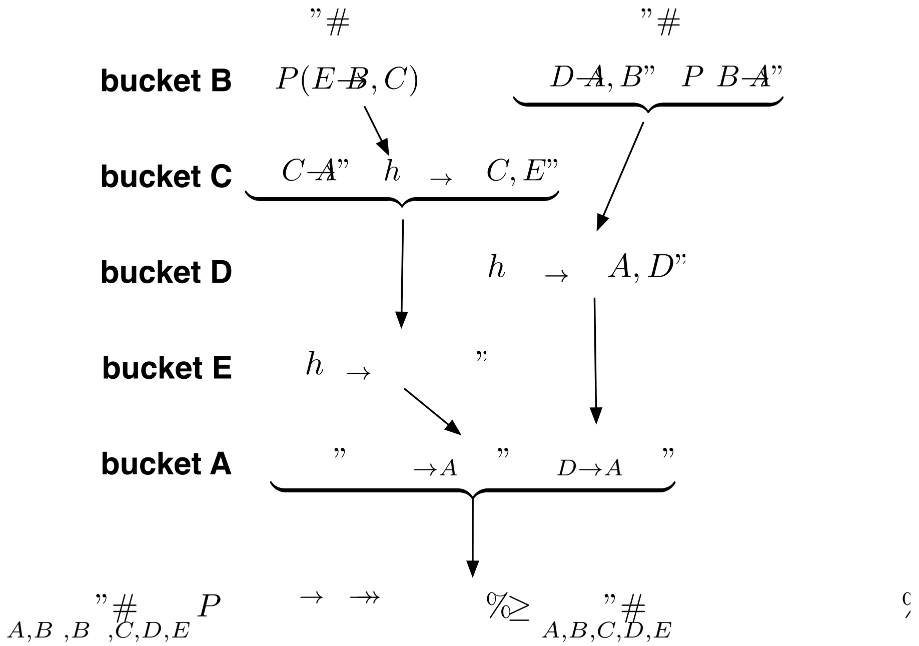


Figure 1.4: A trace of mini-bucket elimination algorithm

Theorem 1.2. *[25] Given a graphical model with variable ordering o having induced width $w^*(o)$ and an i-bound parameter i, the time complexity of the mini-bucket algorithm MBE(i) is $O(n \cdot k^{min(i,w^*(o))+1})$ and space complexity is $O(n \cdot k^{min(i,w^*(o))})$, where n is the number of problem variables and k is the maximum domain size.*

Higher values of i take more computational resources, but yield more accurate bounds. When i is large enough (i.e., $i \geq w^*(o)$), MBE coincides with the full bucket elimination.

As we will describe in greater details in Section 1.2.4.4, the mini-bucket elimination is often used to generate heuristics for both best-first and depth-first branch and bound search over graphical models [50, 51, 53].

■ 1.2.3 Heuristic Search

Search algorithms are used in a vast variety of applications. Though we mostly concentrate on the application of search to optimization tasks over graphical models, we will provide some background on general purpose search schemes as well. For more information see, for example, Pearl [77].

Consider a search space defined implicitly by a set of states (the nodes in the graph), operators that map states to states, having costs or weights (the directed weighted arcs), a starting state n_0 and a set of goal states. We say that a node is *generated*, when its representation code is computed based on the heuristic information and the information about its parent. It is said that the parent of the node is then *explored*, or *expanded*, when all its children have been generated. A node *expansion* consists of generating all successors of a given parent node. We call a path *explored*, if all nodes on this path have been expanded. The task typically assumed is to find the least cost solution path from n_0 to a goal [72], where the cost of a solution path is the sum or the product of the weights on its arcs.

Search procedure, or strategy, is a policy that determines the order, in which nodes are generated. We distinguish between *blind* (or *uninformed*) search and *informed* (or *heuristic*) search. The former operates based only on information obtained during the search process (e.g., the cost of getting from the root to the current node).

A heuristic search algorithm uses partial (heuristical) information about the search space and the goal in order to move towards more promising solutions. A *heuristic function*, denoted $h(n)$, provides an estimate of the cost of the least cost path from each node n to any of the goals. A heuristic function is called *admissible*, if and only if it never overestimates (for minimization task) the true minimal cost, $h^*(n)$, to reach the goal from node n, namely, $\forall n\ h(n) \leq h^*(n)$. A heuristic is called *consistent* or *monotonic*, if for every node n and for every successor n' of n we have:

$$h(n) \leq c(n, n') + h(n') \tag{1.4}$$

where $c(n, n')$ is the weight of the arc (n, n').

The two main heuristic search strategies are best-first search and depth-first branch and bound search. Both of them assess how promising each node is and make decisions concerning node expansions based on a numerical estimation called *evaluation function*, which estimates the minimal cost of the path from start to a goal that passes through node n.

■ 1.2.3.1 Best-First Search

Best-first search (BFS) always expands the node with the best (e.g., smallest for minimization problem) value of the evaluation function. It maintains a graph of explored paths, a list CLOSED of expanded nodes and a frontier of OPEN nodes. BFS chooses from OPEN a node n with lowest value of an evaluation function $f(n)$, expands it, places it on CLOSED, and places its child nodes on OPEN. The most popular variant of best-first search, A*, uses the evaluation function $f(n) = g(n) + h(n)$, where $g(n)$ is the current minimal cost from the root to n, and $h(n)$ is a heuristic function that estimates the optimal cost-to-go $h^*(n)$ from n to a goal node. In the following we assume that the heuristic used by A* is admissible, unless specified otherwise. If $h(n)$ is consistent, then the values of evaluation function $f(n)$ along any path are non-decreasing. A path π is called *C^*-bounded* relative to f, if $\forall n \in \pi : f(n) < C^*$, where C^* is the cost of optimal solution. It is known that, regardless of the tie-breaking rule, A* expands any node n that is reachable by a strictly

C^*-bounded path from the root. Such a node is said to be *surely expanded by A** [24].

A* search possesses a number of attractive properties [72, 77, 24]:

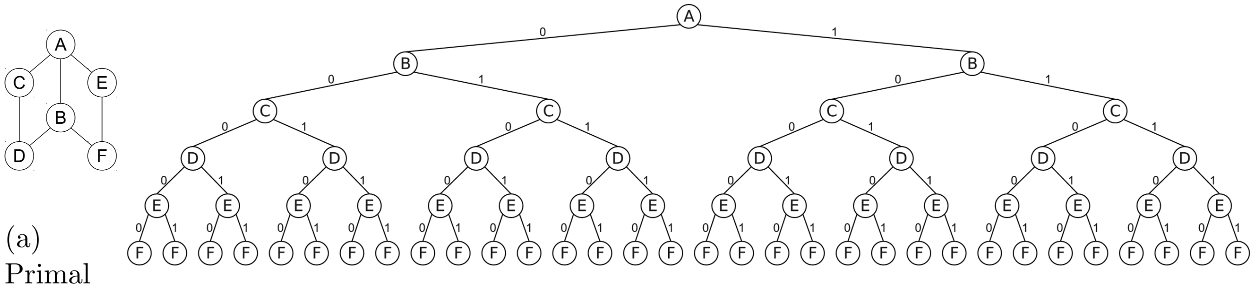
- **Soundness and completeness**: A* terminates with an optimal solution.
- When h is consistent, A* explores only the nodes in the set $S = \{n|f(n) \leq C^*\}$ and it surely expands all the nodes having $S = \{n|f(n) < C^*\}$.
- **Optimal efficiency under consistent heuristic**: When h is consistent, any node surely expanded by A* must be expanded by any other sound and complete search algorithm having access to the same heuristic information. Also, in thise case A* will expand each node at most once, when searching a graph, because at the time of node's expansion A* has found the least-cost path to it.
- **Dominance**: Given two heuristic functions h_1 and h_2, such that $\forall n \ h_1(n) < h_2(n)$, A_1^* will expand every node surely expanded by A_2^*, where A_j^* uses heuristic h_j.

Though best-first search is known to be the best algorithm in terms of number of nodes expanded [24], sometimes it requires storing the whole search space, which means often an exponential memory in the worst-case.

■ 1.2.3.2 Depth-First Branch and Bound

A popular alternative is depth-first branch and bound (DFBB), whose most attractive feature, compared to best-first search, is that it can be executed with linear memory. Yet, when the search space is a graph, it can exploit additional memory to improve its performance by flexibly trading space and time.

Depth-first branch and bound expands nodes in a depth-first manner, maintaining an upper bound UB on the cost of the optimal solution, which equals to the best solution cost found



(a) Primal graph

(b) OR search tree along ordering A, B, C, D, E, F

Figure 1.5: OR search space example problem (after [75])

so far (initially infinity). If the evaluation function of the current node n is greater than the upper bound, the node is *pruned* and the subtree below it is never explored. In the worst case depth-first branch and bound explores the entire search space. In the best case the first solution found is optimal, in which case DFBB's performance can be as good as BFS. However, if the solution depth is unbounded, depth-first search might follow an infinite branch and never terminate. Also, if the search space is a graph, DFBB may expand nodes numerous times, unless it uses additional memory for caching and checks for duplicates.

The average case is hard to characterize for both DFBB and BFS because we do not know C^*, and even if we did, we cannot easily estimate the set $\{n|f(n) \leq C^*\}$. This depends on the values of heuristic function and, for DFBB, on the order in which the solutions are encountered.

■ 1.2.4 Heuristic Search in Graphical Models

Search algorithms provide a way to systematically enumerate all possible assignments of a given graphical model. Optimization problems can be naturally presented as the task of finding an optimal cost path in an appropriate search space.

The simplest variant of a search space is a so-called OR search tree. Each level of this tree corresponds to a variable from the original problem. The nodes correspond to variable assignments and the arc weights are derived from problems' functions. The size of such search tree is $O(k^n)$, where n is the number of variables and k is the maximum domain size.

Throughout this section we illustrate the concepts using the example problem with six variables (A, B, C, D, E, F) and seven pairwise functions. Its primal graph is shown in the Figure 1.5(a). Figure 1.5(b) displays the corresponding OR search tree along lexicographical ordering (after [75]).

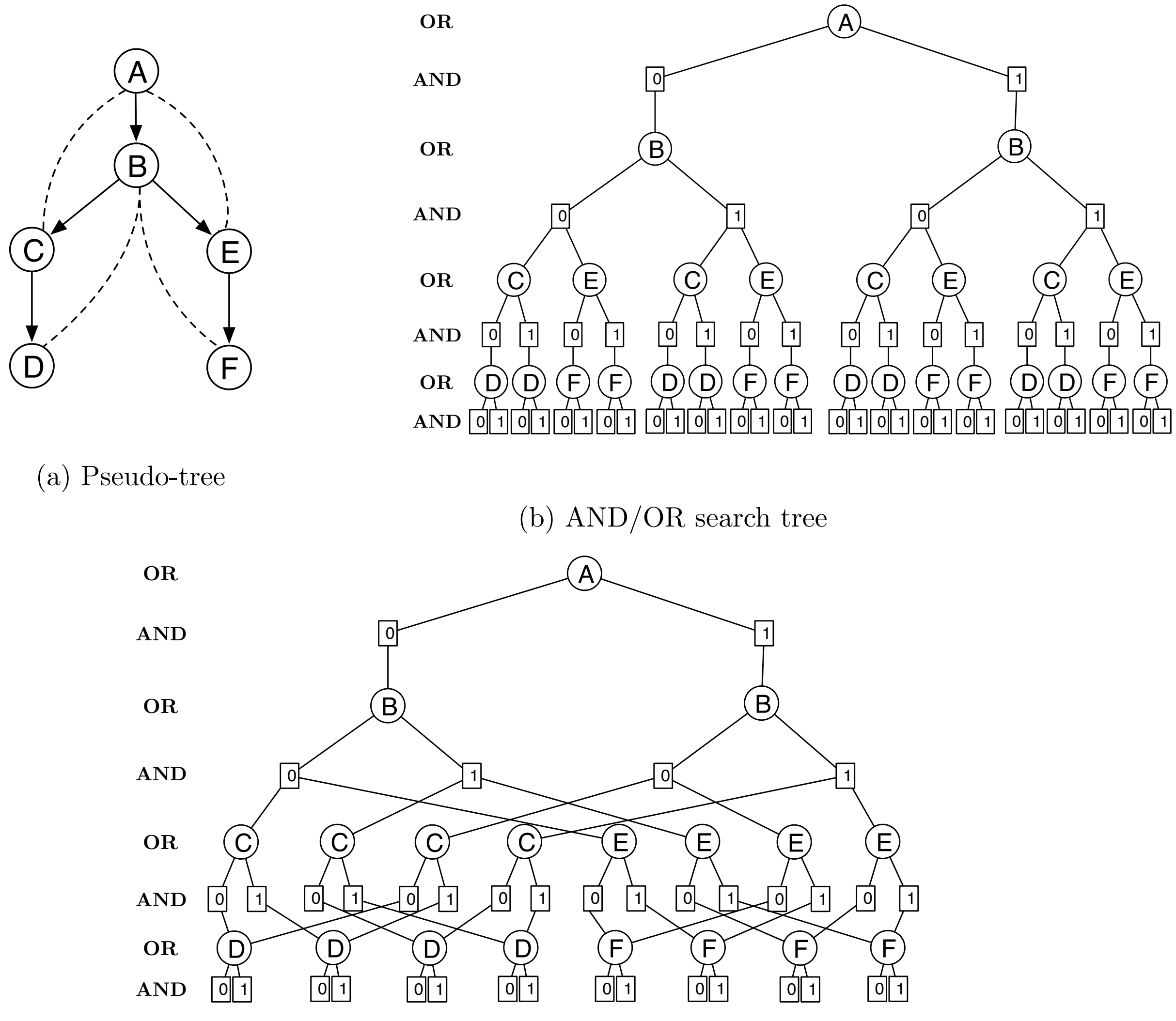
■ 1.2.4.1 AND/OR Search Spaces

OR search trees are blind to the problem decomposition encoded in the graphical models and can therefore be inefficient. They do not exploit the independencies in the model. AND/OR search spaces for graphical models have been introduced to better capture the problem structure [23]. The AND/OR search space is defined by a *pseudo-tree* of the primal graph that captures problem decomposition.

Definition 1.9 (**Pseudo-tree**, [36])**.** *A* pseudo-tree *of an undirected graph $G = (V, E)$ is a directed rooted tree $\mathcal{T} = (V, E')$, such that every arc of G not included in E' can be viewed as a* back-arc *relative to $\mathcal{T}$, namely it is an arc that connects a node to an ancestor relative to $\mathcal{T}$. A node n' is an* ancestor *of n in $\mathcal{T}$ if it appears on the path from the root to n in $\mathcal{T}$.*

Definition 1.10 (**AND/OR search tree**, [23])**.** *Given a graphical model $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \bigotimes \rangle$ with primal graph G and a pseudo-tree $\mathcal{T}$ of G, the* AND/OR search tree *$S_{\mathcal{T}}$ contains alternating levels of OR and AND nodes. Its structure is based on the underlying pseudo-tree $\mathcal{T}$. The root node of $S_{\mathcal{T}}$ is an OR node labelled by the variable at the root of $\mathcal{T}$. The children of an OR node labeled X_j are AND nodes labelled with value assignments $\langle X_j, x_j \rangle$ (or simply $\langle x_j \rangle$); the children of an AND node $\langle X_j, x_j \rangle$ are OR nodes labelled with the children of X_j in $\mathcal{T}$, representing conditionally independent subproblems.*

Figure 1.6(a) shows a pseudo-tree of the example problem. The solid directed edges belong



(a) Pseudo-tree

(b) AND/OR search tree

(c) Context-minimal AND/OR search graph

Figure 1.6: AND/OR search spaces example

to the pseudo-tree. The dashed lines represent the back-arcs in the primal graph depicted in Figure 1.5(a), but are not part of the pseudo-tree. An AND/OR tree corresponding to the pseudo-tree in Figure 1.6(a) is shown in Figure 1.6(b). The arcs from nodes X_j to $\langle X_j, x_j \rangle$ in an AND/OR search tree are annotated by weights that are derived from the cost functions in $\mathbf{F}$ as follows.

Definition 1.11 (**Arc weight**, [23])**.** *The* weight $w(X_j, x_j)$ of the arc $(X_j, \langle X_j, x_j \rangle)$ *is the combination (i.e., sum for WCSP and product for MPE) of all the functions, whose scope* *includes X_j and which are fully assigned by values specified along the path from root to the node to $\langle X_j, x_j \rangle$.*

Theorem 1.3. *[23] Given a pseudo-tree $\mathcal{T}$ of a graphical model having height h, the size of the AND/OR search tree based on $\mathcal{T}$ and the time complexity of an algorithm exploring it are bounded by $O(n \cdot k^h)$, where n is the number of variables and k bounds their domain size.*

Definition 1.12 (**Context**, [23])**.** *Given the primal graph $G = (V, E)$ of a graphical model $\mathcal{M}$ and a corresponding pseudo-tree $\mathcal{T}$, the* context *of a node X_j (referred to originally as* OR context*) in $\mathcal{T}$ is the set of the ancestors of X_j in $\mathcal{T}$ that have connections in G to X_j or its descendants.*

In other words, the context of a variable X_j is a set of variables, for which any partial instantiation separates the subproblem rooted at X_j from the rest of the network. When talking about the context of a subproblem, we imply the context of the subproblem's root node.

Definition 1.13 (**Context-minimal AND/OR search graph**, [23])**.** *A* context-minimal AND/OR search graph*, denoted $C_{\mathcal{T}}$, is obtained from an AND/OR search tree by merging all the identical subproblems that have the same context.*

Theorem 1.4. *[23] Given a graphical model $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \bigotimes \rangle$ with a primal graph G, whose induced width along the pseudo-tree $\mathcal{T}$ is w^*, the size of a context-minimal AND/OR search graph is $O(n \cdot k^{w^*})$.*

Definition 1.14 (**Solution tree**, [65])**.** *A* solution tree *T of a context-minimal AND/OR search graph $C_{\mathcal{T}}$ is a subtree such that: (1) it contains the root node of $C_{\mathcal{T}}$; (2) if an internal AND node n is in T, then all its children are in T; (3) if an internal OR node n is in T, then exactly one of its children is in T; (4) every* tip node *in T (i.e., node with no children, also known as* leaf node*) is a terminal node, namely it has no children in $C_{\mathcal{T}}$.*

Definition 1.15 (Cost of a solution tree, [65]**).** *The* cost of a solution tree *is the product (for MPE task) or sum (for WCSP) of the weights associated with its arcs (not to be confused with computational cost of the tree construction, namely its time and space complexity).*

Each node n in $C_{\mathcal{T}}$ is associated with a *value* $v(n)$ capturing the optimal solution cost of the conditioned subproblem rooted in n. Assuming an MPE/MAP problem, it was shown that $v(n)$ can be computed recursively based on the values of n's successors: OR nodes by maximization and AND nodes by multiplication. In the case of WCSPs, $v(n)$ for OR and AND nodes is computed as a function of their child nodes by minimization and summation, respectively [23].

We next provide an overview of a depth-first branch and bound and a best-first search algorithms that explore AND/OR search spaces [69, 68, 76]. These schemes use heuristics generated either by the mini-bucket elimination scheme (see Section 1.2.4.4 for details) or through soft arc-consistency schemes [68, 69, 83, 16] or their composite [48]. As is customary in the heuristic search literature, when defining algorithms we assume without loss of generality a minimization task (i.e., min-sum optimization problem).

■ 1.2.4.2 AND/OR Best-First Search

The state-of-the-art version of A* for the AND/OR search space for graphical models is the AND/OR Best-First algorithm (AOBF) [69]. AOBF is a variant of AO* [72] that explores the AND/OR context-minimal search graph.

AOBF (Algorithm 3) maintains the explicated part, denoted $\mathcal{G}$, of the context-minimal AND/OR search graph $C_{\mathcal{T}}$, namely all the nodes of $C_{\mathcal{T}}$ that AOBF generated so far and the edges between them. It also keeps track of the current best partial solution tree T^*. AOBF interleaves iteratively a top-down node expansion step (lines 4-16), selecting a non-terminal tip node of T^* and generating its children in the explored search graph $\mathcal{G}$, with a bottom-

Algorithm 3: AOBF(h_i) exploring AND/OR search tree [69]

```
   Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$, pseudo-tree $\mathcal{T}$ rooted in $X_1$, heuristic $h_i$ calculated
          with i-bound $i$;
   Output: Optimal solution to $\mathcal{M}$
1  create root OR node $s$ labelled by $X_1$ and let $\mathcal{G}$ (explored search graph) = $\{s\}$;
2  initialize $v(s) = h_i(s)$ and best partial solution tree $T^*$ to $\mathcal{G}$;
3  while $s$ is not SOLVED do
4      select non-terminal tip node $n$ in $T^*$. If there is no such node then exit;
       // expand node $n$
5      if $n = X_j$ is OR then
6          forall the $x_j \in D(X_j)$ do
7              create AND child $n' = \langle X_j, x_j \rangle$;
8              if $n'$ is TERMINAL then
9                  mark $n'$ SOLVED;
10             $succ(n) \leftarrow succ(n) \cup n'$;
11     else if $n = \langle X_j, x_j \rangle$ is AND then
12         forall the successor $X_k$ of $X_j$ in $\mathcal{T}$ do
13             create OR child $n' = X_k$;
14             $succ(n) \leftarrow succ(n) \cup n'$;
15     initialize $v(n') = h_i(n')$ for all new nodes;
16     add new nodes to the explored search space graph $\mathcal{G} \leftarrow \mathcal{G} \cup succ(n)$;
       // update $n$ and its AND and OR ancestors in $\mathcal{G}$, bottom-up
17     repeat
18         if $n$ is OR node then
19             $v(n) = \min_{k \in succ(n)}(w(n,k) + v(k))$;
20             mark the best successor $k$ of OR node $n$, such that $k = \arg\min_{k \in succ(n)}(w(n,k) + v(k))$
               (maintaining previously marked successor if still the best);
21             mark $n$ as SOLVED if its best marked successor is solved;
22         else if $n$ is AND node then
23             $v(n) = \sum_{k \in succ(n)} v(k)$;
24             mark all arcs to the successors;
25             mark $n$ as SOLVED if all its children are SOLVED;
26         $n \leftarrow p$; //$p$ is a parent of $n$ in $\mathcal{G}$
27     until $n$ is not root node $s$;
28     recompute $T^*$ by following marked arcs from the root $s$;
29 return $\langle v(s), T^* \rangle$;
```

up cost revision step (lines 17-27), updating the values of the internal nodes based on the children's values. If a newly generated child node is terminal, it is marked solved (lines 8-9). During the bottom-up phase the OR nodes having at least one solved child and the AND nodes having all children solved are also marked as solved (lines 21 and 25). AOBF also marks the arc to the best AND child of an OR node, through which the minimum is achieved (line 20). After the bottom-up step, a new best partial solution tree T^* is recomputed (line

28). AOBF terminates when the root node is marked solved. For admissible heuristic at termination T^* is the optimal solution tree with the cost $v(s)$, where s is the root node of the search space.

Theorem 1.5. *[69] The AND/OR Best-First search algorithm traversing the context-minimal AND/OR graph has the time and space complexity of $O(nk^{w^*})$, where n is the number of variable in the problem, w^* is the induced width along the pseudo-tree and k bounds the domain size.*

■ 1.2.4.3 AND/OR Depth-First Branch and Bound

The AND/OR Branch and Bound [68] algorithm traverses the context-minimal AND/OR graph in a *depth-first* rather than best-first manner, while keeping track of the current upper bound on the minimal solution cost.

The algorithm (Algorithm 4) interleaves forward node expansion (lines 4-17) with a backward cost revision (or propagation) step (lines 19-29) that updates node values (capturing the current best solution to the subproblem rooted at each node), until search terminates and the optimal solution is found. A node n is pruned (lines 12-13) if the current upper bound on the best solution rooted in n is higher than the heuristic lower bound, computed recursively using the procedure described in Algorithm 5.

Theorem 1.6. *[66] The worst case time and space complexity of AND/OR Branch and Bound exploring a context-minimal AND/OR search graph of a graphical model with n variables, maximum domain size k, and an induced width w^* along the pseudo-tree $\mathcal{T}$ is $O(nk^{w^*})$.*

In practice AOBB is likely to expand more nodes than AOBF using the same heuristic, but empirical performance of AOBB depends heavily on the order, in which the solutions are encountered, namely on how quickly the algorithm finds an optimal solution, that it will use as an upper bound for pruning.

Algorithm 4: AOBB(h_i) exploring AND/OR search tree [69]

```
Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$, pseudo-tree $\mathcal{T}$ rooted at $X_1$, heuristic $h_i$;
Output: Optimal solution to $\mathcal{M}$
create root OR node $s$ labelled by $X_1$ and let stack of created but not expanded nodes be OPEN = $\{s\}$;
initialize $v(s) = \infty$ and best partial solution tree rooted in $s$ $T^*(s) = \emptyset$; $UB = \infty$;
while OPEN$\neq \emptyset$ do
    select top node $n$ on OPEN.
    //EXPAND
    if $n$ is OR node labelled $X_j$ then
        foreach $x_j \in D(X_j)$ do
            //Expand node $n$:
            add AND child $n' = \langle X_j, x_j \rangle$ to list of successors of $n$;
            initialize $v(n') = 0$, best partial solution tree rooted in $n$ $T^*(n') = \emptyset$;
    if $n$ is AND node labelled $\langle X_j, x_j \rangle$ then
        foreach OR ancestor $p$ of $n$ do
            recursively evaluate the cost of the partial solution tree rooted in $p$, based on heuristic $h_i$, assign its cost
            to $f(p)$; // see evalPartialSolutionTree($T_n^*$, $h_i(n)$) in Algorithm 5
            if evaluated partial solution is not better than current upper bound at $p$ (e.g., $f(p) \geq v(p)$ for
            minimization) then
                prune the subtree below the current tip node $n$;
            else
                foreach successor $X_k$ of $X_j \in \mathcal{T}$ do
                    add OR child $n' = X_k$ to list of successors of $n$;
                    initialize $v(n') = \infty$, best partial solution tree rooted in $n$ $T^*(n') = \emptyset$;
    add successors of $n$ on top of OPEN;
    //PROPAGATE
    //Only propagate if all children are evaluated and the final $v$ are determined
    while list of successors of node $n$ is empty do
        if node $n$ is the root node then
            return solution: $v(n), T^*(n)$ ;
        else
            //update ancestors of $n$, AND and OR nodes $p$, bottom up:
            if $p$ is AND node then
                $v(p) = v(p) + v(n)$, $T^*(p) = T^*(p) \cup T^*(n)$;
            else if $p$ is OR node then
                $v_{new}(p) = w(p, n) + v(n)$;
                if the new value of better than the old one (e.g., $v_{new}(p) > (w(p, n) + v(n))$ for minimization then
                    $v(p) = v_{new}(p)$, $T^*(p) = T^*(p) \cup \langle x_j, X_j \rangle$;
        remove $n$ from the list of successors of $p$;
        move one level up: $n \leftarrow p$;
```

Breadth Rotating AND/OR Branch and Bound (BRAOBB).

AOBB, though shown to be a powerful search scheme for graphical models, however, lacks a proper anytime behavior: at each AND node all but one independent child subproblems have to be solved completely, before the last one is even considered. To remedy this the *Breadth-Rotating AND/OR Branch and Bound* algorithm has been introduced recently [76]. The basic idea is to rotate through different subproblems in a breadth-first manner. The

Algorithm 5: Recursive computation of the heuristic evaluation function [69]

```
function evalPartialSolutionTree($T'_n$, $h(n)$)
Input: Partial solution subtree $T'_n$ rooted at node $n$, heuristic function $h(n)$;
Output: Heuristic evaluation function $f(T'_n)$;
if $succ(n) == \emptyset$ then
    return $h(n)$;
else
    if $n$ is an AND node then
        let $k_1, \dots, k_l$ be the OR children of $n$;
        return $\sum_{j=1}^{l}$ evalPartialSolutionTree($T'_{k_j}$, $h(k_j)$);
    else if $n$ is an OR node then
        let $k$ be the AND child of $n$;
        return $c(n, k)$ + evalPartialSolutionTree($T'_k$, $h(k)$);
```

concepts is illustrated in Figure 1.7. In Figure 1.7(a) the two subproblems on the left have been solved completely before the subproblem on the right was even considered. In Figure 1.7(b) the initial solutions to the three independent subproblems are found in parallel. As an input parameter the algorithm has a threshold z, defining the limit on the number of nodes expanded before switching to the next subproblem.

Algorithm 6 shows the pseudo-code of BRAOBB. As mentioned above, the main difference between this algorithm and AOBB lies in the rotation of the subproblems. The subproblems are added into a global queue ($GLOBAL$), that insures that they are processed in a breadth-first manner. Each individual subproblem is explored as usual in a depth-first manner, using the stack $LOCAL$ for the current subproblem.

In the beginning there is only one subproblem, rooted in the first variable X_1 in the GLOBAL queue (line 3). As long as there are still subproblems in the $GLOBAL$ queue, the next subproblem is taken from the queue and put on the $LOCAL$ stack (line 6). The subproblem is then processed (lines 7-26) until either $LOCAL$ becomes empty, i.e., the current subproblem is fully solved, or the number of nodes expanded reaches the threshold z, or the current subproblem decomposes further. In the latter case the new subproblems are added to queue GLOBAL for subsequent processing (lines 27-28).

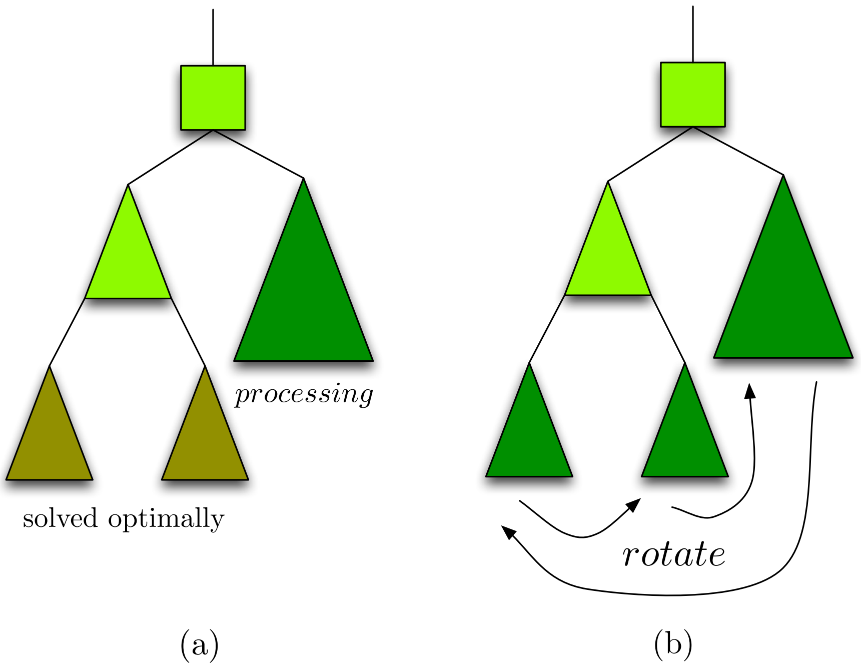


Figure 1.7: Illustration of subproblem processing: (a) default by AOBB, (b) subproblem rotation by BRAOBB

Empirically, BRAOBB was shown to find the first suboptimal solution significantly faster than plain AOBB [76].

■ 1.2.4.4 Mini-Bucket Heuristics

The AND/OR search algorithms presented (AOBF, AOBB and BRAOBB) most often use the *mini-bucket* (also known as *MBE*) *heuristic* $h(n)$, formulated for OR search spaces by Kask [50, 51, 53] and extended for AND/OR search by Marinescu [66].

As shown in [66], the intermediate bucket functions generated by MBE(i) underestimate the optimal solution cost to subproblems rooted in each node of the AND/OR search graph (assuming a minimization query), just as the overall bound returned by MBE(i) is a lower bound on the overall solution. These intermediate functions can therefore be used to derive a heuristic function that is admissible.

```
Algorithm 6: BRAOBB($h_i$)
Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$, pseudo-tree $\mathcal{T}$ rooted at $X_1$; heuristic $h_i$, rotation threshold $Z$;
Output: Optimal solution to $\mathcal{M}$
Create root OR node $s$ labelled by $X_1$ (generate root subproblem)
Initialize $UB = \infty$
$GLOBAL \leftarrow$[ROOT] //put the root subproblem into queue
while $GLOBAL \neq \emptyset$ do
    //next subproblem
    for $z$ from 1 to $Z$ or until LOCAL is empty or until childSubprom(LOCAL)$\neq \emptyset$ do
        $n \leftarrow$top($LOCAL$);
        //next node in subproblem
        // Expand node $n$:
        if $n$ is OR node labelled $X_j$ then
            foreach $x_j \in D(X_j)$ do
                initialize AND child $n' = \langle X_j, x_j \rangle$ as in AOBB (line 7);
                add it on top of $LOCAL$;
        if $n$ is AND node labelled $\langle X_j, x_j \rangle$ then
            foreach OR ancestor $m$ of $n$ do
                evaluate the heuristic and prune like in AOBB (lines 10-13);
                if no pruning happened then
                    $Y_1, \ldots, Y_k \leftarrow$ children of $n$ in the pseudo-tree;
                    generate OR children of $\langle Y_1 \rangle, \ldots, \langle Y_k \rangle$ ;
                    if $k$=1 then
                        //if there is no decomposition
                        push $Y_1$ to top of $LOCAL$;
                    else if $k > 1$ then
                        //problem decomposition
                        for $r \leftarrow 1$ to $k$ do
                            $NEW \leftarrow \{\langle Y_r \rangle\}$;
                            //new child subproblem
                            push $NEW$ to the back of $GLOBAL$;
    if children(n)$\neq \emptyset$ then
        //if $n$ is a leaf
        do propagation as in AOBB (lines 18-30);
        //upward in search space
if $LOCAL \neq \emptyset$ then
    //subproblem is not yet solved
    push LOCAL to the end of GLOBAL;
return value(root) //root node has optimal solution
```

Recently more advanced heuristics for AND/OR search, such as mini-bucket elimination with max-marginal-matching and Joint Graph Linear Programming have been proposed [48]. We will discuss them and evaluate their performance in Chapters 5.

Having established the necessary background, we will now turn to the main part of the thesis, presenting our contributions.

Chapter 2

Bucket Elimination for *M*-best Optimization Task

■ 2.1 Introduction[1]

The usual aim of combinatorial optimization is to find an optimal solution, minimum or maximum, of an objective function. However, in many applications it is desirable to obtain not just a single optimal solution, but a set of the first m best solutions for some integer m. We are motivated by many real-life domains, in which such task arises. For instance, the problem of finding the most likely haplotype in a pedigree can be presented as finding the most probable assignment in a Bayesian network that encodes the genetic information [34]. In practice the data is often corrupted or missing, which makes the single optimal solution unreliable. It is possible to increase the confidence in the answer by finding a set of m best solutions and then choosing the final solution with an expert help or by obtaining additional genetic data. More examples of the m-best task arise in procurement auction problems and in probabilistic expert systems, where certain constraints often cannot be directly incorporated into the model, either because they make the problem infeasibly complex or they are too

[1] Part of this work has already been published in Natalia Flerova, Emma Rollon, and Rina Dechter. ”Bucket and mini-bucket Schemes for M Best Solutions over Graphical Models”, Special Issue of Lecture Notes in Computer Science: Graph structures for knowledge representation and reasoning, 2011.

vague to formalize (e.g., idiosyncratic preferences of a human user). Thus in such domains it may be more practical to first find several good solutions to a relaxed problem and then pick the one that satisfies all additional constraints in a post-processing manner. Additionally, sometimes a set of diverse assignments with approximately the same cost is required, as in reliable communication network design. Finally, in the context of a summation problem, such as probability of evidence or finding the partition function, an approximation can be derived by summing over the m most likely tuples, though we do not expect this bound to be tight in practice.

The problem of finding the m best solutions has been well studied. One of the earliest and most influential works belongs to Lawler [61]. He provided a general scheme that extends any optimization algorithm to the m-best task. The idea is to compute the next best solution successively by finding a single optimal solution for a slightly different reformulation of the original problem that excludes the solutions generated so far. This approach has been extended and improved over the years and is still one of the primary strategies for finding the m best solutions. Other approaches are more direct, trying to avoid the repeated computation inherent to Lawler's scheme. Two earlier works, that are most relevant and provide the highest challenge to our work, are by Nilsson [73] and Aljazzar, et al., [2].

- Nilsson [73] proposed a junction-tree-based message-passing scheme that iteratively finds the m best solutions. He claimed that it has the best runtime complexity among m-best schemes for graphical models. Our analysis (Section 3.5) shows that indeed Nilsson's scheme has the second best worst case time complexity after our algorithm BE+m-BF (Section 3.4.3). However, in practice this scheme is often not feasible for problems having a large induced width.

- Aljazzar, et al., [2] proposed an algorithm called K*, an A* search-style scheme for finding the k shortest paths, that is interleaved with breadth-first search. They used a very specialized data structure, and it is unclear if this approach can be straightforwardly extended to graphical models, a point that we will leave to future work.

One of the popular approximate approaches to solving optimization problems is based on an LP-relaxation of the problem [99]. The m-best extension of this approach [37] does not guarantee exact solutions, but is quite efficient in practice. We will discuss these and other related works further in Chapter 3, in Section 3.5.

Our main focus lies in optimization in the context of graphical models, such as Bayesian networks, Markov networks and constraint networks. However, some of the algorithms developed can be used for more general tasks, such as finding m shortest paths in a graph. Various graph-exploiting algorithms for solving optimization tasks over graphical models were proposed in the past few decades. Such algorithms are often characterized as being either of *inference* type (e.g., message-passing schemes, variable elimination) or of *search* type (e.g., AND/OR search or recursive-conditioning). In this chapter we limit our treatment to the class of inference schemes, as represented by the bucket elimination algorithm (BE) [19]. The discussion of search schemes is deferred till Chapter 3.

Our contribution. We will show that the extension of the bucket elimination to compute the m-best solutions can be achieved by a relatively simple modification of its underlying *combination* and *marginalization* operators [19]. In optimization tasks over probabilistic graphical models the combination operator is a product, while the marginalization operator is maximization or minimization (we assume maximization here). We will show that the extension of the bucket elimination algorithm for the m-best solutions can be facilitated by representing functions as vector functions, by defining the combination operator to be a product between vector functions and the marginalization operator as *m-sorting* (rather than maximization). Applying these modifications yielding the bucket-elimination algorithm elim-m-opt is described within the framework of semirings [85, 1, 56, 7]. This unifying formulation ensures the soundness and completeness of any algorithm applied to any problem that fits into the framework and in particular it implies that elim-m-opt solves the m-best optimization task.

We show that the worst-case complexity of our algorithm increases by a factor of $O(m\cdot\log(m\cdot deg))$ over finding a single best solution, where deg is the highest degree of the bucket-tree that underlies the computation. This yields an overall complexity of $O(m\cdot n\cdot k^{w^*}\cdot\log(m\cdot deg))$ when m is the number of solutions, n is the number of variables in the problem, k bounds the domain size of each variable and w^* bounds the induced-width of the graphical model.

Since the bucket elimination scheme can be approximated by the relaxation and bounding scheme of mini-bucket elimination (MBE) [25], we can extend elim-m-opt straightforwardly to a mini-bucket scheme mbe-m-opt, which computes a bound on each of the m best solutions. More significantly, we also show that the m bounds computed by mbe-m-opt can be used to tighten the bound on the first best solution, since both are generated from the same relaxed problem. In particular, it can facilitate a scheme for tightening any heuristic generation scheme.

The formulation of the m-best problem using semiring framework makes a variety of schemes immediately applicable to this task, such as the Generalized Distributive Law [1] and Iterative Join-Graph Propagation [22].

The remainder of the chapter is organized in the following manner. In Section 2.2 we give the background on the semirings framework and more formally re-define in the context of semirings the combination and marginalization operators, as well as the reasoning task over graphical models, previously introduced in Section 1.2.1.2. We also axiomatically state the correctness of the bucket elimination algorithm, based on the results by Shenon and Shafer [88, 87]. Section 2.3 presents the formulation of the m-best reasoning task in the semiring framework. In Section 2.4 we describe the extension of bucket elimination algorithm to the

m-best. Section 2.5 is devoted to the extension of mini-bucket elimination to the m-best task. Section 2.6 presents empirical evaluation. Section 2.7.

■ 2.2 Preliminaries and Background

Before moving on to our main contributions, in this section we give more formal definitions of some graphical models concepts previously presented in Section 1.2.1.2 and axiomatically describe the properties of the algorithms for solving the reasoning tasks over graphical models. Most of the results are based on the works of Shenon and Shafer [88, 87] and Bistarelli, et al., [8]. Though some of the definitions included in this section may seem familiar from the previous chapter, the mathematically formal formulations presented here are crucial for proving the correctness of the new m-best algorithm we introduce in Section 2.4.

As before, let $\mathbf{X} = \{X_1, \ldots, X_n\}$ be an ordered set of variables and $\mathbf{D} = \{\mathbf{D}_1, \ldots, \mathbf{D}_n\}$ an ordered set of domains. Domain $\mathbf{D}_j$ is a finite set of potential values for X_j. Let us denote the assignment (i.e., instantiation) of variable X_j with $x_j \in \mathbf{D}_j$ as $X_j = x_j$. A *tuple* is an ordered set of assignments to different variables $(X_1 = x_1, \ldots, X_k = x_k)$. A *complete assignment* is an assignment to all variables in $\mathbf{X}$. Let t and s be two tuples having the same instantiations to the common variables. Their *join*, noted $t \cdot s$, is a new tuple which contains the assignments of both t and s. We use the symbol $\cdot$ for both join and multiplication, however, the context usually makes the meaning unambiguous.

Definition 2.1 (**Field, valuations**). *A* field *is an algebraic structure with notions of addition, subtraction, multiplication, and division, satisfying certain axioms. A* valuation *is a function on the elements of a field that provides a measure of size or multiplicity of elements of the field.*

Alternatively, we can define valuations the following way. Let the scope of a function f be

$\mathbf{Y}$. For a subset of variables $\mathbf{Y}$, we denote by $\mathbf{D_Y}$ the set of tuples over $\mathbf{Y}$, namely the set of all possible assignments to variables in $\mathbf{Y}$. Let $f : \mathbf{D_Y} \rightarrow \mathbf{A}$ be a function having scope $\mathbf{Y}$, $\mathbf{A}$ is a set of valuations. When the correspondence between valuations and variable sets can be ambiguous, we more specifically talk about *valuations on* $\boldsymbol{Y}$ denoted $\mathbf{A_Y}$. Typical ranges of valuations $\mathbf{A}$ are natural, real and boolean numbers. For example, in the case of Bayesian networks a valuation is a set of non-negative, real-valued numbers.

For each set of valuations $\mathbf{A}$ there can be at most one valuation $\zeta \in \mathbf{A}$ called a *zero valuation.* In case of a Bayesian network the zero valuation is such valuation ζ that the values of the function $f : \mathbf{D_Y} \rightarrow \zeta$ are zero for all configurations of $\mathbf{Y}$. For a subset of variables $\mathbf{Y}$ and function $f : \mathbf{D_Y} \rightarrow \mathbf{A}$ a subset of valuations $\mathcal{P} \in \mathbf{A}$ is called *proper valuations* if $\mathcal{P} = \mathbf{A} \setminus \zeta$, where ζ is a zero valuation. The notion of proper valuations is important, as it will enable us to define combinability of valuations. In the probability case, a valuation A is said to be proper if the values of the function $\mathbf{D_Y} \rightarrow \mathbf{A}$ are not zero for all configurations of $\mathbf{Y}$.

We assume two binary operations over valuations: $\otimes : \mathbf{A} \times \mathbf{A} \rightarrow \mathbf{A}$ called combination (or multiplication) and $\oplus : \mathbf{A} \times \mathbf{A} \rightarrow \mathbf{A}$ called addition or marginalization. Both operators are associative and commutative.

Following [88] we define these operators in the following way:

Definition 2.2 (Combination). *We assume there is a mapping* $\otimes : \boldsymbol{A} \times \boldsymbol{A} \rightarrow \boldsymbol{A}$*, called* combination*, such that:*

1. *If* $G_{\boldsymbol{Y}}$ *and* $H_{\boldsymbol{Z}}$ *are valuations on* $\boldsymbol{Y}$ *and* $\boldsymbol{Z}$*, then* $G_{\boldsymbol{Y}} \otimes H_{\boldsymbol{Z}}$ *is a valuation on* $\boldsymbol{Y} \cup \boldsymbol{Z}$*;*
2. *If either* $G_{\boldsymbol{Y}}$ *or* $H_{\boldsymbol{Z}}$ *is not a proper valuation, then* $G_{\boldsymbol{Y}} \otimes H_{\boldsymbol{Z}}$ *is not a proper valuation;*
3. *If* $G_{\boldsymbol{Y}}$ *and* $H_{\boldsymbol{Z}}$ *are both proper valuations, then* $G_{\boldsymbol{Y}} \otimes H_{\boldsymbol{Z}}$ *may or may not be a proper valuation.*

If $G_{\boldsymbol{Y}} \otimes H_{\boldsymbol{Z}}$ is not a proper valuation, then we shall say that $G_{\boldsymbol{Y}}$ and $H_{\boldsymbol{Z}}$ are not combinable. If $G_{\boldsymbol{Y}} \otimes H_{\boldsymbol{Z}}$ is a proper valuation, then we shall say that $G_{\boldsymbol{Y}}$ and $H_{\boldsymbol{Z}}$ are combinable and that $G_{\boldsymbol{Y}} \otimes H_{\boldsymbol{Z}}$ is the combination of $G_{\boldsymbol{Y}}$ and $H_{\boldsymbol{Z}}$.

As was pointed out in [88], intuitively, combination corresponds to aggregation. If $G_{\mathbf{Y}}$ and $H_{\mathbf{Z}}$ represent information about variable sets $\mathbf{Y}$ and $\mathbf{Z}$, respectively, then $G_{\mathbf{Y}} \otimes H_{\mathbf{Z}}$ represents the aggregated information for variables in $\mathbf{Y} \cup \mathbf{Z}$. In the Bayesian networks case, combination corresponds to multiplication.

Definition 2.3 (**Marginalization** or **addition**). *We assume that for each subset of variables $\boldsymbol{Y}$ and subset $\boldsymbol{Z} \subseteq \boldsymbol{Y}$ there is a mapping $\oplus_{\boldsymbol{Z}} : \boldsymbol{A} \times \boldsymbol{A} \to \boldsymbol{A}$, called* marginalization *to $\boldsymbol{Z}$, such that*

1. *If $G_{\boldsymbol{Y}}$ is a valuation on $\boldsymbol{Y}$ and $\boldsymbol{Z} \subseteq \boldsymbol{Y}$, then $\oplus_{\boldsymbol{Z}} G_{\boldsymbol{Y}}$ is a valuation on $\boldsymbol{W}$, where $\boldsymbol{W} = \boldsymbol{Y} \setminus \boldsymbol{Z}$ is a relative complement of a set $\boldsymbol{Z}$ with respect to a set $\boldsymbol{Y}$, namely the set of elements in $\boldsymbol{Y}$, but not in $\boldsymbol{Z}$;*
2. *If $G_{\boldsymbol{Y}}$ is a proper valuation, then $\oplus_{\boldsymbol{Z}} G_{\boldsymbol{Y}}$ is a proper valuation;*
3. *If $G_{\boldsymbol{Y}}$ is not a proper valuation, then $\oplus_{\boldsymbol{Z}} G_{\boldsymbol{Y}}$ is not a proper valuation.*

Intuitively, marginalization corresponds to narrowing the focus of a valuation. If $G_{\mathbf{Y}}$ is a valuation on $\mathbf{Y}$ representing some information about variables in $\mathbf{Y}$, and $\mathbf{Z} \subseteq \mathbf{Y}$, then $\oplus_{\mathbf{Z}} G_{\mathbf{Y}}$ represents the information for variables in $\mathbf{Y}$ implied by $G_{\mathbf{Y}}$, if we disregard variables in $\mathbf{Y} \setminus \mathbf{Z}$. [88]. In the belief network case the marginalization corresponds to summation.

Definition 2.4 (**Valuation structure**). *A* valuation structure *is a triple $K = (A, \otimes, \oplus)$ such that A is an arbitrary set of valuations.*

In order to be able to formally define the reasoning task and discuss the algorithms which can solve it we extend the combination and addition operators to operate also over functions.

Definition 2.5 (Combination operator over functions). *Let $f_{\boldsymbol{S}_j} : \boldsymbol{D}_j \to \boldsymbol{A}$ and $g_{\boldsymbol{S}_p} : \boldsymbol{D}_p \to \boldsymbol{A}$ be two functions. Their* combination, *noted $f_{\boldsymbol{S}_j} \bigotimes g_{\boldsymbol{S}_p}$ relative to $\boldsymbol{A}$, is a new function $h_{\boldsymbol{S}_k}$ with scope $\boldsymbol{S}_k = \boldsymbol{S}_j \cup \boldsymbol{S}_p$, which returns for each tuple $t \in D_{\boldsymbol{S}_j \cup \boldsymbol{S}_p}$ the combination of valuations of $f_{\boldsymbol{S}_j}$ and $g_{\boldsymbol{S}_p}$. Formally,*

$$\forall t \in \boldsymbol{D}_h,\ h_k(t) = f_j(t) \otimes g_p(t)$$

In the framework of graphical models combination operator $\otimes$ is defined by enumeration as $\otimes \in \{\prod, \sum, \bowtie\}$.

Definition 2.6 (Marginalization operator over functions). *Let $f_{\boldsymbol{S}_j} : \boldsymbol{D}_j \to \boldsymbol{A}$ be a function and $\boldsymbol{W} \subseteq \boldsymbol{X}$ be a set of variables. The* marginalization *of $\boldsymbol{W}$ from $f_{\boldsymbol{S}_j}$, noted $f_{\boldsymbol{S}_j} \Downarrow_{\boldsymbol{W}}$ relative to $\boldsymbol{A}$, is a new function $h_{\boldsymbol{S}_k}$ with the scope $\boldsymbol{S}_k = \boldsymbol{S}_j \setminus \boldsymbol{W}$ that returns for each tuple $t \in D_{\boldsymbol{S}_j - \boldsymbol{W}}$ the addition of the valuations over the different extensions to $\boldsymbol{W}$. Formally,*

$$\forall t \in \boldsymbol{D}_k,\ h_{\boldsymbol{S}_k}(t) = \oplus_{t' \in \boldsymbol{D}_{\boldsymbol{W}}} f_{\boldsymbol{S}_j}(t \cdot t')$$

In the framework of graphical models marginalization operator $\Downarrow$ is defined by enumeration as $\Downarrow = \{\max, \min, \prod, \sum\}$.

Example 2.1. *For example, consider three variables X_1, X_2 and X_3 with domains $\boldsymbol{D}_1 = \boldsymbol{D}_2 = \boldsymbol{D}_3 = \{1, 2, 3\}$. Let $f(X_1, X_2) = X_1 X_2$ and $g(X_2, X_3) = 2X_2 + X_3$ be two functions. If the combination operator is product (i.e., $\cdot$), then $(f \times g)(X_1, X_2, X_3) = X_1 X_2 \cdot (2X_2 + X_3)$. If the marginalization operator is* max, *then $(f \Downarrow_{X_1})(X_2) = \max\{1X_2, 2X_2, 3X_2\} = 3X_2$.*

For completeness we re-state here the previously given definition of the graphical model and the reasoning task, keeping in mind that they assume the new, more mathematically strict, definitions of the combination and marginalization operators.

Definition 2.7 (Graphical model). *A* graphical model *is a tuple $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes \rangle$,* *where: $\boldsymbol{X} = \{X_1, \ldots, X_n\}$ is a set of variables; $\boldsymbol{D} = \{\boldsymbol{D}_1, \ldots, \boldsymbol{D}_n\}$ is the set of their finite domains of values; $\boldsymbol{A}$ is a set of valuations $(\boldsymbol{A}, \otimes, \oplus)$; $\boldsymbol{F} = \{f_1, \ldots, f_r\}$ is a set of discrete functions, where $var(f_j) \subseteq \boldsymbol{X}$ and $f_j : \boldsymbol{D}_{f_j} \rightarrow \boldsymbol{A}$; and $\bigotimes$ is the combination operator over functions (see Definition 2.5). The graphical model $\mathcal{M}$ represents the function $C(\boldsymbol{X}) = \bigotimes_{f \in \boldsymbol{F}} f$.*

Definition 2.8 (Reasoning task). *A* reasoning task *is a tuple $\mathcal{P} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes, \Downarrow \rangle$ where $\langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes \rangle$ is a graphical model and $\Downarrow$ is a marginalization operator over functions as defined in Definition 2.6. The reasoning task is to compute $F(\boldsymbol{X}) \Downarrow_{\boldsymbol{X}}$.*

For a reasoning task $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \bigotimes, \Downarrow)$ the choice of $(\mathbf{A}, \otimes, \oplus)$ determines the combination $\bigotimes$ and marginalization $\Downarrow$ operators over functions, and thus the nature of the graphical model and its reasoning task. For example, if $\mathbf{A}$ is the set of non-negative reals and $\bigotimes$ is product, the graphical model is a Markov network or a Bayesian network. If $\Downarrow$ is max, the task is to compute the Most Probable Explanation (MPE), while if $\Downarrow$ is sum, the task is to compute the Probability of the Evidence.

The correctness of the algorithmic techniques for computing a reasoning task relies on the properties of the set of valuations and the combination and marginalization operators. These properties are axiomatically described by means of an algebraic structure over $(\mathbf{A}, \otimes, \oplus)$.

In other words, reasoning tasks on graphical models can be axiomatically described. As a result, any new problem expressed in that axiomatization immediately inherits the techniques developed for the existing ones.

Two main axiomatizations proposed in the literature are the Shenoy-Shafer [88] and the semiring CSP framework [9]. In the first one, the triplet $(\mathbf{A}, \otimes, \oplus)$ is a commutative semiring, while in the second one, the triplet $(\mathbf{A}, \otimes, \oplus)$ is a so-called c-semiring, that is, a commutative semiring with $\oplus$ being idempotent. While Shenoy and Shafer focus in their work mainly on the tasks over belief networks, Bistarelli, et al., [9] propose SCSP (Semiring CSP), a general framework for constraint satisfaction problems, that provide unification for classical CSPs, fuzzy CSPs, weighted CSPs, partial constraint satisfaction, and others. The Shenoy-Shafer framework ensures the correctness of inference algorithms, while the semiring CSP framework also ensures the correctness of search algorithms within a unified definition of a search space.

We are going to use Shenoy-Shafer framework in our work and thus focus on it in this background section.

The correctness of the inference algorithms using local computations, such as bucket elimination, is possible when the combination and marginalization operators obey the following set of axioms, which we here present in the context of our operators over functions:

Axiom 2.1 (Commutativity and associativity of combination)**.** *: Let $f_j : \boldsymbol{D}_j \rightarrow \boldsymbol{A}$, $g_p : \boldsymbol{D}_p \rightarrow \boldsymbol{A}$ and $h_k : \boldsymbol{D}_k \rightarrow \boldsymbol{A}$ be three functions. Then $f_j \otimes g_p = g_p \otimes f_j$ and $f_j \otimes (g_p \otimes h_k) = (f_j \otimes g_p) \otimes h_k$.*

Axiom 2.2 (Consonance of marginalization)**.** *Let $f_j : \boldsymbol{D}_j \rightarrow \boldsymbol{A}$ be a function and $\boldsymbol{Z}$ and $\boldsymbol{W}$ be disjoint subsets of variables. Then $\Downarrow_{\boldsymbol{W}} (\Downarrow_{\boldsymbol{Z}} f_j) = \Downarrow_{\boldsymbol{W} \cup \boldsymbol{Z}} f_j$.*

Axiom 2.3 (Distributivity of marginalization over combination)**.** *Let $f_{\boldsymbol{S}_j}$ and $h_{\boldsymbol{S}_k}$ be functions over scopes $\boldsymbol{S}_j$ and $\boldsymbol{S}_k$, where $\boldsymbol{S}_j$ and $\boldsymbol{S}_k$ are disjoint subsets of variables. Then $\Downarrow_{\boldsymbol{S}_k} (f_{\boldsymbol{S}_j} \otimes h_{\boldsymbol{S}_k}) = f_{\boldsymbol{S}_j} \otimes (\Downarrow_{\boldsymbol{S}_k} h_{\boldsymbol{S}_k})$.*

Shenon and Shafer [88, 87] showed that the reasoning task $\mathcal{P} = \langle \mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \bigotimes, \Downarrow \rangle$ can be solved by the sequential marginalization over subsets of functions, if the triplet $(\mathbf{A}, \bigotimes, \Downarrow)$ obey the axioms above.

Proposition 2.1. *If a triplet $(\boldsymbol{A}, \bigotimes, \Downarrow)$ is a commutative semiring, then the combination operator $\bigotimes$ and marginalization operator satisfy the Shenon-Shafer theorems.*

Definition 2.9 (Commutative semiring). *A* commutative semiring *is a triplet* $(\boldsymbol{A}, \otimes, \oplus)$ *which satisfies the following three axioms:*

A1. *The operation $\oplus$ is associative and commutative, and there is an additive identity element called* 0 *such that $a \oplus 0 = a$ for all $a \in \boldsymbol{A}$. In other words, $(\boldsymbol{A}, \oplus)$ is a commutative monoid.*

A2. *The operation $\otimes$ is also associative and commutative, and there is a multiplicative identity element called* 1 *such that $a \otimes 1 = a$ for all $a \in \boldsymbol{A}$. In other words, $(\boldsymbol{A}, \otimes)$ is also a commutative monoid.*

A3. *$\otimes$ distributes over $\oplus$, i.e., $(a \otimes b) \oplus (a \otimes c) = a \otimes (b \oplus c)$.*

Note that the operators $\bigotimes$ (Definition 2.5) and $\Downarrow$ (Definition 2.6) in fact correspond to operators $\otimes$ and $\oplus$ forming a commutative semiring. This leads to such algorithms as Bucket Elimination [19, 54] and joint-tree tree decomposition [88].

Namely, if the triplet $(\mathbf{A}, \bigotimes, \Downarrow)$ is a commutative semiring then $F(\mathbf{X}) \Downarrow_{X_j}$ can be computed by eliminating variable X_j from the set of functions in $\mathbf{F}$ containing X_j in their scope:

$$F(\mathbf{X}) \Downarrow_{X_j} = \bigotimes_{f_{\mathbf{S}_j} \in \mathbf{F}, X_j \notin \mathbf{S}_j} f \bigotimes (\bigotimes_{f_{\mathbf{S}_j} \in \mathbf{F}, X_j \in \mathbf{S}_j} f) \Downarrow_{X_j}$$

In words, when eliminating variable X_j, the only relevant functions are the ones containing X_j in their scope. This set of functions is the bucket of X_j, denoted $\mathbf{B}_j$. Applying this principle iteratively to a given variable ordering o that we assume, without loss of generality, lexicographical (i.e., $o = \{X_1, X_2, \ldots, X_n\}$), the reasoning task $F(\mathbf{X}) \Downarrow_{\mathbf{X}}$ is computed as:

$$F(\mathbf{X}) \Downarrow_{\mathbf{X}} = (\bigotimes_{f \in \mathbf{B}_1} \cdots (\bigotimes_{f \in \mathbf{B}_{n-1}} (\bigotimes_{f \in \mathbf{B}_n} f) \Downarrow_{X_n}) \Downarrow_{X_{n-1}} \cdots) \Downarrow_{X_1}$$

Theorem 2.1 (**Correctness of BE** [87, 51, 54])**.** *If a graphical model $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes \rangle$ and a reasoning task $\mathcal{P} = \langle \mathcal{M}, \Downarrow \rangle$ satisfy the Shenoy-Shafer axioms, BE is sound and complete.*

Throughout the paper we will assume that the graphical models and reasoning tasks discussed satisfy Shenon-Shafer axioms.

■ 2.3 M-Best Reasoning Task

Having introduced the necessary notations and definitions, we will now present our first contribution of the chapter. In this section we seek to show that the new task we consider, namely finding the m-best solutions to an optimization problems over graphical models, can be formulated as a semiring, which implies immediate applicability and correctness of the existing algorithms for this task, based on Shenoy-Shaffner's axioms.

Let us first consider the usual optimization tasks defined over a set of totally ordered valuations; in other words, the reasoning tasks where the marginalization operator $\Downarrow$ is min or max and the objective is to find a single solution. Without loss of generality, in the following we assume a maximization task (i.e., $\Downarrow$ is max).

More formally, we define such optimization task the following way:

Definition 2.10 (**Optimization task**)**.** *Given a graphical model $\mathcal{M}$, its* optimization task *is $\mathcal{P} = \langle \mathcal{M}, \max \rangle$. The goal is to find a complete assignment t such that $\forall t' \in D_{\boldsymbol{X}}, C(t) \geq C(t')$. $C(t)$ is called the optimal solution.*

This definition is very intuitively extended to the m-best task:

Definition 2.11 (**$\boldsymbol{M}$-best optimization task**)**.** *Assuming a maximization task and given a graphical model $\mathcal{M}$, its m-*best optimization task* $\mathcal{P}^m$ is to find m complete assignments*

$\boldsymbol{T} = \{t_1, \ldots, t_m\}$ *such that* $C(t_1) \geq, \cdots, \geq C(t_m)$ *and* $\forall t' \in \boldsymbol{D_X} \backslash \boldsymbol{T}, C(t_m) \geq C(t')$. *The solution to* $\mathcal{P}^m$ *is the set of valuations* $\{C(t_1), \ldots, C(t_m)\}$, *called* m*-best solutions.*

■ 2.3.1 M-best Valuation Structure

Given an optimization task $\mathcal{P}$ over a graphical model $\mathcal{M}$, in order to phrase the m-best optimization task as a reasoning task over a semiring, we will now introduce a new formal definition of a reasoning task $\mathcal{P}^m$ that corresponds to the set of m best solutions of $\mathcal{M}$. We are going to achieve that by defining a new valuation structure, specified for the m-best task.

First we introduce the set of ordered m-best elements of a subset $S \subseteq \mathbf{A}$.

Definition 2.12 (**Set of ordered m-best elements**). *Let* S *be a subset of a set of valuations* $\boldsymbol{A}$. *The* set of ordered m-best elements *of* S *is* $Sorted^m\{S\} = \{s_1, \ldots, s_j\}$, *such that* $s_1 \geq s_2 \geq \ldots \geq s_j$ *where* $j = m$ *if* $|S| \geq m$ *and* $j = |S|$ *otherwise, and* $\forall s' \notin Sorted^m\{S\}, s_j \geq s'$.

While typical reasoning tasks are defined over valuations, such as real or boolean numbers, the m-best task returns an ordered subset of valuations, so the valuation structure $(\mathbf{A}, \otimes, \oplus)$ needs to be extended over sets.

Definition 2.13 (**$\boldsymbol{M}$-space**). *Let* $\boldsymbol{A}$ *be a set of valuations. The* m-space *of* $\boldsymbol{A}$, *denoted* $\boldsymbol{A}^m$, *is a set of sorted subsets of valuations from* $\boldsymbol{A}$.

The combination and addition operators over the m-space $\mathbf{A}^m$, noted $\otimes^m$ and $sort^m$ respectively, are defined as follows.

Definition 2.14 (**Combination and addition over the m-space**). *Let* $\boldsymbol{A}$ *be a set of valuations, and* $\otimes$ *and* max *be its combination and marginalization operators, respectively. Let* $S, T \in \boldsymbol{A}^m$. *Their* combination, *noted* $S \otimes^m T$, *is the set* $Sorted^m\{a \otimes b \mid a \in S, b \in T\}$, *while their* addition, *noted* $sort^m\{S, T\}$, *is the set* $Sorted^m\{S \cup T\}$.

It is easy to see that the special case of $m = 1$ corresponds to the valuation structure of an ordinary optimization task.

Proposition 2.2. *When $m = 1$, the valuation structure $(\boldsymbol{A}^m, \otimes^m, sort^m)$ is equivalent to $(\boldsymbol{A}, \otimes, \max)$.*

Now that we defined all the necessary entities, let us show that the new valuation structure over the m-space is in fact a semiring.

Theorem 2.2. *The valuation structure $(\boldsymbol{A}^m, \otimes^m, sort^m)$ is a semiring.*

The proof can be found in Appendix A.1.

It is worthwhile to see the ordering defined by the semiring $(\mathbf{A}^m, \otimes^m, sort^m)$, because it would be important in the extension of Mini-Bucket Elimination (Section 2.5). Recall that by definitions 2.12 and 2.13, given two elements $S, T \in \mathbf{A}^m$, $S \geq T$ if $S = Sorted^m\{T \cup W\}$, where $W \in \mathbf{A}^m$. We call S an *m-best bound* of T.

Definition 2.15 (*M*-best bound). *Let T, S be two sets of ordered m-best elements. S is an m-*best bound *of T iff there exists a set W, such that $S = Sorted^m\{T \cup W\}$.*

Let us illustrate the previous definition by the following example. Let $T = \{10, 6, 4\}$, $S = \{10, 7, 4\}$, and $R = \{10, 3\}$ be three sets of ordered 3-best elements. S is not a 3-best bound of T, because there is no set W such that $S = Sorted^3\{T \cup W\}$. Note that a modified set $S' = \{10, 7, 6\}$ is in fact a 3-best bound of T, since there exists a set $W = \{7\}$, such that $Sorted^3\{T \cup W\} = Sorted^3\{\{10, 6, 4\} \cup \{7\}\} = \{10, 7, 6\} = S'$. At the same time, S is a 3-best bound of R because there exists $W' = \{7, 4\}$, so that $Sorted^3\{R \cup W\} = Sorted^3\{\{10, 3\} \cup \{7, 4\}\} = \{10, 7, 4\} = S$.

h_1:

X_1	X_2	
a	a	{4,2}
a	b	{3,1}
b	a	{5}
b	b	{2}

h_2:

X_2	
a	{3,1}
b	{1}

$h_1 \otimes^m h_2$:

X_1	X_2	
a	a	{12,6}
a	b	{3,1}
b	a	{15,5}
b	b	{2}

$sort^m_{X_2}\{h_1\}$:

X_1	
a	$\{4, 3\}$
b	$\{5, 2\}$

Figure 2.1: Combination and marginalization over vector functions for $m = 2$ and $\otimes = \times$. For each pair of values of (X_1, X_2) the result of $h_1 \otimes^m h_2$ is an ordered set of size 2 obtained by choosing the 2 larger elements out of the result of pair-wise multiplication of the corresponding elements of h_1 and h_2. The result of $sort^m_{X_2}\{h_1\}$ is an ordered set containing the two larger values of function h_1 for each value of X_1.

■ 2.3.1.1 Vector Functions

In order to be able to discuss the algorithms solving the m-best, task we will need to have in our toolbox operators defined not just over valuations, but also over functions, as in Section 2.2.

We will refer to functions over the m-space $\mathbf{A}^m$ $f_j : \mathbf{D}_j \rightarrow \mathbf{A}^m$ as *vector functions.* Abusing notation, we extend the $\otimes^m$ and $sort^m$ operators to operate over vector functions similar to how operators $\otimes$ and $\oplus$ were extended to operate over scalar functions in Definition 2.5.

Definition 2.16 (Combination and marginalization over vector functions). *Let $f_j : \boldsymbol{D}_j \rightarrow \boldsymbol{A}^m$ and $g_p : \boldsymbol{D}_p \rightarrow \boldsymbol{A}^m$ be two vector functions. Their* combination*, noted $f_j \overline{\bigotimes} g_p$, is a new function with scope $\boldsymbol{S}_j \cup \boldsymbol{S}_p$, such that*

$$\forall t \in \boldsymbol{D}_{\boldsymbol{S}_j \cup \boldsymbol{S}_p}, \ (f_j \overline{\bigotimes} g_p)(t) = f_j(t) \otimes^m g_p(t)$$

Let $\boldsymbol{W} \subseteq \boldsymbol{X}$ be a set of variables. The marginalization *of f_j over $\boldsymbol{W}$, noted $\underset{\boldsymbol{W}}{sort}^m\{f_j\}$, is a new function whose scope is $\boldsymbol{S}_j \setminus \boldsymbol{W}$, such that*

$$\forall t \in D_{\boldsymbol{S}_j - \boldsymbol{W}}, \ \underset{\boldsymbol{W}}{sort}^m\{f_j\}(t) = sort^m_{t' \in D_W}\{f_j(t \cdot t')\}$$

Example 2.2. *Figure 2.1 shows the combination and marginalization over two example* *vector functions h_1 and h_2 for $m = 2$ and $\otimes = \cdot$, e.g., product.*

■ 2.3.2 M-best Optimization as a Graphical Model

Now we have necessary mathematical apparatus to formally define the m-best optimization task over a graphical models.

The m-best extension of an optimization problem $\mathcal{P}$ is a new reasoning task $\mathcal{P}^m$ that expresses the m-best task over $\mathcal{P}$.

Definition 2.17 (M-best extension of optimization task). *Let $\mathcal{P} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes, \Downarrow \rangle$ be an optimization problem defined over a semiring $(\boldsymbol{A}, \otimes, \max)$. Its m-*best extension *is a new reasoning task $\mathcal{P}^m = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}^m, \boldsymbol{F}^m, \overline{\bigotimes}, sort^m \rangle$ over semiring $(\boldsymbol{A}^m, \otimes^m, sort^m)$. Each function $f : \boldsymbol{D}_f \to \boldsymbol{A}$ in $\boldsymbol{F}$ is trivially transformed into a new vector function $f' : \boldsymbol{D}_f \to \boldsymbol{A}^m$ defined as $f'(t) = \{f(t)\}$. In words, function outcomes of f are transformed to singleton sets in f'. Then, the set $\boldsymbol{F}^m$ contains the new f' vector functions.*

The following theorem shows that the optimum of $\mathcal{P}^m$ corresponds to the set of m-best valuations of $\mathcal{P}$.

Theorem 2.3. *Let $\mathcal{P} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes, \Downarrow \rangle$ be an optimization problem defined over semiring $(A, \otimes, \max)$ and let $\{C(t_1), \ldots, C(t_m)\}$ be its m best solutions. Let $\mathcal{P}^m$ be the m-best extension of $\mathcal{P}$. Then, the optimization task $\mathcal{P}^m$ computes the set of m-best solutions of $\mathcal{P}$. Formally,*

$$sort_{\boldsymbol{X}}{}^m\{\overline{\bigotimes}_{f \in \boldsymbol{F}^m} f\} = \{C(t_1), \ldots, C(t_m)\}$$

Proof. By definition of $sort^m$,

$$sort_{\mathbf{X}}{}^m\{\overline{\bigotimes}_{f \in \mathbf{F}^m} f\} = Sorted^m\{\bigcup_{t \in \mathbf{D_X}} (\overline{\bigotimes}_{f \in \mathbf{F}^m} f(t))\}$$

By definition of $\mathbf{F}^m$,

$$sort_{\mathbf{X}}{}^m\{\overline{\bigotimes}_{f \in \mathbf{F}^m} f\} = Sorted^m\{\bigcup_{t \in \mathbf{D_X}} (\overline{\bigotimes}_{f \in \mathbf{F}}\{f(t)\})\}$$

Since all $\{f(t)\}$ are singletons, then $\{f(t)\} \otimes^m \{g(t)\} = \{f(t) \otimes g(t)\}$. Then,

$$sort_{\mathbf{X}}{}^m\{\overline{\bigotimes}_{f \in \mathbf{F}^m} f\} = Sorted^m\{\bigcup_{t \in \mathbf{D_X}} \{\bigotimes_{f \in \mathbf{F}} f(t)\}\}$$

Since C is the combination of functions f,

$$sort_{\mathbf{X}}{}^m\{\overline{\bigotimes}_{f \in \mathbf{F}^m} f\} = Sorted^m\{\bigcup_{t \in \mathbf{D_X}} \{C(t)\}\}$$

By definition of the set union,

$$sort_{\mathbf{X}}{}^m\{\overline{\bigotimes}_{f \in \mathbf{F}^m} f\} = Sorted^m\{\{C(t) \mid t \in D_{\mathbf{X}}\}\}$$

By definition of the set of ordered m-best elements,

$$sort_{\mathbf{X}}{}^m\{\overline{\bigotimes}_{f \in \mathbf{F}^m} f\} = \{C(t_1), \ldots, C(t_m)\}$$

□

Though the definition of the $sort^m$ operator and the proof of the theorem assumes maximization, it is easy to see how the same extension applies to minimization tasks. The only difference is the set of valuations selected by operator $sort^m$.

In the following we refer to $\mathcal{P}^m$ as the *m-best reasoning task*, implicitly assuming that its valuation structure is always a semiring.

2.4 Bucket Elimination for the M-Best Task

In this section we provide a formal description of the extension of bucket elimination algorithm to the m-best best task, based on the operators over the m-space defined in the previous section. We also provide algorithmic details for the operators and show through an example how the algorithm can be derived from first principles.

2.4.1 Algorithm Definition

Consider an optimization task $\mathcal{P} = \langle \mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \bigotimes, \max \rangle$. Algorithm elim-m-opt (see Algorithm 7, defined here for max-prod task) is the extension of bucket elimination to solve $\mathcal{P}^m$, i.e., the m-best extension of $\mathcal{P}$, as defined in Section 2.3.2.

First, the algorithm transforms scalar functions in $\boldsymbol{F}$ to their equivalent vector functions as described in Definition 2.17 (line 1) and partitions the functions into buckets, just like bucket elimination does (line 2). Then the algorithm processes the buckets from last to first as usual, using the two new combination and marginalization operators $\overline{\bigotimes}$ and $sort^m$, respectively (lines 3-9). Roughly, the elimination of variable X_p from a vector function will produce a new vector function $h_{X_p \rightarrow X_k}$, where X_k is the highest-index variable in the set $Scope(bucket_{X_p}) - X_p$, and $h_{X_p \rightarrow X_k}(t)$ will contain the m-best extensions of t to the eliminated variables $X_{p+1}, \ldots, X_n$ with respect to the sub-problem below the bucket variable in the bucket tree. Once all variables have been eliminated, the resulting zero-arity function h_{X_1} contains the m-best cost extensions to all variables in the problem. In other words, h_{X_1} is the solution of the problem.

Theorem 2.4 (elim-m-opt correctness). *Algorithm elim-m-opt is sound and complete*

Algorithm 7: The elim-m-opt algorithm

```
Input: An optimization task $\mathcal{P} = \langle \mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \prod, \max \rangle$; An ordering of variables $o = \{X_1, \ldots, X_n\}$
Output: A zero-arity function $h_1 : \emptyset \to \mathbf{A}^m$ containing the solution of the $m$-best optimization task
        $\mathcal{P}^m$
//Initialize
Transforms scalar functions in $\boldsymbol{F}$ to their equivalent vector functions;
Partition the functions in $\mathbf{F}$ into $bucket_{X_1}, \ldots, bucket_{X_n}$, where $bucket_{X_p}$ contains all functions
whose highest-index variable according to the ordering $o$ is $X_p$;
//Backward pass
for $p \leftarrow n$ down to 1 do
    Let $g_1, \ldots, g_r$ be the functions in $bucket_{X_p}$ (including both original functions and previously
    generated messages); let $\mathbf{S}_1, \ldots, \mathbf{S}_r$ be the scopes of functions $g_1, \ldots, g_r$;
    if $X_p$ is instantiated ($X_p = x_p$) then
        Assign $X_p = x_p$ to each $g_j$ and put each resulting function into its appropriate bucket;
    else
        Generate the message function $h_{X_p \to X_k}$: $h_{X_p \to X_k} = sort^m_{X_p} \overline{\bigotimes}_j g_j$, where $X_k$ is the
        highest-index variable in $Scope(h_{X_p \to X_k}) = \cup_{j=1}^{r} \mathbf{S}_j - X_p$;
        Add $h_{X_p \to X_k}$ to $bucket_{X_k}$;
//Forward pass
Assign a value to each variable in the ordering $o$ so that the combination of the functions in each
bucket is optimal, according to the marginalization operator $sort^m$;
return the function computed in the bucket of the first variable and the corresponding assignment
```

for finding the m best solutions over an optimization task $\mathcal{P}$.

Proof. The correctness of the algorithm follows from the formulation of the m-best optimization task as a reasoning task over a semiring (Section 2.3.2). □

There could be several ways to generate the set of m-best assignments, one of which is presented next and it uses the $argsort^m$ operator.

Definition 2.18 (*argsort*^{m} **operator**). *Operator $argsort^m_{X_j} f_k$ returns a vector function $\overline{x}_j(t)$ such that $\forall t \in D_{\boldsymbol{S}_k \setminus X_j}$, where $\langle f_k(t \cdot {x_j}^1), \ldots, f_k(t \cdot {x_j}^m) \rangle$ are the m-best valuations extending t to X_j and where ${x_j}^j$ denotes the j^{th} element of $\overline{x}_j(t)$.*

In words, $\overline{x}_j(t)$ is the vector of assignments to X_j that yields the m-best extensions to t.

We recap the main algorithmic issues and demonstrate the intuition behind the method in the next section by deriving elim-m-opt through an example. For clarity reasons, we omit the generation of actual m-best solution assignments.

2.4.2 Illustrating the Algorithm's Derivation through an Example

We have in mind the MPE (most probable explanation) task in probabilistic networks. Consider a graphical model with four variables $\{X, Y, Z, T\}$ having the following functions (for simplicity we use un-normalizes functions):

x	z	$f_1(z,x)$
0	0	2
0	1	2
1	0	5
1	1	1
2	0	4
2	1	3

y	z	$f_2(z,y)$
0	0	6
0	1	7
1	0	2
1	1	4
2	0	8
2	1	2

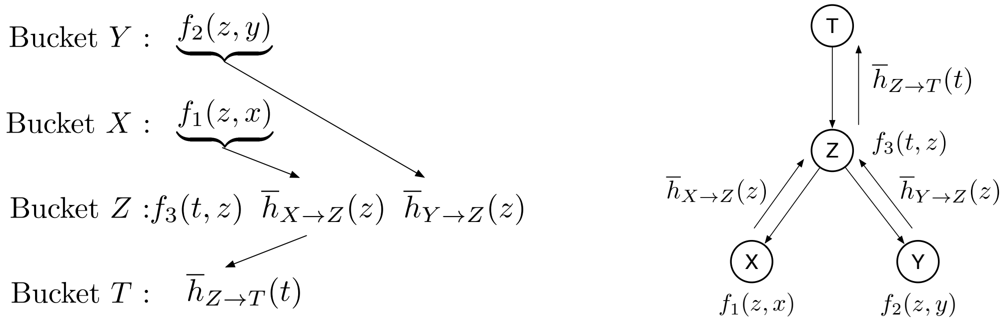
z	t	$f_3(t,z)$
0	0	1
0	1	2
1	0	4
1	1	3

Let $m = 3$. Finding the m best solutions to $P(t, z, x, y) = f_3(t, z) \cdot f_1(z, x) \cdot f_2(z, y)$ can be expressed as finding Sol, defined by:

$$Sol = \underset{t,x,z,y}{sort}^{m} \left(f_3(t, z) \cdot f_1(z, x) \cdot f_2(z, y) \right) \tag{2.1}$$

Since operator $sort^m$ is an extension of operator max, it inherits its distributive properties over multiplication. Due to this distributivity, we can apply symbolic manipulation and migrate each of the functions to the left of the $sort^m$ operator over variables that are not in its scope. In our example we rewrite as:

$$Sol = \underset{t}{sort}^{m} \underset{z}{sort}^{m} \left(f_3(t, z) \left(\underset{x}{sort}^{m} f_1(z, x) \right) \left(\underset{y}{sort}^{m} f_2(z, y) \right) \right) \tag{2.2}$$


Bucket Y : $\underbrace{f_2(z,y)}$
Bucket X : $\underbrace{f_1(z,x)}$
Bucket Z :$f_3(t,z)$ $\overline{h}_{X\to Z}(z)$ $\overline{h}_{Y\to Z}(z)$
Bucket T : $\overline{h}_{Z\to T}(t)$
T
$\overline{h}_{Z\to T}(t)$
Z
$f_3(t,z)$
$\overline{h}_{X\to Z}(z)$
$\overline{h}_{Y\to Z}(z)$
X
Y
$f_1(z,x)$
$f_2(z,y)$

(a) Messages passed between buckets

(b) Bucket-tree

Figure 2.2: Example of applying elim-m-opt algorithm

The output of $sort^m$ is a set, so in order to make (2.2) well defined, we replace the multiplication operator by the combination over vector functions, as in Definition 2.16.

$$Sol = \underset{t}{sort}^m \underset{z}{sort}^m (f_3(t,z) \overline{\bigotimes} (\underset{x}{sort}^m f_1(z,x)) \overline{\bigotimes} (\underset{y}{sort}^m f_2(z,y))) \tag{2.3}$$

BE computes (2.3) from right to left, which corresponds to the elimination ordering $o = \{T, Z, X, Y\}$. We assume the original input functions to be extended to vector functions, e.g., f_j is extended as $\overline{f}_j(t) = \{f_j(t)\}$. Figure 2.2 shows the messages passed between buckets and the bucket tree under o.

For brevity in the following we denote a bucket of variable X_j as $\mathbf{B}_{X_j}$. Bucket $\mathbf{B}_Y$ containing function $f_2(z,y)$ is processed first. The algorithm applies operator $\underset{y}{sort}^m$ to $f_2(z,y)$, generating a *message*, which is a vector function denoted by $\overline{h}_{Y\to Z}(z)$, that is placed in $\mathbf{B}_Z$. Note that this message associates each z with the vector of m-best valuations of $f_2(z,y)$. Namely,

$$\underset{y}{sort}^m f_2(z,y) = (h^1_{Y\to Z}(z), \ldots, h^j_{Y\to Z}(z), \ldots, h^m_{Y\to Z}(z)) = \overline{h}_{Y\to Z}(z) \tag{2.4}$$

where for z each $h^j_{Y\to Z}(z)$ is the j^{th} best value of $f_2(z,y)$. Similar computation is carried in $\mathbf{B}_X$ yielding $\overline{h}_{X\to Z}(z)$ which is also placed in $\mathbf{B}_Z$.

z	$\overline{h}_{X\to Z}(z)$	$\overline{x}$	$\overline{h}_{Y\to Z}(z)$	$\overline{y}$
0	{5,4,2}	{ 1, 2, 0}	{8,6,2}	{2, 0, 1}
1	{3,2,1}	{ 2, 0, 1}	{7,4,2}	{0, 1, 2}

When processing $\mathbf{B}_Z$, we compute (see Eq. 2.3):

$$\overline{h}_{Z\to T}(t) = sort_z^m[\overline{f}_3(t,z)\overline{\bigotimes}\overline{h}_{X\to Z}(z)\overline{\bigotimes}\overline{h}_{Y\to Z}(z)]$$

The result is a new vector function that has m^2 elements for each tuple (t, z) as shown below.

t	z	$f_3(t,z)\overline{\bigotimes}\overline{h}_{X\to Z}(z)\overline{\bigotimes}\overline{h}_{Y\to Z}(z)$
0	0	{40, 32, 30, 16, 24, 12, 10, 8, 4}
0	1	{84, 56, 48, 32, 28, 24, 16, 16, 8}
1	0	{80, 64, 60, 48, 32, 24, 20, 16, 8}
1	1	{63, 42, 36, 24, 21, 18, 12, 12, 6}

Applying $sort_z^m$ to the resulting combination generates the m-best elements out of those m^2 yielding message $\overline{h}_{Z\to T}(t)$ along with its variable assignments:

t	$\overline{h}_{Z\to T}(t)$	$\langle\overline{x}, \overline{y}, \overline{z}\rangle$
0	{84,56,48}	$\{\langle 2,0,1\rangle, \langle 0,0,1\rangle, \langle 2,1,1\rangle\}$
1	{80,64,63}	$\{\langle 1,2,0\rangle, \langle 2,2,0\rangle, \langle 2,0,1\rangle\}$

In Section 2.4.3 we show that it is possible to apply a more efficient procedure that would calculate at most $2m$ elements per tuple (t, z) instead.

Finally, processing the last bucket yields the vector of m best solution costs for the entire problem and the corresponding assignments: $Sol = \overline{h}_T() = \underset{t}{sort}^m \overline{h}_{Z\to T}(t)$ (see Figure 2.2).

$\overline{h}_{Z\to T}(t)$	$\langle\overline{x}, \overline{y}, \overline{z}, \overline{t}\rangle$
{84,80,64}	$\{\langle 2,0,1,0\rangle, \langle 1,2,0,1\rangle, \langle 2,2,0,1\rangle\}$

■ 2.4.3 Bucket Processing

We will next show that the messages computed in a bucket can be obtained more efficiently than through a brute-force application of $\overline{\bigotimes}$ followed by $sort^m$. Consider processing $\mathbf{B}_Z$ (see Figure 2.2(a)). A brute-force computation of

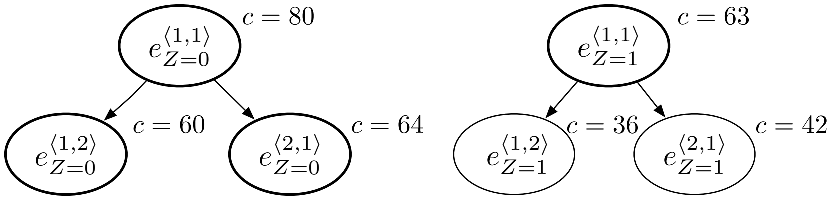
$$\overline{h}_{Z\to T}(t) = \underset{z}{sort}^m(f_3(z,t)\overline{\bigotimes}\overline{h}_{Y\to Z}(z)\overline{\bigotimes}\overline{h}_{X\to Z}(z))$$

for each t combines $f_3(z,t)$, $\overline{h}_{Y\to Z}(z)$ and $\overline{h}_{X\to Z}(z)$ for $\forall z \in D_Z$ first. This results in a vector function with scope $\{T, Z\}$ having m^2 elements that we call *candidate elements* and denote by $E(t,z)$. The second step is to apply $\underset{z}{sort}^m E(t,z)$ yielding the desired m best elements $\overline{h}_{Z\to T}(t)$.

However, since $\overline{h}_{Y\to Z}(z)$ and $\overline{h}_{X\to Z}(z)$ can be kept sorted, we can generate only a small subset of these m^2 candidates as follows. We denote by $e_z^{\langle i,j\rangle}(t)$ the candidate element obtained by the product of the scalar function value $f_3(t,z)$ with the i^{th} element of $\overline{h}_{Y\to Z}(z)$ and j^{th} element of $\overline{h}_{X\to Z}(z)$, having cost $c_z^{\langle i,j\rangle}(t) = \big(f_3(t,z)\cdot h^i_{Y\to Z}(z)\cdot h^j_{X\to Z}(z)\big)$. We would like to generate the candidates $e_z^{\langle i,j\rangle}$ in decreasing order of their costs while taking their respective indices i and j into account.

The *child elements* of $e_z^{\langle i,j\rangle}(t)$, denoted *children*$(e_z^{\langle i,j\rangle}(t))$ are obtained by replacing in the product either an element $h^i_{Y\to Z}(z)$ with $h^{i+1}_{Y\to Z}(z)$, or $h^j_{X\to Z}(z)$ with $h^{j+1}_{X\to Z}(z)$, but not both.

This leads to a forest-like search graph whose nodes are the candidate elements, where each search subspace corresponds to a different value of z denoted by $G_{Z=z}$ and rooted in $e_{Z=z}^{\langle 1,1\rangle}(t)$. Clearly, the cost along any path from a node to its descendants is non-increasing. It is easy to see that the m best elements $\overline{h}_{Z\to T}(t)$ can then be generated using a greedy best-first search across the forest search space $G_{Z=0}\cup G_{Z=1}$. It is easy to show that we do not need to keep more than m nodes on the OPEN list (the frontier of the search) at the same time.



$c = 80$
$e^{\langle 1,1\rangle}_{Z=0}$
$c = 60$
$e^{\langle 1,2\rangle}_{Z=0}$
$c = 64$
$e^{\langle 2,1\rangle}_{Z=0}$
$c = 63$
$e^{\langle 1,1\rangle}_{Z=1}$
$c = 36$
$e^{\langle 1,2\rangle}_{Z=1}$
$c = 42$
$e^{\langle 2,1\rangle}_{Z=1}$

Figure 2.3: The explored search space for $T = 0$ and $m = 3$. The resulting message is $\overline{h_T}(t = 1) = \{80, 64, 63\}$.

Algorithm 8: Bucket processing

```
Input: $\mathbf{B}_X$ of variable $X$ containing a set of ordered $m$-vector functions $\{\overline{h_1}(S_1, X), \cdots, \overline{h_d}(S_d, X)\}$
Output: $m$-vector function $\overline{h_X}(S)$, where $S = \cup_{j=1}^{d} S_j \setminus X$
forall the $t \in D_S$ do
    forall the $x \in D_X$ do
        $OPEN \leftarrow e^{\langle 1,\ldots,1\rangle}_{X=x}(t)$;
        Sort OPEN;
    while $j \leq m$, by +1 do
        $n \leftarrow$ first element $e^{\langle i_1,\cdots,i_d\rangle}_{X=x}(t)$ in OPEN;
        Remove $n$ from OPEN;
        $h^j_X(s) \leftarrow n$; //the $j^{th}$ element is selected $C \leftarrow$ children(n) = $\{e^{\langle i_1,\cdots,i_r+1,\cdots,i_d\rangle}_{X=x}(t) | r = 1..d\}$;
        Insert each $c \in C$ into OPEN maintaining order based on its computed value. Check for
        duplicates;
        Retain the $m$ best nodes in OPEN, discard the rest;
return Return $\overline{h_X}(S)$;
```

The general algorithm is described in Algorithm 8. The trace of the search for the elements of cost message $\overline{h}_{Z \to T}(t = 1)$ for our running example is shown in Figure 2.3.

Proposition 2.3 (complexity of bucket processing). *Given a bucket of a variable X over scope S having j functions $\{\overline{h_1}, ..., \overline{h_j}\}$ of dimension m, where m is the number of best solutions sought and k bounds the domain size, the complexity of bucket processing is $O(k^{|S|} \cdot m \cdot j \log m)$, where $|S|$ is the scope size of S.*

Proof. To generate each of the m solutions, the bucket processing routine removes the current best element from OPEN (in constant time), generates its j children and puts them on OPEN, while keeping the list sorted, which takes $O(\log(m \cdot j))$ per child node, since the maximum length of OPEN is $O(m \cdot j)$. This yields time complexity of $O((m \cdot j) \cdot log(m \cdot j))$

for all m solutions. The process needs to be repeated for each of the $O(k^{|S|})$ tuples, leading to overall complexity $O(k^{|S|} \cdot m \cdot j \log(m \cdot j))$. □

Theorem 2.5 (Complexity of elim-m-opt). *Given a graphical model $\langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \bigotimes \rangle$ having n variables, whose domain size is bounded by k, an ordering o with induced-width $w^*(o)$ and an operator $\Downarrow = max$, the time complexity of elim-m-opt is $O(nk^{w^*(o)+1} m \log m)$ and its space complexity is $O(mnk^{w^*(o)})$.*

Proof. Let deg_p be the degree of the node corresponding to the variable X_p in the bucket-tree. Each bucket $\mathbf{B}_p$ contains deg_p functions and at most $w^*(o)+1$ different variables with largest domain size k. We can express the time complexity of computing a message between two buckets as $O(k^{w^*(o)+1} m \cdot deg_p \log m)$ (Proposition 2.3), yielding the total time complexity of elim-m-opt of $O(\sum_{p=1}^{n} k^{w^*(o)+1} m \cdot deg_p \log m)$. Since $\sum_{p=1}^{n} deg_p \leq 2n$, we get the total time complexity of $O(nmk^{w^*(o)+1} \log m)$. The space complexity is dominated by the size of the messages between buckets, each containing m costs-to-go for each of $O(k^{w^*(o)})$ tuples. Having at most n such messages yields the total space complexity of $O(mnk^{w^*(o)})$. □

■ 2.5 Mini-Bucket Elimination for M-Best

We next extend the elim-m-opt to the mini-bucket scheme. We prove that the new algorithm computes an m-best bound on the set of m-best solutions of the original problem, and describe how the m-best bound can be used to tighten the bound on the best solution of an optimization task.

■ 2.5.1 The Algorithm Definition

Algorithm mbe-m-opt (Algorithm 9) is a straightforward extension of MBE to solve the m-best reasoning task, where the combination and marginalization operators are the ones

Algorithm 9: The mbe-m-opt algorithm

```
Input: An optimization task $\mathcal{P} = \langle \mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \prod, \max \rangle$; An ordering of variables $o = \{X_1, \ldots, X_n\}$;
       parameter $i$.
Output: bounds on each of the $m$-best solution costs and the corresponding assignments for the
        expanded set of variables (i.e., node duplication)
//Initialize:
Generate an ordered partition of functions $\overline{f}(t) = \{f(t)\}$ into buckets $\mathbf{B}_{X_1}, \ldots, \mathbf{B}_{X_n}$, where $\mathbf{B}_{X_p}$
contains all the functions whose highest variable in their scope is $X_p$ along $o$;
//Backward pass:
forall the $p \leftarrow n$ down to 1 do
    //Processing bucket $B_{X_p}$
    Let $g_1, \ldots, g_r$ be the functions in $\mathbf{B}_{X_p}$ (including both original functions and previously
    generated messages); let $\mathbf{S}_1, \ldots, \mathbf{S}_r$ be the scopes of functions $g_1, \ldots, g_r$;
    if $X_p$ is instantiated $(X_p = x_p)$ then
        Assign $X_p = x_p$ to each $g_j$ and put each resulting function into its appropriate bucket;
    else
        Partition functions in $\mathbf{B}_{X_p}$ into mini-buckets, generating the partitioning
        $Q_{X_p} = \{q_p^1, \ldots, q_p^l\}$, where each $q_p^t \in Q_{X_p}$ has no more than $i+1$ variables;
        foreach $q_p^t \in Q_{X_p}$ do
            Generate the message function $h^t_{X_p \to X_k} = sort^m_{X_p} \overline{\bigotimes}_j g^t_j$, where $g^t_j \in q^t_p$ and $X_k$ is the
            highest-index variable in $Scope(h^t_{X_p \to X_k}) = \cup_j Scope(g^t_j) - X_p$;
            Generate assignment using duplicate variables for each mini-bucket:
            $\overline{x}^t_{X_p \to X_k} = argsort^m_{X_p}(\overline{\bigotimes}_j g^t_j$, concatenate with relevant elements of the previously
            generated assignment messages;
            Add $h^t_{X_p \to X_k}$ to $bucket_{X_k}$;
return The set of all buckets, and the vector of m-best costs bounds in the first bucket;
```

defined over vector functions. The input of the algorithm is an optimization task P, and its output is a collection of bounds (i.e., an m-best bound, see Definition 2.15) on the m best solutions of P.

Theorem 2.6 (mbe-m-opt bound). *Given a maximization task $\mathcal{P}$, mbe-m-opt computes an m-best upper bound on the m-best optimization task $\mathcal{P}^m$.*

Proof. Let $C^m = \{C(t_1), \ldots, C(t_m)\}$ be the m-best solutions of P. Let $\tilde{\mathcal{P}}$ be the relaxed version of $\mathcal{P}$ solved by mbe-m-opt, and let $\tilde{C}^m = \{\tilde{C}(t'_1), \ldots, \tilde{C}(t'_m)\}$ be its m-best solutions. We prove that (1) $\tilde{C}^m$ is an m-best upper bound of C^m; and (2) mbe-m-opt($\mathcal{P}$) computes $\tilde{C}^m$.

1. It is clear that $\tilde{C}^m = Sorted^m\{C^m \cup W\}$, where W is the set of solutions for which duplicated variables are assigned different domain values. Therefore, by definition, $\tilde{C}^m$ is an m-best bound of C^m.

2. As shown in Theorem 2.4, elim-m-opt($\tilde{\mathcal{P}}$) computes $\tilde{C}^m$, and by definition of mini-bucket elimination, elim-m-opt($\tilde{\mathcal{P}}$) = mbe-m-opt($\mathcal{P}$). Therefore, mbe-m-opt(P) computes $\tilde{C}^m$.

□

Theorem 2.7 (mbe-m-opt complexity). *Given a maximization task $\mathcal{P}$ and an i-bound i such that $i \leq w^*(o)$, where $w^*(o)$ is the induced width of the ordering used, the time and space complexity of mbe-m-opt is $O(mnk^{i+1}\log(m))$ and $O(mnk^i)$, respectively, where k is the maximum domain size and n is the number of variables.*

Proof. Given a control parameter i, each mini-bucket contains at most $i+1$ variables. Let deg_j be the number of functions in the bucket $\mathbf{B}_j$ of variable X_j, i.e., the degree of the node in the original bucket tree. Let l_j be the number of mini-buckets created from $\mathbf{B}_j$ and let mini-bucket Q_{j_p} contain deg_{j_p} functions, where $\sum_{p=1}^{l_j} deg_{j_p} = deg_j$. The time complexity of computing a message between two mini-buckets is bounded by $O(k^{i+1}m \cdot deg_{j_p} \log m)$ (Proposition 2.3) and the complexity of computing all messages in mini-buckets created out of $\mathbf{B}_j$ is $O(\sum_{p=1}^{l_j} k^{(i+1)}im \cdot deg_{j_p} \log m) = O(k^{i+1}m \cdot deg_j \log m)$. Taking into account that $\sum_{j=1}^{n} deg_j \leq 2n$, we obtain the total runtime complexity of mbe-m-opt of $\sum_{j=1}^{n} k^{i+1}m \cdot deg_{j_p} \log m) = O(nmk^{i+1}\log m)$. The space complexity is bounded by the size of the largest message. □

■ 2.5.2 Using the M-Best Bound to Tighten the First-Best Bound

Here is a simple, but quite fundamental observation: whenever upper or lower bounds are generated by solving a relaxed version of a problem, the relaxed problem's solution set contains all the solutions to the original problem. We next discuss the ramification of this observation.

Proposition 2.4. *Let $\mathcal{P}$ be an optimization problem with an optimal solution p^{opt} and let $\tilde{C} = \{\tilde{p}_1 \geq \tilde{p}_2 \geq, ..., \geq \tilde{p}_m\}$ be the m best approximate solution costs (or valuations) of the m-best task $\mathcal{P}^m$, generated by mbe-m-opt. There can be two cases: either the set $\tilde{C}$ contains the optimal solution p^{opt} or $\tilde{p}_m$ is an upper bound on p^{opt}, and this bound is as tight or tighter than all other bounds $\tilde{p}_1, ...\tilde{p}_{m-1}$.*

Proof. Let $\tilde{C}_{all} = \{\tilde{p}_1 \geq \tilde{p}_2 \geq, ..., \geq \tilde{p}_{N_1}\}$ be an ordered set of valuations of all tuples over the relaxed problem (with duplicate variables). Clearly $\tilde{C}$ is the subset of the first m elements in $\tilde{C}_{all}$. By the nature of any relaxation, $\tilde{C}_{all}$ must also contain all the probability values associated with solutions of the original problem $\mathcal{P}$ denoted by $C_{all} = \{p_1 \geq \cdots \geq p_{N_2}\}$. Let j be the first index such that $\tilde{p}_j$ coincides with $p^{opt} = p_1$. Clearly, $\forall k < j$ $\tilde{p}_k \geq p^{opt}$, with $\tilde{p}_{j-1}$ being the tightest upper bound. If $j \leq m$, then $\tilde{p}_j \in \tilde{C}$ and thus $p^{opt} \in \tilde{C}$. If $j > m$, then $\tilde{p}_m$ is the tightest upper bound on p^{opt} from the set $\tilde{C}$. In particular $\tilde{p}_m$ is tighter than the bound $\tilde{p}_1$. □

In other words, if $j \leq m$, we already have the optimal value, otherwise we can use $\tilde{p}_m$ as our better upper bound. Such tighter bounds would be useful as heuristics for search algorithm such as A*. It is essential therefore to decide efficiently, whether a bound coincides with the exact optimal cost. Luckily, the nature of the MBE relaxation supplies us with an efficient decision scheme, since, as mentioned above, it is known that an assignment in which duplicates of variables take on identical values yields an exact solution.

Benchmark	# inst	n	k	w^*
Pedigrees	12	581-1006	3-7	16-39
Binary Grids	32	144-2500	2	15-90
WCSP	56	25-1057	2-100	5-287
Mastermind	15	1220-3692	2	18-37

Table 2.1: Benchmark parameters: # inst - number of instances, n - number of variables, k - domain size, w^* - induced width.

Proposition 2.5. *Given a set of bounds produced by mbe-m-opt $\tilde{p}_1 \geq \tilde{p}_2 \geq, ... \geq \tilde{p}_m$, deciding if $\tilde{p}_j = p^{opt}$ can be done in polynomial time, more specifically in $O(nm)$ steps.*

Proof. The mbe-m-opt provides both the bounds on the m-best costs and, for each bound, a corresponding tuple maintaining assignments to duplicated variables. The first of such assignment tuples (going from the one with the largest cost to the one with the smallest), whose duplicate variables are assigned identical values, is optimal. And if no such tuple is observed, the optimal value is smaller than $\tilde{p}_m$. Since the above tests require just $O(nm)$ steps applied to m-best assignments, already obtained in polynomial time, the claim follows. □

■ 2.6 Empirical Demonstrations

■ 2.6.1 Overview and Methodology

In our experiments we used four benchmarks[2] used in UAI 2008 competition [16], all, except for binary grids, coming from real world domains:

- Pedigrees
- Binary grids
- WCSP
- Mastermind

[2] http://graphmod.ics.uci.edu/group/Repository

The **pedigrees instances ("pedigree*")** arise from the domain of genetic linkage analysis and are associated with the task of haplotyping. The haplotype is the sequence of alleles at different loci inherited by an individual from one parent, and the two haplotypes (maternal and paternal) of an individual constitute this individual's genotype. When genotypes are measured by standard procedures, the result is a list of unordered pairs of alleles, one pair for each locus. The maximum likelihood haplotype problem consists of finding a joint haplotype configuration for all members of the pedigree which maximizes the probability of data. It can be shown that given the pedigree data the haplotyping problem is equivalent to computing the most probable explanation of a Bayesian network that represents the pedigree [32].

In **binary grid networks ("50-*", "75-*"** and **"90-*")**[3] the nodes corresponding to binary variables are arranged in an N by N square and the functions are defined over pairs of variables and are generated uniformly randomly.

The **WCSP ("*.wcsp")** benchmark includes random binary WCSPs, scheduling problems from the SPOT5 benchmark, and radio link frequency assignment problems, providing a large variety of problem parameters.

The **mastermind** benchmark **("mastermind_")** includes 15 instances. Each problem is a ground instance of a relational Bayesian network, that models differing sizes of the popular game of Mastermind. These networks were produced by the PRIMULA System[4] and used in, for example, Chavira, et al., [13]. The resulting instances are quite large, with the number of binary variables n ranging between 1220 and 3692, each containing n unary and ternary cost functions.

Table 2.1 contains the benchmark parameters: # inst - number of instances, n - number of variables, k - maximum domain size and w^* - induced width of the ordering used. The

[3] http://graphmod.ics.uci.edu/repos/mpe/grids/

[4] http://www.cs.auc.dk/jaeger/Primula

induced width is one of the crucial parameters indicating the difficulty of the problem. Moreover, the difference between the induced width and the mini-bucket i-bound signifies the strength of the heuristic. When the i-bound is considerably smaller than the induced width, the heuristic is weak, while the i-bound equal or greater than the induced width yields an exact heuristic, which in turn yields much faster search. Clearly, a large number of variables or a high domain size suggest harder problems.

We evaluated the performance of mbe-m-opt as an approximate m-best algorithm and compared our algorithm with the BMMF scheme [106].

■ 2.6.2 Weighted Constraint Satisfaction Problems

The first part of our empirical evaluation assumed solving the Weighted CSP task, i.e, summation-minimization problem. We ran mbe-m-opt on 20 WCSP instances using i-bound equal to 10 and number of solutions m equal to 10. Table 2.2 shows for each instance the time in seconds it took mbe-m-opt to solve the 10-best problem and the values of the lower bounds on each of the first ten best solutions. Higher values are preferable. For each problem instance we also show the number of variables n, the largest domain size k and the induced width w^*. Note that 9 of the instances have induced width less than the $i = 10$ and thus are solved exactly. We see that, as the index number of solution goes up, the value of the corresponding lower bound increases, getting closer to the exact best solution. This demonstrates, that there is a potential of improving the bound on the optimal assignment using the m-best bounds as discussed in Section 2.5.2. Figure 2.4 illustrates this observation in graphical form, showing the dependency of the lower bounds on the solution index number for selected instances.

Instance)	n	k	w^*	time	Solution index number							
				(sec)	1	2	3	5	6	7	8	10
1502.uai	209	4	6	0.11	228.955	228.955	228.955	229.053	229.053	229.053	229.054	229.142
29.uai	82	4	14	55.17	147.557	147.557	147.925	148.189	148.189	148.557	148.557	148.925
404.uai	100	4	19	3.96	147.056	148.002	148.056	149.056	149.056	150.001	150.001	151.001
408.uai	200	4	35	80.27	436.551	437.179	437.549	437.551	438.177	438.178	438.179	438.550
42.uai	190	4	26	61.16	219.981	219.981	220.015	220.04857	220.048	220.083	220.083	220.913
503.uai	143	4	9	3.58	225.039	225.039	225.0394	226.038	226.038	226.038	226.038	226.038
GEOM30a_3.uai	30	3	6	0.03	0.008	1.008	2.008	2.008	3.008	3.008	3.008	4.008
GEOM30a_4.uai	30	4	6	0.19	0.008	1.008	2.008	3.008	3.008	4.008	4.008	5.008
GEOM30a_5.uai	30	5	6	0.84	0.008	1.008	2.008	2.008	3.008	3.008	3.008	4.008
GEOM40_2.uai	40	2	5	0	0.008	2.007	2.008	3.008	3.008	3.008	4.008	4.008
GEOM40_3.uai	40	3	5	0.01	0.008	2.007	2.007	3.008	4.008	4.008	4.007	4.007
GEOM40_4.uai	40	4	5	0.11	0.008	2.008	2.008	2.008	2.008	3.007	3.008	4.007
GEOM40_5.uai	40	5	5	0.16	0.008	2.008	2.008	3.008	4.007	4.007	4.007	4.007
le450_5a_2.uai	450	2	293	6.06	0.571	1.571	1.571	1.571	1.571	1.571	2.571	20.569
myciel5g_3.uai	47	3	19	6.39	0.023	1.023	1.023	3.024	4.02	10.023	11.023	11.023
myciel5g_4.uai	47	4	19	129.54	0.024	1.024	2.024	2.023	2.023	2.023	2.023	3.023
queen5_5_3.uai	25	3	18	5.53	0.016	1.016	1.016	2.016	2.016	3.016	3.016	3.016
queen5_5_4.uai	25	4	18	122.26	0.016	1.016	1.016	1.016	2.016	2.016	2.016	2.016

Table 2.2: The lower bounds on the 10 best solutions found by *mbe-m-opt* ran with $i = 10$ and $m = 10$. We also report the runtime in seconds, number of variables n, induced width w^* and largest domain size k.

■ 2.6.3 Most Probable Explanation Problems

For the second part of the evaluation the mbe-m-opt was solving the MPE problem, i.e., max-product task on three sets of instances: Pedigrees, Grids and Mastermind. We search for $m \in [1, 5, 10, 20, 50, 100, 200]$ solutions with i-bound equal to 10.

Pedigrees. Table 2.3 contains the runtimes in seconds for each of the number of solutions m along with the parameters of the problems. We see that the empirical scaling factor for the runtime with m is much smaller than theoretical analyses implies. Figure 2.5 presents the runtime in seconds against the number of solutions m for chosen pedigrees. We see that for small number of solutions ($m \leq 10$) for all instances, but one (pedigree38), the runtime increases very slowly with m.

Figure 2.6 demonstrates the difference between the way the runtime would scale according to the theoretical worst case performance analysis and the empirical runtimes obtained for various values of m. For three chosen instances we plot the experimental runtimes in seconds against the number of solutions m and the theoretical curve obtained by multiplying the value of empirical runtime for $m = 1$ by the factor of $m \log m$ for m equal to 5, 10, 50, 100 and

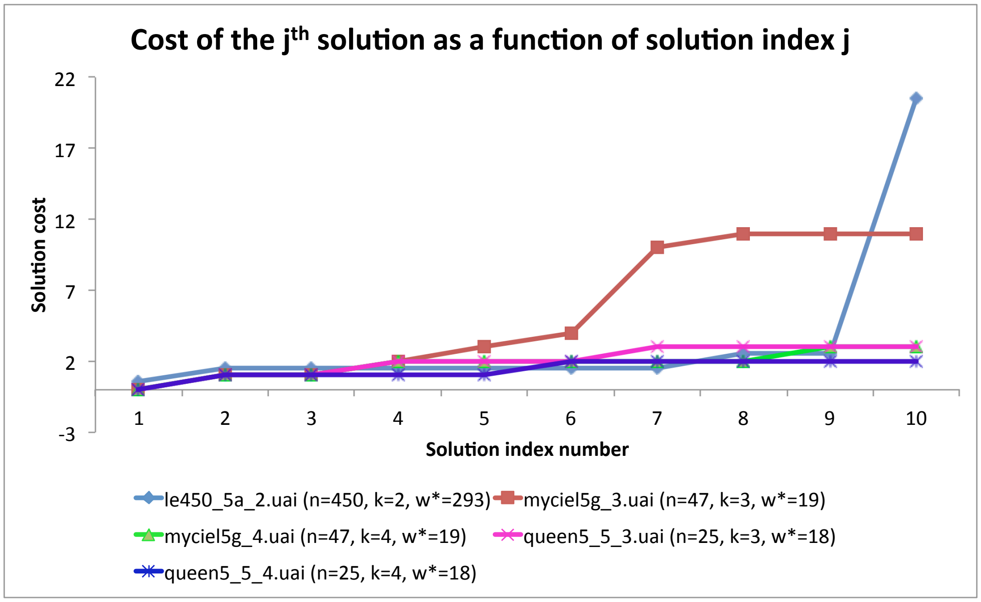


Figure 2.4: The change in the cost of the j^{th} solution as j increases for chosen WCSP instances. Results are obtained by mbe-m-opt with $i = 10$.

200. We see that the empirical curve lays much lower than theoretical for all instances.

Figure 2.7 illustrates the potential usefulness of the upper bounds on m best solutions as an approximation of the best solution. We plot in logarithmic scale the values of upper bounds on the 100 best solutions found by mbe-m-opt for the i-bounds ranging from 10 to 15. When using MBE as an approximation scheme, the common rule of thumb is to run the algorithm with the highest i-bound possible. In general, higher i-bound indeed corresponds to better accuracy, however increasing the parameter by a small amount (one or two) does not provably produce better results, as we can see in our example, where mbe-m-opt with $i = 10$ achieves better accuracy than the ones with $i = 11$ and $i = 12$. Such behavior can be explained by the differences in partitioning of the buckets into mini-buckets due to the changing of the control parameter i, which greatly influences the accuracy of MBE results. On the other hand, the upper bound on each next solution is always at least as good as the previous one. Thus the increase in m never leads to a worse bound and possibly can produce a better one.

However, we acknowledge that the power of mbe-m-opt with larger m for improving the upper bound on the 1^{st} best solution is quite weak compared with using higher i-bound. Although theory suggests that the time and memory complexity of mbe-m-opt is exponential in the parameter i, while only depending as a factor of $m \log m$ on the number of solutions, our experiments show that, in order to obtain a substantial improvement of the bound, it might be necessary to use high values of m. For example, for a problem with binary variables mbe-m-opt with $m = 1$ and a certain i-bound i is equivalent in terms of complexity to mbe-m-opt with $m = 3$ and i-bound $(i - 1)$. We observed that the costs of the first and third solutions are quite close for the instances we considered. In order to characterize, when the use of mbe-m-opt with higher m would add power over increasing the i-bound, a study of additional classes of instances is required.

Instances	n	k	w^*	Runtime (sec)						
				$m = 1$	$m = 5$	$m = 10$	$m = 20$	$m = 50$	$m = 100$	$m = 200$
pedigree1	334	4	15	0.22	0.57	1.01	1.46	3.35	6.87	25.46
pedigree13	1077	3	30	0.64	1.06	1.32	1.65	2.77	5.06	23.80
pedigree19	793	3	21	1.84	4.67	7.65	10.17	24.12	44.17	194.79
pedigree20	437	5	20	0.54	1.22	1.83	2.34	5.00	9.43	50.36
pedigree23	402	5	20	0.92	2.09	2.89	3.58	7.51	14.63	87.22
pedigree30	1289	5	20	0.38	0.66	1.00	1.26	2.48	4.58	19.53
pedigree31	1183	5	28	0.83	1.82	2.68	3.60	7.65	13.16	57.35
pedigree33	798	4	24	0.38	0.76	1.11	1.23	2.60	4.72	27.81
pedigree37	1032	5	20	1.64	3.27	4.56	6.25	14.15	26.43	158.74
pedigree38	724	5	16	4.52	11.77	19.63	28.87	73.21	127.65	552.30
pedigree39	1272	5	20	0.33	0.63	0.89	1.25	2.42	4.64	18.31
pedigree41	1062	5	28	1.45	3.33	4.43	5.56	11.67	20.59	120.79
pedigree51	871	5	39	0.76	1.24	1.65	2.16	3.98	6.97	33.95
pedigree7	867	4	32	0.66	1.17	1.61	2.15	4.45	8.01	39.26
pedigree9	935	7	27	0.85	1.48	2.12	2.77	5.70	9.49	50.58

Table 2.3: Runtime (sec) of mbe-m-opt on pedigree instances searching for the following number of solutions: $m = \in [1, 5, 10, 20, 50, 100, 200]$ with the $i = 10$. We report the number of variables n, largest domain size k and induced width w^*.

Grids. Table 2.4 shows the runtimes in seconds for each value of number of solutions m. Theory suggests that the runtimes for $m = 1$ and $m = 100$ should differ by at least two orders of magnitude, however, we can see that in practice mbe-m-opt scales much better. Figure 2.8 shows graphically the dependency of the runtime in seconds on the number of solutions m for 10 selected instances.

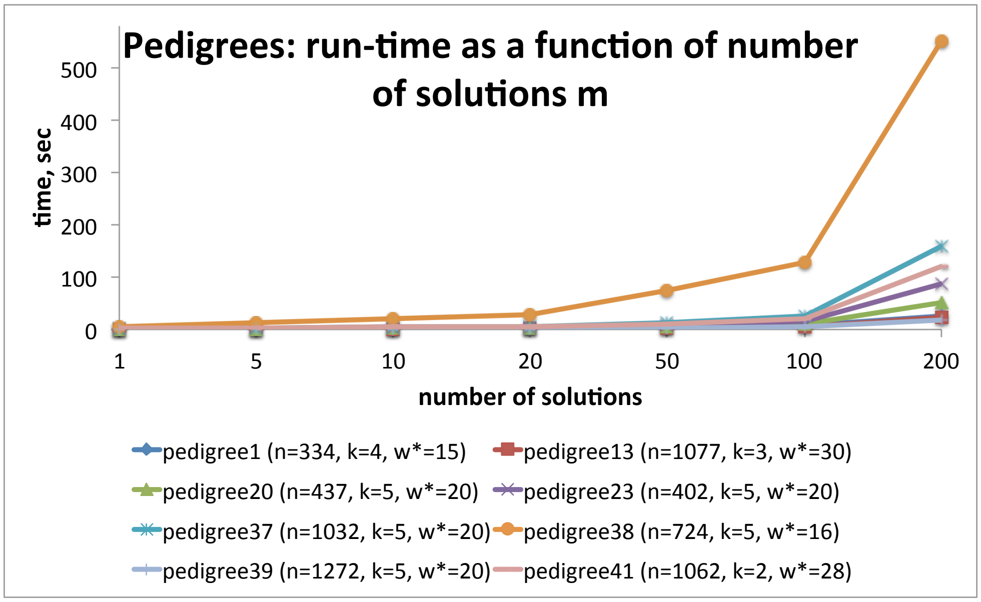


Figure 2.5: The run time (sec) for pedigree instances as a function of number of solutions m. *mbe-m-opt* ran with the $i = 10$.

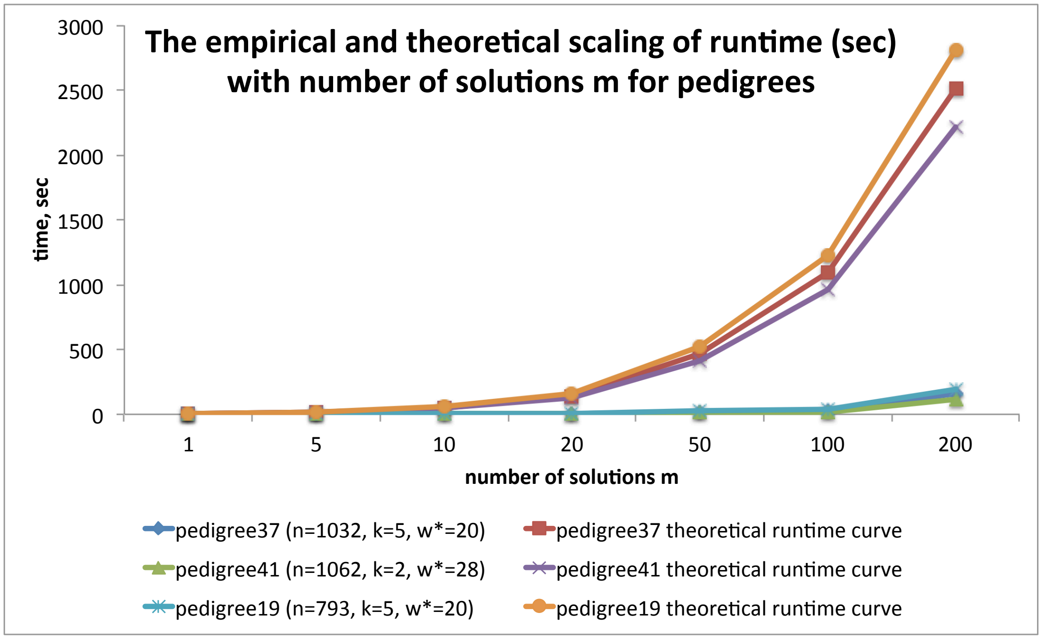


Figure 2.6: The empirical and theoretical runtime scaling with number of solutions m for chosen pedigree instances. The theoretical curve is obtained by multiplying the experimental runtime in seconds obtained for $m = 1$ by the factor of $m \log m$ for values $m \in [5, 10, 20, 50, 100, 200]$.

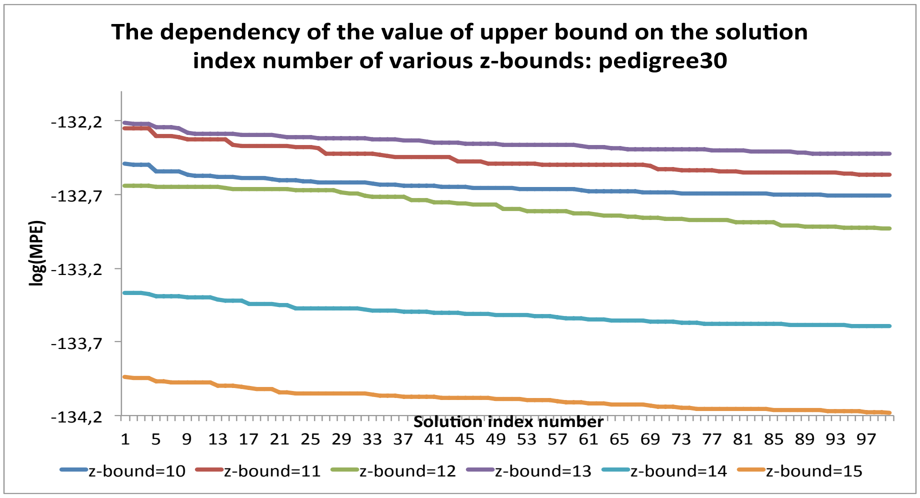


Figure 2.7: The upper bounds on the 100 best solutions (in log scale) found by mbe-m-opt ran with i-bounds∈ $[10, 11, 12, 13, 14, 15]$ for pedigree30 instance. The parameters of the problem: n=1289, k=5, w^*=20.

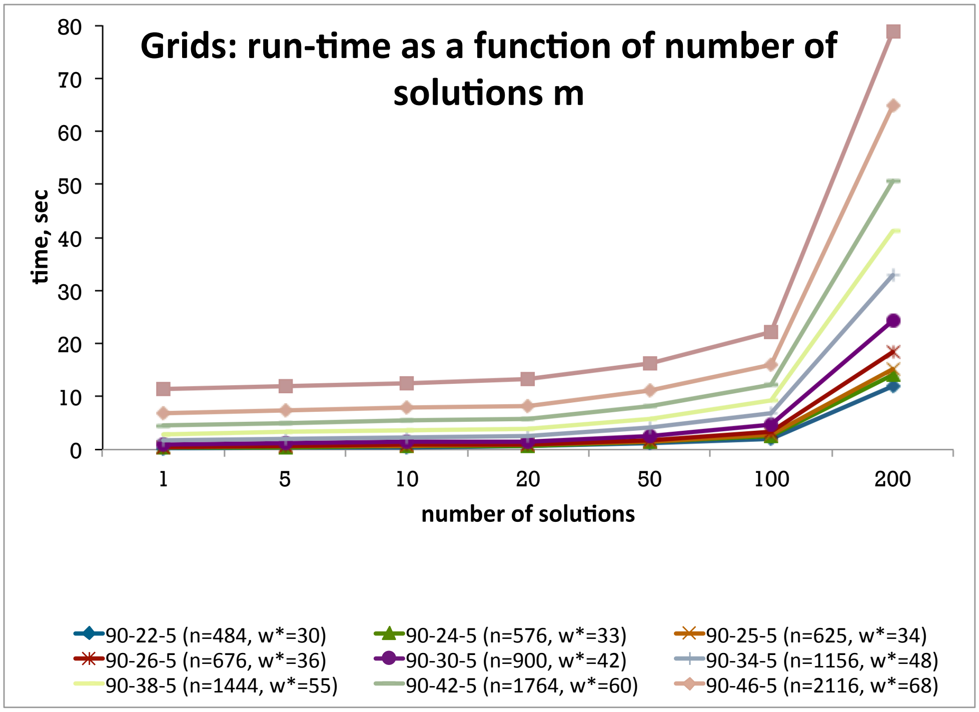


Figure 2.8: Selected binary grid instances: mbe-m-opt run time (sec) as a function of number of solutions m. The $i = 10$.

Instances	n	w^*	Runtime (sec)						
			$m = 1$	$m = 5$	$m = 10$	$m = 20$	$m = 50$	$m = 100$	$m = 200$
50-15-5	144	15	0.07	0.15	0.22	0.30	0.68	1.27	5.68
50-16-5	256	21	0.07	0.17	0.25	0.33	0.68	1.34	6.30
50-17-5	289	22	0.11	0.24	0.33	0.45	1.00	1.87	8.70
50-18-5	324	24	0.14	0.29	0.35	0.52	1.05	2.04	9.13
50-19-5	361	25	0.13	0.29	0.41	0.54	1.15	2.24	9.87
50-20-5	400	27	0.18	0.33	0.44	0.59	1.20	2.28	10.65
75-16-5	256	21	0.08	0.17	0.21	0.27	0.56	1.09	5.92
75-17-5	289	22	0.10	0.21	0.27	0.36	0.75	1.46	7.83
75-18-5	324	24	0.12	0.23	0.30	0.40	0.79	1.58	8.34
75-19-5	361	25	0.14	0.26	0.34	0.47	0.94	1.86	9.22
75-20-5	400	27	0.18	0.30	0.38	0.52	0.97	1.80	9.78
75-21-5	441	28	0.20	0.36	0.44	0.60	1.07	2.03	10.91
75-22-5	484	30	0.25	0.40	0.53	0.68	1.28	2.49	12.40
75-23-5	529	31	0.29	0.47	0.56	0.71	1.36	2.44	13.11
75-24-5	576	32	0.34	0.51	0.65	0.81	1.49	2.87	14.58
75-25-5	625	34	0.41	0.62	0.74	0.93	1.71	3.18	16.08
75-26-5	676	36	0.49	0.73	0.90	1.17	2.06	3.86	19.06
90-20-5	400	27	0.17	0.27	0.35	0.44	0.81	1.57	9.26
90-21-5	441	28	0.02	0.35	0.41	0.52	0.97	1.91	10.72
90-22-5	484	30	0.25	0.41	0.47	0.61	1.10	2.08	11.85
90-23-5	529	31	0.29	0.46	0.55	0.66	1.17	2.27	12.63
90-24-5	576	33	0.34	0.49	0.60	0.74	1.36	2.61	13.98
90-25-5	625	34	0.42	0.58	0.70	0.83	1.50	2.80	15.26
90-26-5	676	36	0.49	0.71	0.85	1.01	1.87	3.42	18.36
90-30-5	900	42	0.93	1.25	1.40	1.59	2.60	4.62	24.26
90-34-5	1156	48	1.69	2.07	2.29	2.60	4.15	6.77	32.93
90-38-5	1444	55	2.86	3.26	3.57	3.98	5.72	9.27	41.33
90-42-5	1764	60	4.57	5.10	5.49	5.88	8.32	12.31	50.70
90-46-5	2116	68	6.81	7.42	7.97	8.33	11.09	16.06	64.88
90-50-5	2500	74	11.3	12.07	12.51	13.2	16.25	22.09	78.70

Table 2.4: Binary grid instances: runtime (sec) of mbe-m-opt for the number of required solutions $m \in [1, 5, 10, 20, 50, 100, 200]$ with the $i = 10$. We report the number of variables n and induced width w^*.

Mastermind. Table 2.5 presents the run time changes with various numbers of best solutions m. We refrain from reporting and discussing the values of the upper bounds found, since mastermind instances in question typically have a large set of solutions with the same costs, making the values of the bounds not particular informative.

■ 2.6.4 Comparison with BMMF

BMMF [106] is a Belief Propagation-based algorithm, which is exact, when ran on junction trees, and approximate, if the problem graph has loops. We compared the performance of mbe-m-opt and BMMF on randomly generated 10 by 10 binary grids. The algorithms differ in the nature of the outputs: BMMF provides approximate solutions with no guarantees,

Instances	n	k	w^*	Runtime (sec)						
				$m = 1$	$m = 5$	$m = 10$	$m = 20$	$m = 50$	$m = 100$	$m = 200$
mastermind_03_08_03-0006	1220	2	19	0.44	0.57	0.64	0.72	1.03	2.68	13.11
mastermind_03_08_03-0007	1220	2	18	0.25	0.33	0.35	0.40	0.67	1.81	8.90
mastermind_03_08_03-0014	1220	2	20	0.68	0.84	0.92	0.98	1.51	3.76	17.77
mastermind_03_08_04-0004	2288	2	30	1.98	2.25	2.28	2.49	3.42	7.11	32.12
mastermind_03_08_04-0005	2288	2	30	1.92	2.20	2.35	2.50	3.44	7.16	32.08
mastermind_03_08_04-0010	2288	2	29	2.53	2.82	2.97	3.09	4.17	8.25	34.89
mastermind_03_08_04-0011	2288	2	29	3.55	3.85	4.00	4.16	5.40	9.90	38.48
mastermind_03_08_05-0001	3692	2	37	6.33	6.73	7.02	7.24	9.10	15.83	59.03
mastermind_03_08_05-0005	3692	2	37	6.33	6.85	7.04	7.29	9.08	15.6	58.77
mastermind_03_08_05-0009	3692	2	37	3.44	3.72	3.81	3.97	5.18	9.59	38.62
mastermind_03_08_05-0010	3692	2	37	6.23	6.57	6.90	7.10	8.80	14.87	56.43
mastermind_04_08_03-0000	1418	2	24	1.12	1.30	1.41	1.49	2.16	4.51	20.33
mastermind_04_08_03-0013	1418	2	23	1.12	1.33	1.43	1.51	2.19	4.60	20.73
mastermind_05_08_03-0004	1616	2	27	1.22	1.43	1.47	1.57	2.22	4.60	20.39
mastermind_05_08_03-0006	1616	2	27	0.21	0.23	0.25	0.26	0.37	0.82	3.84

Table 2.5: The runtime (sec) of mbe-m-opt for the mastermind instances. Number of required solutions $m \in [1, 5, 10, 20, 50, 100, 200]$, $i = 10$. We report the number of variables n, induced width w^*, domain size k.

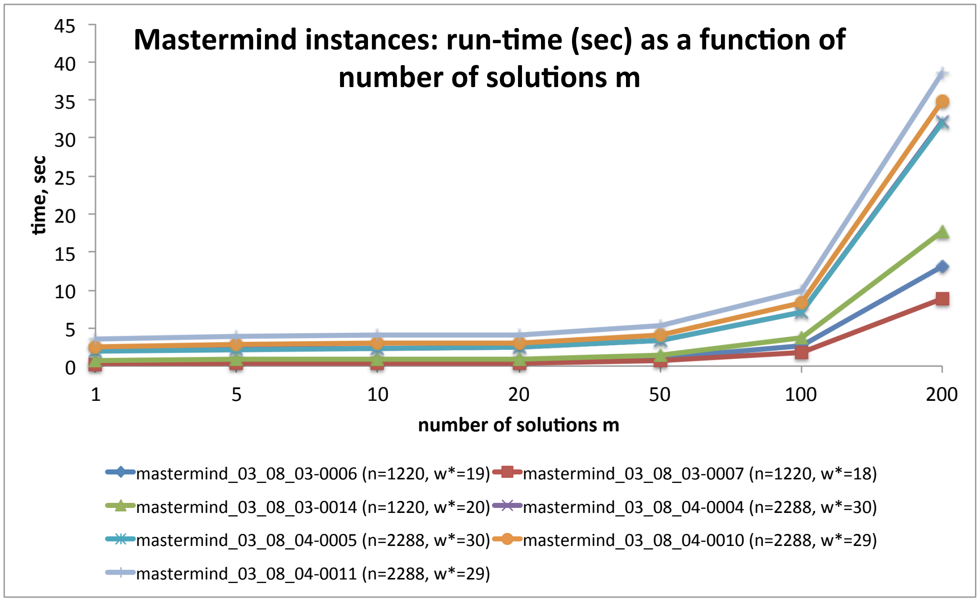


Figure 2.9: The runtime (sec) of mbe-m-opt run time as a function of number of solutions m for the mastermind instances. The $i = 10$.

while mbe-m-opt generates bounds on all the m-best solutions. Moreover, the runtimes of the algorithms are not comparable since our algorithm is implemented in C and BMMF in Matlab, which is inherently slower. For most instances that mbe-m-opt can solve exactly in under a second, BMMF takes more than 5 minutes.

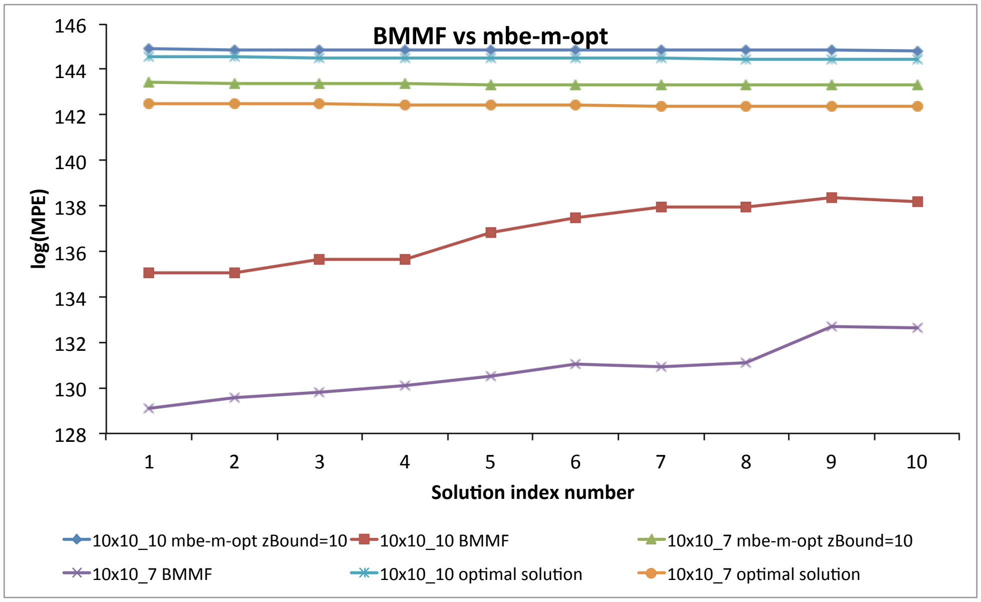


Figure 2.10: Comparison of mbe-m-opt with i-bound equal to 10 and BMMF on random 10x10 grids. The exact solutions obtained by elim-m-opt. The mbe-m-opt provides upper bounds on the solutions, BMMF gives no guarantees whether it outputs an upper or a lower bound. In this particular example BMMF outputs lower bounds on the exact solutions.

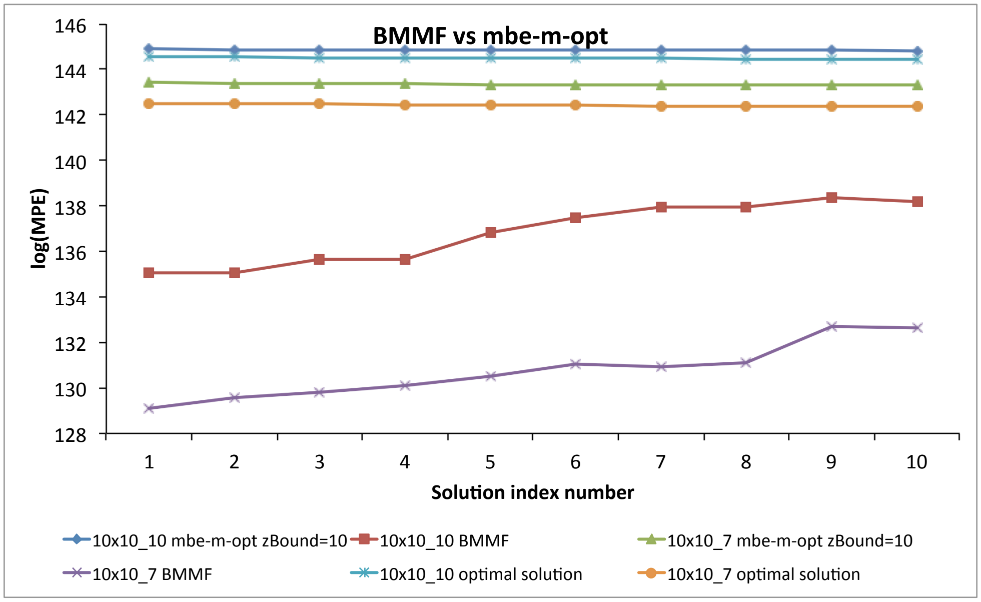


Figure 2.11: Comparison of mbe-m-opt with i-bound equal to 10 and BMMF on random 10x10 grids. The exact solutions obtained by elim-m-opt. The mbe-m-opt provides upper bounds on the solutions, BMMF gives no guarantees whether it outputs an upper or a lower bound. In this particular example BMMF outputs lower bounds on the exact solutions.

Still, some information can be learned from viewing the two algorithms side by side, as is demonstrated by typical results in Figure 2.11. For two chosen instances we plot the values of the 10-best bounds reported by both algorithms in logarithmic scale as a function of the solution index. We also show the exact solutions found by the algorithm elim-m-opt. We can see that mbe-m-opt with the i-bound equal to 10 can produce upper bounds that are considerably closer to the exact solutions than the results reported by BMMF.

■ 2.7 Conclusion

We presented a formulation of the m-best reasoning task within a framework of semirings, thus making all existing inference and search algorithms immediately applicable for the task via the definition of the combination and elimination operators. We then focused on inference algorithms and provided a bucket elimination algorithm, elim-m-opt, for the m-best task. We analyzed its performance and empirically evaluated the algorithm, demonstrating that in practice mbe-m-opt scales as a function of m better than worst-case analysis predicted.

We emphasize that the practical significance of the algorithm is primarily for approximation through the mini-bucket scheme, since other exact schemes have better worst-case performance. Indeed, as we will show in the next chapter, heuristic search methods are quite efficient for the m-best task, because they can benefit from the power of the guiding heuristic function. The promise of the elim-m-opt inference algorithm is in its potential to yield viable lower- and upper-bounds for the m-best solutions via the mini-bucket algorithm, as we discussed. Furthermore, it could also lead to loopy propagation message-passing schemes that are highly popular for approximations in graphical models.

Chapter 3

Heuristic Search for *M*-best Task

■ 3.1 Introduction[1]

In Chapter 2 we extended inference schemes as represented by the bucket elimination algorithm (BE) [19] to the task of finding the m best solutions. However, due to their large space requirements, variable elimination algorithms, including bucket elimination, cannot be used in practice for finding exact solutions to combinatorial optimization tasks when the problem's tree width is high. Depth-first branch and bound, or DFBB, and best-first search, or BFS, both presented in Section 1.2.3 are more flexible and can trade space for time. This chapter explores the possibilities of adapting such search algorithms to finding the m best solutions.

Our contribution lies in extending the heuristic search algorithms to the m best solutions task. We described general purpose m-best variants of both depth-first branch and bound and best-first search (A*), yielding algorithms m-BB and m-A* respectively, and analyzed their properties. Specifically, we showed that m-A* inherits all of A*'s desirable properties [24], most significantly it is optimally efficient compared to any other exact search-based scheme. We also discussed the size of the search space explored by m-BB. We then extended

[1] Part of this work has already been published in Rina Dechter, Natalia Flerova, and Radu Marinescu. ”Search Algorithms for m Best Solutions for Graphical Models” in Proceedings of AAAI 2012.

our new m-best algorithms to graphical models by exploring the AND/OR search space.

We evaluated the resulting algorithms on six benchmarks having more than 300 instances in total, examining the impact of the number of solutions m on the algorithms' behavior. In particular, we observed that the runtime of most of the schemes (except for the depth-first branch and bound exploring an AND/OR tree) scales much better with m than what worst case theoretical analysis suggests. We also showed that a m-A* search armed with exact bucket elimination heuristic (a scheme we called BE+m-BF) is highly efficient on easier problems but suffers severely from memory issues over dense graphs, far more than the A*-based schemes using approximate mini-bucket heuristic. Finally, we compared our schemes with some of the most efficient algorithms based on the LP-relaxation [37, 5], showing competitiveness and even superiority for large values of m ($m \geq 10$), while providing optimality guarantees.

The chapter is organized as follows. Section 3.2 presents the extension of best-first search to the m-best task. In particular, we define m-A*, the extension of A* algorithm to the m-best (3.2.1), and prove its main properties (3.2.2). Section 3.3 describes algorithm m-BB, an extension of depth-first branch and bound algorithm to solving the m-best solution task. In Section 3.4 we discuss the adaptation of the two newly proposed m-best search algorithms for AND/OR search spaces over graphical models, including a hybrid method BE+m-BF that incorporates both variable elimination and heuristic search. Section 3.5 elaborates on the related work and contrasts it with our methods. Section 3.6 presents the empirical evaluation of our m-best schemes and Section 3.7 concludes.

■ 3.2 Best-First Search for M-best Solutions

Extending best-first search (Section 1.2.3) and in particular its most popular version, A*, to the m-best task is fairly straightforward and was suggested, for example, by Charniak [12]. Instead of stopping after finding the optimal solution, the algorithm continues exploring the search space, reporting the next discovered solutions up until m of them are obtained. Our contribution is in showing that these solutions are indeed the m best and that they are found in a decreasing order of their optimality. In particular, the second solution reported is the second best solution and, in general, the i^{th} solution discovered is the i^{th} best. Moreover, we explore and prove the main properties of the newly formulated algorithm.

■ 3.2.1 m-A*: Definition

The m-best tree-search variant of A* denoted m-A* (Algorithm 10, assumes a consistent heuristic) solves an m-best optimization problem over any general search graph. We will show later how it can be extended to general admissible heuristics.

The scheme expands the nodes in the order of increasing value of f in the usual A* manner. It keeps the lists of generated nodes OPEN and expanded nodes CLOSED, as usual, maintaining a search tree, denoted by Tr. Beginning with the start node s, m-A* picks the node with the smallest evaluation function $f(n)$ on OPEN and puts it on CLOSED (line 7). If the node is a goal, a new solution is reported (lines 8-13). Otherwise, the node is expanded and its children are created (lines 14-23). The algorithm may encounter each node multiple times and will maintain up to m its copies on OPEN and CLOSED lists combined (line 17), with separate paths to each copy in the explored search tree (lines 22-23). Nodes encountered beyond m times are discarded (line 18). We denote by C^*_i the i^{th} best solution cost, by $f^*_i(n)$ the cost of the i^{th} best solution going through node n, by $f_i(n)$ the heuristic evaluation function estimating $f^*_i(n)$ and by $g_i(n)$ and $h_i(n)$ the estimates of the i^{th} best costs from s to n and from n to a goal, respectively.

If the heuristic is not consistent, whenever the algorithm reaches a node it has seen before (if the search space is a graph and not a tree), there exists a possibility of the new path improving on the previously discovered ones. Therefore, lines 17-18 should be revised in the following way to account for the possibility that a better path to n' is discovered:

```
17  If $n'$ appears already more than $m$ times in the union of OPEN or CLOSED then
18      If $g(n')$ is strictly smaller than $g_m(n')$, the current $m$-best path to $n'$ then
19          Keep $n'$ with a pointer to $n$ and put n back in OPEN
20          Discard the earlier subtree rooted at $n$
```

Figure 3.1 shows an example of m-A* finding the $m = 3$ shortest paths on a toy problem. On the left, Figure 3.1(a) shows the problem graph with 7 variables and 8 edges, along with the admissible heuristic functions for each node. Note that the heuristic is not consistent. For example, $h(A) > h(C) + c(A, C)$. A is the start node, G is the goal node. On the right, Figure 3.1(b) presents the trace of m-A*, with the evaluation function for each copy of a node, created by the time that the 3^{rd} solution is found. The white nodes are on CLOSED, the grey one (node G_4) was created, but never put on OPEN.The algorithm expands the nodes on OPEN in increasing order of the evaluation functions. We assume that ties are broken in favor of deeper nodes. First m-A* discovers the solution $A - C - D - F - G$ with cost $C_1^* = 8$, next the solution $A - C - D - E - G$ with cost $C_1^* = 10$. The third solutions was $A - B - D - F - G$ with cost $C_1^* = 10$. Note that two copies of each node D, E and F and four copies of G were created. The goal node G_4 was discarded, because we bound the total number of copies of a particular node by $m = 3$.

Theorem 3.1. *Given a graphical model $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \bigotimes \rangle$ with n variables whose domain size is bounded by k, the worst case time and space complexity of m-A* exploring an OR search tree of $\mathcal{M}$ is $O(k^n)$.*

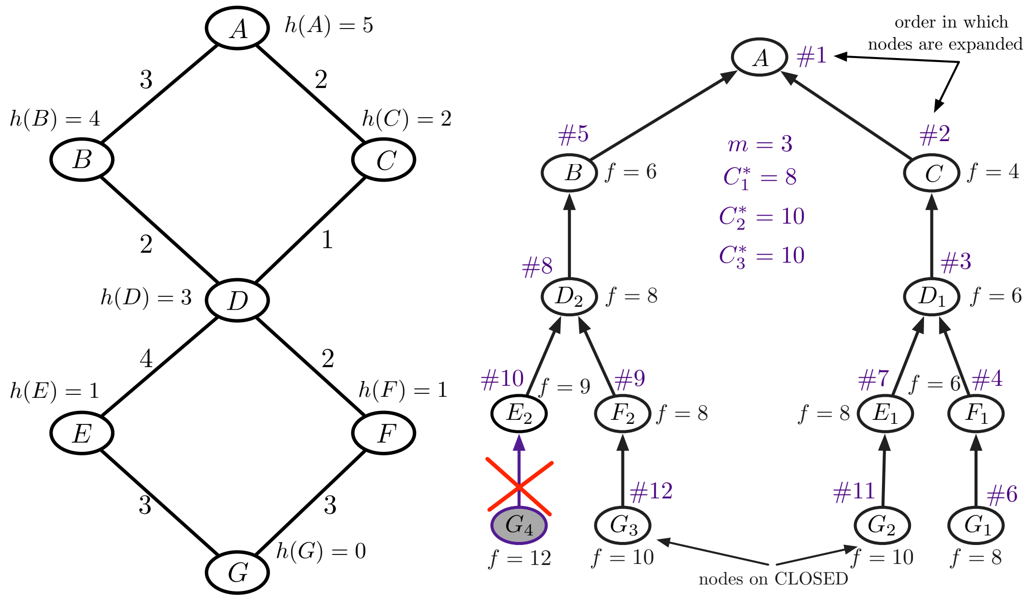
Algorithm 10: m-A* exploring a graph, assuming consistent heuristic

```
   Input: An implicit directed search graph $G = (N, E)$, with a start node $s$ and a set of goal nodes
          $Goals$, a consistent heuristic evaluation function $h(n)$, parameter $m$
   Output: the $m$ best solutions
1  Initialize: OPEN=$\emptyset$, CLOSED=$\emptyset$, a tree $Tr = \emptyset$, $i = 1$ ($i$ counts the current solution being searched
   for)
2  OPEN $\leftarrow \{s\}$; $f(s) = h(s)$;
3  Make $s$ the root of $Tr$;
4  while $i \leq m$ do
5      if OPEN is empty then
6          return the solutions found so far;
7      Remove a node, denoted $n$, in OPEN having a minimum $f$ (break ties arbitrarily, but in favor of
       goal nodes and deeper nodes) and put it in CLOSED;
8      if $n$ is a goal node then
9          Output the current solution obtained by tracing back pointers from $n$ to $s$ (pointers are
           assigned in step 22); denote this solution as $Sol_i$;
10         if $i = m$ then
11             return;
12         else
13             $i \leftarrow i + 1$;
14     else
15         Expand node $n$, generating all its children $Ch$ ;
16         foreach $n' \in Ch$ do
17             if $n'$ already appears in OPEN or CLOSED $m$ times then
18                 Discard node $n'$;
19             else
20                 Compute current path cost $g(n') = g(n) + c(n, n')$;
21                 Compute evaluation function $f(n') = g(n') + h(n')$ ;
22                 Attach a pointer from $n'$ back to $n$ in $Tr$;
23                 Insert $n'$ into the right place in OPEN based on $f(n')$;
24 return The set of the $m$ best solutions found
```

Proof. In worst case m-A* would explore the entire OR search tree, whose size is $O(k^n)$ (Section 1.2.4). Since the underlying search space is a tree, the algorithm will never encounter any of the nodes more than once, thus no nodes will be duplicated. □

■ 3.2.2 Properties of m-A*

In this section we extend the desirable properties of A*, listed in Section 1.2.3, to the m-best case. For simplicity and without loss of generality, we assume throughout that the search graph accommodates at least m distinct solutions.



(a) Primal graph

(b) Trace of m-A*

Figure 3.1: Example problem. On the left: problem graph. $h(n)$ - heuristic values. On the right: trace of m-A*, solving minimization-summation problem for $m = 3$, $f(n)$ - evaluation function. White nodes are on CLOSED, the grey one was created, but discarded.

Theorem 3.2. *Given an optimization task over a graphical model and some integer parameter $m \geq 1$, m-A* aimed with admissible heuristic possesses the following properties:*

1. *Soundness and completeness:* m-A* *terminates with the m best solutions generated in order of their costs.*

2. *Optimal efficiency under consistent heuristic: Any node that is surely expanded*[2] *by* m-A* *must be expanded by any other search algorithm guaranteed to find the m best solutions having the same heuristic information.*

3. *Optimal efficiency for node expansions:* m-A* *expands each node at most m times when the heuristic is consistent. The i^{th} path found to a node is the i^{th} best path.*

4. *Dominance: Given two heuristic functions h_1 and h_2, such that for every n $h_1(n) <$*

[2] to be precisely defined in Section 3.2.2.3

$h_2(n)$, m-A$_1$ will expand every node surely expanded by m-A*$_2$, when m-A*$_i$ is using heuristic h_i.*

We prove the properties of m-A* in Sections 3.2.2.1-3.2.2.2.

■ 3.2.2.1 Soundness and Completeness

Algorithm m-A* maintains up to m copies of each node and discards the rest. We will next show that this restriction does not compromise completeness.

Proposition 3.1. *Any node discarded by m-A* does not lead to any of the m-best solutions.*

Proof. Consider a consistent heuristic first (as described in Algorithm 10). At the moment when m-A* discovered a node n for the $(m+1)^{th}$ time, m copies of n reside on OPEN or CLOSED, and the algorithm maintains m distinct paths to each. Let π_m be the $(m+1)^{th}$ path. As we will prove in Theorem 3.8, when node n is discovered for the $(m+1)^{th}$ time, the cost C_{new} of the newly discovered path π_{new} is the $(m+1)^{th}$ best, namely it is no better than the costs already discovered: $C_{new} \geq C_{\pi_m}$. Therefore, the eliminated $(m+1)^{th}$ path to node n is guaranteed to be worse than the remaining m ones and thus cannot be a part of any of the potential m-best optimal solutions that might be passing through node n.

If the heuristic is not consistent, m-A* can be modified to replace the worst of the previously discovered paths π_m with the newly found π_{new}, if the cost of the latter is better and place the new copy in OPEN. Thus, again, it is safe to bound the number of copies by m. □

It is clear that along any particular solution path π the evaluation function over all the nodes on π is bounded by the path's cost $C(\pi)$, when the heuristic is admissible.

Proposition 3.2. *The following is true regarding m-A*:*

1. *For any solution path π, for all nodes $n \in \pi$, $f(n) \leq C(\pi)$.*
2. *Unless π was already discovered by m-A*, there is always a node n on π which resides in OPEN.*
3. *Therefore, as long as m-A* did not discover π, there must be a node in OPEN having $f(n) \leq C(\pi)$.*

Proof. 1. $f_\pi(n) = g_\pi(n) + h(n)$ and since $h(n) \leq c_\pi(n,t)$ due to admissibility, where $c_\pi(n,t)$ is the actual cost from n to the goal node t along π, we conclude that $f(n) \leq g_\pi(n) + h(n) = C(\pi)$.
2. Any path reachable from the root always has a leaf on OPEN unless all the nodes along the path are expanded and are on CLOSED.
3. Follows easily from 1 and 2. □

It follows immediately from Proposition 3.2 (similar to [74]) that:

Proposition 3.3. *[Necessary condition for node expansion.] Any node n expanded during m-A*, when searching for the i^{th} best solution ($1 \leq i \leq m$), satisfies $f(n) \leq C^*_i$.*

and it is also clear that

Proposition 3.4. *[Sufficient condition for node expansion.] Every node n on OPEN, such that $f(n) < C^*_i$, must be expanded by m-A* before the i^{th} best solution is found.*

Soundness and completeness of m-A* follows quite immediately.

Theorem 3.3 (**Soundness and completeness**). *Algorithm m-A* generates the m-best solutions in order, namely, the i^{th} solution generated is the i^{th} best solution.*

Proof. Let us assume that it is not the case. Let the i^{th} generated solution path π_i be the first one that is not generated according to the best-first order. Namely the i^{th} solution generated has a cost C such that $C > C_i^*$. However, when the algorithm selected the goal t_i along π_i, its evaluation function was $f(t_i) = g_{\pi_i}(t_i) = C$, while, based on Proposition 3.2, there was a node n' on OPEN whose evaluation function was at most C_i^*. Thus n' should have been selected for expansion instead of t_i - a contradiction. □

■ 3.2.2.2 The Impact of the Heuristic Strength

Like for A*, the performance of m-A* improves with more accurate heuristic.

Proposition 3.5. *Consider two heuristic functions h_1 and h_2. Let us denote by m-A*$_1$ the algorithm that uses heuristic h_1 and by m-A*$_2$ the one using heuristic h_2. If the heuristic h_1 is more informed than h_2, namely for every node n $h_2(n) < h_1(n)$, algorithm m-A*$_2$ will expand every node that will be expanded by the algorithm m-A*$_1$ before finding the j^{th} solution for any $j \in [1, m]$, assuming the same tie-breaking rule.*

Proof. Since h_1 is more informed than h_2, $h_1(n) > h_2(n)$ for every non-goal node n. Let us assume that m-A*$_1$ expands some non-terminal node n before finding the j^{th} best solution with cost C_j^*. If node n is expanded, it means that (a) at some point it is on OPEN and (b) its evaluation function satisfies $f_1(n) = g(n) + h_1(n) \leq C_j^*$ (Proposition 3.3). Consider the current path π from start node to n. Each node $n' \in \pi$ on the path was selected at some point for expansion and thus the evaluation functions of all these nodes are also bounded by the cost of the j^{th} best solution: $f_1(n') \leq C_j^*$. Since $h_1(n') > h_2(n')$ for every node n' along the path π, their evaluation functions according to heuristic $h_2(n)$ obeys:

$$f_2(n') = g(n') + h_2(n') < g(n') + h_1(n') < C_j^* \tag{3.1}$$

and thus each node n' must also be expanded by m-A*$_2$. □

Consider the case of the exact heuristic. It is easy to show that

Theorem 3.4. *If $h = h^*$ is the exact heuristic, then m-A* generates solutions only on j-optimal paths $1 \leq j \leq m$.*

Proof. Since h is exact, the f values on OPEN are expanded in sequence of values $C_1^* \leq C_2^* \leq \ldots \leq C_i^* \ldots \leq C_m^*$. All the generated nodes having evaluation function $f = C_1^*$ are by definition on optimal paths (since $h = h^*$), all those who have $f = C_2^*$ must be on paths that can be second best and so on. Notice that some solutions can have the same costs. □

When $h = h^*$, m-A*'s complexity is clearly linear in the number of nodes having evaluation function $f^* \leq C_m^*$. However, when the cost function has only a small range of values, there may be an exponential number of solution paths having the cost C_m^*. To avoid this exponential frontier we chose the tie-breaking rule of expanding deeper nodes first, yielding a number of node expansions bounded by $m \cdot n$, when n bounds the solution length. Clearly then:

Theorem 3.5. *When m-A* has access to $h = h^*$, then using a tie-breaking rule in favor of deeper nodes it expands at most $\#N = \sum_i \#N_i$ nodes, where $\#N_i$ is the length of the i^{th} optimal solution path. Clearly, $\#N \leq m \cdot n$.*

■ 3.2.2.3 m-A* with Consistent Heuristic

When m-A* uses a consistent heuristic, it has several useful properties.

Optimal efficiency under consistent heuristic. Algorithm A* is known to be optimally efficient for consistent heuristic [24]. Namely, any other algorithm that extends search paths from the root and uses the same heuristic information as A* will expand every node that is *surely expanded by A**, i.e., it will expand every n, such that $f(n) < C^*$. We extend the notion of nodes surely expanded by A* for m-best case:

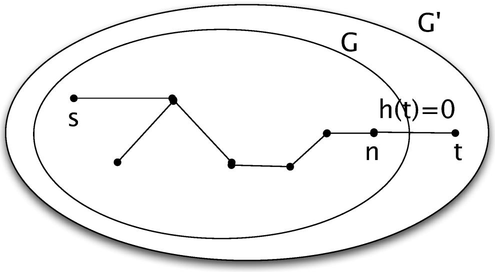


Figure 3.2: The graph G' represents a new problem instance constructed by appending a branch leading to a new goal node t to node n.

Proposition 3.6. *Algorithm m-A* will expand any node n reachable by a strictly C^*_m-bounded path from the root, regardless of the tie-breaking rule. The set of such nodes is referred to as* surely expanded by m-A*s*.*

Proof. Let us consider the strictly C^*_m-bounded path $\pi = \{s, n_1, n_2, \ldots n\}$. The start node s is clearly expanded at the beginning of the search and its children, including node n_1, are placed on OPEN. Since $f(n_1) < C^*_m$, node n_1 must be expanded by m-A* before finding the m^{th} best solution (Proposition 3.4), its children, including n_2, in turn are placed on OPEN. The same is true for all nodes of π, including n. □

Theorem 3.6 (M-optimal efficiency). *Any search algorithm that is guaranteed to find the m-best solutions and that explores the same search graph as m-A* and has the same consistent heuristic will have to expand any node that is surely expanded by m-A*. Namely it will expand every node that lies on any path π dominated by C^*_m, i.e., $f(n') < C^*_m, \forall n' \in \pi$.*

The proof idea is similar to [24]. Namely we can show that any algorithm that does not expand a node n, surely expanded by m-A*, can miss one of the m-best solutions, when applied to a slightly modified problem:

Proof. Let us consider a problem having the search graph G and a consistent heuristic h. Assume that node n is surely expanded by m-A* before finding the j^{th} best solution. Let

B be an algorithm that uses the same heuristic h and is guaranteed to find the m best solutions. Let also assume that node n is not expanded by B.

We can create a new problem graph G' (see Figure 3.2) by adding a new goal node t with $h(t) = 0$, connecting it to n by an edge having cost $c = h(n) + \delta$, where $\delta = 0.5(C_j^* - D)$ and $D = \max\limits_{n' \in S_j} f(n')$. S_j is the set of nodes surely expanded by *m-A** before finding the j^{th} solution, namely $S_j = \{n | f(n) < C_j^*\}$. It is possible to show that the heuristic h is admissible for the graph G' [24]. Since $\delta = 0.5(C_j^* - D)$, $C^* = D - 2\delta$. By construction, the evaluation function of the new goal node is:

$$f(t) = g(t) + h(t) = g(n) + c = g(n) + h(n) + \delta = f(n) + \delta \leq D + \delta = C_j^* - \delta < C_j^* \quad (3.2)$$

which means that t is reachable from s by a path whose cost is strictly bounded by C_j^*. That guarantees that m-A* will expand t (Proposition 3.6), discovering a solution with cost $C_j^* - \delta$. On the other hand, algorithm B, that does not expand node n in the original problem, will still not expand it, thus not reaching node t, and will only discover the solution with cost C_j^*, not returning the true set of m best solutions to the modified problem. From the contradiction the theorem follows. □

Proposition 3.7. *If the heuristic function employed by m-A* is consistent, the values of the evaluation function f of the sequence of expanded nodes are non-decreasing.*

The proof is a straightforward extension of a result from [72].

Proof. Let node n_2 be expanded immediately after n_1. If n_2 was already on OPEN at the time when n_1 was expanded, then from the node selection rule it follows that $f(n_1) \leq f(n_2)$. If n_2 was not on OPEN, then it must have been added to it as a result of expansion of n_1, i.e., be a child of n_1. In this case the cost of getting to n_2 from the start node is $g(n_2) = g(n_1) + c(n_1, n_2)$

and the evaluation function of node n_2 is $f(n_2) = g(n_2) + h(n_2) = g(n_1) + c(n_1, n_2) + h(n_2)$. Since $h(n)$ is consistent, $h(n_1) \leq c(n_1, n_2) + h(n_2)$ and $f(n_2) \geq g(n_1) + h(n_1)$. Namely, $f(n_2) \geq f(n_1)$. □

If the heuristic function is consistent, we have a stronger condition of Proposition 3.4:

Theorem 3.7. *Algorithm m-A* using a consistent heuristic function:*

1. *expands all nodes n such that $f(n) < C^*_m$*

2. *never expands any nodes with evaluation function $f(n) > C^*_m$*

3. *expands some nodes such that $f(n) = C^*_m$, subject to a tie-breaking rule*

Proof. 1. Assume that there exists a node n such that $f(n) < C^*_m$ and node n is never expanded by m-A*. Such situation can only arise if node n has never been on OPEN list, otherwise it would have been expanded, according to Proposition 3.4. That implies that the parent of node n in the search space (let us denote is node p) has never been expanded. However, similarly how it is done in the proof of Proposition 3.7, it is easy to show that $f(p) \leq f(n)$ and, consequently $f(p) < C^*_m$. Thus node p must also have never been on OPEN, otherwise it would be expanded. Clearly, this is true for all the ancestors of n, up to the start node s. Since node s is clearly on OPEN at the beginning of the search, the initial assumption is incorrect and property follows.

2. and 3. Directly follow from Proposition 3.3 □

Figure 3.3 provides a schematic summary of the search space explored by m-A* having a consistent heuristic.

Optimal efficiency for node expansions. Whenever a node n is selected for expansion for the first time by m-A*, the algorithm has already found the shortest path to that node.

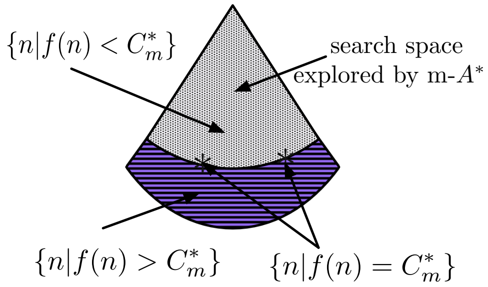


Figure 3.3: The nodes explored by m-A* algorithm with consistent heuristic.

We can extend this property as follows:

Theorem 3.8. *Given a consistent heuristic h, when* m-A* *selects a node n for expansion for the i^{th} time, then $g(n) = g_i^*(n)$, namely it has found the i^{th} best path from start node s to n.*

Proof. By induction. For the $i = 1$ the theorem holds [72]. Assume that it also holds for $(i-1)^{th}$ expansion of node n. Let us consider the i^{th} case, $i > 1$. We have already expanded the node n $(i-1)$ times and due to the induction hypothesis we have already found the $(i-1)$ distinct best paths to the node n. Let us assume that the cost of the newly found solution path is greater than the i^{th} optimal one, i.e., $g_i(n) > g_i^*(n)$. Then there exists a different, undiscovered path π from s to n with cost $g_\pi(n) = g_i^*(n) < g_i(n)$. From Proposition 3.2 there exists on OPEN a node $n_0 \in \pi$. Obviously node n_0 must be located between the start node s and node n. Denoting by $C_\pi(n_0, n) = c(n_0, n_1) + \cdots + c(n_k, n)$, from the heuristic consistency it easily follows that $h(n_0) < C_\pi(n_0, n) + h(n)$ and that the evaluation function of node n_0 along path π is $f_\pi(n_0) = g_\pi(n_0) + h(n_0) < g_\pi(n_0) + C_\pi(n_0, n) + h(n)$. Seeing that the cost of path π from s to n is $g_\pi(n) = g_\pi(n_0) + C_\pi$, we conclude that $f_\pi(n_0) < f_\pi(n)$. However, that contradicts our assumption that node n was expanded for the i^{th} time before node n_0. The theorem follows. □

■ 3.2.2.4 The Impact of the Required Number of Best Solutions m

The sequence of the sizes of search space explored by m-A* as a function of m is obviously monotonically increasing with m. Denoting by j-A* and i-A* the versions of m-A* algorithm that search respectively for j and i best solutions, we can make the following straightforward characterization:

Proposition 3.8. *Given a search graph and consistent heuristic,*

1. *Any node expanded by i-A* is expanded by j-A* if $i < j$ and if both use the same tie-breaking rule.*
2. *The set $S(i,j)$ of nodes defined by $S(i,j) = \{n | C^*_i < f(n) < C^*_j\}$ will surely be expanded by j-A* and surely not expanded by i-A*.*
3. *If $C^*_j = C^*_i$, the difference in the number of nodes expanded by i-A* and j-A* is determined by the tie-breaking rule.*

The proof follows trivially from Theorem 3.7. As a result, larger discrepancy between the respective costs $C^*_j - C^*_i$ yields larger difference in the search spaces explored by j-A* and i-A*. This difference, however, also depends on the granularity with which the values of a sequence of observed evaluation functions increase, which is related to the arc costs (or weights) of the search graph. If $C^*_i = C^*_j = C$, then the search space explored by i-A* and j-A* will differ only in the frontier of nodes satisfying $f(n) = C$. Figure 3.4 schematically represents the explored search spaces of i-A* algorithm.

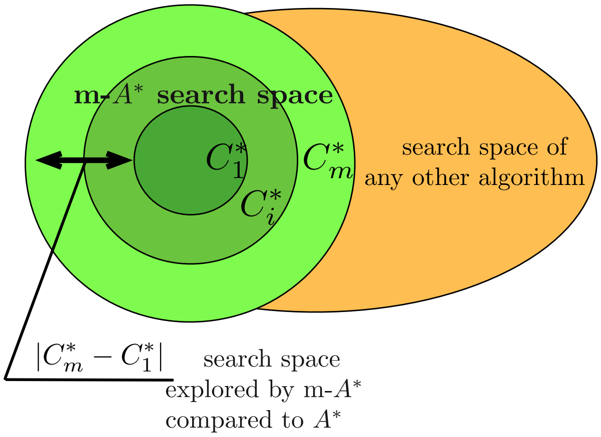

m-A^* search space
C_1^*
C_m^*
C_i^*
search space of any other algorithm
$|C_m^* - C_1^*|$
search space explored by m-A^* compared to A^*

Figure 3.4: The schematic representation of the search spaces explored by m-A* algorithm, depending on m and cost C_m^*

■ 3.3 Depth-First Branch and Bound for Finding the M-best Solutions

Along with its valuable properties, m-A* inherits also the disadvantages of A*: its exponential space complexity, which makes the algorithm infeasible for many applications. An alternative approach is searching using depth-first branch and bound (*DFBB*), which can be implemented in linear space if necessary and is therefore often more practical. DFBB finds the optimal solution by exploring the search space in a depth first manner. The algorithm maintains a cost U of the best solution encountered so far and prunes search nodes whose lower-bounding evaluation function $f(n) = g(n) + h(n)$ is larger than U. Extending DFBB to the m-best task is straightforward, as we describe next.

■ 3.3.1 The m-BB Algorithm

Algorithm m-BB, the depth-first branch and bound extension to the m-best task, that explores a search tree is presented in Algorithm 11. As usual, the algorithm maintains lists of OPEN and CLOSED nodes. The algorithm also maintains a sorted list of CANDIDATE that contains the best m solutions found so far. Nodes on OPEN are organized in a ”last in - first out” manner in order to facilitate depth-first exploration of the search space. At each step m-BB expands the next node n on OPEN (line 5). If it is a goal node, a new complete solution is found (lines 7-8) and it is stored on the CANDIDATE list (line 9), which is then re-sorted (line 10). Only up to m best solutions are maintained (lines 11-13).

The main modification of depth-first branch and bound, when extended to the m-best task, is in its pruning condition. Let $U_1 \leq U_2 \leq \ldots \leq U_m$ denote the costs of the m best solutions encountered thus far,then U_m is the upper bound used for pruning. Before m solutions are discovered, no pruning takes place. Algorithm m-BB expands the current node n, generates its children (lines 15-17) and computes their evaluation function (line 18-19). It prunes a subproblem below n iff $f(n) \geq U_m$ (lines 20-23). It is easy to see that when the algorithm terminates, it outputs the m-best solutions to the problem.

Theorem 3.9. *Algorithm m-BB is sound and complete for the m-best solution task.*

Proof. Algorithm m-BB explores the search space systematically. The only solutions that are skipped are the ones satisfying $f(n) \geq U_m$ (see lines 20-21). Since $U_m \geq C^*_m$, where C^*_m is the m best solution cost, it implies $f(n) \geq C^*_m$ and therefore that path cannot lead to a newly discovered m-best cost. □

Theorem 3.10. *Given a graphical model $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \bigotimes \rangle$, the worst case time complexity of m-BB that explores an OR search tree of $\mathcal{M}$ is $O(k^n + \log m)$, where n is the number of variables, k is the domain size and m is the number of required solutions. Space complexity is $O(n)$.*

Proof. In worst case m-BB would explore the entire OR search tree of size $O(k^n)$. The maintaining of CANDIDATE list introduces additional time overhead of $O(\log m)$. Since the OR search tree yields no caching, m-BB uses space linear in the number of variables. □

Algorithm 11: Algorithm m-BB exploring OR tree

```
   Input: An implicit directed search graph $G = (N, E)$ with a start node $n_0$ and a set of goal nodes
          $Goals$. A heuristic evaluation function $h(n)$. Parameter $m$ (the number of desired solutions).
   Output: the $m$ best solutions
1  Initialize: OPEN=$\emptyset$, CLOSED=$\emptyset$, a tree $Tr = \emptyset$, sorted list CANDIDATE = $\emptyset$, UpperBound = $-\infty$,
   $i = 1$ ($i$ counts the current solution being searched for)
2  Put the start node $n_0$ in OPEN, $g(n_0) = 0$, $f(n_0) = h(n_0)$
3  Assign $n_0$ to be the root of $Tr$
4  while OPEN is not empty do
5      Remove the top node from OPEN, denoted $n$, and put it on CLOSED.
6      if $n$ is a goal node then
7          $sol_i \leftarrow$ solution obtained by tracing back pointers from $n$ to $n_0$ (pointers assigned at step 17)
8          $C_i \leftarrow$ cost of $sol_i$
9          place solution $sol_i$ on CANDIDATE
10         sort CANDIDATE in increasing order of solution costs
11         if size of CANDIDATE list $\geq m$ then
12             $U_m \leftarrow$ cost of the $m^{th}$ element in CANDIDATE
13             Keep first $m$ elements of CANDIDATE, discard the rest
14         else
15             Expand node $n$, generating its children
16             forall the $n' \in Successors\ of\ n$ do
17                 Attach a pointer from $n'$ back to $n$ in $Tr$
18                 $g(n') = g(n) + c(n, n')$;
19                 $f(n') = g(n') + h(n')$
20                 if $f(n') < U_m$ then
21                     Place $n'$ on OPEN
22                 else
23                     Discard $n'$
24 return the solutions on CANDIDATE list
```

3.3.2 Characterization of Search Space Explored by m-BB

We have already shown that m-A* is superior to any exact search algorithm for m-best solutions when heuristic is consistent (Theorem 3.6). In particular, m-BB must expand all the nodes that are surely expanded by m-A*, namely the set of nodes $\{n | f(n) < C^*_m\}$. From Theorem 3.6 and the pruning condition it is clear that:

Proposition 3.9. *Given a consistent heuristic m-BB must expand any node in the set* $\{n | f(n) < C^*_m\}$. *Also, there are instances for which m-BB will expands nodes satisfying* $f(n) > C^*_m$.

Several sources of overhead of m-BB are discussed next.

m-BB vs. BB. Pruning in m-BB does not occur until the upper bound on the current m^{th} best solution is assigned a valid value, i.e., until m solutions are found. In the absence of determinism, when all solutions are consistent, the time it takes to find m arbitrary solutions in depth-first manner is $O(m \cdot n)$, where n is the length of solution and is equal to the number of problem variables. If the problem contains determinism it may be difficult to find even a single solution. This means that for m-BB the search may be exhaustive for quite some time.

The impact of solution order. The difference in the number of nodes expanded by BB and m-BB depends greatly on the variance between the solution costs. If all the solutions have the same cost, then $U_1 = U_m$. However, such a situation is unlikely and therefore the conditions for m-BB's node expansions are impacted by the order in which solutions are discovered. Let $\{U_m^1, \dots, U_m^j\}$ be the non-increasing sequence of the upper bounds on the m^{th} best solution, up to a point when m-BB uncovered the j^{th} solution. Initially $U_m^j = \infty$, for $j \in [1, m-1]$.

Proposition 3.10. *Between the discovery of the $(j-1)^{th}$ and the j^{th} solutions the set of nodes expanded by m-BB are included in $S_j = \{n|f(n) \leq U_m^{j-1}\}$, where $C_m^* \leq U_m^j \leq U_m^{j-1} \leq \infty$.*

Proof. Between discovering the $(j-1)^{th}$ and j^{th} solutions m-BB expands only nodes satisfying $\{n|f(n) \leq U_m^{j-1}\}$, hence $\forall j : C_j \leq U_m^{j-1}$. Once the j^{th} solution is found, it either replaces the previous bound on m^{th} solution $U_m^j = C_j$ or some k^{th} upper bound, $k \in [1, m-1]$, yielding $U_m^j = U_{m-1}^{j-1}$. Either way, $C_m^* \leq U_m^j \leq U_m^{j-1}$. □

Ordering overhead. The need to keep a list of m sorted solutions (the CANDIDATE list) implies $O(\log m)$ overhead for each new solution discovered. The total number of solutions encountered before termination is hard to characterize.

Caching overhead. The overhead related to caching arises only when m-BB explores a search graph and uses caching. This version of the algorithm (not explicitly presented) stores the m best partial solutions to any fully explored subproblems (and a subset of m when only a partial set is discovered) and re-uses these results whenever the subproblem is encountered again. In order to implement caching, m-BB requires to store a list of length m for each node that is cached. Moreover, the cached partial solutions need to be sorted, which yields an $O(m \log m)$ time overhead *per cached node*.

■ 3.4 M-best Best-First Search for Graphical Models

Our main task is to find the m best solutions to optimization tasks over graphical models. Therefore we adapt the m-best search algorithms m-A* and m-BB to explore the AND/OR search space over graphical models, yielding algorithms m-AOBF and m-AOBB. We will also describe a hybrid algorithm *BE+m-BF*, combining Bucket Elimination and m-A*.

■ 3.4.1 Introducing M-best Best-First Search to Graphical Models

The extension of algorithm AOBF (Section 1.2.4.2) to the m-best task seems fairly straightforward, in principle. m-AOBF is AOBF that continues searching after discovering the first solution, until the required number of m best solutions is obtained. The actual implementation requires several modifications as we discuss next.

It is not easy to extend AOBF's bottom-up node values updates and corresponding arc marking mechanism to the m-best task. Therefore, in order to keep track of the current best partial solution tree while searching for the i^{th} best solution we adopt a naive approach that maintains explicitly a list OPEN containing entire partial solution trees (not just nodes), sorted in ascending order of their heuristic evaluation costs. Algorithm 12 presents the pseudo-code of our simple scheme that explores the AND/OR search tree and generates solutions one by one in order of their costs. At each step, the algorithm removes the next partial solution tree T' from OPEN (line 4). If T' is a complete solution, it is added to the list of solutions along with its cost (lines 5-8), otherwise the algorithm expands a tip node n of T', generating its successors (line 10-17). Each such newly generated node n' is added to T' separately, yielding a new partial solution tree T'' (lines 19-23), whose cost is recursively evaluated using Algorithm 5, as in AOBB (line 28). These new partial trees are then placed in OPEN (line 29). Search stops when all m solutions have been found.

We note that the maintenance of the OPEN list containing explicit partial solution subtrees is a source of significant additional overhead which will become apparent in the empirical evaluation from Section 3.6. Thus, the question whether the performance of m-AOBF can be improved further is open and is therefore a rich topic of future work.

All m-A* properties (Section 3.2.2) can be extended to m-AOBF. In particular, algorithm m-AOBF with an admissible heuristic is sound and complete, terminating with the m best solutions generated in order of their costs. m-AOBF is also optimal in terms of the number of nodes expanded compared with any other algorithm that explores the same AND/OR search space with the same consistent heuristic function.

■ **3.4.1.0.1 Complexity of m-AOBF** We discuss the algorithm's complexity as a function of the underlying search space, considering both the tree- and graph-exploring versions.

Theorem 3.11 (**Complexity of m-AOBF**). *The complexity of algorithm m-AOBF traversing either the AND/OR search tree or the context-minimal AND/OR search graph is time and space $O(k^{deg^{h-1}})$, where h is the depth of the underlying pseudo-tree, k is the maximum domain size, and deg bounds the degree of the nodes in the pseudo-tree. If the pseudo-tree is balanced (i.e., each internal node has exactly deg child nodes), then the time and space complexity is $O(k^n)$, where n is the number of variables.*

Algorithm 12: m-AOBF exploring an AND/OR search tree

```
Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, pseudo-tree $\mathcal{T}$ rooted at $X_1$, heuristic function $h(\cdot)$,
       parameter $m$;
Output: The $m$ best solutions to $\mathcal{M}$
Create root OR node $s$ labelled by $X_1$, let $\mathcal{G} = \{s\}$ (explored search space) and $T = \{s\}$ (partial
solution tree);
Initialize $\mathcal{S} \leftarrow \emptyset$; $OPEN \leftarrow \{T\}$; $i = 1$; ($i$ counts the current solution being searched for);
while $i \leq m$ and $OPEN \neq \emptyset$ do
    Select the top partial solution tree $T'$ and remove it from OPEN;
    if $T'$ is a complete solution then
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{\langle f(T'), T' \rangle\}$;
        $i \leftarrow i + 1$;
        continue;
    Select a non-terminal tip node $n$ in $T'$;
    // Expand node $n$
    if $n$ is OR node labeled $X_i$ then
        forall the $x_i \in D(X_i)$ do
            Create AND child $n'$ labeled $\langle X_i, x_i \rangle$;
            $succ(n) \leftarrow succ(n) \cup \{n'\}$;
    else if $n$ is AND node labeled $\langle X_i, x_i \rangle$ then
        forall the successor $X_j$ of $X_i$ in $\mathcal{T}$ do
            Create an OR child $n'$ labeled $X_j$;
            $succ(n) \leftarrow succ(n) \cup \{n'\}$;
    $\mathcal{G} \leftarrow \mathcal{G} \cup \{succ(n)\}$;
    // Generate new partial solution trees
    $\mathcal{L} \leftarrow \emptyset$;
    forall the $n' \in succ(n)$ do Initialize $v(n') = h(n')$;
    if $n$ is OR node then
        forall the $n' \in succ(n)$ do
            Create a new partial solution tree $T'' \leftarrow T' \cup \{n'\}$;
            $\mathcal{L} \leftarrow \mathcal{L} \cup \{T''\}$;
    else if $n$ is AND node then
        Create a new partial solution tree $T'' \leftarrow T' \cup \{succ(n)\}$;
    forall the $T'' \in \mathcal{L}$ do
        Recursively evaluate and assign to $f(T'')$ the cost of the partial solution tree $T''$, based on
        heuristic function $h(\cdot)$; // see Algorithm 5
        Place $T''$ in OPEN, keeping it sorted in the ascending order of costs $f(T'')$;
return The $m$ best solutions found $\mathcal{S}$;
```

The proof of the theorem can be found in Appendix B.1.

Algorithm 13: m-AOBB exploring an AND/OR search tree

```
Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, pseudo-tree $\mathcal{T}$ rooted at $X_1$, heuristic function $h(\cdot)$, parameter $m$;
Output: The $m$ best solutions to $\mathcal{M}$
// INITIALIZE
Create root OR node $s$ labeled by $X_1$ and let the stack of created but not expanded nodes $OPEN = \{s\}$;
Initialize $\overline{v}(s) = \overline{\infty}$ (a set of bounds on $m$ best solutions under $s$) and a set of best partial solution trees rooted in $s$ $\overline{T^*}(s) = \emptyset$; $UB = \infty$, sorted list $CANDIDATE = \emptyset$;
while $OPEN \neq \emptyset$ do
    Select top node $n$ in OPEN;
    // EXPAND
    if $n$ is OR node labeled $X_i$ then
        foreach $x_i \in D(X_i)$ do
            Add AND child $n'$ labeled $\langle X_i, x_i \rangle$ to list $succ(n)$ containing the successors of $n$;
            Initialize $\overline{v}(n') = \overline{0}$, a set of best partial solution trees rooted in $n$ $\overline{T^*}(n') = \overline{\emptyset}$;
    if $n$ is AND node labeled $\langle X_i, x_i \rangle$ then
        Let $p$ be ancestor of $n$;
        Recursively evaluate and assign to $f(p)$ the cost of the partial solution tree rooted in $p$, based on the heuristic $h(\cdot)$; // see Algorithm 5
        if $v_m(p) < \infty$ and $f(p) \geq v_m(p)$ then
            Prune the subtree below the current tip node $n$;
        else
            foreach successor $X_j$ of $X_i \in \mathcal{T}$ do
                Add OR child $n'$ labeled $X_j$ to list $succ(n)$ containing the successors of $n$;
                Initialize $\overline{v}(n') = \overline{\infty}$, a set of best partial solution trees rooted in $n$ $\overline{T^*}(n') = \overline{\emptyset}$;
    Remove $n$ from OPEN and add $succ(n)$ on top of OPEN;
    // PROPAGATE
    while list of successors of node n is empty do
        if $n$ is the root node then
            return a set of solutions rooted at n and their costs: $\overline{T^*}(n), \overline{v}(n)$ ;
        else
            Update ancestors of $n$, AND and OR nodes $p$, bottom up:
            if $p$ is AND node then
                Combine the set of the partial solution trees to the subproblem rooted in $p$ $\overline{T^*}(p)$ and the set of partial solution trees rooted in $n$ $\overline{T^*}(n)$ and their costs $\overline{v}(p)$ and $\overline{v}(n)$; // see Algorithm 14
                Assign the resulting set of the costs and the set of the best partial solution trees respectively to $\overline{v}(p)$ and $\overline{T^*}(p)$;
            else if $p$ is OR node then
                foreach solution cost $v_i(n)$ in the set $\overline{v}(n)$ do
                    Update the cost with the weight of the arc, creating a new set of costs $\overline{v'}(n)$: $v'_i(n) = c(p, n) + v_i(n)$;
                Merge the sets of partial solutions $\overline{v}(n)$ and $\overline{v}(p)$ and the sets of partial solution trees rooted in $p$ and $n$: $\overline{T^*}(p)$ and $\overline{T^*}(n)$, keeping $m$ best elements; //Algorithm 15
                Assign results of merging respectively to $\overline{v}(p)$ and $\overline{T^*}(p)$;
            Remove $n$ from the list of successors of $p$;
            Move one level up: $n \leftarrow p$;
return $\overline{v}(s)$ and $\overline{T^*}(s)$
```

■ 3.4.2 m-AOBB: M-best BB for Graphical Models

Algorithm m-AOBB extends the AND/OR Branch and Bound search (AOBB, Section 1.2.4.3) to the m-best task. The main difference between AOBB and m-AOBB is in the value function computed for each node.

m-AOBB tracks the costs of the m best partial solutions of each solved subproblem. Thus it extends the node value $v(n)$ and solution tree $T^*(n)$ rooted by n in AOBB to ordered sets of length m, denoted by $\overline{v}(n)$ and $\overline{T^*}(n)$, respectively, where $\overline{v}(n) = \{v_1(n), \ldots, v_m(n)\}$ is an ordered set of the costs of the m best solutions to the subproblem rooted by n, and $\overline{T^*}(n) = \{T_1^*(n), \ldots, T_m^*(n)\}$ is a set of corresponding solution trees. This extension arises due to the depth-first manner of search space exploration of m-AOBB in conjunction with the AND/OR decomposition. Therefore, due to the AND/OR decomposition m-AOBB needs to completely solve the subproblems rooted in all the children n' of an AND node n, before even a single solution to a subproblem above n is acquired (unlike the m-BB case). Consequently, during the bottom-up phase sets of m costs have to be propagated and updated. m-AOBF on the other hand, only maintains a set of partial solution trees.

Unlike m-AOBF that discovers solutions one by one in order of their costs, m-AOBB (pseudo-code in Algorithm 13) reports the entire set of m solutions at once, at termination. m-AOBB interleaves forward node expansion (lines 5-18) with a backward propagation (or cost revision) step (lines 19-33) that updates node values until search terminates. A node n will be pruned (lines 12-13) if the current upper bound on the m^{th} solution under n, $v_m(n)$, is lower than the node's evaluation functions $f(n)$, which is computed recursively as in AOBB (Algorithm 5). During the bottom-up propagation phase at each AND node the partial solutions to the subproblems rooted in the node's children are combined (line 24-26, Algorithm 14). At each parent OR node p $\overline{v}(p)$ and $\overline{T^*}(p)$ are updated to incorporate the new and possibly better partial solutions rooted in a child node n (lines 27-31, Algorithm 15).

■ 3.4.2.0.2 Characterizing node processing overhead

In addition to the increase in the explored search space that m-BB experiences compared with BB due to the reduced pruning (Section 3.3.2), AND/OR search introduces additional overhead for m-AOBB. The

Algorithm 14: Combining the sets of costs and partial solution trees

```
function Combine($\overline{v}(n)$, $\overline{v}(p)$, $\overline{T^*}(n)$,$\overline{T^*}(n)$)
Input: Input sorted sets of costs $\overline{v}(n)$, $\overline{v}(p)$, corresponding partial solution trees $\overline{T^*}(n)$, $\overline{T^*}(p)$,
       number of required solutions $m$
Output: A set of costs $m$ best combined solutions $\overline{v'}(p)$, corresponding partial solution trees $\overline{T^{*'}}(p)$
// INITIALIZE
Sorted list OPEN, initially empty; //contains potential cost combinations
$\overline{v'}(p) \leftarrow \emptyset$; $\overline{T^{*'}}(p) \leftarrow \emptyset$;
$k = 1$; //number of partial solutions already assembled, up to $m$ in total
// Search over possible combinations
OPEN$\leftarrow v_1(n) + v_1(p)$;
while $k < m$ and $OPEN$ is not empty do
    Remove the top node $V$ on OPEN, where $V = Sv_i(n) + v_j(p)$;
    $v'_k(p) \leftarrow V$;
    $T^{*'}(p) \leftarrow T^*_i(n) \cup T^*_j(p)$;
    if $v_{i+1}(n) + v_j(p)$ not in $OPEN$ then
        Put $v_{i+1}(n) + v_j(p)$ in $OPEN$;
    if $v_i(n) + v_{j+1}(p)$ not on $OPEN$ then
        Put $v_i(n) + v_{j+1}(p)$ in $OPEN$;
    $k \leftarrow k + 1$;
return $\overline{v'}(p)$, $\overline{T^*(p)}$;
```

propagation of a set of m costs and of m partial solution trees leads to an increase in memory by a factor of m per node. Processing the partial solutions at both OR and AND nodes introduces an additional overhead.

Theorem 3.12 (Complexity of m-AOBB). *Algorithm m-AOBB exploring the* AND/OR search tree *has a time overhead of* $O(m \cdot deg \cdot \log m)$ *per AND node and* $O(m \cdot k)$ *per OR node, where deg bounds the degree of the pseudo-tree and k is the largest domain size. Assuming* $k < deg \cdot \log(m)$*, the total worst case time complexity is* $O(n \cdot k^h deg \cdot m \log(m))$ *and the space complexity is* $O(m \cdot n)$*. The time complexity of m-AOBB exploring the* AND/OR search graph *is* $O(n \cdot k^{w^*} deg \cdot m \log(m))$*, space complexity* $O(mn \cdot k^{w^*})$*.*

Proof. Combining the sets of current m-best partial solutions (Algorithm 14) introduces an overheard of $O(m \log(m))$. The resulting time overhead per AND node is $O(deg \cdot m \log(m))$. Merging two sorted sets of costs (Algorithm 15) can be done in $O(m)$ steps. If an OR node has $O(k)$ children, the resulting overhead is $O(m \cdot k)$. Assuming $k < deg \cdot \log(m)$,

Algorithm 15: Merging the sets of costs and partial solution trees

```
function Merge($\overline{v}(n)$,$\overline{v}(p)$,$\overline{T^*}(n)$,$\overline{T^*}(p)$)
Input: Input sorted cost sets $\overline{v}(n)$ and $\overline{v}(p)$, sets of corresponding partial solution trees $\overline{T^*}(n)$ and
       $\overline{T^*}(p)$, number of required solutions $m$
Output: $\overline{v'}(p)$, a merged set of $m$ best solution costs, $\overline{T^{*'}}(p)$ a set of corresponding partial solution
        trees
// INITIALIZE
$\overline{v'}(p) \leftarrow \emptyset$;
$\overline{T^{*'}}(p) \leftarrow \emptyset$;
$i, j \leftarrow 1$; //indices in the cost sets
$k \leftarrow 1$; //index in the resulting array
// Merge two sorted sets
while $k \leq m$ do
    if $v_i(p) \leq v_j(n)$ then
        $v'_k(p) \leftarrow v_i(p)$;
        $T^{*'}_k(p) \leftarrow T^*_i(p)$;
        $i \leftarrow i + 1$;
        $k \leftarrow k + 1$;
    else
        $v'_k(p) \leftarrow v_j(n)$;
        $T^{*'}_k(p) \leftarrow T^*_j(n)$;
        $j \leftarrow j + 1$;
        $k \leftarrow k + 1$;
return $\overline{v'}(p)$ and $\overline{T^{*'}}(p)$;
```

the complexity is dominated by processing of the AND nodes. In the worst case, the tree version of m-AOBB, called m-AOBB-tree, would explore the complete search space of size $O(n \cdot k^h)$, where h bounds the depth of the pseudo -tree, while the graph version, called m-AOBB-graph, would visit a space of of size $O(n \cdot k^{w^*})$, where w^* is the induced width of the pseudo-tree. The space complexity of m-AOBB-tree follows from the need to propagate the sets of $O(m)$ partial solutions of length $O(n)$. The time overhead for m-AOBB is the same for AND/OR trees and AND/OR graphs. The space complexity of m-AOBB-graph is explained by the need to store m partial solutions for each cached node. □

3.4.3 Algorithm BE+m-BF

It is known that exact heuristic for graphical models can be generated by the bucket elimination (BE, [19]) algorithm described in Section 1.2.2.1. We can therefore first compile the exact heuristics along an ordering using BE and then apply m-A* (or m-AOBF, both will work the same at this point), using these exact heuristics. The resulting algorithm is called BE+m-BF. Worst-case analysis of this algorithm will show that it yields the best worst-case complexity compared with any known m-best algorithm for graphical models.

Theorem 3.13 (**Complexity of BE+m-BF**). *The time complexity of BE+m-BF is bounded by $O(nk^{w*+1}+nm)$ when n is the number of variables, k is the largest domain size, w^* is the induced width of the problem and m is the desired number of solutions. The space complexity is $O(nk^{w*}+nm)$.*

Proof. BE's time complexity is $O(nk^{w*+1})$ and space complexity of $O(nk^{w*})$ [19]. Since BE compiles an exact heuristic function, m-A* with this exact heuristic expands nodes for which $f(n) = C^*_j$ only while searching for i^{th} solution. If the algorithm breaks ties in favor of deeper nodes, it will only expand nodes on solution paths. Each path has length n, yielding total time and space complexity of this step of the algorithm equal to $O(n \cdot m)$. □

■ 3.5 Related Work

We can distinguish several primary approaches employed by earlier m-best exact algorithms, some mentioned already in the introduction. Note that some of the original works we discuss here do not include space complexity analysis and the bounds provided are often our own.

The first and most influential approach was introduced by Lawler [61]. It aimed to use of-the-shelf optimization schemes for best solutions. Lawler showed how to extend any given optimization algorithm to the m-best task. At each step the algorithm seeks the best solution to a re-formulation of the original problem that excludes the solutions already discovered. The scheme has been improved over the years and is still one of the primary strategies for finding the m-best solutions. The time and space complexity bounds of Lawler's scheme are $O\big(n \cdot m \cdot T(n)\big)$ and $O(S(n))$ respectively, where $T(n)$ and $S(n)$ are the time and space complexity of finding a single best solution. For example, if we use AOBF as an underlying optimization algorithm, the use of Lawler's method yields time complexity of $O(n^2mk^{w^*})$ and space complexity of $O(nk^{w^*})$.

Hamacher and Queyranne [42] built upon Lawler's work, but used as building blocks algorithms that find both the first and second best solutions. Once two best solutions are generated, a new problem is formulated so that the second best solution is the best solution to the new problem. Then the second best solution for the new problem becomes the overall third best solution and the procedure is repeated. The algorithm has time complexity of $O(m \cdot T_2(n))$ and space complexity of $O(S_2(n))$, where $T_2(n)$ and $S_2(n)$ are respectively the time and space for finding the second best solution. The complexity of this method is always bounded from above by that of Lawler, seeing as Lawler's scheme can be used as an algorithm for finding the second best solution. Using m-AOBB to solve the 2-best task, we obtain time complexity of $O(2mnk^{w^*}deg\log 2)$ and space complexity $O(2nk^{w^*})$, or $O(mnk^{w^*}deg)$ and $O(nk^{w^*})$, respectively, if we discard the constants.

Nilsson [73] applied Lawler's method using a join-tree algorithm. On top of that, his algorithm reuses computations from previous iterations. His scheme, called max-flow algorithm, applies message-passing on a junction tree to calculate the initial max-marginal functions for each cluster (e.g., probability values of the most probable assignments task), yielding the best solution. Note that this step is equivalent to running the bucket-elimination algorithm. Subsequent solutions are recovered by conditioning search, which consults the generated functions. The time complexity [73] is $O(2p|C| + 2mp|R| + pm\log(pm))$, where p is the number of cliques in the joint tree, $|C|$ is the size of the largest clique and $|R|$ is the size of the largest residual (i.e., the number of variables in a particular cluster, but not in neighboring clusters). The space complexity can be bounded by $O(p|C| + p(|S|))$, where $|S|$ is the size of a separator between the clusters. If applied to a bucket-tree, Nilsson's scheme has time and space complexity of $O(2nk^{w*+1} + mn(2k + \log(mn))$ and $O(nk^{w^*+1} + nk^{w^*})$ respectively, since the the bucket tree has $p = n$ cliques, whose size is bounded by $|C| = k^{w^*+1}$, and the residual in each cluster is $|R| = k$ (the domain of a single variable). Thus the algorithm has better time complexity than all other schemes mentioned so far, except for BE+m-BF.

More recently, Yanover and Weiss [106] developed an iterative scheme based on belief propagation, called BMMF. At each iteration BMMF uses loopy Belief Propagation to solve two new problems obtained by restricting the values of certain variables. When applied to a junction tree having induced width w^* (whose largest cluster size is bounded by k^{w^*+1}), it is an exact algorithm having time complexity $O(2mnk^{w*+1})$ and space complexity $O(nk^{w*} + mnk)$. When applied on a loopy graph, BMMF is not guaranteed to find exact solutions.

Another approach based on Lawler's idea uses optimization via the LP-relaxation [99], formulated by Fromer and Globerson [37]. Their method, called Spanning TRee Inequalities and Partitioning for Enumerating Solutions (STRIPES) also partitions the search space, while systematically excluding all previously determined assignments. At each step new constraints are added to a LP optimization problem, which is solved via an off-the-shelf LP-solver. In general the algorithm is approximate. However, on trees or junction-trees it is exact, if the underlying LP solver reports solutions within the time limit. PESTEELARS is an extension of the above scheme by Batra [5]. It solves the LP relaxation using message-passing approach that, unlike conventional LP solvers, exploits the structure of the problem's graph. The complexity of these LP-based algorithm is hard to characterize using the usual graph parameters.

Another approach extends variable elimination (or dynamic programming) schemes to directly obtain the m best solutions. In Chapter 2 we extended bucket elimination and mini-bucket elimination to the m-best solutions, yielding an exact scheme elim-m-opt and its approximate version mbe-m-opt. The time complexity of the elim-m-opt is $O(nk^{w^*+1}m \log m)$, space is $O(mnk^{w^*})$.

Two related dynamic programming based ideas are by Seroussi and Golmard [84] and by Elliot [29]. Seroussi and Golmard extract the m solutions directly, by propagating the m best partial solutions along a junction tree. Given a junction tree with p cliques, largest cluster size $|C|$, separator size bounded by $|S|$ and branching degree deg, the time complexity of the algorithm is $O(m^2 \cdot p \cdot |C| \cdot deg)$ and the space complexity is $O(m \cdot p \cdot |S|)$. Adapted to bucket tree, this algorithm has a time complexity equal to $O(m^2nk^{w^*+1}deg)$ and a space complexity of $O(mnk^{w^*})$. Elliot propagates the m best partial solutions along a representation called Valued And-Or Acyclic Graph, also known as a smooth deterministic decomposable negation normal form (sd-DNNF) [15]. The time complexity of Elliot's algorithm is $O(nk^{w^*+1}m \log{(m \cdot deg)})$ and the space complexity is $O(mnk^{w^*+1})$.

Several methods focus on search schemes obtaining multiple optimal solution for the k shortest paths task (KSP). For a survey see [30]. The majority of these algorithms assume that the entire search graph is available in memory, and thus are not directly applicable. A recent exception is by Aljazzar, et al., [2], whose K^* algorithm finds the k shortest paths during search "on-the-fly" and thus can be potentially useful for graphical models. The algorithm interleaves A* search on the problem's implicit graph G and Dijkstra's algorithm [27] on a specific path graph structure denoted $P(G)$. $P(G)$ is a directed graph, the vertices of which correspond to edges in the problem graph G. Given a consistent heuristic, when applied to an AND/OR search graph, K^* has time and space complexity $O(nk^{w^*}w^* \log(n \cdot k) + m)$.

More recently, Gosh, et al., [39] introduced a best-first search algorithm for generating ordered solutions for *explicit* AND/OR trees or graphs. The time complexity of their algorithm can be bounded by $O(mnk^{w*})$, when applied to a context-minimal AND/OR search graph. The space complexity is bounded by $O(s \cdot nk^{w*+1})$, where s is the number of candidate solutions generated and stored by the algorithm, hard to quantify using usual graph parameters.

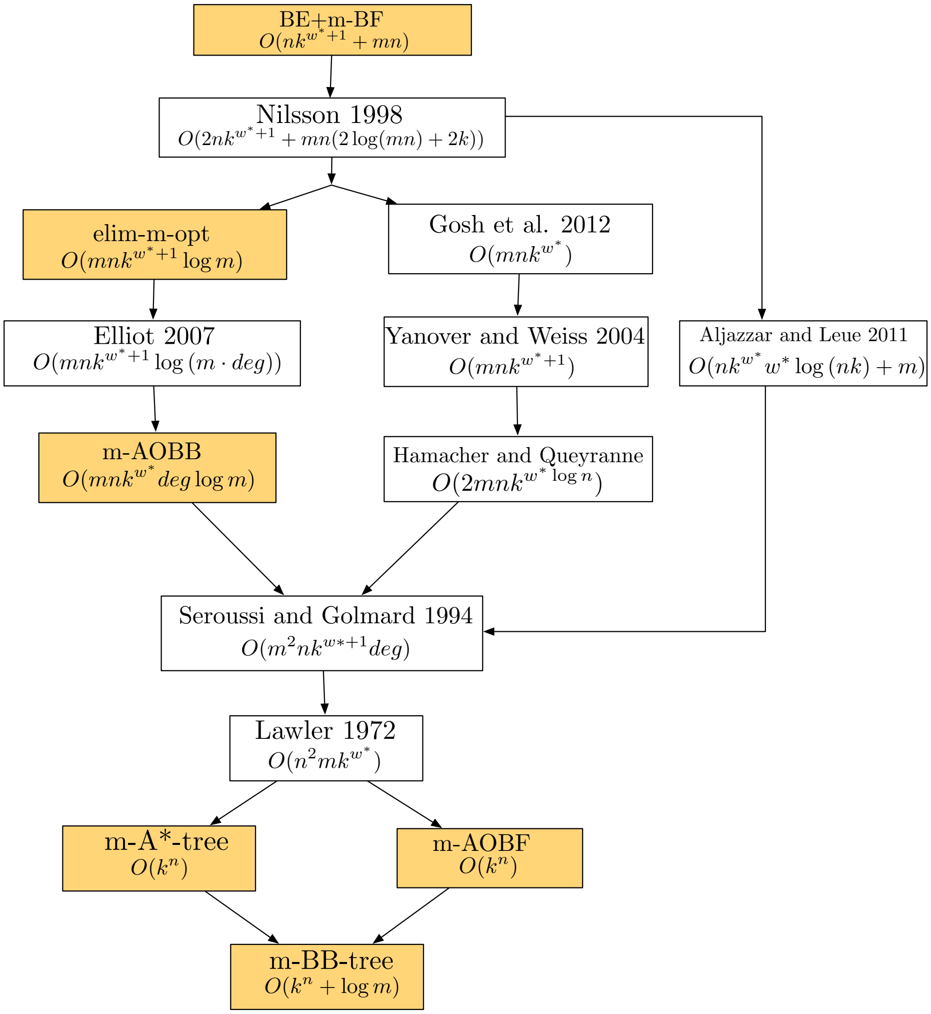


Figure 3.5: Time complexity comparison of the exact m-best algorithms specified for bucket tree. A parent node in the graph has a better complexity than its children. Problem parameters: n - number of variables, k - largest domain size, w^* - induced width, deg - the degree of the join (bucket elimination) tree. Our algorithms are highlighted.

However, this approach, which explores the space of complete solutions, does not seem to be practical for graphical models because it requires the entire AND/OR search space to be fully explicated in memory before attempting to generate even the second best solution. In contrast, our algorithms generate the m best solutions while traversing the space of partial solutions.

Figure 3.5 provides a visual comparison between the worst-case time complexity bounds of the discussed schemes in a form of a directed graph, where each node corresponds to an algorithm and a parent in the graph has a better complexity than its child. We assume in our analysis that $n >> m > k > 2$.

We see that the best scheme, as far as worst-case performance goes, is BE+m-BF. However, since it requires compiling of the exact heuristics, it is often infeasible. We see also that algorithm elim-m-opt appears to have relatively good time complexity, superior, e.g., to m-AOBB search. However, as we showed in the Chapter 2, it is quite limited empirically. Note that the worst-case analysis often fails to capture the practical behavior of algorithms, either because it ignores the power of the cost function in bounding the performance, or because the algorithms that have good worst-case runtime performance require too much memory.

■ 3.6 Experimental Results

Our experiments consist of two parts: evaluation of the m-best search algorithms on the benchmarks from recent UAI and Pascal2 competitions and comparison of our schemes with some of the previously developed algorithms on randomly generated networks, whose parameters and structure had to be restricted due to the limitations of the available implementations of the competing schemes. We defer the discussion of the second part of experiments till Section 3.6.5, concentrating now on evaluating our m-best search schemes only.

■ 3.6.1 Overview and Methodology

Evaluation on real-world problems. We used 6 benchmarks, all, except for binary grids, came from real world domains:

- Pedigrees
- Binary grids

Benchmark	# inst	n	k	w^*	h_T
Pedigrees	13	581-1006	3-7	16-39	52-104
Grids	32	144-2500	2	15-90	48-283
WCSP	61	25-1057	2-100	5-287	11-337
Promedas	86	197-2113	2	5-120	34-187
Proteins	72	15-242	18-81	5-16	7-44
Segmentation	47	222-234	2-21	15-18	47-67

Table 3.1: Benchmark parameters: # inst - number of instances, n - number of variables, k - domain size, w^* - induced width, h_T - pseudo-tree height.

- WCSP
- Promedas
- Proteins
- Segmentation

The benchmark parameters are shown in Table 3.1. Pedigree, Binary Grids and WCSP benchmark have been previously described in Section 2.6.1.

Protein side-chain prediction ("pdb*") networks correspond to side-chain conformation prediction tasks in the protein folding problem [105]. The resulting instances have relatively few nodes, but very large variable domains, generally rendering most instances very complex.

Promedas ("or_chain_*") and **segmentation ("*_s.binary")** are probabilistic networks that come from the set of problems used in the 2011 Probabilistic Inference Challenge[3]. **Promedas** instances are based on a Bayesian network model developed for expert systems for medical diagnosis [102]. **Segmentation** is a common benchmark used in computer vision, modeling as a MPE problem the task of image segmentation, namely assigning a label to every pixel in an image, such that pixels with the same label share certain characteristics.

■ 3.6.1.1 Algorithms

We can distinguish 6 algorithms: **BE+m-BF**, **m-A*-tree** and **m-BB-tree** exploring a regular OR search tree and their modifications that explore an AND/OR search tree, denoted

[3] http://www.cs.huji.ac.il/project/PASCAL/archive/mpe.tgz

m-AOBF-tree and **m-AOBB-tree**. We also consider a variant of m-AOBF that explores the AND/OR search graph, **m-AOBF-graph**. We did not implement the m-AOBB over AND/OR search graph, because the overhead due to book-keeping looked prohibitive. We used m-AOBF as representative for AND/OR graph search, and, as we will see, it proved indeed to be not cost-effective. All algorithms were guided by pre-compiled mini-bucket heuristics, described in Section 1.2.4.4. We used 10 i-bounds, ranging from 2 to 22. However, for some hard problems computing the mini-bucket heuristic with the larger i-bounds proved infeasible, so the actual range of i-bounds varies among the benchmarks and among instances within a benchmark. All algorithms were restricted to a static variable ordering computed using a min-fill heuristic [55]. Both AND/OR schemes explored the same pseudo-tree. In our implementation algorithms m-BB, m-BF and m-AOBF break ties lexicographically, algorithm m-AOBB solves the independent subproblems rooted at an AND node in increasing order of their lower bound heuristic estimates.

The algorithms were implemented in C++ (32-bit) and the experiments were run on a 2.6 GHz quad-core processor. The memory limit was set for 4 GB per problem, the time limit to 3 hours. We report the CPU time (in seconds) and the number of nodes expanded during search. For uniformity we consider the task throughout to be the maximization-product problem, also known as Most Probable Explanation task (MPE or MAP). We focus on complete and exact solutions only and thus do not report the results if the algorithm found less than m solutions (for best-first schemes) or if the optimality of the solutions was not proved (for branch and bound schemes).

■ 3.6.1.2 Goals of the Empirical Evaluation

We will address the following aspects:

1. Comparing best-first and depth-first branch and bound approaches

2. The impact of AND/OR decomposition on the search performance
3. Scalability of the algorithms with the number of required solutions m
4. Comparison with earlier proposed algorithms

■ 3.6.2 The Main Trends in the Behavior of the Algorithms

Tables 3.2, 3.4, 3.6, 3.8, 3.10, and 3.12 present for each of our algorithms the raw results in the form of runtime in seconds and number of expanded nodes for select instances from each benchmark, selected to best illustrate the prevailing trends. For each benchmark we show the results for two values of i-bound, corresponding, in most cases, to relatively weak and strong heuristics. Note that the i-bound has no impact on the BE+m-BF, since it always calculates the exact heuristic. We show three values of number of solutions m, equal to 1 (ordinary optimization problem), 10 and 100.

In order to see the bigger picture, in Figures 3.6-3.11 we show bar charts representing for each benchmark a median runtime and a number of instances solved by each algorithm for a particular level of heuristic for $m \in \{1, 2, 5, 10, 100\}$. The y-axis is on logarithmic scale. The numbers above the bars indicate the actual values of median time in seconds and number of solved instances, respectively. It is important to note that in these figures we only account for harder instances, for which the i-bound did not yield exact heuristic. We acknowledge that the median times are not strictly comparable since they are calculated over a varied number of instances solved by each algorithm. However, this metric is robust to outliers and gives us an intuition about the algorithms' relative success. In addition, Tables 3.3, 3.5, 3.7, 3.9, 3.9, 3.11 and 3.13 show for each benchmark the number of instances, for which a given algorithm is the best in terms of runtime and in terms of number of expanded nodes. If several algorithms show the same best result, it counts towards the score of all of them.

instance	i-bound	algorithm	number of solutions					
(n,k,w^*,h)			$m = 1$		$m = 10$		$m = 100$	
			time	nodes	time	nodes	time	nodes
503.wcsp	4	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	OOM		OOM		OOM	
		m-BB tree	Timeout		Timeout		Timeout	
		m-AOBB tree	Timeout		Timeout		Timeout	
		BE+m-BF	**0.05**	**6647**	**0.06**	**6671**	**0.06**	**6984**
(144, 4, 9, 44)	8	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	OOM		OOM		OOM	
		m-BB tree	1269.0	348648825	1275.65	348648869	1255.46	348651775
		m-AOBB tree	22.99	2320223	164.72	12110559	8010.42	568148386
		BE+m-BF	**0.05**	**6647**	**0.06**	**6671**	**0.06**	**6984**
myciel5g_3.wcsp	4	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	OOM		OOM		OOM	
		m-BB tree	OOT		OOT		OOT	
		m-AOBB tree	**1461.76**	**46419482**	**2389.32**	**74629839**	**3321.47**	**83802828**
		BE+m-BF	OOM		OOM		OOM	
(47,2, 19, 46)	8	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	OOM		OOM		OOM	
		m-BB tree	151.49	33563300	**152.27**	33609110	**148.08**	36255491
		m-AOBB tree	**107.03**	**4274313**	185.66	**7245553**	251.98	**8319419**
		BE+m-BF	OOM		OOM		OOM	
satellite01ac.wcsp	4	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	OOM		OOM		OOM	
		m-BB tree	Timeout		Timeout		Timeout	
		m-AOBB tree	**5760.25**	**14260410**	OOT		OOT	
		BE+m-BF	OOM		OOM		OOM	
(79, 8, 19, 56)	8	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	793.56	2579416	OOM		OOM	
		m-BB tree	Timeout		Timeout		Timeout	
		m-AOBB tree	**484.69**	**1530768**	**551.67**	**1995114**	**858.29**	**3507104**
		BE+m-BF	OOM		OOM		OOM	
29.wcsp	4	m-AOBF tree	10.23	464134	10.22	464182	10.29	464698
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	11.59	938812	11.58	938869	11.57	939508
		m-BB tree	12.96	2243619	12.89	2245137	12.77	2279587
		m-AOBB tree	1.81	87717	2.63	147851	115.3	9189667
		BE+m-BF	**0.0**	**111**	**0.0**	**168**	**0.01**	**739**
(83, 4, 18, 58)	8	m-AOBF tree	0.05	2347	0.05	2395	0.08	2899
		m-AOBF graph	0.09	1401	0.09	1447	0.13	1482
		m-A* tree	0.02	2098	0.02	2155	**0**.02	2724
		m-BB tree	0.17	37629	0.17	38463	0.25	55125
		m-AOBB tree	0.02	1577	0.33	24239	79.38	6731546
		BE+m-BF	**0.0**	**111**	**0.0**	**168**	**0.01**	**739**

Table 3.2: WCSP: CPU time (in seconds) and number of nodes expanded. A 'Timeout' stands for exceeding the time limit of 3 hours. 'OOM' indicates out of 4GB memory. In bold we highlight the best time and number of nodes for each m. Parameters: n - number of variables, k - domain size, w^* - induced width, h - pseudo-tree height.

We next provide some elaboration and interpretation of the results.

■ 3.6.2.1 WCSP

Table 3.2 shows the results for two values of i-bound for select instances chosen to best illustrate the common trends seen across the WCSP benchmark. Figure 3.6 presents the median time and number of solved instances for each algorithm for $i = 16$. Table 3.3 shows for the same i-bound the number of instances for which each of the schemes had the best

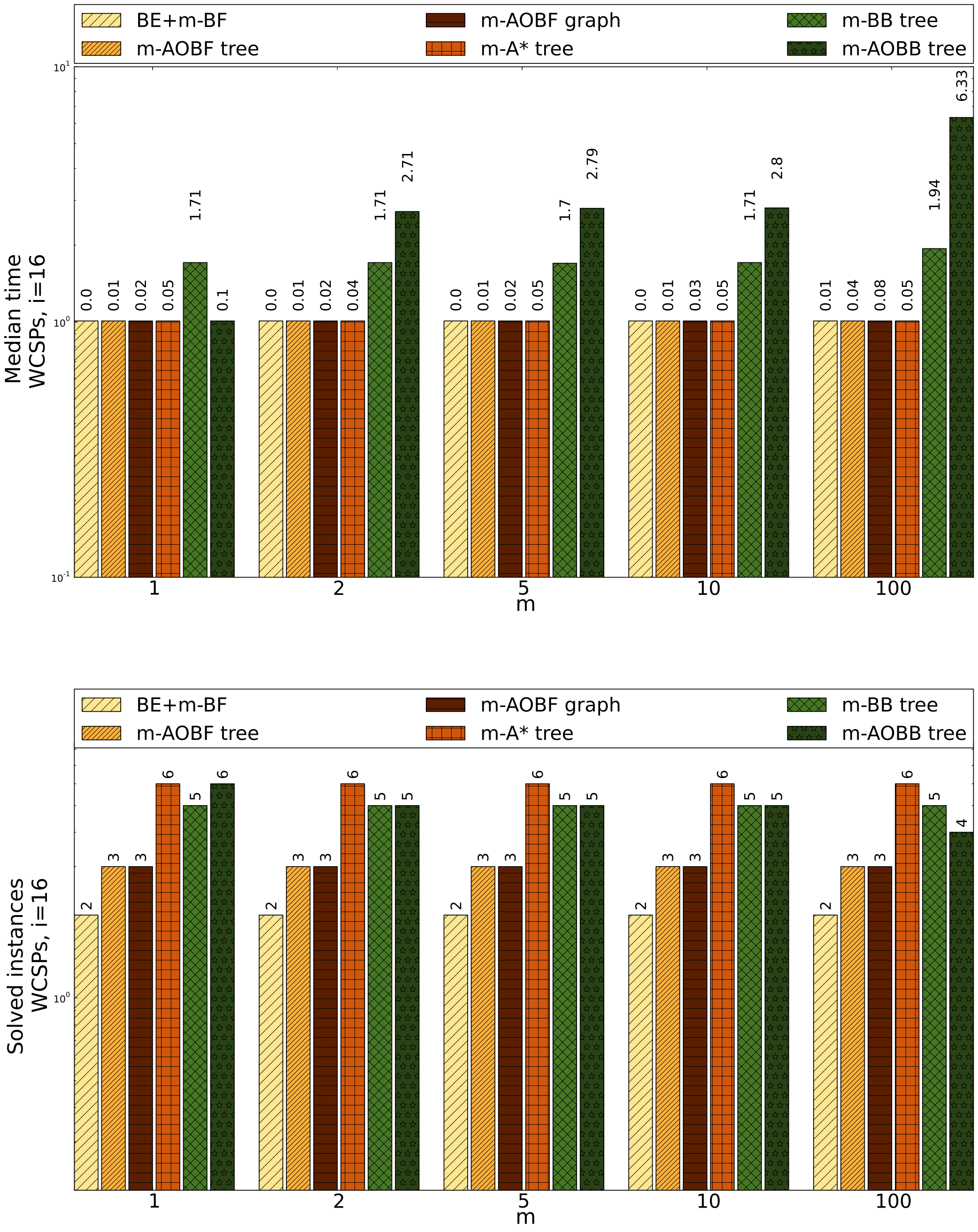


Figure 3.6: Median time and number of solved instances (out of 49) for select values of m for WCSPs, $i = 16$. Numbers above bars - actual values of time (sec) and # instances. Total instances in benchmark: 61, discarded instances due to exact heuristic: 12.

algorithm	WCSPs: # inst=61, n=14-1058 k=2-100, w^*=6-287, h_T=8-585, i-bound=16				
	$m = 1$	$m = 2$	$m = 5$	$m = 10$	$m = 100$
	#BT / #BN	#BT / #BN	#BT / #BN	#BT / #BN	#BT / #BN
Not solved	43	43	43	43	43
m-AOBF tree	1 / 2	1 / 1	0 / 1	1 / 1	0 / 0
m-AOBF graph	1 / 2	0 / 2	0 / 2	0 / 2	0 / 3
m-A* tree	5 / 1	4 / 3	5 / 3	4 / 3	5 / 2
m-BB tree	1 / 0	2 / 0	1 / 0	1 / 0	0 / 0
m-AOBB tree	1 / 2	2 / 0	0 / 0	0 / 0	0 / 0
BE+m-BF	2 / 1	2 / 1	2 / 1	2 / 1	2 / 1

Table 3.3: Number of instances, for which each algorithm has the best runtime (#BT) and best number of expanded nodes (#BN), WCSPs. Out of 61 instances 12 have exact heuristics. The table accounts for remaining 49, $i = 16$.

runtime and best number of expanded nodes. For many problems of this benchmark the mini-bucket elimination with the large i-bounds is infeasible, thus we present the results for a small and a medium i-bounds.

BE+m-BF. As suggested by theory, whenever BE+m-BF does not run out of memory, it is the most efficient scheme. See for example Table 3.2, 503.wcsp and 29.wcsp. However, calculation of the exact heuristic is only feasible for easier instances and, as Figure 3.6 shows, it can only solve 2 WCSP instances. As seen in Table 3.3, on these two instances BE+m-BF demonstrated the best runtime among all the schemes.

m-AOBB-tree. For a number of problems for small values of m, m-AOBB-tree is superior to m-BB-tree both in terms of the runtime and in number of expanded nodes. For example, for 29.wcsp, $i = 4$, $m = 10$ m-AOBB-tree requires 2.63 seconds to solve the problem and expands 147851 nodes while the runtime of m-BB-tree is 12.89 seconds and it expands 2245137 nodes. However, on the majority of instances m-AOBB-tree is slower than all other schemes, as seen in Figure 3.6. Moreover, m-AOBB-tree scales poorly with the number of solutions. For $m = 100$ very often it has both the worst runtime and the largest explored search space among all the schemes, e.g., $i = 8$, 503.wcsp. Such striking decrease in performance as m grows is consistent across various benchmarks and can be explained by the need to combine sets of partial solutions at AND nodes, as we described earlier. The overhead connected to AND/OR decomposition also accounts for the larger time per node ratio of m-AOBB-tree, compared to other schemes. For example, in Table 3.2 for instance myciel5g_3.wcsp, $i = 8$, for $m = 10$ and $m = 100$ m-AOBB-tree expands less nodes than m-BB-tree, but its runtime is larger. Nevertheless, m-AOBB-tree has its benefits. Since it is more space efficient than the other algorithms, it is often the only scheme able for find solutions for the harder instances, especially when the heuristic is weak, as we see for myciel5g_3.wcsp for i=4 and satellite01ac.wcsp for both $i = 4$ and $i = 8$.

m-BB-tree. In Figure 3.6 we see that m-BB-tree solves almost the same number of problems as m-AOBB-tree while having considerably better median time.

m-AOBF-tree and m-AOBF-graph. Unsurprisingly, best-first search algorithms often run out of space on problems feasible for branch and bound, such as 503.wcsp and myciel5g_3.wcsp for $i = 8$. m-AOBF-based schemes are overall inferior to other algorithms, solving, as Figure 3.6 shows, the least number of problems. Both schemes run out of memory much more often than m-A*-tree. We believe this is due to overhead of maintaining an OPEN list of partial solution trees, as opposed to an OPEN list of individual nodes as m-A*-tree does. Whenever the m-AOBF schemes do manage to find solutions, as for example for instance 29.wcsp, $i = 8$, m-AOBF-graph explores the smallest search space among the schemes, except for BE+m-BF. At the same time m-AOBF-tree sometimes expands more nodes than both m-AOBF-graph and m-A*-tree. m-AOBF-tree and m-AOBF-graph have almost the same median time and number of solved problems, as seen in Figure 3.6.

m-A*-tree. Out of the three best first algorithms m-A*-tree is overall the best. In Figure 3.6 we see that it solves more instances than all other schemes for all values of m and its median runtime is close to that by BE+m-BF. Table 3.3 proves that for $i = 16$ this scheme is the fastest among all the schemes on largest number of instances, showing best runtime on 4-5 instances, depending on m.

instance	i-bound	algorithm	number of solutions					
(n,k,w^*,h)			$m=1$		$m=10$		$m=100$	
			time	nodes	time	nodes	time	nodes
		m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
pedigree33	12	m-A* tree	OOM		OOM		OOM	
		m-BB tree	Timeout		Timeout		Timeout	
		m-AOBB tree	**7814.77**	**145203641**	Timeout		Timeout	
		BE+m-BF	OOM		OOM		OOM	
		m-AOBF tree	OOM		OOM		OOM	
(798, 4, 24, 132)		m-AOBF graph	OOM		OOM		OOM	
	22	m-A* tree	**1.32**	73625	**1.55**	**77138**	**3.76**	**112422**
		m-BB tree	2.98	145717	4.15	177397	21.48	655141
		m-AOBB tree	2.88	**70644**	Timeout		Timeout	
		BE+m-BF	OOM		OOM		OOM	
		m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
pedigree30	12	m-A* tree	OOM		OOM		OOM	
		m-BB tree	Timeout		Timeout		Timeout	
		m-AOBB tree	**2510.59**	**33453995**	OOT		OOT	
		BE+m-BF	OOM		OOM		OOM	
		m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
(1290, 5, 20, 105)	16	m-A* tree	65.43	4866388	65.86	4867551	67.01	4882985
		m-BB tree	84.28	12243789	85.72	12298570	127.25	13027245
		m-AOBB tree	594.36	6907399	Timeout		Timeout	
		BE+m-BF	**0.31**	**5039**	**0.4**	**6202**	**1.73**	**21636**
		m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
pedigree23	12	m-A* tree	8.44	713664	8.46	715729	11.15	904802
		m-BB tree	23.0	4446224	24.56	4676953	35.46	6179124
		m-AOBB tree	32.11	831096	1077.9	75355901	Timeout	
		BE+m-BF	**7.11**	**630**	**7.24**	**2482**	**7.68**	**19297**
		m-AOBF tree	0.12	867	0.21	1425	1.31	7912
		m-AOBF graph	0.16	**379**	0.32	**395**	2.03	**634**
(403, 5, 21, 64)	22	m-A* tree	**0.01**	493	**0.07**	2558	**0.5**	19297
		m-BB tree	0.03	1917	0.16	9913	1.74	111736
		m-AOBB tree	0.05	1474	610.93	44099247	Timeout	
		BE+m-BF	**0.01**	493	**0.07**	2558	**0.5**	19297
		m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
pedigree20	12	m-A* tree	27.0	2321986	26.66	2324701	27.47	2353927
		m-BB tree	34.23	7239379	37.63	7434961	66.81	9155747
		m-AOBB tree	88.85	4365855	1019.76	63940515	Timeout	
		BE+m-BF	**24.95**	**491**	**24.99**	**3482**	**26.16**	**32643**
		m-AOBF tree	1.89	16101	2.11	17049	4.32	26525
(438, 5, 20, 65)		m-AOBF graph	2.85	1512	3.18	**1554**	6.67	**1581**
	22	m-A* tree	0.14	6350	**0.23**	9065	**1.28**	38291
		m-BB tree	2.76	163985	3.12	184127	8.08	474074
		m-AOBB tree	**0.03**	**1230**	704.0	37534080	Timeout	
		BE+m-BF	0.14	6350	**0.23**	9065	**1.28**	3829

Table 3.4: Pedigrees: CPU time (in seconds) and number of nodes expanded. A 'Timeout' stands for exceeding the time limit of 3 hours. 'OOM' indicates out of 4GB memory. In bold we highlight the best time and number of nodes for each m.Parameters: n - number of variables, k - domain size, w^* - induced width, h - pseudo-tree height.

■ 3.6.2.2 Pedigrees

Table 3.4 displays the results for select instances from Pedigree benchmark for two i-bounds each. Overall, the difference between the results for the algorithms greatly diminishes as the heuristic strength increases. Figure 3.7 shows the median time and number of solved instances for select values of m for $i = 16$. The number of instances for which each of the schemes had the best runtime and best number of expanded nodes for the same i-bound is presented in Table 3.5.

algorithm	Pedigrees: # inst=13, n=335-1290 k=3-7, w^*=15-47, h_T=52-204, i-bound=16				
	$m = 1$	$m = 2$	$m = 5$	$m = 10$	$m = 100$
	#BT / #BN	#BT / #BN	#BT / #BN	#BT / #BN	#BT / #BN
Not solved	6	6	6	6	6
m-AOBF tree	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0
m-AOBF graph	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0
m-A* tree	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1
m-BB tree	0 / 0	0 / 0	1 / 1	1 / 1	1 / 1
m-AOBB tree	1 / 1	1 / 1	0 / 0	0 / 0	0 / 0
BE+m-BF	4 / 4	4 / 4	4 / 4	4 / 4	4 / 4

Table 3.5: Number of instances, for which each algorithm has the best runtime (#BT) and best number of expanded nodes (#BN), Pedigrees. Out of 13 instances 1 have exact heuristics. The table accounts for remaining 12, $i = 16$.

BE+m-BF. BE+m-BF is again often superior to other algorithms, especially when the other schemes use lower i-bounds, e.g., pedigree23, $i = 12$, all ms. For large i-bounds, and thus a more accurate heuristic, the difference is much smaller, e.g., pedigree23, $i = 22$. Table 3.5 shows that BE+m-BF is overall the fastest. We see that on the Pedigree benchmark this scheme is quite successful, as is evident from the many instances it solved (see Figure 3.7).

m-AOBB-tree. For low values of m m-AOBB-tree is slightly superior to all other algorithms, solving the most number of instances, (see Figure 3.7). On the other hand, its median time is the largest. It fails to solve any instances for $m = 100$. From Table 3.4 we see that m-AOBB-tree is the slowest, (e.g., pedigree23, $i = 22$, all ms). Yet, for instance pedigree33, $i = 12$, $m = 1$, this scheme is the only one to find any solution.

m-BB-tree. As expected, m-BB-tree is inferior to the best-first schemes unless the latter run out of memory. As was the case for WCSP, this scheme is often faster than m-AOBB-tree, as for example, for pedigree30, $i = 16$, all values of m. The bar charts show that m-BB-tree has the second worst median time for all values of m, but solves the most number of problems for $m = 100$.

m-AOBF schemes. Both m-AOBF algorithms are unsuccessful on Pedigree benchmark. They often run out of memory even for $m = 1$ (e.g., pedigree33, $i = 22$). For most instances,

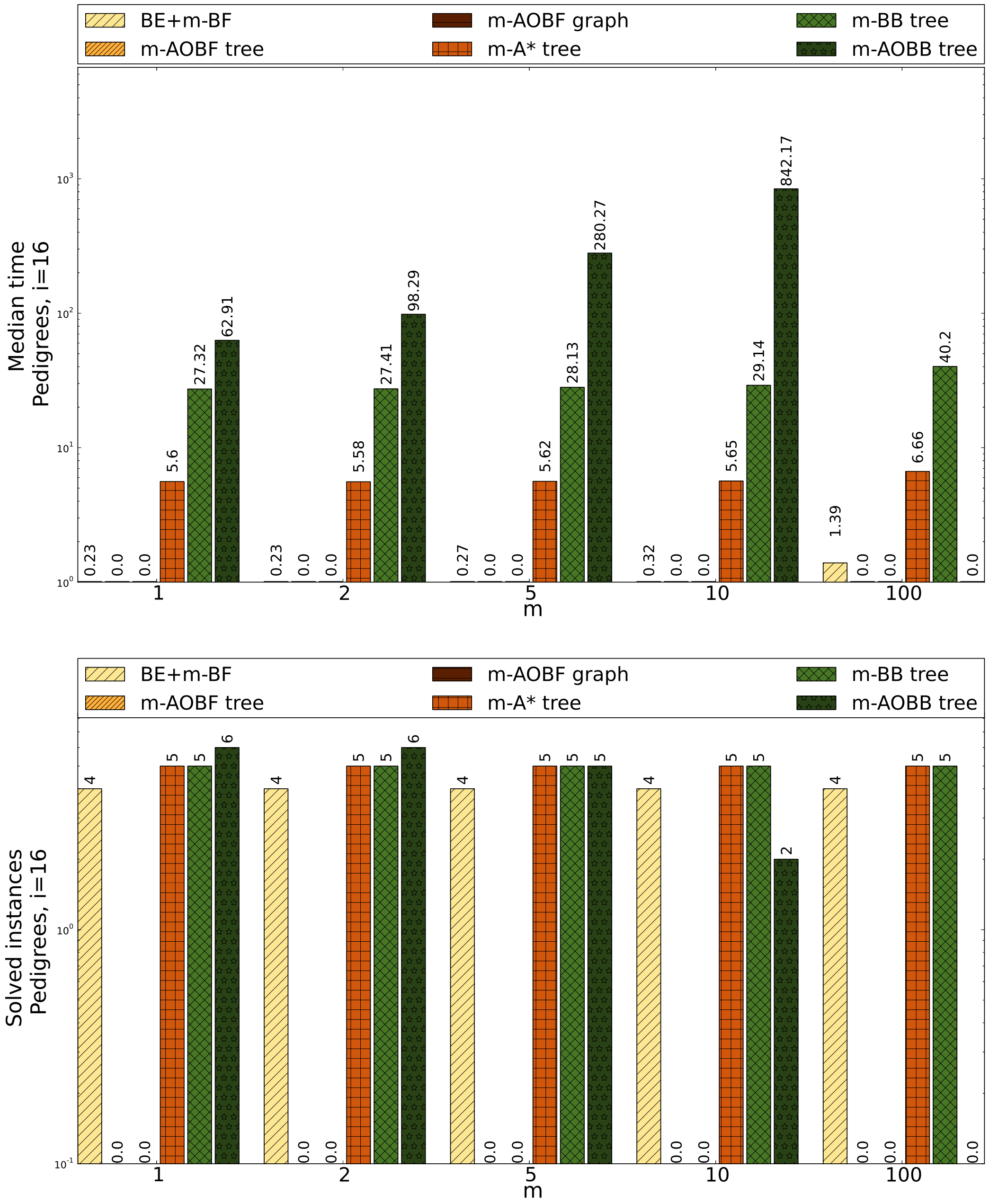


Figure 3.7: Median time and number of solved instances (out of 12) for select values of m for Pedigrees, $i = 16$. Numbers above bars - actual values of time (sec) and # instances. Total instances in benchmark: 13 , discarded instances due to exact heuristic: 1.

where they do report solution, m-AOBF-tree is slightly faster than m-AOBF-graph.

m-A*-tree. As we saw for WCSPs, on some pedigree instances m-A*-tree is faster than

instance (n,k,w^*,h)	i-bound	algorithm	number of solutions					
			$m = 1$		$m = 10$		$m = 100$	
			time	nodes	time	nodes	time	nodes
50-15-5 (400, 2, 27, 99)	10	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	8.83	866865	11.97	1177549	20.22	1931039
		m-BB tree	8.75	1967152	11.91	2647393	22.06	4708311
		m-AOBB tree	3.29	251502	34.28	2485393	Timeout	
		BE+m-BF	**2.11**	**225**	**2.11**	**1469**	**2.30**	**11240**
	18	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	OOM		OOM		OOM	
		m-BB tree	Timeout		Timeout		Timeout	
		m-AOBB tree	**347.24**	**17332742**	**692.59**	**28676212**	**2277.92**	**75442102**
		BE+m-BF	OOM	OOM	OOM	OOM	OOM	OOM
50-17-5 (289, 2, 22, 84)	10	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	OOM		OOM		OOM	
		m-BB tree	Timeout		Timeout		Timeout	
		m-AOBB tree	82.95	3290939	368.18	16431707	Timeout	
		BE+m-BF	**18.45**	**289**	**18.47**	**1220**	**18.67**	**9534**
	18	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	**0.35**	51205	**0.39**	55783	**0.89**	104621
		m-BB tree	1.39	355700	1.83	421798	4.92	892065
		m-AOBB tree	1.79	85289	116.77	7505310	Timeout	
		BE+m-BF	18.45	**289**	18.47	**1220**	18.67	**9534**
90-20-5 (400, 2, 27, 99)	10	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	OOM		OOM		OOM	
		m-BB tree	Timeout		Timeout		Timeout	
		m-AOBB tree	**347.24**	**17332742**	**692.59**	**28676212**	**2277.92**	**75442102**
		BE+m-BF	OOM	OOM	OOM	OOM	OOM	OOM
	18	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	**1.3**	**153399**	**1.88**	**211547**	**3.53**	**362344**
		m-BB tree	74.83	14968683	76.93	15403354	85.42	16631321
		m-AOBB tree	46.53	2450725	118.26	4940247	563.22	18306275
		BE+m-BF	OOM	OOM	OOM	OOM	OOM	OOM
75-19-5 (361, 2, 25, 89)	10	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	OOM		OOM		OOM	
		m-BB tree	Timeout		Timeout		Timeout	
		m-AOBB tree	3591.1	**1**19431966	Timeout		Timeout	
		BE+m-BF	**143.11**	**361**	**143.11**	**2330**	**144.11**	**16897**
	18	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	**14.3**	1609506	**18.76**	2029844	**28.04**	2995437
		m-BB tree	16.27	4005082	22.28	5320573	37.26	8191215
		m-AOBB tree	39.66	1367955	94.0	3480629	Timeout	
		BE+m-BF	143.11	**361**	143.11	**2330**	144.11	**16897**

Table 3.6: Grids: CPU time (in seconds) and number of nodes expanded. A 'Timeout' stands for exceeding the time limit of 3 hours. 'OOM' indicates out of 4GB memory. In bold we highlight the best time and number of nodes for each m. Parameters: n - number of variables, k - domain size, w^* - induced width, h - pseudo-tree height.

the two m-AOBF schemes, e.g., pedigree23, $i = 22$, all values of m. Moreover, it is superior on harder instances infeasible for both m-AOBF schemes and BE+m-BF, e.g., pedigree33, $i = 22$. As shown in Figure 3.7, it solves 5 instances for $i = 16$, for all ms, which is the best or second best results, depending on the number of solutions. However, the median time of m-A*-tree is considerably larger than that of BE+m-BF for $i = 16$, while for this i-bound the latter solves only a single instance less.

■ 3.6.2.3 Binary Grids

Table 3.6 shows the results for select instances from the grid networks domain. Figure 3.8 shows the median runtime and number of solved instances for $i = 18$, while Table 3.7 presents the number of instances, for which an algorithm is the best, for the same i-bound. Most trends in the algorithms' behavior observed on WCSP and Pedigree benchmarks can be also noticed on Grid benchmark. In particular, m-AOBB-tree is very successful when m is small, even solving the most instances, as seen in Figure 3.8. But it shows worse results for $m = 100$ and for any number of solutions has the largest median time. m-BB-tree has smaller median time for all ms, but is still considerably slower than any of the best-first schemes. m-A*-tree presents the best compromise between a small medium running time and a relatively large number of solved instances. Table 3.7 shows that for majority of grid instances it is the fastest algorithm. The two m-AOBF schemes have results quite similar to each other, solving almost the same number of instances for all ms with little difference in median runtimes, as is shown in Figure 3.8. They both are consistently inferior to all other schemes except for BE+m-BF, which often runs out of memory. The main difference of the Grid benchmark compared with the previously discussed ones lies in the behavior of BE+m-BF when the i-bound is high. Even though it expands less nodes, for many problems BE+m-BF is slower than the other schemes, due to the large time required to compute the exact heuristic. For example, for instance 75-19-5, $i = 18$, for $m = 10$ the runtime of BE+m-BF is 143.11 seconds, while even m-AOBB-tree, known to be slow, terminates in just 94.0 seconds. At the same time, for this instance BE+m-BF explores the smallest search space for all values of m.

■ 3.6.2.4 Promedas

Table 3.8 shows the results for the Promedas benchmark. Figure 3.9 presents the median time and number of solved instances for the benchmark for $i = 16$. Table 3.9 shows for the

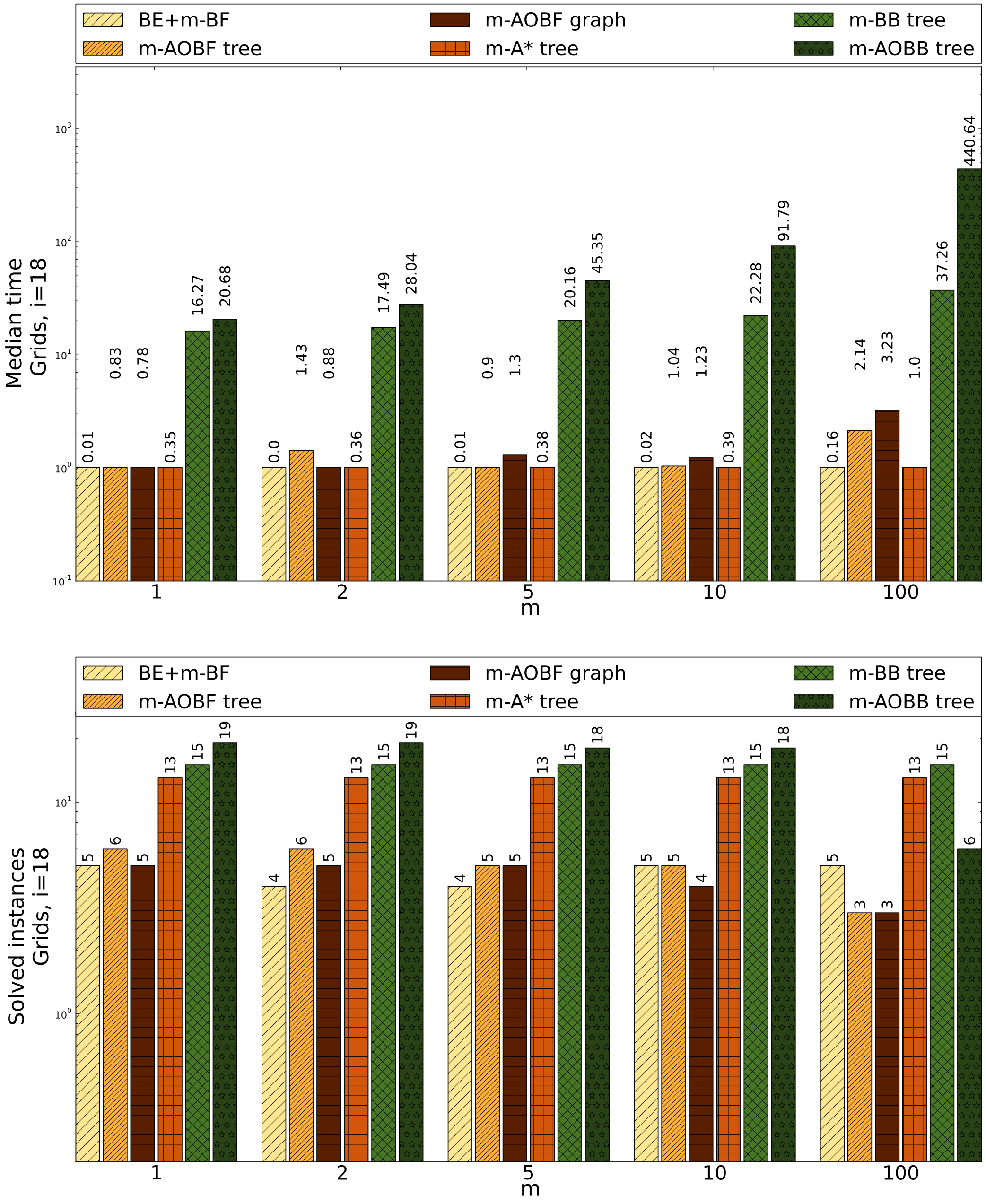


Figure 3.8: Median time and number of solved instances (out of 31) for select values of m for Grids, $i = 18$. Numbers above bars - actual values of time (sec) and # instances. Total instances in benchmark: 32, discarded instances due to exact heuristic: 1.

same i-bound the number of instances, for which each of the schemes had the best runtime and best number of expanded nodes. A significant fraction of the instances is not solved by

algorithm	Grids: # inst=32, n=144-2500 k=2-2, w^*=15-74, h_T=48-312, i-bound=18				
	$m = 1$	$m = 2$	$m = 5$	$m = 10$	$m = 100$
	#BT / #BN	#BT / #BN	#BT / #BN	#BT / #BN	#BT / #BN
Not solved	12	12	13	13	14
m-AOBF tree	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0
m-AOBF graph	0 / 0	0 / 1	0 / 2	0 / 1	0 / 3
m-A* tree	8 / 2	8 / 2	8 / 6	9 / 6	7 / 6
m-BB tree	1 / 0	1 / 0	0 / 0	1 / 0	2 / 2
m-AOBB tree	7 / 10	7 / 10	5 / 5	5 / 5	2 / 2
BE+m-BF	5 / 7	5 / 6	6 / 5	6 / 6	6 / 4

Table 3.7: Number of instances, for which each algorithm has the best runtime (#BT) and best number of expanded nodes (#BN), Grids. Out of 32 instances 1 have exact heuristics. The table accounts for remaining 31, $i = 18$.

any of the algorithms, especially for low and medium i-bounds. Unlike the other benchmarks, on Promedas m-AOBB-tree not only solves the most instances for small ms, but also is quite successful for $m = 100$, solving only one instance less than the best scheme for this value of m, m-BB-tree. Moreover, sometimes m-AOBB-tree is the only scheme to report any solutions, especially for weak heuristic, e.g., or_chain_50.fg and or_chain_212.fg, $i = 12$. BE+m-BF runs out of memory on most instances, as seen in Table 3.8. Overall, the variance of the algorithms' performance is more significant for Promedas than for the previously discussed benchmarks. For example, as we see in Figure 3.9, for $i = 16$ m-A*-tree, m-BB-tree and m-AOBB-tree solve between 25 and 33 instances for $m \in [1, 10]$, while BE+m-BF and both m-AOBB-based schemes solve only between 4 and 8 instances. Table 3.9 demonstrates that m-A*-tree most often is the fastest of the algorithms.

■ 3.6.2.5 Protein

Table 3.10 shows select Protein instances for $i = 4$ and $i = 8$. Figure 3.10 and Table 3.11 show the summary of the results for $i = 4$. This benchmark is fairly difficult due to very large domain size (up to 81). The heuristic calculation is not feasible for higher i-bounds. In particular, BE+m-BF has considerable problems in calculating the exact heuristic. Even for low i-bounds only relatively easy instances are solved. Both m-AOBF-tree and m-AOBF-

instance	i-bound	algorithm	number of solutions					
(n,k,w^*,h)			$m = 1$		$m = 10$		$m = 100$	
			time	nodes	time	nodes	time	nodes
or_chain_107.fg (620, 2, 30, 64)	16	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	**7.89**	**919865**	**18.89**	**2108122**	**44.61**	**4641627**
		m-BB tree	14.58	3139711	35.15	7051974	102.6	18494630
		m-AOBB tree	67.95	1398364	229.49	5134280	627.94	13594667
		BE+m-BF	OOM		OOM		OOM	
	22	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	**9.2**	**1093564**	**20.83**	**2465364**	**56.59**	**6356871**
		m-BB tree	17.0	3861414	42.36	9205755	100.45	19217427
		m-AOBB tree	122.01	3214924	418.46	11123810	855.84	21388619
		BE+m-BF	OOM		OOM		OOM	
or_chain_141.fg (676, 2, 30, 70)	12	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	OOM		OOM		OOM	
		m-BB tree	9553.91	1276222668	OOT		OOT	
		m-AOBB tree	**272.0**	**9878480**	**721.67**	**25481595**	**2091.26**	**64400241**
		BE+m-BF	OOM		OOM		OOM	
	16	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	**14.16**	**1261489**	**38.48**	**3379926**	OOM	
		m-BB tree	279.61	56821714	460.15	87947802	**885.72**	160581726
		m-AOBB tree	140.9	6490042	315.2	14103095	909.48	**33842266**
		BE+m-BF	OOM		OOM		OOM	
or_chain_212.fg (773, 2, 33, 79)	12	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	OOM		OOM		OOM	
		m-BB tree	Timeout		Timeout		Timeout	
		m-AOBB tree	**1772.8**	**49808550**	**4206.07**	**111853485**	Timeout	
		BE+m-BF	OOM		OOM		OOM	
	22	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	**9.91**	**1118792**	**33.87**	**3669711**	**78.37**	**8186757**
		m-BB tree	78.08	15922806	141.66	27615033	342.51	58246101
		m-AOBB tree	584.83	11336657	1239.88	24717964	5032.11	86444575
		BE+m-BF	OOM		OOM		OOM	
or_chain_50.fg 661, 2, 36, 76)	12	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	OOM		OOM		OOM	
		m-BB tree	Timeout		Timeout		Timeout	
		m-AOBB tree	**1404.27**	**33495406**	**3748.85**	**93992107**	**10070.0**	**245628104**
		BE+m-BF	OOM		OOM		OOM	
	22	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	**53.87**	**5673948**	OOM		OOM	
		m-BB tree	91.15	18515503	**176.14**	**34915510**	**447.46**	**85945673**
		m-AOBB tree	Timeout		Timeout		Timeout	
		BE+m-BF	OOM		OOM		OOM	

Table 3.8: Promedas: CPU time (in seconds) and number of nodes expanded. An 'Timeout' stands for exceeding the time limit of 3 hours. 'OOM' indicates out of 4GB memory. In bold we highlight the best time and number of nodes for each m. Parameters: n - number of variables, k - domain size, w^* - induced width, h - pseudo-tree height.

graph fail to find any solutions within the memory limit on the majority of instances, e.g., pdb1b2v and pdb1cxy, $i = 4$. There is not much difference between the runtimes of all algorithms, with an exception of m-AOBB-tree. For example, for pdb1b2v, $i = 8$, m-AOBB-tree requires 6.46 seconds to find $m = 10$ solutions, while the runtimes of other algorithms range from 0.03 to 0.09 seconds (except for BE+m-BF which runs out of memory). However, the slow performance of m-AOBB-tree on easier problems, that are feasible for all algorithms, is compensated by the fact that for many instances it is the only scheme to report

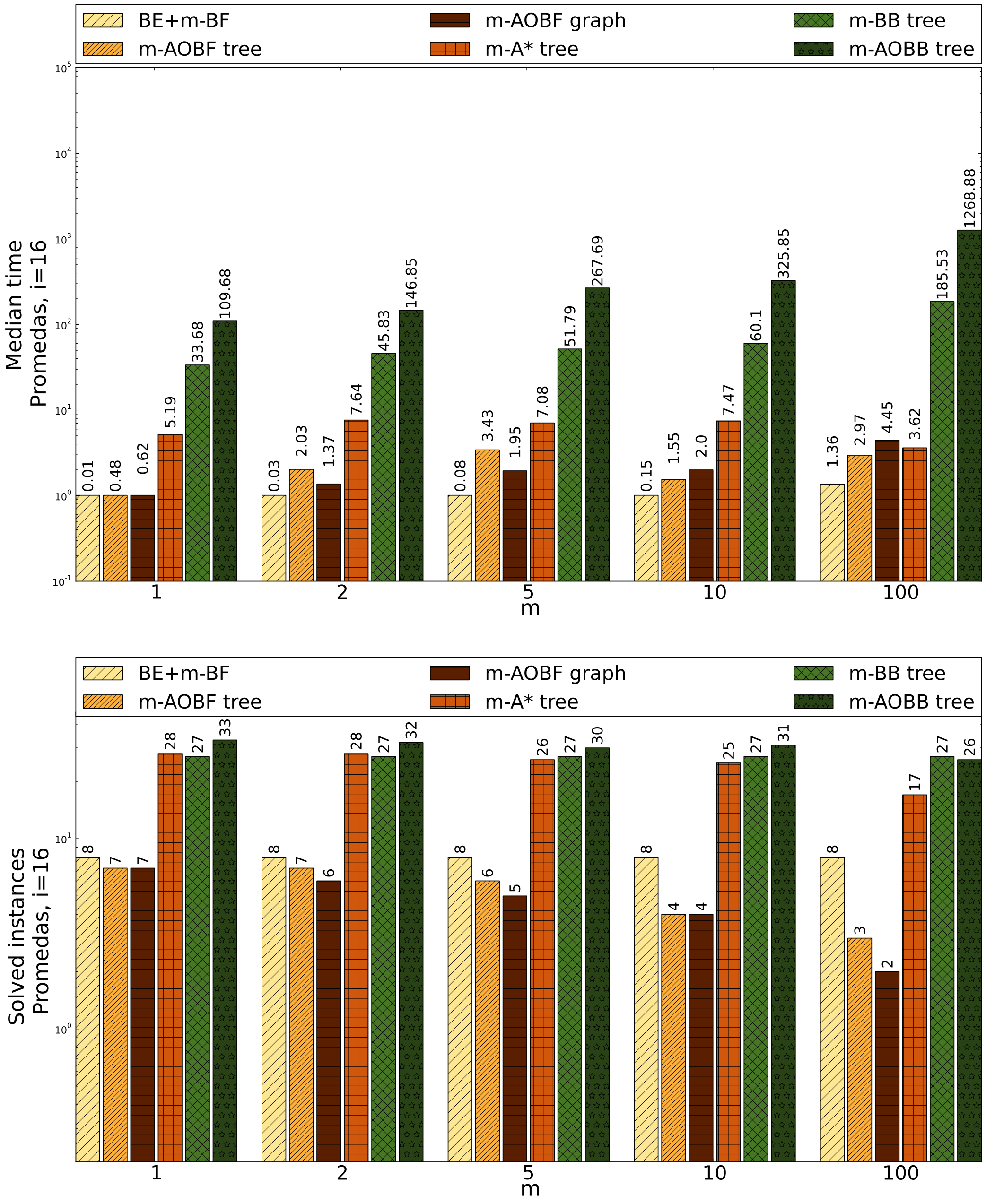


Figure 3.9: Median time and number of solved instances (out of 86) summary for select values of m for Promedas, $i = 16$. Numbers above bars - actual values of time (sec) and # instances. Total instances in benchmark: 75, discarded instances due to exact heuristic: 11.

any solution, solving most instances by considerable amount for $m \in [1, 10]$ (Figure 3.10). Table 3.11 shows that m-AOBB-tree is the best both in terms of time and space for the

algorithm	Promedas: # inst=86, n=197-2113 k=2-2, w^*=5-120, h_T=34-187, i-bound=16				
	$m = 1$	$m = 2$	$m = 5$	$m = 10$	$m = 100$
	#BT / #BN	#BT / #BN	#BT / #BN	#BT / #BN	#BT / #BN
Not solved	42	42	45	44	46
m-AOBF tree	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0
m-AOBF graph	0 / 1	0 / 2	0 / 2	0 / 3	0 / 2
m-A* tree	22 / 17	21 / 17	18 / 17	18 / 15	9 / 9
m-BB tree	1 / 0	1 / 0	1 / 0	2 / 0	10 / 5
m-AOBB tree	4 / 8	4 / 8	3 / 5	4 / 8	2 / 7
BE+m-BF	8 / 7	8 / 6	8 / 6	8 / 5	8 / 6

Table 3.9: Number of instances, for which each algorithm has the best runtime (#BT) and best number of expanded nodes (#BN), Promedas. Out of 86 instances 11 have exact heuristics. The table accounts for remaining 75, $i = 16$.

overwhelming majority of problems for all values of m except for $m = 100$.

■ 3.6.2.6 Segmentation

Table 3.12 shows the results for the select instances from the Segmentation benchmark for two i-bounds. Figure 3.11 and Table 3.13 present the summary of the results for $i = 12$. Unlike WCSP, for this benchmark we chose to display relatively low i-bounds not because calculating heuristic with larger i's is infeasible, but because the problems have low induced width and we wished to avoid displaying results obtained with exact heuristics. The main peculiarity of this benchmark is the striking success of BE+m-BF. Overall it solves as many instances as usually superior m-A*-tree and m-BB-tree, as is seen in Figure 3.11. Moreover, its runtime is superior to the other schemes, as is true for all instances in Table 3.12 and is also illustrated by the results in the Table 3.13. When the heuristic is very weak, m-AOBB-tree is fairly successful, for example, finding solutions for all values of m for 12_4_s.binary, $i = 4$, which is infeasible for any other scheme, except for BE+m-BF. However, as usual, m-AOBB-tree is the overall slowest of the schemes.

instance	i-bound	algorithm	number of solutions					
(n,k,w^*,h)			$m = 1$		$m = 10$		$m = 100$	
			time	nodes	time	nodes	time	nodes
pdb1b2v	4	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	**0.12**	**2508**	**0.17**	**3186**	**0.34**	**6249**
		m-BB tree	10.23	948337	11.09	1034645	14.4	1404370
		m-AOBB tree	6.51	100584	35.14	827365	4462.0	230849005
		BE+m-BF	OOM		OOM		OOM	
(133, 36, 13, 33)	8	m-AOBF tree	0.02	**95**	0.06	294	0.42	1956
		m-AOBF graph	0.03	**95**	0.09	**108**	0.64	**135**
		m-A* tree	**0.0**	139	**0.03**	597	**0.15**	3051
		m-BB tree	0.01	2401	0.07	8861	0.5	67330
		m-AOBB tree	0.29	6563	6.46	256588	Timeout	
		BE+m-BF	OOM		OOM		OOM	
pdb1cxy	4	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	**0.38**	**3708**	**0.48**	**4434**	**0.94**	**10854**
		m-BB tree	0.4	51020	0.6	73849	1.45	191203
		BE+m-BF	OOM		OOM		OOM	
(70, 81, 9, 19)	8	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	**0.01**	**121**	**0.04**	**480**	**0.1**	**2870**
		m-BB tree	0.03	5791	0.07	11429	0.27	53702
		m-AOBB tree	0.66	7029	2.04	34567	44.28	1335157
		BE+m-BF	OOM		OOM		OOM	
pdb1ctj	4	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	10.43	35400	13.23	43538	19.74	65340
		m-BB tree	844.08	76609260	1039.96	94422614	1325.84	120786833
		m-AOBB tree	**5.64**	74833	18.29	306054	157.43	5307198
		BE+m-BF	**0.01**	**62**	**0.02**	**265**	**0.07**	**1050**
(62, 81, 8, 21)	8	m-AOBF tree	**0.0**	49	0.07	302	0.42	1825
		m-AOBF graph	0.02	**45**	0.11	**74**	0.75	**95**
		m-A* tree	0.01	62	0.03	265	0.08	1057
		m-BB tree	0.01	1118	0.03	5385	0.15	31066
		m-AOBB tree	0.22	3098	2.32	54324	71.86	3273324
		BE+m-BF	0.01	62	**0.02**	265	**0.07**	1050
pdb1dlw	4	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	46.17	579108	46.26	579405	46.49	582375
		m-BB tree	47.27	6380302	47.33	6391107	50.72	6762911
		m-AOBB tree	187.38	1451906	544.55	12759004	OOT	
		BE+m-BF	**0.01**	**294**	**0.05**	**635**	**0.39**	**4265**
(84, 81, 8, 29)	8	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	0.14	6900	0.18	7240	0.52	10855
		m-BB tree	1.06	162913	1.09	167037	1.86	280189
		m-AOBB tree	18.53	154850	157.01	8632114	OOT	
		BE+m-BF	**0.01**	**294**	**0.05**	**635**	**0.39**	**4265**

Table 3.10: Protein: CPU time (in seconds) and number of nodes expanded. An 'Timeout' stands for exceeding the time limit of 3 hours. 'OOM' indicates out of 4GB memory. In bold we highlight the best time and number of nodes for each m. Parameters: n - number of variables, k - domain size, w^* - induced width, h - pseudo-tree height.

■ 3.6.3 Best-First vs Depth-First Branch and Bound for M-best Solutions

Let us again consider the data presented in Tables 3.2-3.13 and Figures 3.6-3.11 in order to summarize our observations and contrast the performance of best-first and depth-first branch and bound schemes. Among the best-first schemes m-A*-tree is the most successful. It is often very effective when armed with a good heuristic and requires less space than the other best-first schemes. As we already noted, BE+m-BF shows good results on Segmentation benchmark, where it is the best algorithm in terms of the mean runtime, while solving at

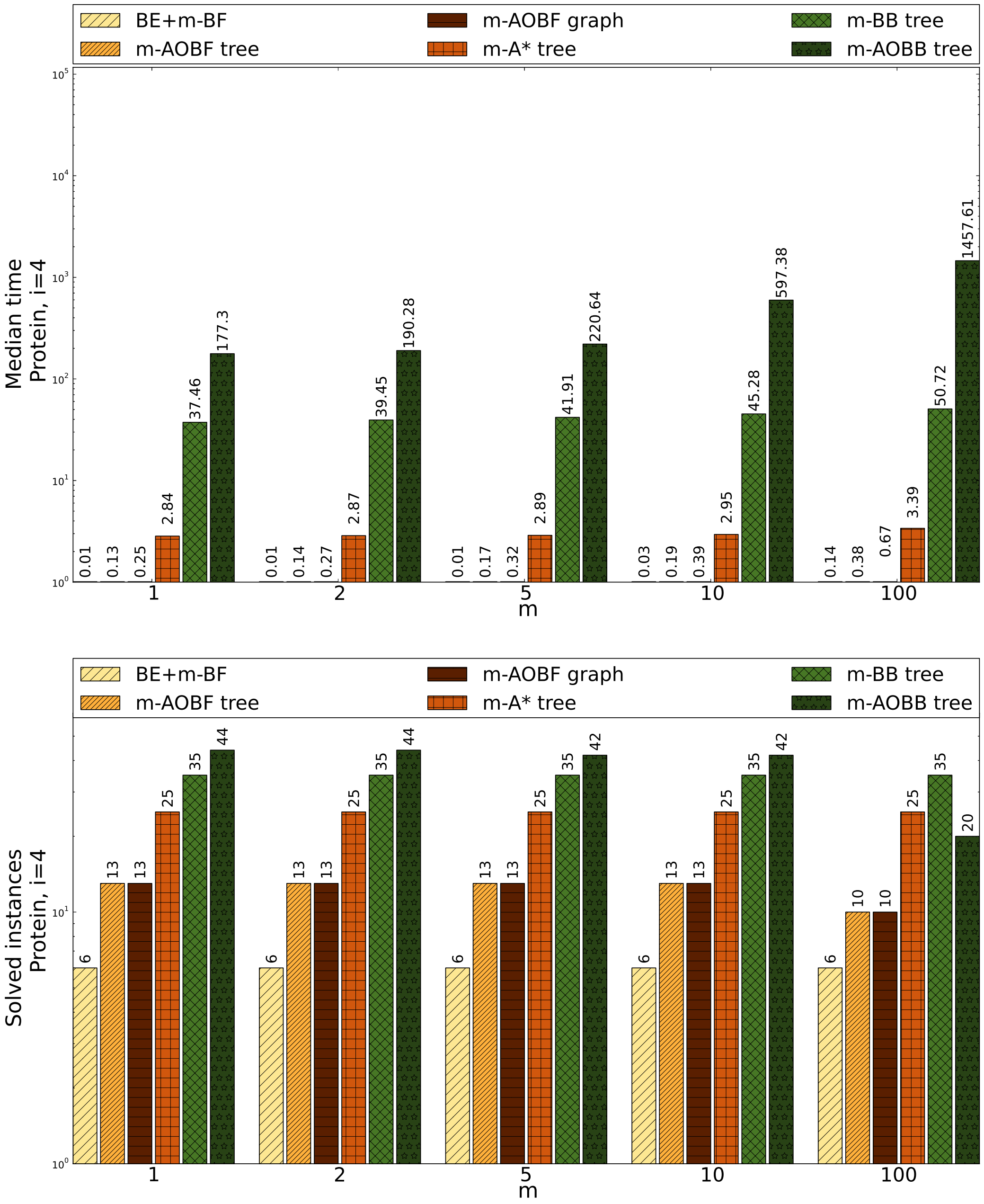


Figure 3.10: Median time and number of solved instances (out of 72) for select values of m for Protein, $i = 4$. Numbers above bars - actual values of time (sec) and # instances. Total instances in benchmark: 72, discarded instances due to exact heuristic: 0.

least the same number of problems as the other schemes. However, on the other benchmarks the calculation of the exact heuristic is often infeasible. The two m-AOBF-based schemes

algorithm	Protein: # inst=72, n=15-242 k=18-81, w^*=5-16, h_T=7-44, i-bound=4				
	$m = 1$	$m = 2$	$m = 5$	$m = 10$	$m = 100$
	#BT / #BN	#BT / #BN	#BT / #BN	#BT / #BN	#BT / #BN
Not solved	26	26	27	27	34
m-AOBF tree	8 / 3	7 / 1	7 / 1	7 / 0	5 / 0
m-AOBF graph	1 / 10	2 / 12	2 / 12	1 / 13	1 / 10
m-A* tree	11 / 9	9 / 9	12 / 9	14 / 10	17 / 13
m-BB tree	8 / 1	9 / 2	8 / 3	6 / 3	11 / 10
m-AOBB tree	21 / 22	21 / 21	18 / 19	17 / 17	3 / 3
BE+m-BF	6 / 4	6 / 2	5 / 2	5 / 2	6 / 2

Table 3.11: Number of instances, for which each algorithm has the best runtime (#BT) and best number of expanded nodes (#BN), Protein. Out of 72 instances 0 have exact heuristics. The table accounts for remaining 72, $i = 4$.

are overall inferior due to prohibitively large memory, solving less instances then the other algorithms. Both branch and bound algorithms are more robust in terms of memory requirements and dominate on many benchmarks in terms of the number of instances solved. However, they tend to have considerably larger median time and expand more nodes compared to m-A*-tree and other best-first schemes. In particular, m-AOBB-tree does not scale well with the number of solutions and for large values of m the runtime increases drastically. Overall, whenever the calculation of the exact heuristic is feasible, BE+m-BF should be the algorithm of choice. Otherwise, m-A*-tree is superior for the relatively easy problems, while m-AOBB-tree is the best scheme for hard memory-intensive instances.

3.6.4 Scalability of the Algorithms with the Number of Required Solutions

Figures 3.12-3.14 present the plots showing the runtime in seconds and the number of expanded nodes as a function of number of solutions m (on a log scale) for two instances from each benchmark. Figure 3.12 displays results for WCSP and Pedigree benchmarks, Figure 3.13 - for Grids and Promedas, Figure 3.14 - for Proteins and Segmentation. Lower values (on a y-axis) are preferable. Each row contains two instances from each benchmarks for a specific value of i-bound, the runtime plots above the expanded nodes plots. The examples are chosen to best illustrate the prevailing tendencies.

instance (n,k,w^*,h)	i-bound	algorithm	number of solutions					
			$m = 1$		$m = 10$		$m = 100$	
			time	nodes	time	nodes	time	nodes
12_4_s.binary	4	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	OOM		OOM		OOM	
		m-BB tree	Timeout		Timeout		Timeout	
		m-AOBB tree	164.91	5653312	505.82	18888321	4371.05	189179726
		BE+m-BF	**0.0**	**225**	**0.02**	**1619**	**0.21**	**11194**
(225, 2, 16, 48)	12	m-AOBF tree	7.31	103327	10.36	143333	OOM	
		m-AOBF graph	10.47	**1843**	OOM		OOM	
		m-A* tree	0.03	3754	0.06	5692	0.3	18616
		m-BB tree	0.04	8251	0.21	24349	1.32	131571
		m-AOBB tree	0.08	4158	1.62	118074	489.57	40961080
		BE+m-BF	**0.0**	**225**	**0.02**	**1619**	**0.21**	**11194**
16_16_s.binary	4	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	OOM		OOM		OOM	
		m-BB tree	Timeout		Timeout		Timeout	
		m-AOBB tree	71.71	2733703	360.14	14906212	OOT	
		BE+m-BF	**0.01**	**227**	**0.02**	**1365**	**0.19**	**11157**
(227, 2, 16, 57)	12	m-AOBF tree	0.23	3338	3.75	46121	OOM	
		m-AOBF graph	0.33	799	5.72	1827	OOM	
		m-A* tree	**0.01**	585	0.09	9103	0.38	30542
		m-BB tree	0.05	10687	0.19	30119	1.2	141591
		m-AOBB tree	0.21	11076	14.28	1054628	OOT	
		BE+m-BF	**0.01**	**227**	**0.02**	**1365**	**0.19**	**11157**
7_9_s.binary	4	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	OOM		OOM		OOM	
		m-BB tree	Timeout		Timeout		Timeout	
		m-AOBB tree	127.17	3337949	505.08	17976200	OOT	
		BE+m-BF	**0.01**	**234**	**0.03**	**1337**	**0.21**	**10212**
(234, 2, 16, 53)	12	m-AOBF tree	8.85	122663	OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	0.02	1978	0.06	4170	0.28	15807
		m-BB tree	0.03	4415	0.11	13357	0.95	89675
		m-AOBB tree	0.05	2750	10.54	806490	OOT	
		BE+m-BF	**0.01**	**234**	**0.03**	**1337**	**0.21**	**10212**
11_4_s.binary	4	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	OOM		OOM		OOM	
		m-BB tree	Timeout		Timeout		Timeout	
		m-AOBB tree	110.19	4227437	555.6	23302165	OOT	
		BE+m-BF	**0.01**	**231**	**0.03**	**1615**	**0.28**	**14241**
(231, 2, 16, 57)	12	m-AOBF tree	OOM		OOM		OOM	
		m-AOBF graph	OOM		OOM		OOM	
		m-A* tree	1.02	102671	1.17	115407	1.86	167983
		m-BB tree	2.07	428791	2.9	527967	7.1	1010155
		m-AOBB tree	0.75	39170	11.99	809403	8497.93	617227854
		BE+m-BF	**0.01**	**231**	**0.03**	**1615**	**0.28**	**14241**

Table 3.12: Segmentation: CPU time (in seconds) and number of nodes expanded. An 'Timeout' stands for exceeding the time limit of 3 hours. 'OOM' indicates out of 4GB memory. In bold we highlight the best time and number of nodes for each m. Parameters: n - number of variables, k - domain size, w^* - induced width, h-pseudo-tree height.

Note that the theoretical analysis suggests that runtime of BE+m-BF should scale with m the best among the algorithms, since its worst case complexity is $O(nk^{w^*} + mn)$. The theoretical complexity of the best-first schemes m-AOBF-tree and m-A*-tree is linear in the number of solutions, while for m-BB-tree the overhead due to m-best task is a factor of $(m \cdot \log m)$ and for m-AOBB-tree it is $(m \log m \cdot deg)$, where deg is the degree of the pseudo-tree for m-AOBB. We observed that compared to other schemes the runtime of BE+m-BF indeed rises quite slowly as the number of solutions increases, even as m reaches 100. The

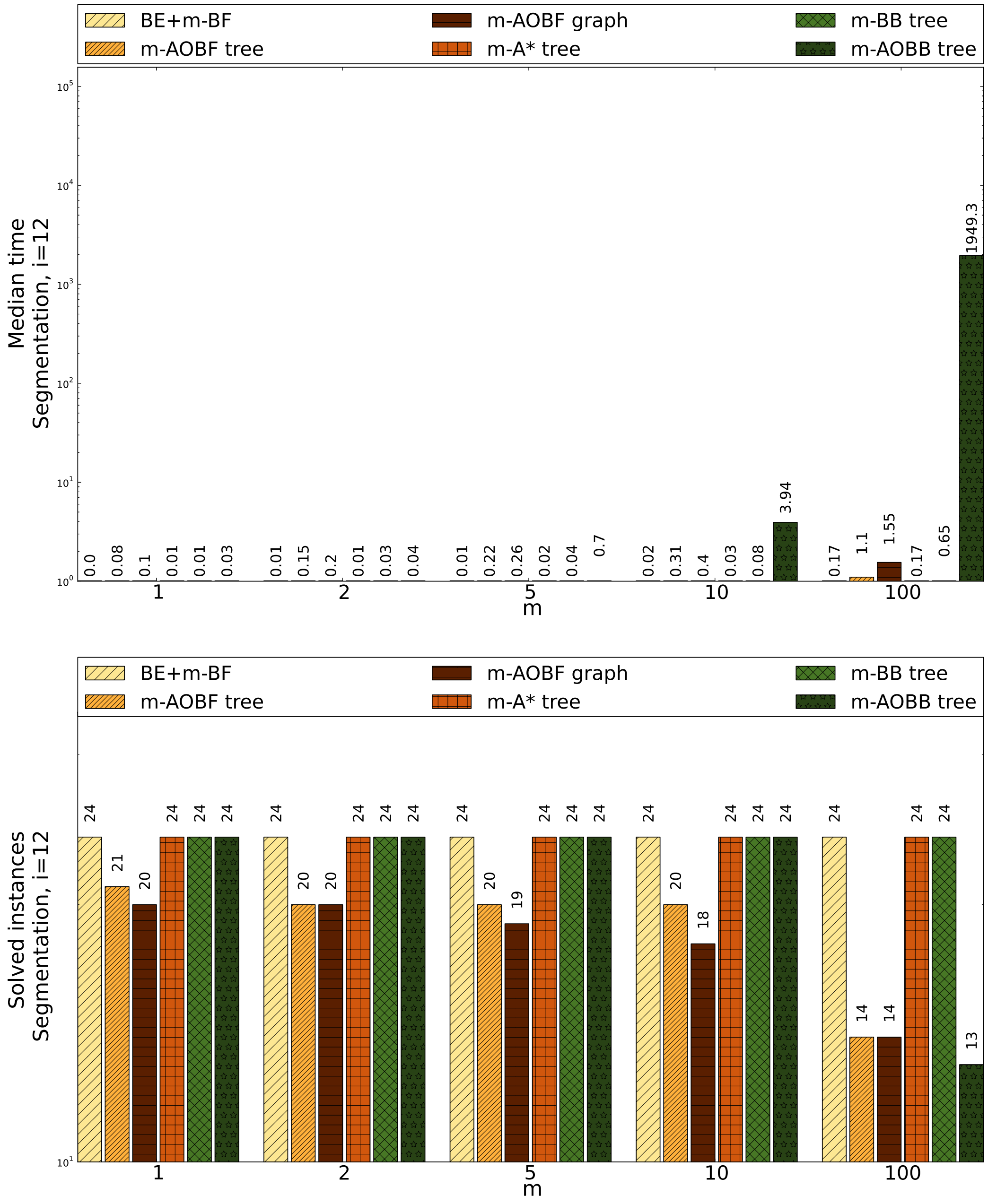


Figure 3.11: Median time and number of solved instances (out of 47) for select values of m for Segmentation, $i = 12$. Numbers above bars - actual values of time (sec) and # instances. Total instances in benchmark: 47, discarded instances due to exact heuristic: 0.

runtime m-A*-tree also scales well with m. The behavior of m-BB-tree depends a lot on a benchmark. On Pedigrees and Protein its runtime changes little on most instances as the

algorithm	Segmentation: # inst=47, n=222-234 k=2-21, w^*=15-18, h_T=47-67, i-bound=12				
	$m = 1$	$m = 2$	$m = 5$	$m = 10$	$m = 100$
	#BT / #BN	#BT / #BN	#BT / #BN	#BT / #BN	#BT / #BN
Not solved	23	23	23	23	23
m-AOBF tree	0 / 5	0 / 0	0 / 0	0 / 0	0 / 0
m-AOBF graph	0 / 5	0 / 7	0 / 11	0 / 15	0 / 14
m-A* tree	15 / 0	11 / 0	12 / 0	11 / 0	3 / 0
m-BB tree	7 / 0	5 / 0	3 / 0	2 / 0	0 / 0
m-AOBB tree	0 / 0	3 / 0	0 / 0	0 / 0	0 / 0
BE+m-BF	21 / 19	20 / 17	22 / 13	24 / 9	24 / 10

Table 3.13: Number of instances, for which each algorithm has the best runtime (#BT) and best number of expanded nodes (#BN), Segmentation. Out of 47 instances 0 have exact heuristics. The table accounts for remaining 47, $i = 12$.

number of solutions grows, but on the other benchmarks, the runtime for $m = 100$ tends to be significantly larger than for $m = 1$. m-AOBF-tree and m-AOBF-graph often do not provide any solutions even for $m = 1$ or, alternatively, run out of memory as m slightly increases ($m \in [2, 10]$). These algorithms are clearly not successful in practice. Both the runtime and number of expanded nodes of m-AOBB-tree increase drastically as m gets larger.

3.6.5 Comparison with Competing Algorithms

We compare our methods with a number of previously developed schemes described in more details in Section 3.5: STRIPES, PESTEELARS and Nilsson's algorithm. The first two schemes are based on ideas of LP relaxations and are approximate, but are known to often find exact solutions, though they provide no guarantees of optimality. Nilsson's algorithm is an exact message-passing scheme operating on a junction tree. For the first set of experiments (on a tree benchmark) we also show results for STILARS algorithm, an older version of the PESTEELARS algorithm. However, this scheme is consistently inferior to the other two LP-based schemes and is not considered for the other two benchmarks.

Randomly generated benchmarks.

The comparison was performed on three benchmarks: random trees, random binary grids

Benchmark	# inst	n	k	w^*	h_T
Random trees	12	10-5994	2-4	1	5-132
Random Binary Grids	24	16-3192	2	6-79	9-221
Random submodular graphs	12	16-3192	2	4-74	9-208

Table 3.14: Benchmark parameters: # inst - number of instances, n - number of variables, k - domain size, w^* - induced width, h_T - pseudo-tree height.

		i-bound=4. k=2				
instance	algorithm	$m = 1$	$m = 2$	$m = 5$	$m = 10$	$m = 100$
		time	time	time	time	time
tree_nnodes245_ps1_k2	m-AOBF tree	0.02	0.02	0.05	0.09	0.61
	m-AOBF graph	0.02	0.03	0.08	0.13	0.95
	m-A* tree	0.0	0.0	0.01	0.02	0.12
	m-BB tree	0.0	0.0	0.02	0.03	0.37
(245, 2, 2, 32)	m-AOBB tree	0.02	0.02	0.06	14.14	3045.25
	STILARS	0.0	0.04	7.93	33.3	1757.41
	STRIPES	0.09	0.17	0.4	0.88	13.88
	PESTEELARS	0.0	0.13	0.51	1.32	47.32
tree_nnodes880_ps1_k4	m-AOBF tree	0.3	0.46	1.06	1.9	OOM
	m-AOBF graph	0.48	0.76	1.8	3.28	OOM
	m-A* tree	0.08	0.17	0.24	0.48	3.67
	m-BB tree	0.1	0.3	0.54	1.23	14.38
(880, 4, 2, 52)	m-AOBB tree	1.17	1.37	52.36	927.12	Timeout
	STILARS	0.0	0.11	28.19	81.21	2440.22
	STRIPES	5.67	11.26	28.09	56.41	607.01
	PESTEELARS	0.0	0.87	6.13	9.26	79.0
tree_nnodes5994_ps1_k4	m-AOBF tree	OOM	OOM	OOM	OOM	OOM
	m-AOBF graph	OOM	OOM	OOM	OOM	OOM
	m-A* tree	5.44	10.68	18.31	37.21	206.26
	m-BB tree	5.77	29.04	36.49	97.73	1112.2
(5994, 4, 2, 189)	m-AOBB tree	851.48	922.19	Timeout	Timeout	Timeout
	STILARS	0.05	2.72	64.48	250.36	7325.4
	STRIPES	248.53	506.25	1279.87	2576.87	Timeout
	PESTEELARS	0.05	18.28	91.17	169.39	Timeout

Table 3.15: Random trees, $i = 4$. Timeout - out of time, OOM - out of memory. 3 GB, 1 h.

and random graphs with submodular potentials, that we call "submodular graphs" in the remainder of the section. Table 3.14 shows the parameters of the benchmarks. The instances where generated in the following manner. First a vector of 12 logarithmically spaced integers between 10 and $10^{3.5}$ was generated, serving as the number of variables for the instances. For binary grids benchmarks each value was used to generate two problems with the same number of variables. The edges between the variables were generated uniformly randomly, while making sure that the end graph is a tree, a grid or a loopy graph, depending on the benchmark. For each edge we define a binary potential and each vertex a uniary potential in an exponential form: $f = e^{\theta}$, where θ is a real number sampled from a uniform distribution. For the third benchmark the potentials are further modified to be submodular. On the

		i-bound=20			
instance	algorithm	$m = 2$	$m = 5$	$m = 10$	$m = 25$
		time	time	time	time
grid_nnodes132_ps1_k2 (132, 2, 13, 33)	m-AOBF tree	0.01	0.01	0.03	0.06
	m-AOBF graph	0.01	0.02	0.04	0.1
	m-A* tree	0.01	0.0	0.01	0.03
	m-BB tree	0.01	0.02	0.03	0.08
	m-AOBB tree	0.02	0.05	0.43	10.15
	Nilsson	9.53	62.49	148.86	OOM
	STRIPES	0.68	5.22	13.3	Timeout
	PESTEELARS	3.01	8.94	18.95	Timeout
grid_nnodes380_ps2_k2 (380, 2, 25, 61)	m-AOBF tree	OOM	OOM	OOM	OOM
	m-AOBF graph	OOM	OOM	OOM	OOM
	m-A* tree	0.2	0.23	0.26	0.36
	m-BB tree	0.32	0.36	0.58	0.95
	m-AOBB tree	7.62	12.82	67.59	1964.18
	Nilsson	110.4	757.14	1820.45	OOM
	STRIPES	2.23	19.41	38.54	Timeout
	PESTEELARS	3.98	11.5	24.34	Timeout
grid_nnodes3192_ps2_k2 (3192, 2, 75, 217)	m-AOBF tree	OOM	OOM	OOM	OOM
	m-AOBF graph	OOM	OOM	OOM	OOM
	m-A* tree	OOM	OOM	OOM	OOM
	m-BB tree	Timeout	Timeout	Timeout	Timeout
	m-AOBB tree	Timeout	Timeout	Timeout	Timeout
	Nilsson	OOM	OOM	OOM	OOM
	STRIPES	123.45	658.05	3035.29	Timeout
	PESTEELARS	26.86	81.27	172.35	Timeout

Table 3.16: Random binary grids, $i = 20$. Timeout - out of time, OOM - out of memory. 3 GB, 1 h.

random trees the m-best optimization LP problem is guaranteed to be tight, on the graphs with submodular potentials the LP optimization problem is tight, but its m-best extension is not, and on the arbitrary loopy graphs, including grids, the algorithms provide no guarantees.

Tables 3.15-3.17 show the runtimes for select instances from each of the three random benchmarks for our 5 m-best search schemes and the competing LP schemes STILARS, PESTEELARS and STRIPES. The time limit was set to 1 hour, memory limit to 3 GB. Since we observed that STILARS is always inferior to the other two schemes, we excluded it from the remainder of evaluation, instead adding for comparison Nilsson's max-flow algorithm. The implementations of all these algorithms were provided by Dhruv Batra. In the following we collectively refer to these 4 algorithms as *”competing schemes”*. The behavior of the algorithms is quite consistent across the instances.

STILARS and Nilsson's schemes are always dominated by the other two competing schemes

instance	algorithm	i-bound=20			
		$m = 2$	$m = 5$	$m = 10$	$m = 25$
		time	time	time	time
gen_nnodes132_ps1_k2 (132, 2, 13, 34)	m-AOBF tree	0.01	0.02	0.03	0.05
	m-AOBF graph	0.01	0.03	0.06	0.09
	m-A* tree	0.0	0.01	0.02	0.02
	m-BB tree	0.0	0.0	0.03	0.05
	m-AOBB tree	0.03	0.09	5.44	120.67
	Nilsson	9.34	60.81	144.93	394.26
	STRIPES	0.5	1.32	3.13	13.24
	PESTEELARS	2.9	8.52	18.56	48.76
gen_nnodes380_ps1_k2 (380, 2, 25, 61)	m-AOBF tree	OOM	OOM	OOM	OOM
	m-AOBF graph	OOM	OOM	OOM	OOM
	m-A* tree	0.47	0.51	0.57	0.72
	m-BB tree	0.54	0.61	0.73	1.03
	m-AOBB tree	51.77	110.96	141.68	2027.05
	Nilsson	105.58	728.0	1753.98	4817.09
	STRIPES	2.07	6.2	13.21	76.0
	PESTEELARS	4.38	14.09	29.96	75.04
gen_nnodes1122_ps1_k2 (1122, 2, 43, 112)	m-AOBF tree	OOM	OOM	OOM	OOM
	m-AOBF graph	OOM	OOM	OOM	OOM
	m-A* tree	OOM	OOM	OOM	OOM
	m-BB tree	Timeout	Timeout	Timeout	Timeout
	m-AOBB tree	Timeout	Timeout	Timeout	Timeout
	Nilsson	OOM	OOM	OOM	OOM
	STRIPES	16.46	57.96	107.73	282.4
	PESTEELARS	9.69	28.84	61.04	158.7

Table 3.17: Random loopy graphs with submodular potentials, $i = 20$. Timeout - out of time, OOM - out of memory. 3 GB, 1 h.

in terms of runtime. STRIPES and PESTEELARS are sometimes faster than all our schemes for $m = 1$, e.g., tree_nnodes880_ps1_k4 in Table 3.15. However, on all three benchmark they scale rather poorly with m. For $m \geq 5$ they are almost always inferior to our algorithms, provided that the latter report any results, with occasional exception with m-AOBB-tree, which also tends to be slow for large m. The only problems, on which PESTEELARS and STRIPES are superior to our search schemes, are the largest networks having over a 1000 variables, such as grid_nnodes3192_ps2_k2, which are infeasible for our algorithms. Overall, our five m-best algorithms proved superiority over the considered competing schemes on the majority of instances, often having better runtime, especially when $m > 2$, while guaranteeing solution optimality.

■ 3.7 Conclusion

Most of the previous work on finding the m best solutions over graphical models was focused on either iterative schemes based on Lawler's idea or on dynamic programming (e.g., variable-elimination or tree-clustering). We showed for the first time that for combinatorial optimization defined over graphical models the traditional heuristic search paradigms are not only directly applicable, but often superior.

Specifically, we extended best-first and depth-first branch and bound search algorithms to solve the m-best optimization tasks, presenting m-A* and m-BB. We showed that the properties of A* extend to the m-A* algorithm and, in particular, proved that m-A* is superior to any other search scheme for the m-best task. We also analyzed the overhead of both algorithms caused by the need to find multiple solutions. We introduced BE+m-BF, a hybrid of variable elimination and best-first scheme, and showed that it has the best worst-case time complexity among all m-best algorithms over graphical models known to us.

We evaluated our schemes empirically. We observed that the AND/OR decomposition of the search space, which significantly boosts the performance of traditional heuristic search schemes, was not cost-effective for m-best search algorithms, at least with our current implementation. As expected, the best-first schemes dominate the branch and bound algorithms whenever sufficient space is available, but fail on the more memory-intensive problems. We compared our schemes with 4 previously developed algorithms: three approximate schemes based on LP-relaxation of the problem and an algorithm performing message passing on a junction tree. We showed that our schemes often dominate the competing schemes, known to be efficient, in terms of runtime, especially when the required number of solutions is large. Moreover, our schemes guarantee solution optimality.

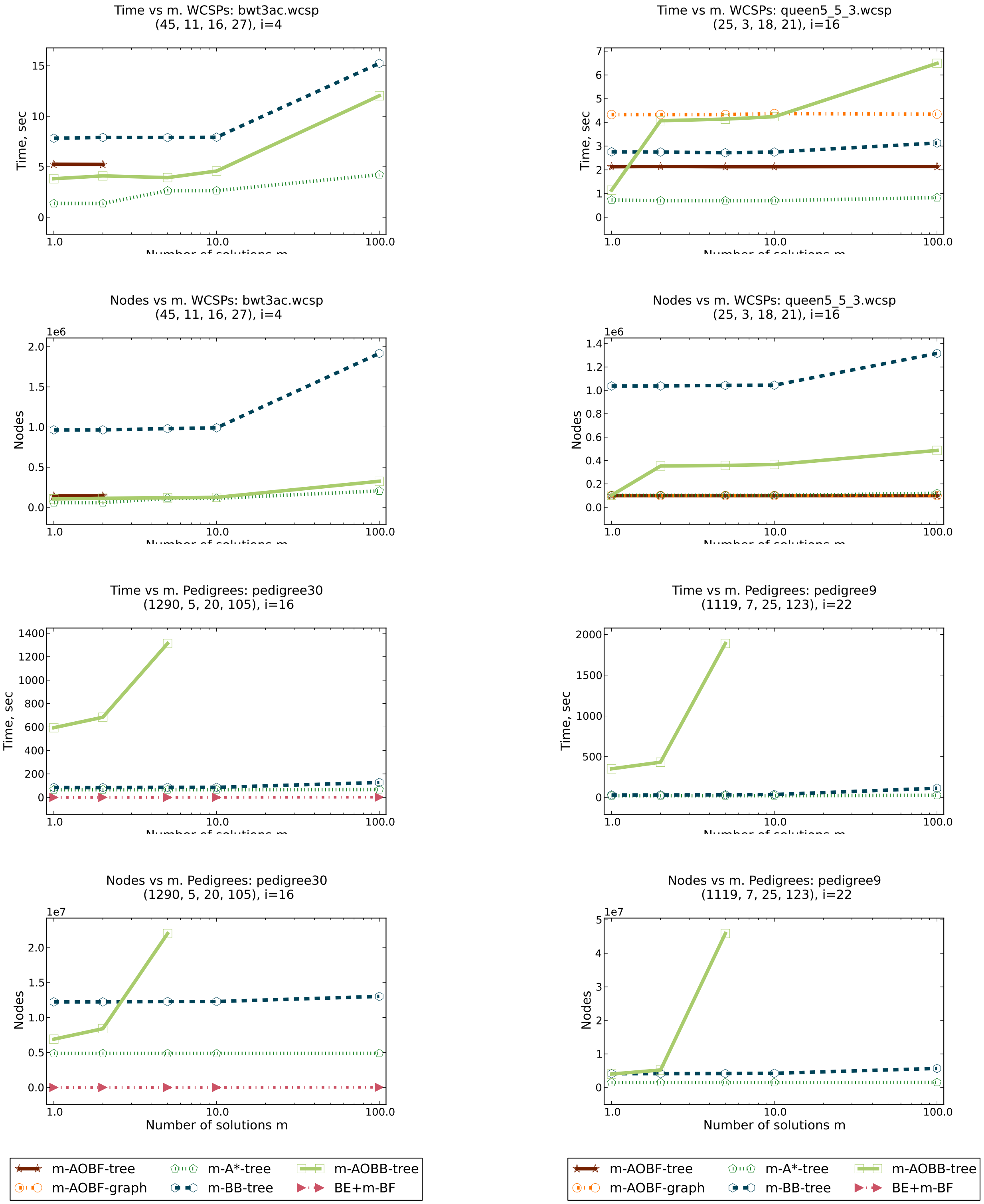


Figure 3.12: CPU time in seconds and number of expanded nodes as a function of number of solutions m. WCSP and Pedigrees, 4 GB, 3 hours.

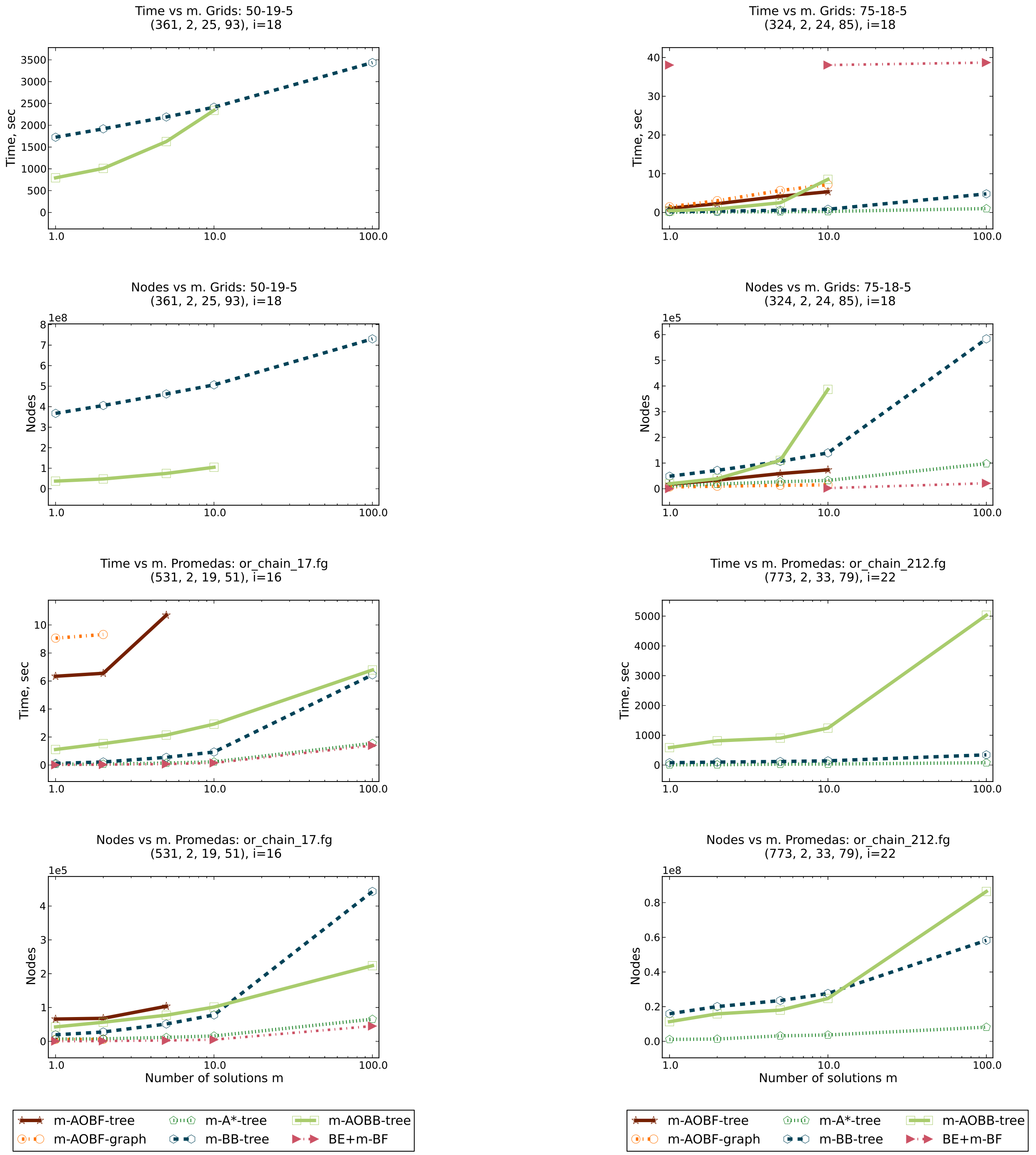


Figure 3.13: CPU time in seconds and a number of expanded nodes as a function of number of solutions m. Grids and Promedas, 4 GB, 3 hours.

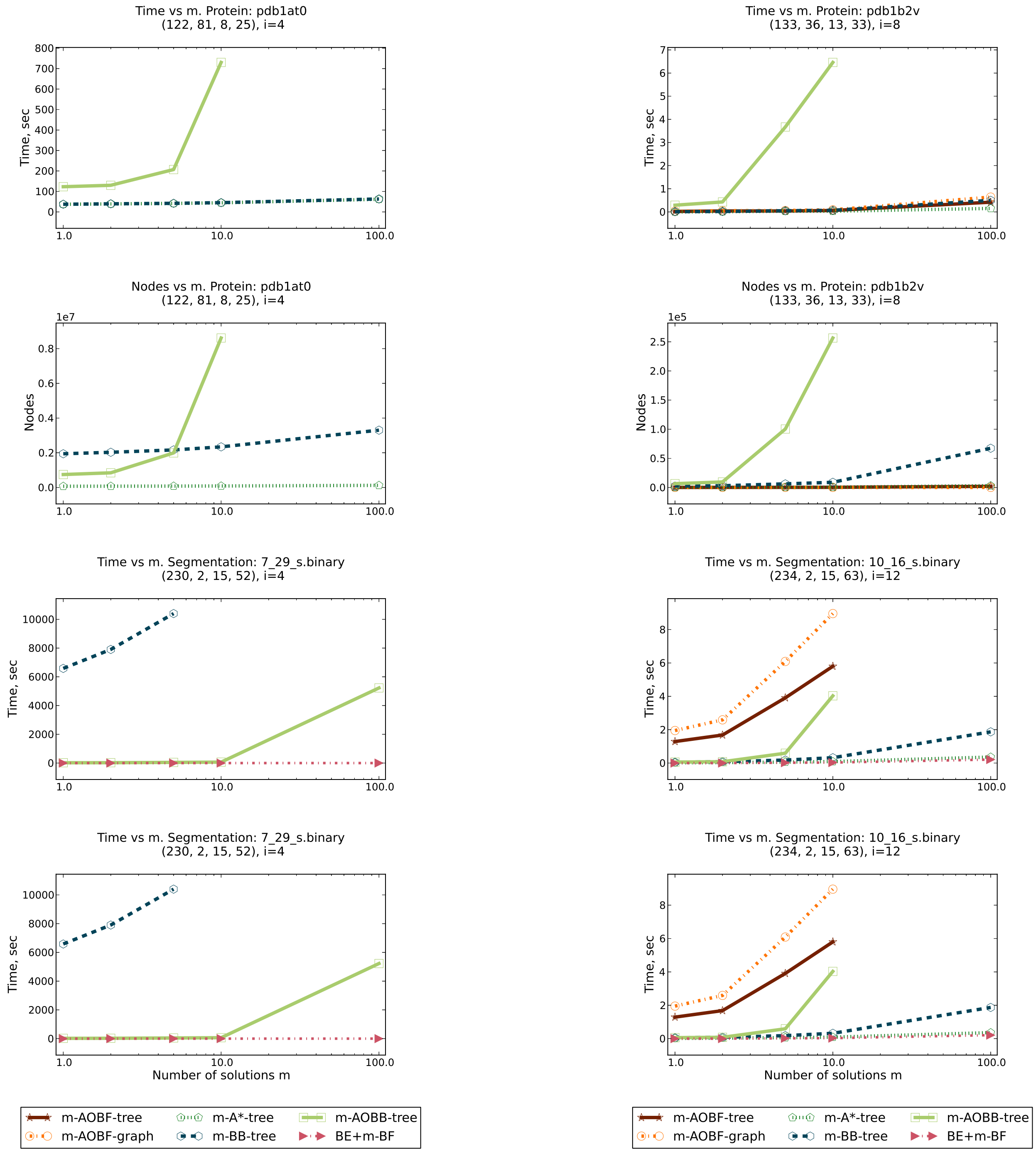


Figure 3.14: CPU time in seconds and number of expanded nodes as a function of number of solutions m. Protein and Segmentation, 4 GB, 3 hours.

Chapter 4

Anytime Weighted Heuristic Search for Graphical Models

■ 4.1 Introduction[1]

In this chapter we focus on the pure optimization problems, such as Most Probable Explanation and Weighted Constraint Satisfaction Problems (Section 1.2.1.2), describing all our algorithms in terms of summation-minimization task. Solving such problems is known to be NP-hard, can require considerable time and would often be infeasible within the existing time and space limits for an exact search algorithm. It is therefore often desirable to obtain an approximate solution or provide solutions in an anytime manner, namely report a suboptimal solution fast and gradually improve its accuracy over time.

One well known method, that allows a search algorithm to produce approximate solutions faster, is the idea of weighting the heuristic evaluation function, which guides the search, by a fixed (or varying) constant [79]. This idea was revived in recent years in the context of path-finding domains, where a variety of algorithms using this concept emerged. The

[1] Part of this work has already been published in Natalia Flerova, Radu Marinescu and Rina Dechter. ”Evaluating Weighted DFS Branch and Bound over Graphical Models ” in Proceedings of Symposium on Combinatorial Search (SoCS), 2014, and in Natalia Flerova, Radu Marinescu, Pratyaksh Sharma and Rina Dechter. ”Weighted Best-First Search for W-Optimal Solutions over Graphical Models.” in Proceedings of Planning, optimization and search (a workshop of AAAI’15), 2015.

attractiveness of this scheme of *weighted heuristic search* is in transforming best-first search into an anytime scheme, where the weight serves as a control parameter trading-off time, memory and accuracy. The common approach is to have multiple executions, gradually reducing the weight along some schedule. A valuable by-product of these schemes is that the weight offers a sub-optimality bound on the generated cost.

In this chapter we investigate the potential of weighted heuristic search for probabilistic and deterministic graphical models queries. Because graphical models are characterized by having many solutions which are all at the same depth, they are typically solved by depth-first schemes. These schemes allow flexible use of memory and they are inherently anytime (though require a modification for AND/OR spaces). Best-first search schemes, on the other hand, do not offer a significant advantage over depth-first schemes for this domain, yet they come with a significant memory cost and lack of anytime behavior, and therefore are rarely used. In this chapter we show that weighted heuristics can facilitate an effective and competitive best-first search scheme, useful for graphical models as well. The following paragraphs elaborate.

In path-finding domain, where solution length varies (e.g., planning), best-first search, and especially its popular variant A* [44], is clearly favored among the exact schemes. However, A*'s exponential memory needs, coupled with its inability to provide a solution any time before termination, lead to extension into more flexible anytime schemes based on the *Weighted A** (WA*) [79]. Several anytime weighted heuristic best-first search schemes were proposed in the context of path-finding problems in the past decade [43, 64, 97, 80, 96, 80].

Our contribution.

We extend and evaluate weighted heuristic search for graphical models. As a basis we used AND/OR Best First search (AOBF,[69]) and AND/OR Branch and Bound search (AOBB, [68]), both described in Section 1.2.4. We compare against a variant called Breadth-Rotating

AND/OR Branch and Bound (BRAOBB [76]). As already mentioned, BRAOBB (under the name daoopt) was instrumental in winning the 2011 Probabilistic Inference Challenge[2] in all optimization categories. This algorithm also won second place for 20 minutes and 1 hour time bounds in MAP category and first place for all time bounds in MMAP category in UAI Inference Challenge 2014[3]. We also compare our schemes against traditional depth-first branch and bound (DFBB) and A* exploring OR search graph, and against Stochastic Local Search (SLS).

We explored a variety of weighted heuristic schemes. After an extensive preliminary empirical evaluation, the two best-first schemes that emerged as most promising were wAOBF and wR-AOBF. Both apply weighted heuristic best-first search iteratively while decreasing w. wAOBF starts afresh at each iteration, while wR-AOBF reuses search efforts from previous iterations, extending ideas presented in Anytime Repairing A* (ARA*) [64]. Our empirical analysis revealed that weighted heuristic search can be competitive with BRAOBB on a significant number of instances from a variety of domains.

We also explored the benefit of weighting for depth-first search, resulting in wAOBB and wBRAOBB schemes. The weights facilitate an alternative anytime approach and most importantly equip those schemes with sub-optimality guarantees. Our empirical evaluation showed that for many instances our algorithms yielded best results.

To explain the behavior of weighted heuristic search we introduce a notion of *focused search* that yields a fast search. Moreover, we derive the optimal value of the weight that a) yields a greedy search with least loss of accuracy; b) when computed over an arbitrary solution path provides a guarantee on the solution accuracy.

Note that for the purpose of this work we intentionally focus primarily on the complete

[2] http://www.cs.huji.ac.il/project/PASCAL/realBoard.php

[3] http://www.hlt.utdallas.edu/ vgogate/uai14-competition/leaders.html

schemes that guarantee optimal solutions if given enough time and space. Thus many approximate schemes developed for graphical models, e.g., [47, 71, 35, 89, 100], remain beyond the scope of our consideration, as do a number of exact methods, developed for weighted CSP problems, e.g., [63, 26]. A relevant work on generating both a lower bound and an upper-bound in an anytime fashion and providing a gap of optimality when terminating was done by Cabon, et al., [10], though their approach is orthogonal to our investigation of the power of weighted heuristic search in generating anytime schemes with optimality guarantees.

The chapter is organized as follows. In Section 4.2 we present relevant background information on weighted heuristic search. In Section 4.3 we consider the characteristics of the search space explored by the weighted heuristic best-first search and reason about values of the weights that make this exploration efficient. Section 4.4 presents our extension of anytime weighted heuristic Best-First schemes to graphical models. Section 4.5 shows the empirical evaluation of the resulting algorithms. It includes the overview of methodology used (4.5.1), shows the impact of the weight on runtime and accuracy of solutions found by the weighted heuristic best-first (4.5.2), reports on our evaluation of different weight policies (4.5.3) and compares the anytime performances of our two anytime weighted heuristic best-first schemes against the previously developed schemes (4.5.4). Section 4.6 introduces the two anytime weighted heuristic depth-first branch and bound schemes (4.6.1) and presents their empirical evaluation (4.6.2). Section 4.7 presents the evaluation of weighted heuristic search schemes when using advanced heuristics. Section 4.8 summarizes and concludes.

■ 4.2 Background

We previously discussed the A* search in Section 1.2.3.1. Here we give some background on the weighted A* search. Note that throughout the chapter we define all algorithms for min-sum problem, as is the convention in weighted heuristic search literature.

Weighted A* Search (WA*) [79] differs from A* only in using the evaluation function: $f(n) = g(n) + w \cdot h(n)$, where $w > 1$. Higher values of w typically yield greedier behavior, finding a solution earlier during search and with less memory. WA* is guaranteed to terminate with a solution cost C such that $C \leq w \cdot C^*$, where C^* is the optimal solution's cost. Such solution is called w-optimal.

Formally, after [79]:

Theorem 4.1. *The cost C of the solution returned by Weighted A* is guaranteed to be within a factor of w from the optimal cost C^*.*

Proof. Consider an optimal path to the goal t. If all nodes on the path were expanded by WA*, the solution found is optimal and the theorem holds trivially. Otherwise, let n' be the deepest node on the optimal path, which is still on the OPEN list when WA* terminates. It is known from the properties of A* search that the unweighted evaluation function of n', equal to $g(n') + h(n')$, is bounded by the optimal cost C^*, namely: $g(n') + h(n') \leq C^*$ [77]. Using some algebraic manipulations: $f(n') = g(n') + w \cdot h(n')$, $f(n') \leq w \cdot (g(n') + h(n'))$. Consequently, $f(n') \leq w \cdot C^*$.

Let n be an arbitrary node expanded by WA*. Since it was expanded before n', $f(n) \leq f(n')$ and $f(n) \leq w \cdot C^*$. It holds true to all nodes expanded by WA*, including goal node t: $g(t) + w \cdot h(t) \leq w \cdot C^*$. Since $g(t) = C$ and $h(t) = 0$, $C \leq w \cdot C^*$. □

■ 4.3 Some Properties of Weighted Heuristic Search

In this section we present our new original exploration of the interplay between the weight w and heuristic function h and their impact on the explored search space. It was observed early on that the search space explored by WA* when $w > 1$ is often smaller than the one explored by A*. Intuitively the increased weight of the heuristic h transforms best-first search into a greedy search. Consequently, the number of nodes expanded tends to decrease as w increases, because a solution may be encountered early on. In general, however, such behavior is not guaranteed [103]. For some domains greedy search can be less efficient than A*.

A search space is a directed graph having a root node. Its leaves are solution nodes or dead-ends. A greedy depth-first search always explores the subtree rooted at the current node representing a partial solution path. This leads us to the following definition.

Definition 4.1 (**Focused search space**). *An explored search space is* focused *along a path π, if for any node $n \in \pi$ once n is expanded, the only nodes expanded afterwards belong to the subtree rooted at n.*

Having a focused explored search space is desirable, because it would yield a fast and memory efficient search. However, if not every path leads to a goal node, e.g., if we have determinism in the problem, focused search can lead to a dead-end and not to a solution, and thus is not likely to be effective. In the following paragraphs we will show that there exists a weight w_h that guarantees a focused search for WA*, and that its value depends on the costs of the arcs on the solution paths and on the heuristic values along the path.

Proposition 4.1. *Let π be a solution path in a rooted search space. Let arc $(n, n') \in \pi$ be such that $f(n) > f(n')$. If n is expanded by A* guided by f, then a) any node n'' expanded after n and before n' satisfies that $f(n'') \leq f(n')$, b) n'' belongs to the subgraph rooted at n, and c) under the weaker condition that $f(n) \geq f(n')$, parts a) and b) still holds given that the algorithm breaks ties in favor of deeper nodes.*

Proof. a). From the definition of best-first search, the nodes n'' are chosen from OPEN (which after expansion of n include all n's children, and in particular n'). Since n'' was chosen before n' it must be that $f(n'') \leq f(n')$.

b) Consider the OPEN list at the time when n is chosen for expansion. Clearly, any node q on OPEN satisfy that $f(q) \geq f(n)$. Since we assured $f(n) > f(n')$, it follows that $f(q) > f(n')$ and node q will not be expanded before n', and therefore any expanded node is in the subtree rooted at n.

c) Assume $f(n) \geq f(n')$. Consider any node q on OPEN: it either has an evaluation function $f(q) > f(n)$, and thus $f(q) > f(n')$, or $f(q) = f(n)$ and thus $f(q) \geq f(n')$. However, node q has smaller depth than n, otherwise it would have been expanded before n (as they have the same f value), and thus smaller depth than n', which is not expanded yet and thus is the descendant of n. Either way, node q will not be expanded before n'. □

In the following we assume that the algorithms we consider always break ties in favor of deeper nodes.

Definition 4.2 (**f non-increasing along a path**). *Given a path $\pi = \{s, \ldots, n, n' \ldots, t\}$ and a heuristic evaluation function $h \leq h^*$, if $f(n) \geq f(n')$ for every n' (a child of n along π), f is said to be* monotonically non-increasing *along π.*

From Proposition 4.1 it immediately follows:

Theorem 4.2. *Given a solution path π, along which evaluation function f is monotonically non-increasing, the search space is focused along path π.*

We will next show that this focused search property can be achieved by WA* when w passes a certain threshold. We denote by $c(n, n')$ the cost of the arc from node n to its child n'.

Definition 4.3 (**The h-weight of an arc**). *Restricting ourselves to problems where $h(n) -$*

$h(n') \neq 0$, *we denote the* h-weight of an arc *as*

$$w_h(n, n') = \frac{c(n, n')}{h(n) - h(n')}$$

Assumption 1. We will assume that for any arc (n, n'), $w_h(n, n') \geq 0$.

Assumption 1 is satisfied iff $c(n, n')$ and $h(n) - h(n')$ have the same sign, and if $h(n) - h(n') \neq 0$. Without loss of generality we will assume that for all (n, n'), $c(n, n') \geq 0$.

Definition 4.4 (The h-weight of a path). *Consider a solution path π. Then its* h-weight *is*

$$w_h(\pi) = \max_{(n,n') \in \pi} w_h(n, n') = \max_{(n,n') \in \pi} \frac{c(n, n')}{h(n) - h(n')} \tag{4.1}$$

Theorem 4.3. *Given a solution π in a search graph and a heuristic function h, such that $w_h(\pi)$ is well defined, then WA* using $w > w_h(\pi)$ yields a focused search along π.*

Proof. We will show that under the theorem's conditions f is monotonically non-increasing along π. Consider an arbitrary arc $(n, n') \in \pi$. Since $w \geq w_h(\pi)$, then

$$w \geq \frac{c(n, n')}{h(n) - h(n')}$$

or, equivalently,

$$c(n, n') \leq w \cdot h(n) - w \cdot h(n').$$

Adding $g_\pi(n)$ to the both sides and some algebraic manipulations yields

$$g_\pi(n) + c(n, n') + w \cdot h(n') \leq g_\pi(n) + w \cdot h(n)$$

which is equivalent to

$$g_\pi(n') + w \cdot h(n') \leq g_\pi(n) + w \cdot h(n)$$

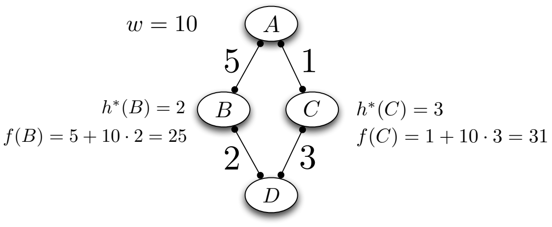


Figure 4.1: WA* with the exact heuristic

and therefore, for the weighted evaluation functions we have $f(n') \leq f(n)$. Namely, f is monotonically non-increasing. From Theorem 4.1 it follows that WA* is focused along π with this w. □

Clearly therefore,

Corollary 4.1. *If WA* uses $w > w_h(\pi)$ for each solution path π, then WA* performs a greedy search, assuming ties are broken in favor of deeper nodes.*

Corollary 4.2. *When $h = h^*$, then on an optimal path π, $\frac{c(n,n')}{h(n)-h(n')} = 1$. Therefore, any value $w \geq 1$ will yield a focused search relative to all optimal paths.*

Clearly, when h is exact, the weight $w = 1$ should be preferred since it guarantees the optimal solution. But if $w > 1$ and $h = h^*$, the solution found by the greedy search may not be optimal.

Example 4.1. *Consider the graph in Figure 4.1. Given $w = 10$, WA* will always find the incorrect path A-B-D instead of the optimal solution path A-C-D.*

Notice that, if the search is focused only along some solution paths, it can still be very unfocused relative to the entire solution space. More significantly, as we consider smaller weights, the search would be focused relative to a smaller set of paths, and therefore less contained. Yet with smaller weights upon termination WA* yields a superior guarantee on the solution quality. We next provide an explicit condition showing that under certain conditions the weight on a path can provide a bound on the relative distance of the cost of the path from the optimal cost.

Theorem 4.4. *Given a search space, and given an admissible heuristic function h having $w_h(\pi)$ function for each π. Let π be a solution path from s to t, satisfying:*

1) for all *arcs* $(n, n') \in \pi$, $c(n, n') \geq 0$, *and for one arc at least* $c(n, n') > 0$

2) for all *arcs* $(n, n') \in \pi$, $h(n) - h(n') > 0$,

then the cost of the path C_π is within a factor of $w_h(\pi)$ from the optimal solution cost C^*. Namely,

$$C_\pi \leq w_h(\pi) \cdot C^* \tag{4.2}$$

Proof. Denote by $f(n) = f_\pi(n)$ the weighted evaluation function of node n using weight $w = w_h(\pi)$: $f_\pi(n) = g(n) + w_h(\pi) \cdot h(n)$. Clearly, based on Theorem 4.2 we have that $\forall (n, n') \in \pi$, $f(n) \geq f(n')$. Namely, that search is focused relative to π.

Since for any arc (n, n') on path π, starting with s and ending with t, f is monotonically non-increasing when using $w_h(\pi)$, we have $f(s) \geq f(t)$. Since $g(s) = 0$, $f(s) = w_h(\pi) \cdot h(s)$ and since $h(t) = 0$, $f(t) = g(t) = C_\pi$, we get that

$$w_h(\pi) \cdot h(s) \geq C_\pi$$

Since h is admissible, $h(s) \leq h^*(s)$ and $h^*(s) = C^*$, we have $h(s) \leq C^*$ and

$$w_h(\pi) \cdot h(s) \leq w_h(\pi) \cdot h^*(s) = w_h(\pi) \cdot C^*$$

We get from the above two inequalities that

$$w_h(\pi) \cdot C^* \geq C_\pi \qquad \text{or} \qquad \frac{C_\pi}{C^*} \leq w_h(\pi)$$

□

In practice, conditions (1) and (2) hold for many problems formulated over graphical models, in particular for many instances in our datasets (described in Section 4.5.1), when MBE heuristic is used. However, in the presence of determinism the h-value is sometimes not well-defined, since for certain arcs $c(n, n') = 0$ and $h(n) - h'(n) = 0$.

In the extreme we can use $w_{max} = \max_\pi w_h(\pi)$ as the weight, which will yield a focused search relative to all paths, but we only guarantee that the accuracy factor will be bounded by w_{max} and in the worst case the bound may be loose.

It is interesting to note that

Proposition 4.2. *If the heuristic evaluation function $h(n)$ is consistent and if for all arcs $(n, n') \in \pi$, $h(n) - h(n') > 0$ and $c(n, n') > 0$, then $w_h(\pi) \geq 1$.*

Proof. From definition of consistency: $h(n) \leq c(n, n') + h(n')$. After some algebraic manipulation it is easy to obtain: $\max_{(n,n') \in E_\pi} \frac{c(n,n')}{h(n)-h(n')} \geq 1$ and thus $w_h(\pi) \geq 1$. □

We conclude

Proposition 4.3. *For every π, and under the conditions of Theorem 4.4*

$$C_\pi \geq C^* \geq \frac{C_\pi}{w_h(\pi)} \quad \textit{and therefore,} \quad \min_\pi\{C_\pi\} \geq C^* \geq \max_\pi\{\frac{C_\pi}{w_h(\pi)}\}$$

Algorithm 16: AOBF(h_i, w_0) exploring AND/OR search tree [69]

```
Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$, weight $w_0$ (default value 1), pseudo-tree $\mathcal{T}$ rooted at
       $X_1$, heuristic $h_i$ calculated with i-bound $i$;
Output: Optimal solution to $\mathcal{M}$
create root OR node $s$ labelled by $X_1$ and let $\mathcal{G}$ (explored search graph) = $\{s\}$;
initialize $v(s) = w_0 \cdot h_i(s)$ and best partial solution tree $T^*$ to $\mathcal{G}$;
while $s$ is not SOLVED do
    select non-terminal tip node $n$ in $T^*$. If there is no such node then exit;
    // expand node $n$
    if $n = X_i$ is OR then
        forall the $x_i \in D(X_i)$ do
            create AND child $n' = \langle X_i, x_i \rangle$;
            if $n'$ is TERMINAL then
                mark $n'$ SOLVED;
            $succ(n) \leftarrow succ(n) \cup n'$;
    else if $n = \langle X_i, x_i \rangle$ is AND then
        forall the successor $X_j$ of $X_i$ in $\mathcal{T}$ do
            create OR child $n' = X_j$;
            $succ(n) \leftarrow succ(n) \cup n'$;
    initialize $v(n') = w_0 \cdot h_i(n')$ for all new nodes;
    add new nodes to the explores search space graph $\mathcal{G} \leftarrow \mathcal{G} \cup succ(n)$;
    // update $n$ and its AND and OR ancestors in $\mathcal{G}$, bottom-up
    repeat
        if $n$ is OR node then
            $v(n) = \min_{k \in succ(n)}(c(n,k) + v(k))$;
            mark best successor $k$ of OR node $n$, such that $k = \arg\min_{k \in succ(n)}(c(n,k) + v(k))$
            (maintaining previously marked successor if still best);
            mark $n$ as SOLVED if its best marked successor is solved;
        else if $n$ is AND node then
            $v(n) = \sum_{k \in succ(n)} v(k)$;
            mark all arcs to the successors;
            mark $n$ as SOLVED if all its children are SOLVED;
        $n \leftarrow p$; //$p$ is a parent of $n$ in $\mathcal{G}$
    until $n$ is not root node $s$;
    recompute $T^*$ by following marked arcs from the root $s$;
return $\langle v(s), T^* \rangle$;
```

In summary, the above analysis provides some intuition as to why the weighted heuristic best-first search is likely to be more focused and therefore more time efficient for larger weights and how it can provide a user-control parameter exploring the trade-off between time and accuracy. There is clearly room for exploration of the potential of Proposition 4.3 that we leave for future work.

■ 4.4 Tailoring Weighted BFS to Graphical Models

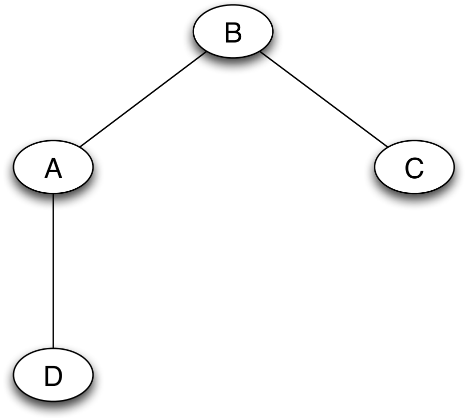
After analyzing a number of existing weighted heuristic search approaches we extended some of the ideas to the AND/OR search space over graphical models. In this section, we describe wAOBF and wR-AOBF - the two approaches that proved to be the most promising after our initial empirical evaluation (not reported here).

■ 4.4.1 Weighted AOBF

The fixed-weighted version of the AOBF algorithm is obtained by multiplying the mini-bucket heuristic function by a weight $w > 1$ (i.e., substituting $h_i(n)$ by $w \cdot h_i(n)$, where $h_i(n)$ is the heuristic obtained by mini-bucket elimination with i-bound equal to i). This scheme is identical to WAO*, an algorithm introduced by [11], but it is adapted to the specifics of AOBF. Clearly, if $h_i(n)$ is admissible, which is the case for mini-bucket heuristics, the cost of the solution discovered by weighted AOBF is w-optimal, same as is known for WA* [79] and WAO* [11].

Consider an example problem with four binary variables in Figure 4.2 that we will use to illustrate the work of our algorithms. Figure 4.2(a) shows the primal graph. We assume weighted CSP problem, namely the functions are not normalized and the task is the min-sum one: $C^* = \min_{A,B,C,D} \big(f(A,B) + f(B,C) + f(B) + f(A,D)\big)$. Figure 4.2(c) presents the AND/OR search graph of the problem, showing the heuristic functions and the weights derived from functions defined in Figure 4.2(b) on the arcs. Since the problem is very easy, the MBE heuristic is exact.

Figure 4.3 shows the part of the search space explored by AOBF (on the left) and weighted AOBF with weight $w = 10$ (on the right). The numbers in boxes mark the order of node expansions. For clarity the entire sequence of node values' updates is not presented and



(a) Primal graph.

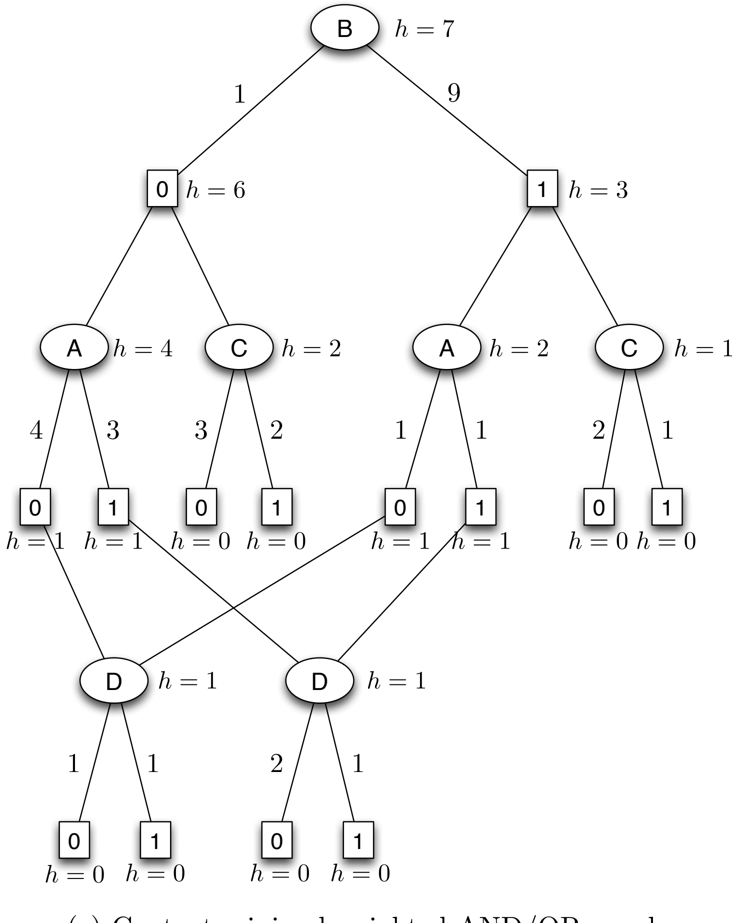
$f(A,B)$	A	B
4	0	0
1	0	1
3	1	0
1	1	1

$f(B,C)$	B	C
3	0	0
2	0	1
2	1	0
1	1	1

$f(A,D)$	A	D
1	0	0
1	0	1
3	1	0
1	1	1

$f(B)$	B
1	0
9	1

(b) Functions.



(c) Context-minimal weighted AND/OR graph.

Figure 4.2: Example problem with four variables, the functions defined over pair of variables and resulting AND/OR search graph.

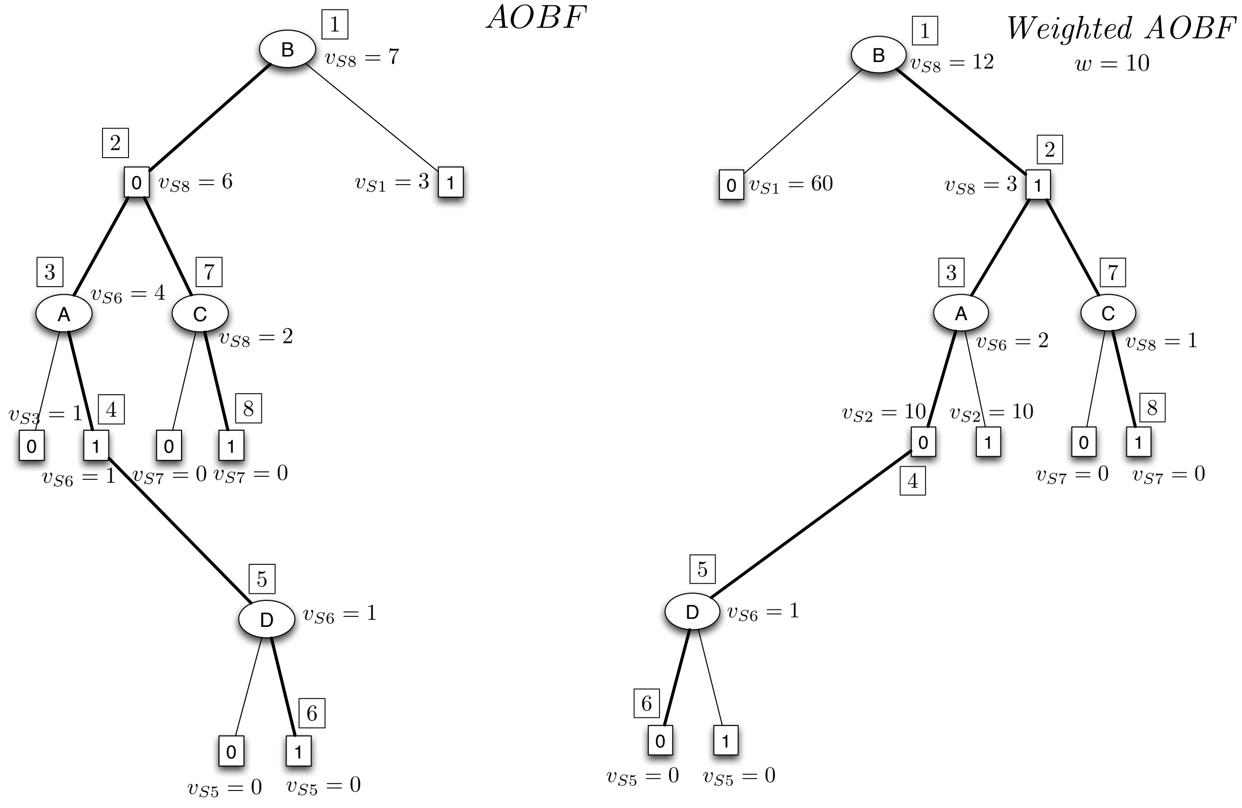


Figure 4.3: Search graphs explored by (a) AOBF, (b) Weighted AOBF, w=10. Boxed numbers indicate the order in which the nodes are expanded. v_{SN} indicates that value v was last assigned during step N, i.e., while expanding the N^{th} node.

only the latest assigned values v are shown. We use notation v_{SN} to indicate that the value was last updated during step N, namely when expanding the N^{th} node. The reported solution subtree is highlighted in bold. AOBF finds the exact solution with cost $C^* = 7$ and assignment $\mathbf{x}^* = \{A = 1, B = 0, C = 1, D = 1\}$. Weighted AOBF discovers a suboptimal solution with cost $C = 12$ and assignment $\mathbf{x} = \{A = 0, B = 1, C = 1, D = 0\}$. In this example both algorithms expand 8 nodes.

■ 4.4.2 Iterative Weighted AOBF (wAOBF)

Since Weighted AOBF yields w-optimal solutions, it can be extended to an anytime scheme **wAOBF** (Algorithm 17), decreasing the weight from one iteration to the next. This ap-

Algorithm 17: wAOBF(w_0, h_i)

```
Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$; heuristic $h_i$ calculated with i-bound $i$; initial weight $w_0$,
       weight update schedule $S$
Output: Set of suboptimal solutions $\mathcal{C}$
Initialize $w = w_0$ and let $\mathcal{C} \leftarrow \emptyset$;
while $w >= 1$ do
    $\langle C_w, T^*_w \rangle \leftarrow$ AOBF($w \cdot h_i$);
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{\langle w, C_w, T^*_w \rangle\}$;
    Decrease weight $w$ according to schedule $S$;
return $\mathcal{C}$;
```

Algorithm 18: wR-AOBF(h_i, w_0)

```
Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$; pseudo-tree $\mathcal{T}$ rooted at $X_1$; heuristic $h_i$ for i-bound=$i$;
       initial weight $w_0$, weight update schedule $S$
Output: Set of suboptimal solutions $\mathcal{C}$
initialize $w = w_0$ and let $\mathcal{C} \leftarrow \emptyset$;
create root OR node $s$ labelled by $X_1$ and let $\mathcal{G} = \{s\}$;
initialize $v(s) = w \cdot h_i(s)$ and best partial solution tree $T^*$ to $\mathcal{G}$;
while $w >= 1$ do
    expand and update nodes in $\mathcal{G}$ using AOBF($w$,$h_i$) search with heuristic function $w \cdot h_i$;
    if $T^*$ has no more tip nodes then $\mathcal{C} \leftarrow \mathcal{C} \cup \{\langle w, v(s), T^* \rangle\}$;
    decrease weight $w$ according to schedule $S$;
    for all leaf nodes in $n \in \mathcal{G}$, update $v(n) = w \cdot h_i(n)$. Update the values of all nodes in $\mathcal{G}$ using the
    values of their successors. Mark best successor of each OR node.;
    recalculate $T^*$ following the marked arcs;
return $\mathcal{C}$;
```

proach is identical to the Restarting Weighted A* by Richter, et al. [80] applied to AOBF.

Theorem 4.5. *Worst case time and space complexity of a single iteration of wAOBF with caching is $O(n \cdot k^{w^*})$, where n is the number of variables, k is the largest domain size and w^* is the induced width of the problem.*

Proof. During each iteration wAOBF executes AOBF from scratch with no overhead. The number of iterations depends on the start weight and on weight decreasing policy. □

■ 4.4.3 Anytime Repairing AOBF (wR-AOBF).

Running each search iteration from scratch seems redundant, so we introduce Anytime Repairing AOBF (Algorithm 18), which we call **wR-AOBF**. It is an extension of the *Anytime*

Repairing A (ARA*)* algorithm [64] to the AND/OR search spaces over graphical models. The original ARA* algorithm utilizes the results of previous iterations by recomputing the evaluation functions of the nodes with each weight change, and thus re-using the inherited OPEN and CLOSED lists. The algorithm also keeps track of the previously expanded nodes, whose evaluation function changed between iterations and re-inserts them back to OPEN before starting each iteration.

Extending this idea to the AND/OR search space is fairly straightforward. Since AOBF does not maintain explicit OPEN and CLOSED lists, *wR-AOBF* keeps track of the partially explored AND/OR search graph, and after each weight update it performs a bottom-up update of all the node values starting from the leaf nodes (whose h-values are multiplied by the new weight) and continuing towards the root node (line 8). During this phase, the algorithm also marks the best AND successor of each OR node in the search graph. These markings are used to recompute the best partial solution tree T'. Then, the search resumes in the usual manner by expanding a tip node of T' (line 9).

Like ARA*, wR-AOBF is guaranteed to terminate with a solution cost C such that $C \leq w \cdot C^*$, where C^* is the optimal solution's cost.

Theorem 4.6. *Worst case space complexity of a single iteration of wR-AOBF with caching is $O(n \cdot k^{w^*})$, where n is the number of variables, k is the largest domain size and w^* is the induced width of the problem. The worst case time complexity can be loosely bounded by $O(2 \cdot n \cdot k^{w^*})$. Total number of iterations depends on the weight parameters.*

Proof. In worst case at each iteration wR-AOBF explores the entire search space of size $O(n \cdot k^{w^*})$, having the same theoretical space complexity as AOBF and wAOBF. In practice there exist an additional space overhead due to required book-keeping.

The time complexity comprises the time it takes to expand nodes and time overhead due to updating node values during each iteration. Every time the weight is decreased, wR-AOBF needs to update the values of all nodes in the partially explored AND/OR graph, at most $O(n \cdot k^{w^*})$ of them. □

The updating of node values is a costly step because it involves all nodes the algorithm has ever generated. Thus, though wR-AOBF expands less nodes that wAOBF, in practice it is considerably slower, as we will see next in the experimental section.

■ 4.5 Empirical Evaluation of Weighted Heuristic BFS

Our empirical evaluation of weighted heuristic search schemes consists of two parts. In this section we focus on the two weighted heuristic best-first algorithms described in Section 4.4: wAOBF and wR-AOBF. In Section 4.6.2 we additionally compare these algorithms with the two weighted heuristic depth-first branch and bound schemes that will be introduced in Section 4.6.1.

■ 4.5.1 Overview and Methodology

In this section we evaluate the behavior of wAOBF and wR-AOBF and contrast their performance with a number of previously developed algorithms. The main point of reference for our comparison is the depth-first branch and bound scheme, BRAOBB [76], which is known to be one of the most efficient anytime algorithms for graphical models [4,5]. In a subset of experiments we also compare against A* search [44] and depth-first branch and bound (DFBB) [62], both exploring an OR search tree, and against Stochastic Local Search (SLS) [47, 52]. We implemented our algorithms in C++ and ran all experiments on a 2.67GHz Intel Xeon X5650, running Linux, with 4 GB allocated for each job.

[4] http://www.cs.huji.ac.il/project/PASCAL/realBoard.php

[5] http://www.hlt.utdallas.edu/ vgogate/uai14-competition/leaders.html

All schemes traverse the same context-minimal AND/OR search graph, defined by a common variable ordering, obtained using well-known MinFill ordering heuristic [55]. The algorithms return solutions at different time points until either the optimal solution is found, until a time limit of 1 hour is reached or until the scheme runs out of memory. No evidence was used.

All schemes use the Mini-Bucket Elimination heuristic, described in Section 1.2.4.4, with 10 i-bounds, ranging from 2 to 20. However, for some hard problems computing mini-bucket heuristic with the larger i-bounds proved infeasible, so the actual range of i-bounds varies among the benchmarks and among instances within a benchmark.

For a subset of experiments we also evaluated the impact of more sophisticated heuristics that we will discuss in more detail in Chapter 5: Mini-Bucket Elimination with Max-marginal-Matching (MBE-MM, Section 5.4) and Join-Graph Linear Programming (JGLP, Section 5.3).

We evaluated the algorithms on 4 benchmarks: Binary grids, Pedigree networks, Weighted CSPs and Type4 genetic networks. The former three were already discussed in Section 2.6.1. **Type4** instances come from the domain of genetic linkage analysis, just as the Pedigree problems, but are known to be significantly harder. Table 4.1 describes the benchmark parameters.

For each anytime solution by an algorithm we record its cost, CPU time in seconds and the corresponding weight (for weighted heuristic schemes) at termination. For uniformity we consider all problems as solving the max-product task, also known as Most Probable Explanation problem (MPE). The computed anytime costs returned by the algorithms are lower bounds on the optimal solutions. In our experiments we evaluate the impact of the weight on the solution accuracy and runtime, analyze the choice of weight decreasing policy, and study the anytime behavior of the schemes and the interaction between heuristic strength and the weight.

Benchmark	# inst	n	k	w^*	h_T
Pedigrees	11	581-1006	3-7	16-39	52-104
Grids	32	144-2500	2-2	15-90	48-283
WCSP	56	25-1057	2-100	5-287	11-337
Type4	10	3907-8186	5-5	21-32	319-625

Table 4.1: Benchmark parameters: # inst - number of instances, n - number of variables, k - domain size, w^* - induced width, h_T - pseudo-tree height.

■ 4.5.2 The Impact of Weights on the Weighted AOBF Performance

One of the most valuable qualities of weighted heuristic search is the ability to flexibly control the trade-off between speed and accuracy of the search using the weight, which provides a w-optimality bound on the solution.

In order to evaluate the impact of the weight on the solution accuracy and runtime we run wAOBF and we consider iterations individually. Each iteration j is equivalent to a single run of AOBF(w_j, h_i), namely the AND/OR Best First algorithm that uses MBE heuristic h_i, obtained with an i-bound equal to i, multiplied by weight w_j.

Table 4.2 reports the results for selected weights (w=2.828, 1.033, 1.00), for several selected instances representative of the behavior prevalent over each benchmark. Following the names and parameters of each instance, the table is vertically split into two blocks, corresponding to two i-bounds. In the second column of each block we report the time in seconds it took BRAOBB to find the optimal solution to the problem (the higher entry in each row) and the solution cost on a logarithmic scale (the lower entry in each row). The symbol "—" indicates that the corresponding algorithm ran out of memory. The next three columns show the runtime in seconds and the cost on the log scale obtained by AOBF when using a specific weight value. The entries mentioned in this section are highlighted. Note that, since calculation of the mini-bucket heuristics is time and space exponential in i-bound, for some instances the heuristics can't be obtained for large i-bounds (e.g., 1502.wcsp, $i = 10$).

Comparison between the exact results by AOBF obtained with weight $w = 1$ (columns 5

Instance (n, k, w^*, h_T)	BRAOBB	**AOBF**(w, h_i) **weights**			BRAOBB	**AOBF**(w, h_i) **weights**		
		2.828	1.033	1.00		2.828	1.033	1.00
	time C^*	time log(cost)	time log(cost)	time log(cost)	time C^*	time log(cost)	time log(cost)	time log(cost)
Grids	I-bound=6				I-bound=20			
50-16-5 (256, 2, 21, 79)	2601.46 -16.916	0.16 -21.095	—	—	7.16 -16.916	7.01 -17.57	7.01 -16.916	7.02 -16.916
50-17-5 (289, 2, 23, 77)	1335.44 -17.759	0.05 -23.496	—	—	9.42 -17.759	9.44 -17.829	9.44 -17.759	9.44 -17.759
75-18-5 (324, 2, 24, 85)	390.72 -8.911	0.42 -10.931	74.69 -8.911	88.53 -8.911	13.52 -8.911	13.95 -9.078	13.95 -8.911	13.96 -8.911
75-20-5 (400, 2, 27, 99)	time out	1.78 -16.282	—	—	22.52 -12.72	19.35 -14.067	24.96 -12.72	27.85 -12.72
90-21-5 (441, 2, 28, 106)	187.75 -7.658	1.13 -8.871	41.38 -7.658	42.48 -7.658	17.01 -7.658	17.32 -9.476	17.65 -7.658	17.74 -7.658
Pedigrees	I-bound=6				I-bound=16			
pedigree9 (935, 7, 27, 100)	time out	0.83 -137.178	—	—	1082.02 -122.904	6.24 -133.063	34.66 -122.904	—
pedigree13 (888, 3, 32, 102)	time out	0.18 -88.563	—	—	time out	4.13 -76.429	—	—
pedigree37 (726, 5, 20, 72)	4.36 -144.882	0.08 -163.325	4.42 -145.082	9.99 -144.882	388.36 -144.882	388.96 -155.259	389.02 -145.341	389.07 -144.882
pedigree39 (953, 5, 20, 77)	time out	0.11 -174.304	—	—	4.34 -155.608	4.3 -162.381	4.37 -155.608	4.83 -155.608
WCSP	I-bound=2				I-bound=10			
1502.wcsp (209, 4, 5, 11)	time out	0.0 -1.258	0.01 -1.258	0.0 -1.258	—	—	—	—
42.wcsp (190, 4, 26, 72)	time out	—	—	—	1563.44 -2.357	11.69 -2.418	—	—
bwt3ac.wcsp (45, 11, 16, 27)	2.47 -0.561	0.93 -0.561	1.84 -0.561	1.85 -0.561	54.34 -0.561	54.88 -0.561	54.92 -0.561	54.92 -0.561
capmo5.wcsp (200, 100, 100, 100)	time out	1.18 -0.262	—	—	time out	24.04 -0.262	—	—
myciel5g_3.wcsp (47, 3, 19, 24)	2661.91 -64.0	—	—	—	12.93 -64.0	2.5 -72.0	—	—
Type4	I-bound=6				I-bound=16			
type4b_100_19 (3938, 5, 29, 354)	time out	5.02 -1309.91	—	—	time out	33.32 -1171.002	—	—
type4b_120_17 (4072, 5, 24, 319)	time out	4.16 -1483.588	—	—	time out	26.06 -1362.607	104.37 -1327.776	—
type4b_140_19 (5348, 5, 30, 366)	time out	7.28 -1765.403	—	—	time out	44.94 -1541.883	—	—
type4b_150_14 (5804, 5, 32, 522)	time out	16.15 -2007.388	—	—	time out	38.22 -1727.035	—	—
type4b_170_23 (5590, 5, 21, 427)	time out	9.65 -2191.859	—	—	time out	18.62 -1978.588	38.4 -1925.883	—

Table 4.2: Runtime (sec) and cost (on logarithmic scale) obtained by AOBF(w, h_i) for selected w, and by BRAOBB (that finds C^* - optimal cost). Instance parameters: n - number of variables, k - max domain size, w^* - induced width, h_T - pseudo-tree height. "—" - running out of memory. 4 GB memory limit, 1 hour time limit, MBE heuristic. The entries mentioned in the text are highlighted.

and 9) and by BRAOBB (columns 2 and 6) with any one of the other columns reveals that abandoning optimality yields run time savings and allows to find approximate solutions when exact ones cannot be obtained within an hour.

In more details, let us consider, for example, the columns of Table 4.2 where the costs generated are guaranteed to be a factor of 2.828 away from the optimal. We see orders of magnitude time savings compared to BRAOBB, for both i-bounds. For example, for pedigree9, $i = 16$, for $w = 2.828$ weighted AOBF's runtime is merely 6.24 seconds, while BRAOBB's is 1082.02 seconds. For WCSP networks, the algorithms' runtimes are often quite similar. For example, for bwt3ac.wcsp, $i = 10$, BRAOBB takes 54.34 seconds and weighted AOBF takes 54.88 seconds. On some WCSP instances, such as myciel5g_3.wcsp, $i = 2$, BRAOBB is clearly superior, finding an optimal solution within the time limit, while weighted AOBF runs out of memory and does not report any solution for $w = 2.828$.

Comparing columns 5 and 9, exhibiting full AOBF with $w = 1$ (when it did not run out of memory) against $w = 2.828$ we see similar behavior. For example, for grid 75-18-5, $i = 6$, we see that exact AOBF ($w = 1$) requires 88.53 seconds, which is about 200 times longer than with weight $w = 2.828$ which requires 0.42 seconds.

More remarkable results can be noticed when considering the column of weight $w = 1.033$, especially for the higher i-bound (strong heuristics). These costs are just a factor of 1.033 away from optimal, yet the time savings compared with BRAOBB are impressive. For example, for pedigree9, $i = 16$ weighted AOBF runtime for $w = 1.033$ is 34.66 seconds as opposed to 1082.02 seconds by BRAOBB. Observe that often the actual results are far more accurate than the bound suggests. In particular, in a few of the cases, the optimal solution is obtained with $w > 1$. For example, see grid 75-18-5, $i = 20$, $w = 1.003$. Sometimes exact AOBF with $w = 1$ is faster than BRAOBB.

Impact of heuristic strength. The i-bound parameter allows to flexibly control the strength of mini-bucket heuristics. Clearly, more accurate heuristics yield better results for any heuristic search and thus should be preferred. However, running the mini-buckets with sufficiently high i-bound is not always feasible due to space limitations and has a considerable time overhead, since the complexity of Mini-Bucket Elimination algorithm is exponential in the i-bound. Thus we are interested to understand how the heuristic strength influences the behavior of weighted heuristic best-first schemes when the value of the i-bound is considerably smaller than the induced width of the problem.

Comparing the results Table 4.2 across i-bounds for the same algorithm and the same weight, we observe a number of instances where more accurate heuristic comes at too high a price. For example, for pedigree37 weighted AOBF finds a w-optimal solution with $w = 2.828$ in 0.08 seconds for $i = 6$, but takes 388.96 seconds for $i = 16$. One of the examples to the contrary, where the higher i-bound is beneficial, is grid 90-21-5, where weighted AOBF takes 41.38 seconds to terminate for $w = 1.033$ when $i = 6$, but only 17.65 seconds, when $i = 20$.

Table 4.2 shows that weighted AOBF is less sensitive to the weak heuristics compared with BRAOBB. For example, for grid 90-21-5 and for $i = 20$, BRAOBB terminates in 17.01 seconds. However, if the heuristic is weak ($i = 6$), it requires 187.75 seconds, 2 orders of magnitude more. On the other hand, for the same instance weighted AOBF with weight $w = 1.033$ has much smaller difference in performance for the two i-bounds. weighted AOBF terminates in 17.65 seconds for $i = 20$ and in 41.38 seconds for $i = 6$. This may suggest that wAOBF could be preferable when the i-bound is small relative to the problem's induced width.

Overall, weighted AOBF solves some hard problems that are infeasible for the exact scheme and often yields solutions with tight bounds considerably faster than the optimal solutions obtained by BRAOBB or exact AOBF.

■ 4.5.3 Exploring Weight Policies

How should we choose the starting weight value and weight decreasing policy? Previous works on weighted heuristic search usually avoid disclosing the details of how the starting weight is defined and how it is decreased at each iteration, e.g., [43], [64], etc. To answer this we evaluated 5 different policies.

The first two policies we considered were *subtract*, which decreases the weight by a fixed quantity, and *divide*, which at each iteration divides the current weight by a constant. These policies lay on the opposite ends of the strategies spectrum. The first method changes the weight very gradually and consistently, leading to a slow improvement of the solution. The second approach yields less smooth anytime behavior, since the weight rapidly approaches 1.0 and much fewer intermediate solutions are found. This could potentially allow the schemes to produce the exact solution fast, but on hard instances presents a danger of leaping directly to a prohibitively small weight and thus failing prematurely due to memory issues. The other policies we considered were constructed manually based on the intuition that it is desirable to improve the solution rapidly by decreasing the weight fast initially and then "fine-tune" the solution as much as the memory limit allows, by decreasing the weight slowly as it approaches 1.0.

Overall, we evaluated the following five policies, each for several values of parameters. We denote by w_j the weight used at the j^{th} iteration of the algorithm, k and d denote real-valued policy parameters, where appropriate. Given k and d, assuming $w_1 = w_0$ (start weight), for $j > 1$:

- *subtract*(k): $w_j = w_{j-1} - k$
- *divide*(k): $w_j = w_{j-1}/k$
- *inverse*: $w_j = w_1/j$

- *piecewise*(k, d): if $w_j \geq d$ then $w_j = w_1/j$ else $w_j = w_{j-1}/k$

- *sqrt*(k): $w_j = \sqrt{w_{j-1}}/k$

The initial weight value needs to be large enough a) to explore the schemes' behavior on a large range of weights; b) to make the search focused enough initially to solve harder instances, known to be infeasible for regular BF within the memory limit. After some preliminary experiments (not included) we chose the starting weight w_0 to be equal to 64. We noticed that further increase of w_0 typically did not yield better results. Namely, the instances that were infeasible for wAOBF and wR-AOBF with $w = 64$ also did not fit in memory when weight was larger. Such behavior can be explained by many nodes having evaluation function values so similar, that even a very large weight did not yield much difference between them, resulting in a memory-prohibitive search frontier.

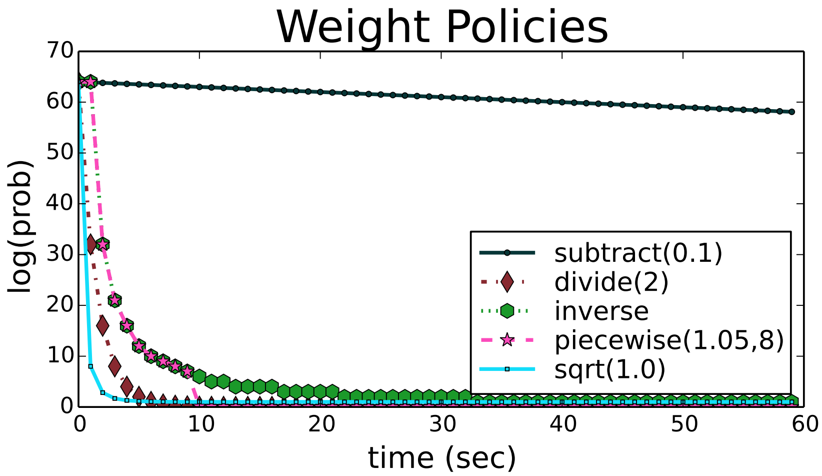


Figure 4.4: The dependency of the weight value on iteration index according to considered weight policies, showing first 50 iterations, starting weight w_0= 64.

Figure 4.4 illustrates the weight changes during the first 50 iterations according to the considered policies. We use the parameter values that proved to be more effective in the preliminary evaluation: *subtract*$(k = 0.1)$, *divide*$(k = 2)$, *inverse*$()$, *piecewise*$(k = 1.05, d = 8)$, *sqrt*$(k = 1.0)$.

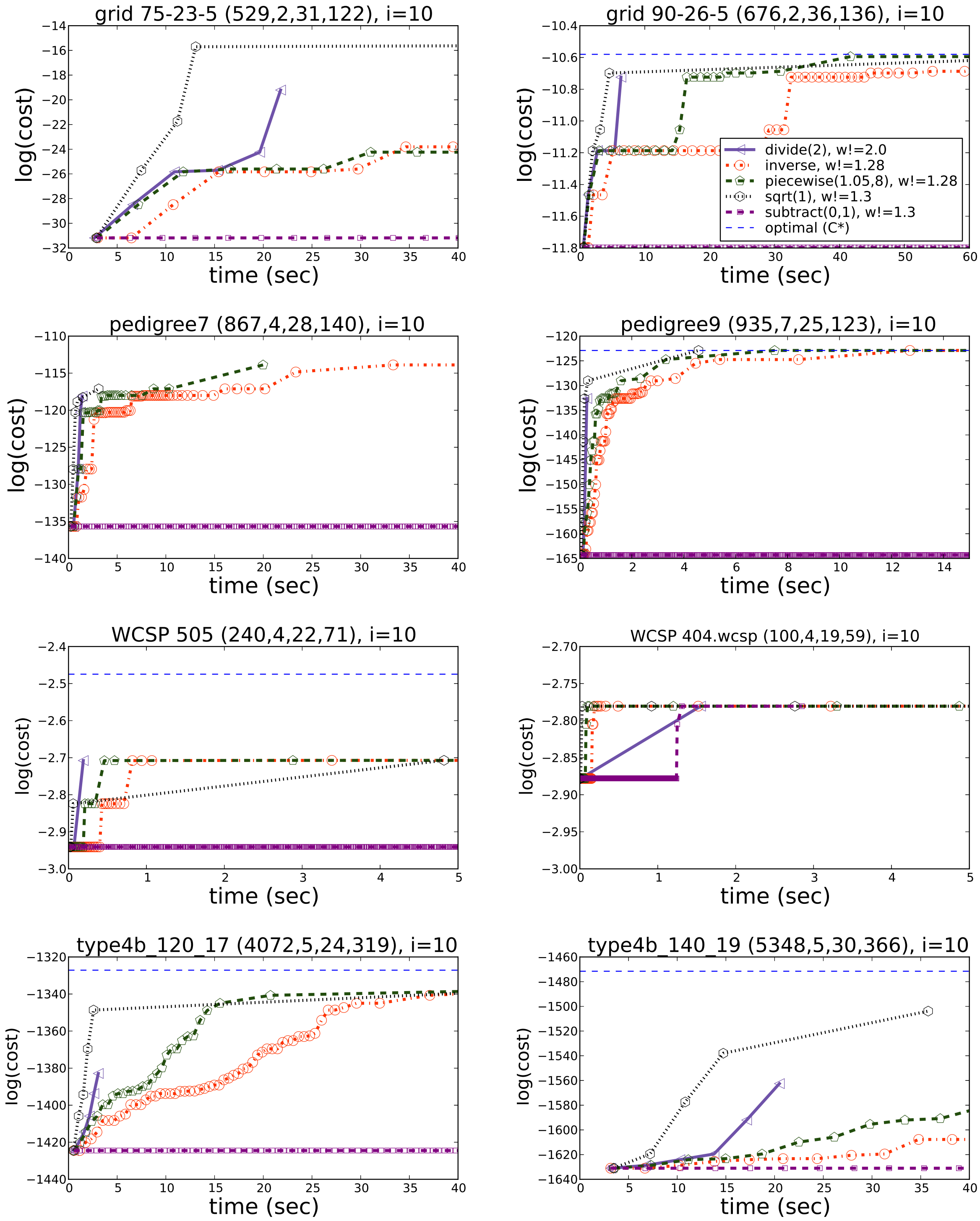


Figure 4.5: wAOBF: solution cost (on logarithmic scale) vs time (sec) for different weight policies, starting weight = 64. Instance parameters are in format (n,k,w^*,h_T), where n - number of variables, k - max. domain size, w^* - induced width, h_T - pseudo-tree height. Time limit - 1 hour, memory limit - 2 GB, MBE heuristic.

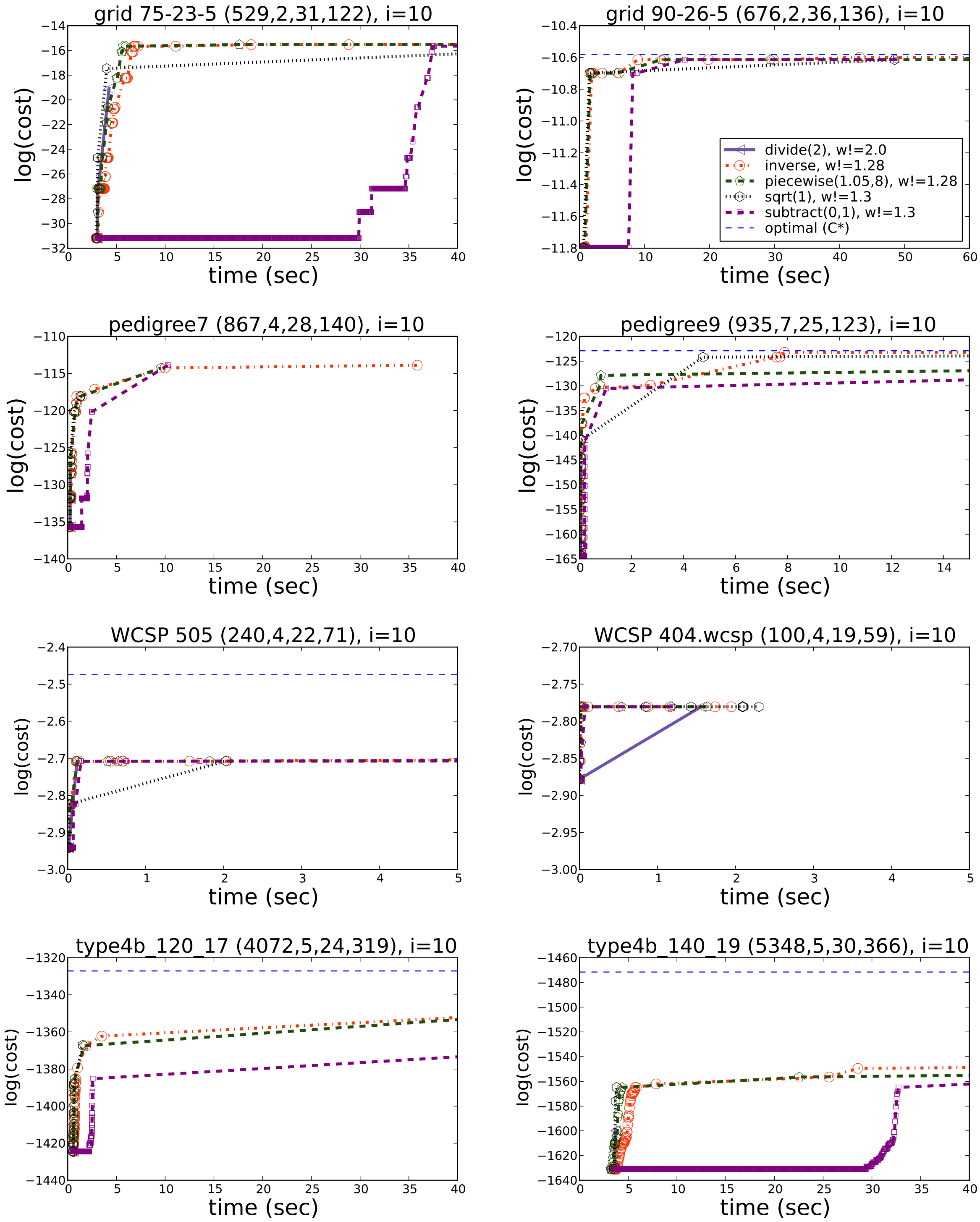


Figure 4.6: wR-AOBF: solution cost (on logarithmic scale) vs time (sec) for different weight policies, starting weight = 64. Instance parameters are in format (n,k,w^*,h_T), where n - number of variables, k - max. domain size, w^* - induced width, h_T - pseudo-tree height. Time limit - 1 hour, memory limit - 2 GB, MBE heuristic.

Figures 4.5 and **4.6** show the anytime performance of wAOBF and wR-AOBF with various weight scheduling schemes, namely how the solution cost changes as a function of time in seconds. We plot the solution cost on a logarithmic scale. Figure 4.5 displays the results for wAOBF using each of our five weight policies. We display results for an i-bound from mid-range, on two instances from each of the benchmarks: Grids, Pedigrees, WCSPs and Type4. Figure 4.6 shows analogous results for wR-AOBF, on the same instances.

Comparing the anytime performances of two schemes, we consider as better the one that finds the initial solutions faster and whose solutions are more accurate (i.e., have higher costs). Graphically, the curves closer to the left top corner of the plot are better.

Several values of numerical parameters for each policies were tried, only the ones that yielded the best performance are presented. The starting weight is 64 and $w!$ denotes the weight at the time of algorithms termination. The behavior depicted here was quite typical across instances and i-bounds. In this set of experiments the memory limit was 2 GB, with time limit of 1 hour and MBE heuristic was used.

We observe in Figure 4.5 that for most Pedigrees, Grids and Type4 problems wAOBF finds the initial solution the fastest using the *sqrt* policy (the reader is advised to consult the colored graph online). This can be seen, for example, on grid instance 75-23-5 and on type4b_120_17. The *sqrt* policy typically facilitates the fastest improvement of the initial solutions. For most of the WCSP instances, however, there is no clear dominance between the weight policies. On some instances (not shown) the *sqrt* policy is again superior. On others, such as instance 505, the difference in negligible.

Figure 4.6 depicts the same information for wR-AOBF. The variance between the results yielded by different weight policies is often very small. On many instances, such as pedigree31 or instance 505, it is almost impossible to tell which policy is superior. The dominance of *sqrt* policy is less obvious for wR-AOBF, than is was for wAOBF. On a number of problems *piecewise* and *inverse* policies are superior, often yielding almost identical results, see for example, pedigree7 or WCSP 404. However, there are still many instances, for which *sqrt* policy performs well, for example, pedigree7.

Overall, we chose to use the *sqrt* weight policy in our subsequent experiments, as it is superior on more instances for wAOBF than other policies, and is often either best or close second best for wR-AOBF.

Instance	Algorithm	Time bounds				
		10	30	60	600	3600
		log(cost) weight	log(cost) weight	log(cost) weight	log(cost) weight	log(cost) weight
Grids, i-bound=10						
75-16-5 (256,2,21,73)	wAOBF	-8.1262 1.0671	-8.0642 1.0082	-8.0642 1.0	-8.0642 1.0	-8.0642 1.0
	wR-AOBF	8.0642 1.0	-8.0642 1.0	-8.0642 1.0	-8.0642 1.0	-8.0642 1.0
75-26-5 (676,2,36,129)	wAOBF	-24.5951 2.8284	-23.0522 1.6818	-23.0522 1.6818	-23.0522 1.6818	-23.0522 1.6818
	wR-AOBF	-25.2884 1.6818	-25.2884 1.6818	-25.2884 1.6818	-25.2884 1.6818	-25.2884 1.6818
Pedigrees, i-bound=10						
pedigree7 (867,4,32,90)	wAOBF	-114.4256 1.2968	-114.4256 1.2968	-113.8887 1.1388	-113.8887 1.1388	-113.8887 **1**.1388
	wR-AOBF	-118.8305 1.2968	-118.8305 1.2968	-114.5481 1.1388	-114.5481 1.1388	-114.5481 1.1388
pedigree41 (885,5,33,100)	wAOBF	-123.6391 1.2968	-121.3366 1.1388	-121.3366 1.1388	-121.3366 1.1388	-121.3366 1.1388
	wR-AOBF	-124.656 1.2968	-121.3366 1.1388	-121.3366 1.1388	-121.3366 1.1388	-121.3366 1.1388
WCSPs, i-bound=10						
408.wcsp (200,4,34,87)	wAOBF	-2.6798 1.2968	-2.6798 1.2968	-2.6798 1.2968	-2.6798 1.2968	-2.6798 1.2968
	wR-AOBF	-2.6811 1.2968	-2.6811 1.2968	-2.6811 1.2968	-2.6811 1.2968	-2.6811 1.2968
capmo5.wcsp (200,100,100,100)	wAOBF	—	-0.2622 2.8284	-0.2622 2.8284	-0.2622 2.8284	-0.2622 2.8284
	wR-AOBF	—	-0.2622 2.8284	-0.2622 2.8284	-0.2622 2.8284	-0.2622 2.8284
Type4, i-bound=10						
type4b_150_14 (5804,5,32,522)	wAOBF	—	-1698.1897 1.6818	-1652.7112 1.2968	-1652.7112 1.2968	-1652.7112 1.2968
	wR-AOBF	—	-1763.7714 1.2968	-1763.7714 1.2968	-1763.7714 1.2968	-1763.7714 1.2968
type4b_190_20 (8186,5,29,625)	wAOBF	—	—	-2605.3849 1.2968	-2605.3849 1.2968	-2605.3849 1.2968
	wR-AOBF	—	—	-2803.4548 1.2968	-2603.6145 1.1388	-2603.6145 1.1388

Table 4.3: Solution cost (on logarithmic scale) and corresponding weight for a fixed time bound for wAOBF and wR-AOBF. ”—” denotes no solution found by the time bound. 4 GB memory, 1 hour time limit, MBE heuristic. The entries mentioned in the text are highlighted.

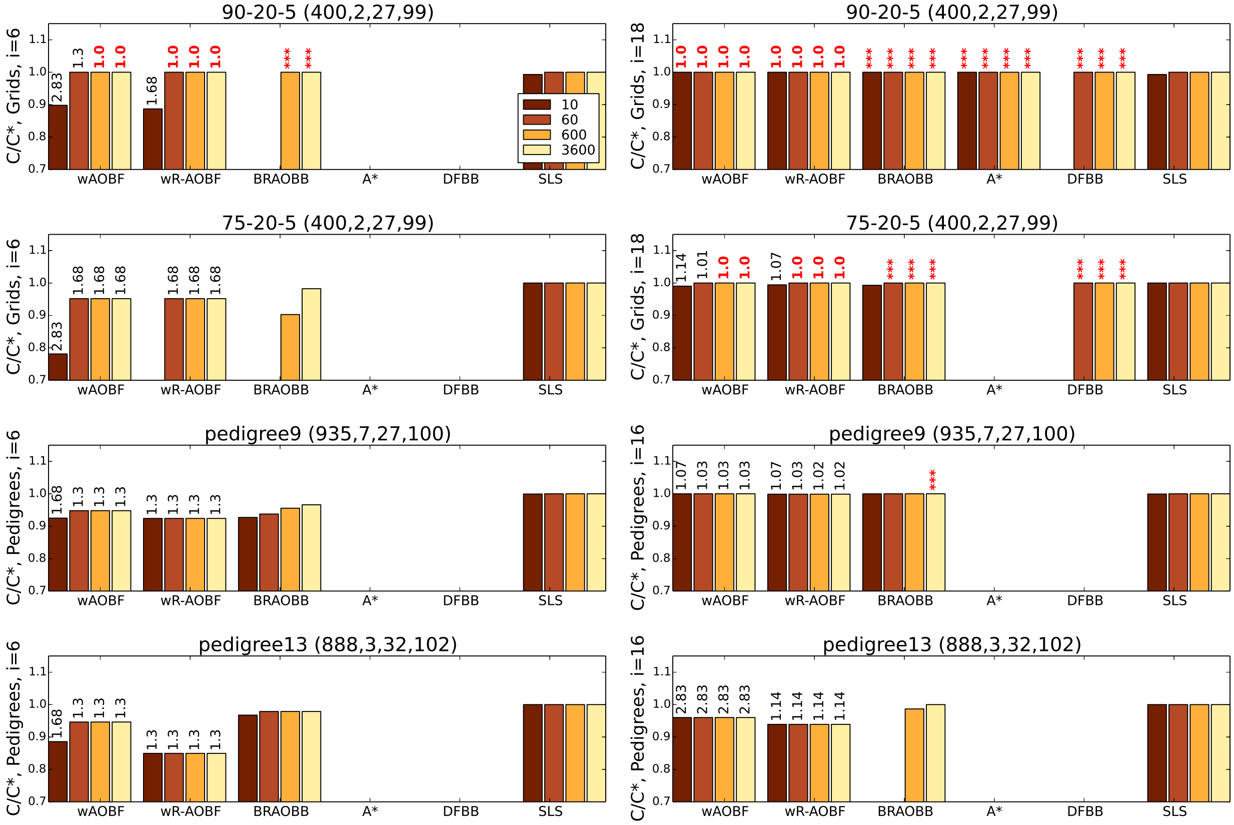


Figure 4.7: Ratio of the cost obtained by some time point (10, 60, 600 and 3600 sec) and max cost (optimal, if known, otherwise - best cost found for the problem by any of the schemes). Corresponding weight - above the bars. '***' indicated proven solution optimality. Instance parameters are in format (n,k,w^*,h_T), where n - number of variables, k - max. domain size, w^* - induced width, h_T - pseudo-tree height. Memory limit 4 GB, time limit 1 hour. Grids and Pedigrees benchmarks, MBE heuristic.

■ 4.5.4 Anytime Behavior of Weighted Heuristic Best-First Search

We now turn to our main focus of evaluating the anytime performance of our two iterative weighted heuristic best-first schemes wAOBF and wR-AOBF and comparing against BRAOBB, A*, depth-first branch and bound of OR search tree (DFBB) and Stochastic Local Search (SLS). We ran each scheme on all instances from the same 4 benchmarks with MBE heuristic using the i-bound ranging from 2 to 20. We recorded the solutions at different time points, up until either the optimal solution was found or until the algorithm ran out of 4 GB of memory or the time cut off of 3600 seconds was reached. When comparing two anytime algorithms, we consider one to be superior to another if: 1) it discovers the initial solution faster and also 2) for a fixed time it returns a more accurate solution. We rarely encounter conflict on these two measures.

We first illustrate the results using a selected set of individual instances in the Table 4.3 and in bar charts in Figures 4.7 and 4.8. Then we provide summaries over all instances using bar charts in Figures 4.9-4.12 and scatterplots in Figure 4.13.

■ 4.5.4.1 wAOBF vs wR-AOBF

Table 4.3 shows solution cost and corresponding weight by wAOBF and wR-AOBF for 2 selected instances from each benchmark, for medium i-bound, for several time bounds. We observe that for these instances (that are quite representative), the simpler scheme wAOBF provides more accurate solutions than wR-AOBF, (e.g., pedigree7, for 10-30 seconds). Still, on some instances the solution costs are equal for the same time bound (e.g., capmo05.wcsp, time between 30 and 3600 seconds), and there are examples, where wR-AOBF manages to find a more accurate solution in a comparable time, such as type4b_190_20, for 600 or 3600 seconds.

Figures 4.7 and **4.8** present the anytime performance of wAOBF, wR-AOBF, BRAOBB, A*, DFBB and SLS for representative instances using bar charts. Figure 4.7 shows results for Grids and Pedigrees, Figure 4.8 - for WCSPs and Type4. Each two rows display two problems from the same benchmark for two i-bounds, smaller i-bounds on the left and larger on the right. The height of each bar is proportional to the ratio between the solution cost generated by an algorithm at a time point (at 10, 60, 600 and 3600 seconds) and the optimal cost (if known) or overall maximal cost. The closer the ratio is to 1, the better. For readability we display the ratios greater than 0.7. Above each bar we also show the weight corresponding to the returned solution, where $w = 1$ is shown in red. The symbol '***' above bars indicates

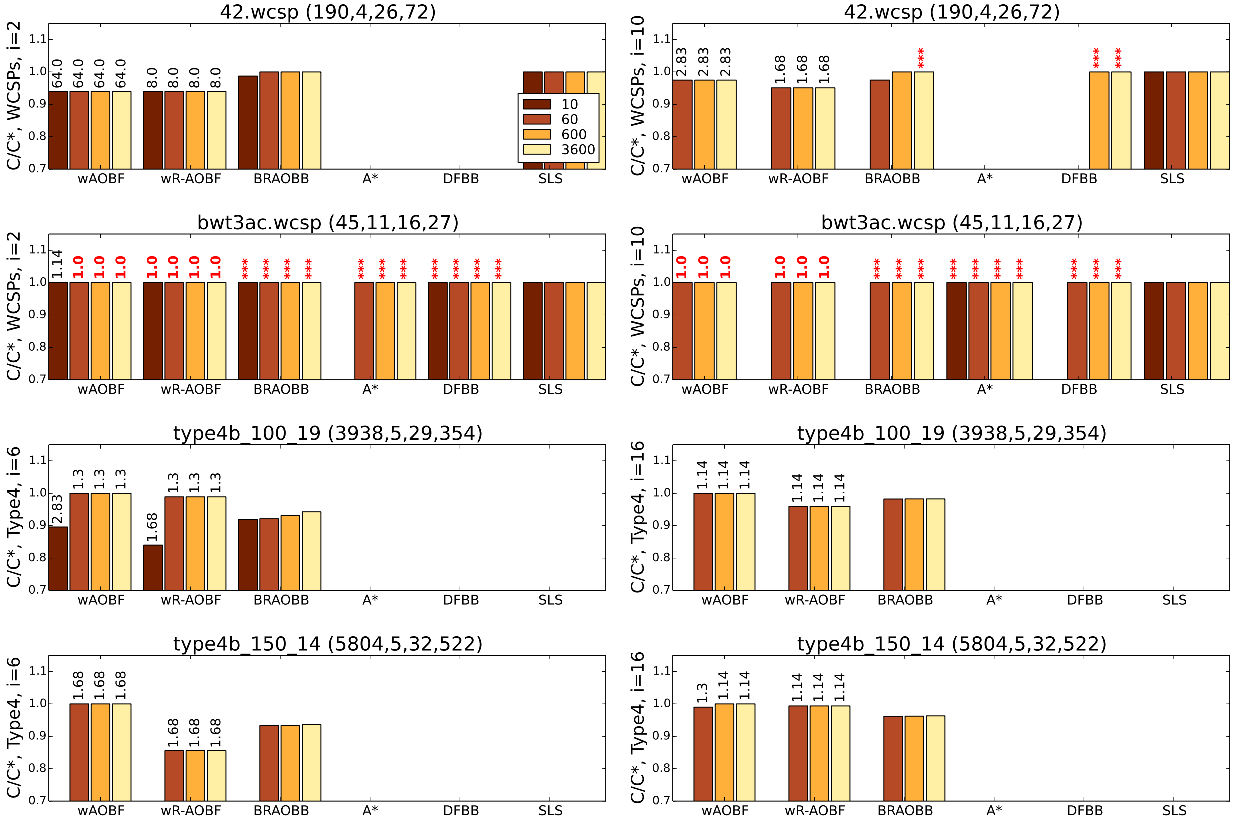


Figure 4.8: Ratio of the cost obtained by some time point (10, 60, 600 and 3600 sec) and max cost (optimal, if known, otherwise - best cost found for the problem by any of the schemes). Corresponding weight - above the bars. '***' indicated proven solution optimality. Instance parameters are in format (n,k,w^*,h_T), where n - number of variables, k - max. domain size, w^* - induced width, h_T - pseudo-tree height. Memory limit 4 GB, time limit 1 hour. WCSPs and Type4 benchmarks, MBE heuristic.

proven solution optimality.

We again observe that wR-AOBF has worse anytime behavior than wAOBF. For example, in Figure 4.7 for pedigree9, $i = 6$, wAOBF finds better solutions for the same time bounds.

These observations, based on the selected instances displayed in the table and in the figures, are quite representative, as we will see from the summaries in the next section.

The overall superiority of wAOBF may be explained by the large overhead of wR-AOBF at each iteration, due to the need to keep track of already expanded nodes and to update their evaluation function as the weight changes. As a result, wR-AOBF explores the search space slower and discovers new improved solutions less frequently than wAOBF.

■ 4.5.4.2 Comparing Against State of the Art

Figures 4.7 and 4.8 provide a different perspective of the strength of weighted schemes compared against several state of the art anytime algorithms. We observe that the dominant scheme varies from benchmark to benchmark. In **Figure 4.7** we see that wAOBF is superior on the Grid instances when the heuristic is weak. For example, for grid 75-20-5, $i = 6$ wAOBF is the only scheme that finds a solution within 10 seconds, aside from SLS. Note that SLS does not provide any guarantees, while wAOBF and wR-AOBF report for time bounds of 60 seconds or more solutions that are guaranteed to be within a factor of 1.68 from the optimal. A* and DFBB are always inferior to all other schemes. The former managed to find any solution for a small number of instances and only for highly accurate heuristics, e.g., 90-20-5, $i = 18$. The latter often reports solutions of such low accuracy that they are not seen on the plots, which only show solutions of relative accuracy greater than 0.7.

On Pedigrees BRAOBB seems to be superior among the complete schemes for weak heuristics, e.g., pedigree9, $i = 6$. SLS reports optimal costs quite fast, usually under 10 seconds. When the heuristics are stronger, the performance is less varied, except for both OR schemes on Grids and many of the Pedigrees. A* and DFBB are usually inferior even when the i-bound is high, e.g., pedigree13, $i = 16$.

Figure 4.8 shows that the weighted heuristic best-first schemes are superior to all other schemes for Type4 instances shown, for all levels of heuristic strengths. Interestingly, on most of Type4 problems even SLS did not report solutions with relative accuracy greater than 0.7 and thus its results are not shown on our plots, while wAOBF often quickly finds solutions close to optimal.

Notice that the weighted schemes provide tight w-optimality guarantees in many cases. For example, on the two Type4 instances presented in Figure 4.8 wAOBF and wR-AOBF guarantee that the costs reported at 60 seconds are 1.14-optimal. This is in contrast to competing schemes (e.g., BRAOBB) that do not prove optimality within the time limit.

On WCSPs, however, wAOBF and wR-AOBF are less effective. For example, both are inferior to BRAOBB and SLS on 42.wcsp instance, for both i-bounds. For stronger heuristic even DFBB, which in general is not particularly successful, is superior to the weighted heuristic BF schemes.

As expected, A* and DFBB, both exploring an OR search tree, are clearly inferior to AND/OR schemes, in particular to BRAOBB (see, for example, [66, 67] for more comparisons between AND/OR and OR search). Additionally, A* is not an inherently anytime scheme. Stochastic Local Search, though it often finds accurate solutions quickly, is not a complete algorithm and thus is outside state of the art schemes with which we systematically compared. It also does not provide any bound on its solution.

We now show data summarizing the results from all the instances. **Figures 4.9-4.12** provide summaries of our experimental results in the form of bar charts. Note that the summaries include also a weighted depth-first branch and bound algorithm wAOBB that will be introduced in Section 4.6. We defer the comparison with wAOBB until Section 4.6.2.2.

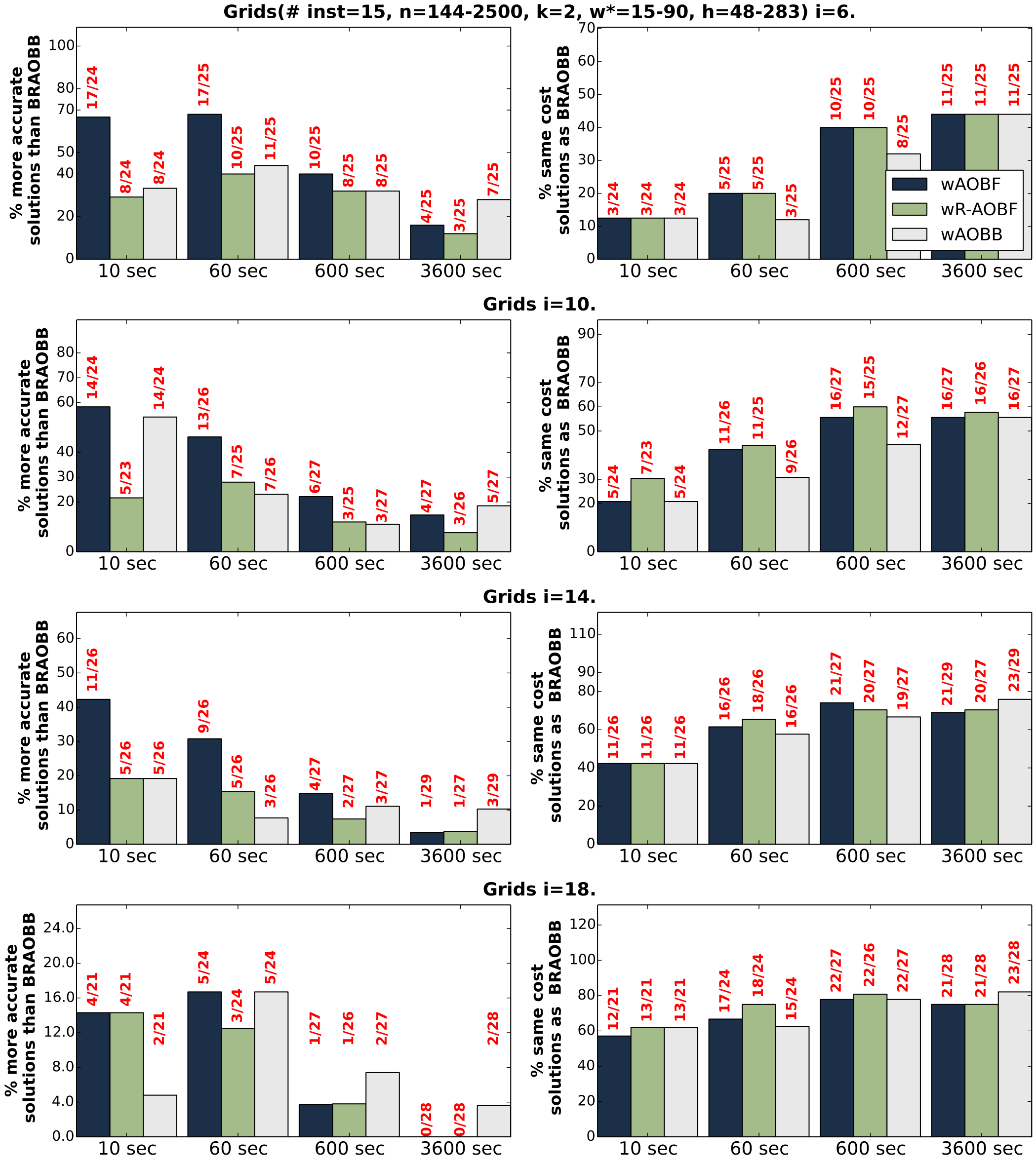
The figures compare each of the three algorithms (wAOBF, wR-AOBF and wAOBB) with BRAOBB. There are two columns of bar charts in each figure. The left one summarizes the percentage of instances in which an algorithm is more accurate compared with BRAOBB, and the right one provides the percentages on instances where an algorithm yields the same costs as BRAOBB. Therefore, the heights of the bars in Figures 4.9-4.12 are proportional to these percentages.

The ratio at the top of each bar is the actual number of instances where an algorithm is better (left column) or equal (right column), divided by the total number of instances solved by the algorithm. Tables 4.4 and 4.5 present the summary of the results in table form, including an additional time bound. For each benchmark for 5 time bounds and for 4 i-bounds we show the percentages of the instances for which a particular weighted algorithm found a more accurate solution than BRAOBB ($\%X$), and for which the algorithm found the solution of the same cost as BRAOBB ($\%Y$). We also show the number of instances solved by each algorithm (N).

From these figures we see that on two out of four our benchmarks the weighted heuristic schemes dominated, finding costs of better accuracy within the time limits. This superior behavior on Grids and Type4 benchmarks was mostly consistent across time bounds and heuristics of various strengths and can be observed in Figures 4.9 and 4.11. Specifically, for certain i-bounds and time bounds wAOBF returned more accurate costs than BRAOBB on up to 68% of the Grid instances and on 90-100% of the Type4 instances. Interestingly, for Type4 benchmark the weighted schemes almost never found the same solution costs as BRAOBB, as is indicated by the zeros in the plots on the right side of Figure 4.11.

The comparative strength of wAOBF is more pronounced for short time intervals and for weak heuristics. The dependence of wR-AOBF's performance on time is less predictable. We observe here that for most i-bounds and for most time limits wAOBF performs better than wR-AOBF. This is especially obvious on Type4 benchmark and for small i-bound, e.g., for $i = 6$, 3600 seconds wR-AOBF is superior to BRAOBB only on 50% of the instances, while wAOBF dominates on 90%.

Figure 4.13 uses scatter diagrams to compare between the more successful of the two weighted heuristic best-first schemes, wAOBF, and BRAOBB. Each plot corresponds to a specific time point. The x-axis coordinate gives the relative accuracy of the solution obtained


Grids(# inst=15, n=144-2500, k=2, w*=15-90, h=48-283) i=6.

Grids i=10.

Grids i=14.

Grids i=18.

Figure 4.9: Grids: percentage of instances for which each algorithm found solutions of more accurate cost than BRAOBB at a specific time bound (left), percentage of instances for which algorithm is tied with BRAOBB, i.e., found solution of equal cost (right). Above bars - number of instances where algorithm is better than BRAOBB (tied with BRAOBB) and number of instances solved by the algorithm. Parameters: # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo-tree height. 4 GB memory, 1 hour time limit.

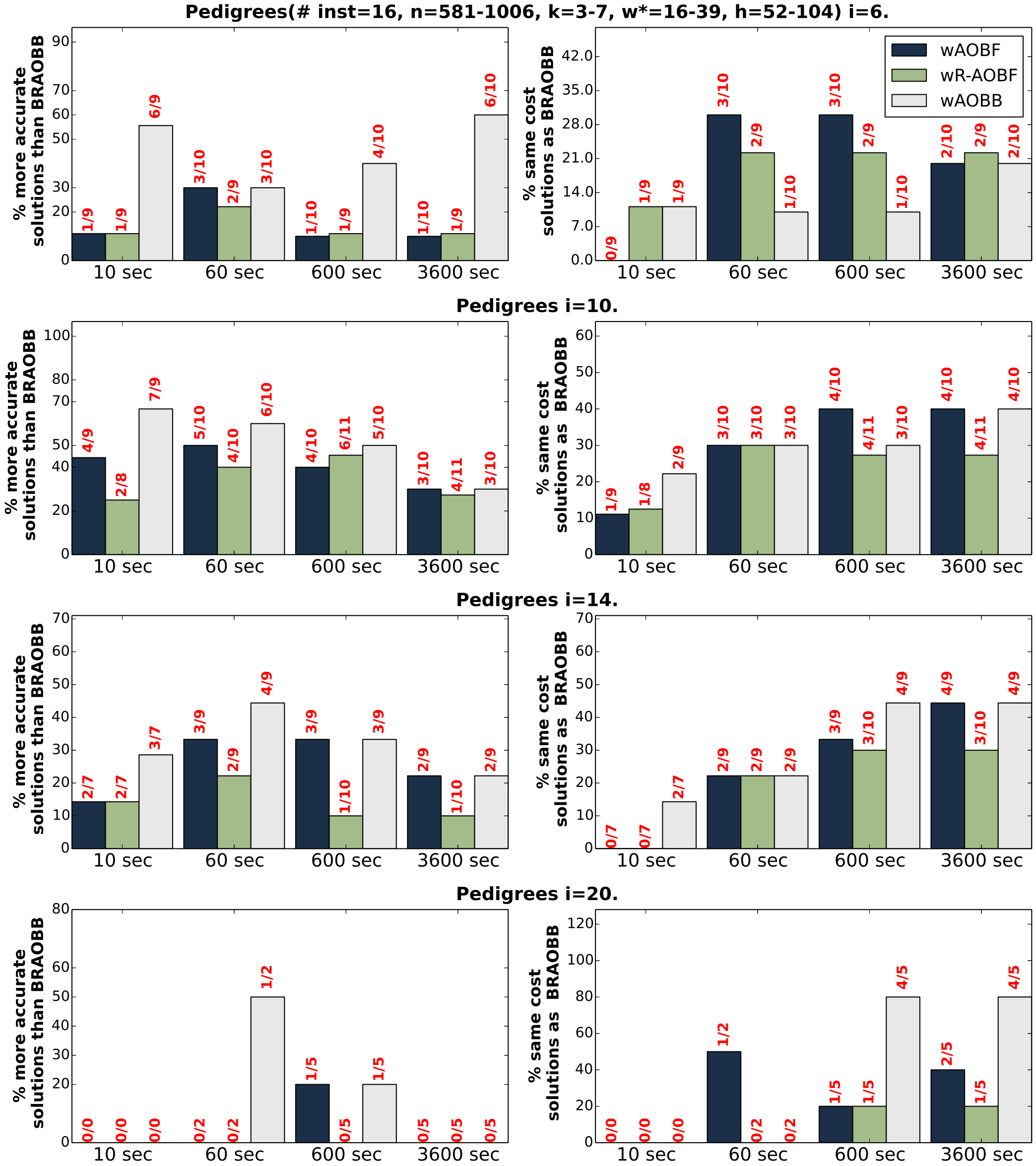


Figure 4.10: Pedigrees: percentage of instances for which each algorithm found solutions of more accurate cost than BRAOBB at a specific time bound (left), percentage of instances for which algorithm is tied with BRAOBB, i.e., found solution of equal cost (right). Above bars - number of instances where algorithm is better than BRAOBB (tied with BRAOBB) and number of instances solved by the algorithm. Parameters: # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo-tree height. 4 GB memory, 1 hour time limit.

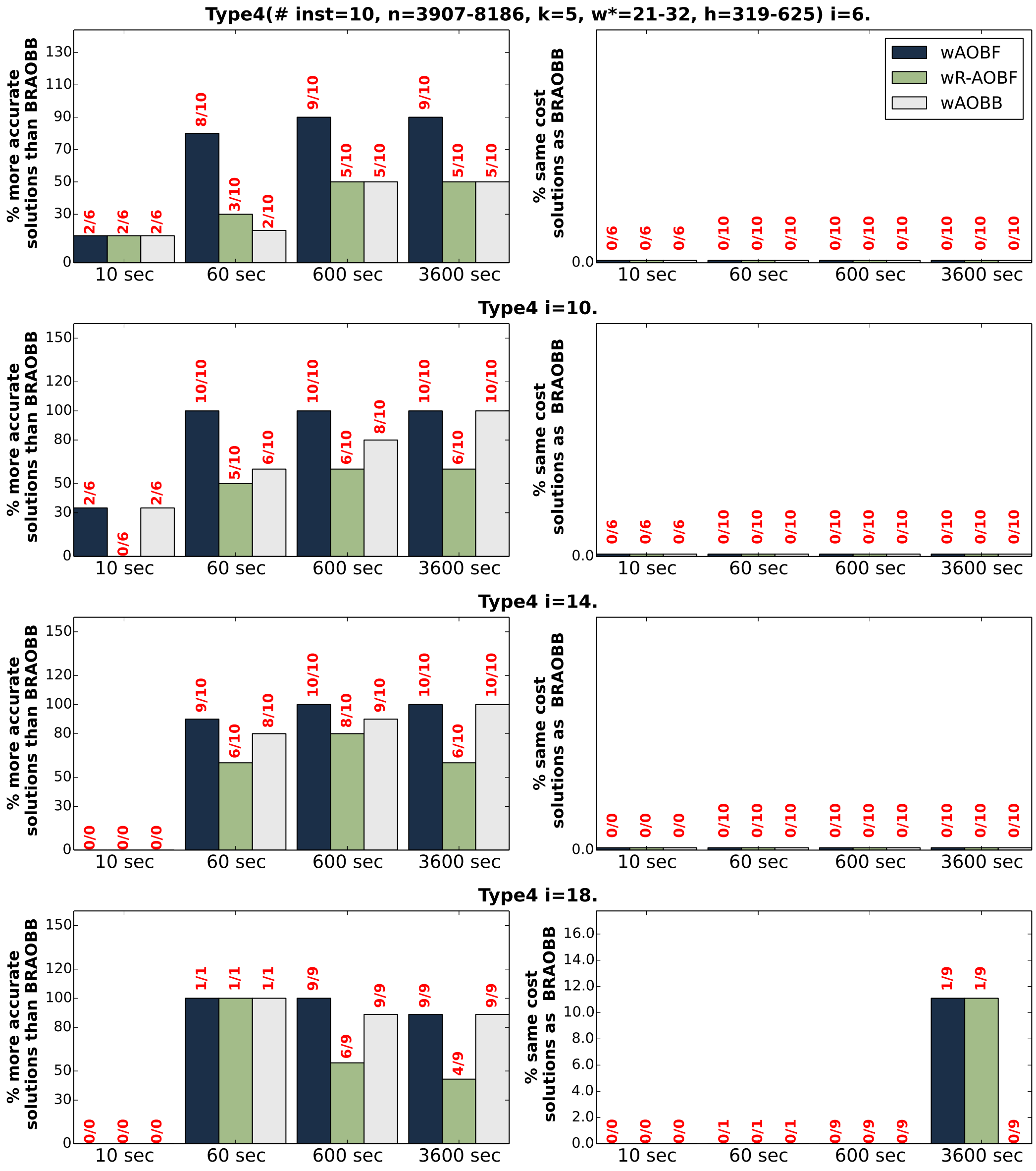


Figure 4.11: Type4: percentage of instances for which each algorithm found solutions of more accurate cost than BRAOBB at a specific time bound (left), percentage of instances for which algorithm is tied with BRAOBB, i.e., found solution of equal cost (right). Above bars - number of instances where algorithm is better than BRAOBB (tied with BRAOBB) and number of instances solved by the algorithm. Parameters: # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo-tree height. 4 GB memory, 1 hour time limit.

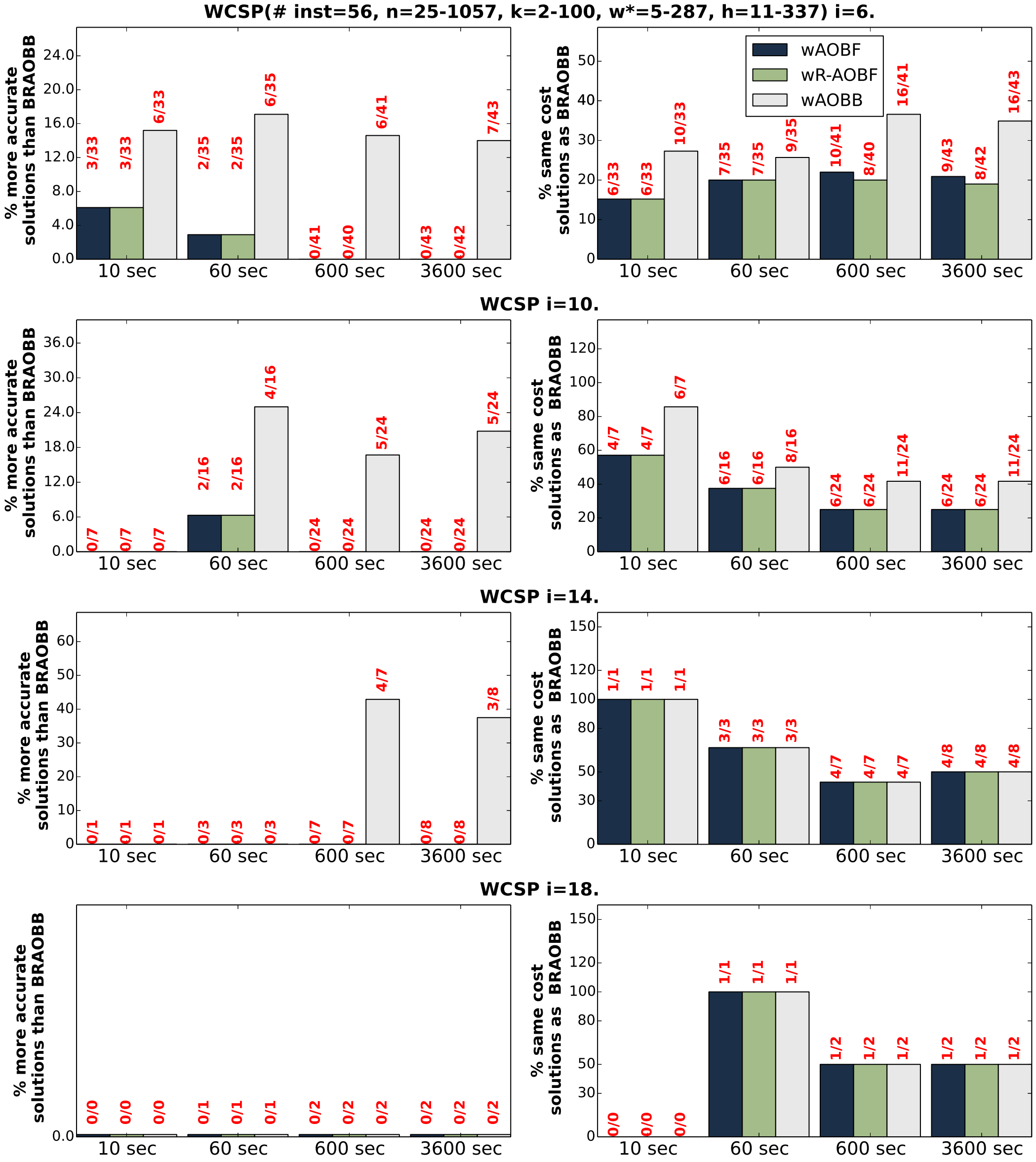


Figure 4.12: WCSP: percentage of instances for which each algorithm found solutions of more accurate cost than BRAOBB at a specific time bound (left), percentage of instances for which algorithm is tied with BRAOBB, i.e., found solution of equal cost (right). Above bars - number of instances where algorithm is better than BRAOBB (tied with BRAOBB) and number of instances solved by the algorithm. Parameters: # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo-tree height. 4 GB memory, 1 hour time limit.

I-bound	Algorithm	Time bounds				
		10	30	60	600	3600
		X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N
Grids (# inst=32, n=144-2500, k=2, w^*=15-90, h_T=48-283)						
$i = 6$	wAOBF	66.7 / 12.5 / 24	64.0 / 16.0 / 25	68.0 / 20.0 / 25	40.0 / 40.0 / 25	16.0 / 44.0 / 25
	wR-AOBF	29.2 / 12.5 / 24	45.8 / 16.7 / 24	40.0 / 20.0 / 25	32.0 / 40.0 / 25	12.0 / 44.0 / 25
	wAOBB	33.3 / 12.5 / 24	40.0 / 12.0 / 25	44.0 / 12.0 / 25	32.0 / 32.0 / 25	28.0 / 44.0 / 25
$i = 10$	wAOBF	58.3 / 20.8 / 24	56.0 / 32.0 / 25	46.2 / 42.3 / 26	22.2 / 55.6 / 27	14.8 / 55.6 / 27
	wR-AOBF	21.7 / 30.4 / 23	37.5 / 33.3 / 24	28.0 / 44.0 / 25	12.0 / 60.0 / 25	7.7 / 57.7 / 26
	wAOBB	54.2 / 20.8 / 24	44.0 / 24.0 / 25	23.1 / 30.8 / 26	11.1 / 44.4 / 27	18.5 / 55.6 / 27
$i = 14$	wAOBF	42.3 / 42.3 / 26	38.5 / 46.2 / 26	30.8 / 61.5 / 26	14.8 / 74.1 / 27	3.4 / 69.0 / 29
	wR-AOBF	19.2 / 42.3 / 26	19.2 / 50.0 / 26	15.4 / 65.4 / 26	7.4 / 70.4 / 27	3.7 / 70.4 / 27
	wAOBB	19.2 / 42.3 / 26	19.2 / 46.2 / 26	7.7 / 57.7 / 26	11.1 / 66.7 / 27	10.3 / 75.9 / 29
$i = 18$	wAOBF	14.3 / 57.1 / 21	16.7 / 66.7 / 24	16.7 / 66.7 / 24	3.7 / 77.8 / 27	0.0 / 75.0 / 28
	wR-AOBF	14.3 / 61.9 / 21	12.5 / 75.0 / 24	12.5 / 75.0 / 24	3.8 / 80.8 / 26	0.0 / 75.0 / 28
	wAOBB	4.8 / 61.9 / 21	8.3 / 58.3 / 24	16.7 / 62.5 / 24	7.4 / 77.8 / 27	3.6 / 82.1 / 28
Pedigrees (# inst=11, n=581-1006, k=3-7, w^*=16-39, h_T=52-104)						
$i = 6$	wAOBF	11.1 / 0.0 / 9	11.1 / 11.1 / 9	30.0 / 30.0 / 10	10.0 / 30.0 / 10	10.0 / 20.0 / 10
	wR-AOBF	11.1 / 11.1 / 9	11.1 / 22.2 / 9	22.2 / 22.2 / 9	11.1 / 22.2 / 9	11.1 / 22.2 / 9
	wAOBB	55.6 / 11.1 / 9	22.2 / 11.1 / 9	30.0 / 10.0 / 10	40.0 / 10.0 / 10	60.0 / 20.0 / 10
$i = 10$	wAOBF	44.4 / 11.1 / 9	50.0 / 30.0 / 10	50.0 / 30.0 / 10	40.0 / 40.0 / 10	30.0 / 40.0 / 10
	wR-AOBF	25.0 / 12.5 / 8	44.4 / 22.2 / 9	40.0 / 30.0 / 10	45.5 / 27.3 / 11	27.3 / 27.3 / 11
	wAOBB	66.7 / 22.2 / 9	60.0 / 20.0 / 10	60.0 / 30.0 / 10	50.0 / 30.0 / 10	30.0 / 40.0 / 10
$i = 14$	wAOBF	14.3 / 0.0 / 7	28.6 / 14.3 / 7	33.3 / 22.2 / 9	33.3 / 33.3 / 9	22.2 / 44.4 / 9
	wR-AOBF	14.3 / 0.0 / 7	14.3 / 14.3 / 7	22.2 / 22.2 / 9	10.0 / 30.0 / 10	10.0 / 30.0 / 10
	wAOBB	28.6 / 14.3 / 7	42.9 / 14.3 / 7	44.4 / 22.2 / 9	33.3 / 44.4 / 9	22.2 / 44.4 / 9
$i = 20$	wAOBF	0 / 0 / 0	0 / 0 / 0	0.0 / 50.0 / 2	20.0 / 20.0 / 5	0.0 / 40.0 / 5
	wR-AOBF	0 / 0 / 0	0 / 0 / 0	0.0 / 0.0 / 2	0.0 / 20.0 / 5	0.0 / 20.0 / 5
	wAOBB	0 / 0 / 0	0 / 0 / 0	50.0 / 0.0 / 2	20.0 / 80.0 / 5	0.0 / 80.0 / 5

Table 4.4: X% - percentage of instances for which each algorithm is the better than BRAOBB at a specific time bound, Y% - percentage of instances for which algorithm ties with BRAOBB, N - number of instances for which at least one of algorithms found a solution. # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo-tree height. 4 GB memory, 1 hour time limit, MBE heuristic.

by wAOBF, the y-axis coordinate gives the relative accuracy of BRAOBB. As for the bar charts, the accuracy is defined relative to the optimal cost, if known, or the maximum cost available. Values closer to 1.0 indicate better results. In each row we show two time bounds for a particular benchmark. In parenthesis we show the number of instances, for which at least one of the displayed algorithms found a solution, and the total number of instances in benchmark. Each marker represents an instance. The markers under the red diagonal correspond to problems where wAOBF is superior. We do not account in these plots for the

I-bound	Algorithm	Time bounds				
		10	30	60	600	3600
		X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N
WCSP (# inst=56, n=25-1057, k=2-100, w^*=5-287, h_T=11-337)						
$i = 6$	wAOBF	6.1 / 15.2 / 33	3.0 / 18.2 / 33	2.9 / 20.0 / 35	0.0 / 22.0 / 41	0.0 / 20.9 / 43
	wR-AOBF	6.1 / 15.2 / 33	3.0 / 18.2 / 33	2.9 / 20.0 / 35	0.0 / 20.0 / 40	0.0 / 19.0 / 42
	wAOBB	15.2 / 27.3 / 33	15.2 / 27.3 / 33	17.1 / 25.7 / 35	14.6 / 36.6 / 41	14.0 / 34.9 / 43
$i = 10$	wAOBF	0.0 / 57.1 / 7	7.1 / 28.6 / 14	6.3 / 37.5 / 16	0.0 / 25.0 / 24	0.0 / 25.0 / 24
	wR-AOBF	0.0 / 57.1 / 7	7.1 / 28.6 / 14	6.3 / 37.5 / 16	0.0 / 25.0 / 24	0.0 / 25.0 / 24
	wAOBB	0.0 / 85.7 / 7	28.6 / 50.0 / 14	25.0 / 50.0 / 16	16.7 / 41.7 / 24	20.8 / 41.7 / 24
$i = 14$	wAOBF	0.0 / 100.0 / 1	0.0 / 50.0 / 2	0.0 / 66.7 / 3	0.0 / 42.9 / 7	0.0 / 50.0 / 8
	wR-AOBF	0.0 / 100.0 / 1	0.0 / 50.0 / 2	0.0 / 66.7 / 3	0.0 / 42.9 / 7	0.0 / 50.0 / 8
	wAOBB	0.0 / 100.0 / 1	0.0 / 50.0 / 2	0.0 / 66.7 / 3	42.9 / 42.9 / 7	37.5 / 50.0 / 8
$i = 18$	wAOBF	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 50.0 / 2	0.0 / 50.0 / 2
	wR-AOBF	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 50.0 / 2	0.0 / 50.0 / 2
	wAOBB	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 50.0 / 2	0.0 / 50.0 / 2
Type4 (# inst=10, n=3907-8186, k=5, w^*=21-32, h_T=319-625)						
$i = 6$	wAOBF	16.7 / 0.0 / 6	44.4 / 0.0 / 9	80.0 / 0.0 / 10	90.0 / 0.0 / 10	90.0 / 0.0 / 10
	wR-AOBF	16.7 / 0.0 / 6	33.3 / 0.0 / 9	30.0 / 0.0 / 10	50.0 / 0.0 / 10	50.0 / 0.0 / 10
	wAOBB	16.7 / 0.0 / 6	11.1 / 0.0 / 9	20.0 / 0.0 / 10	50.0 / 0.0 / 10	50.0 / 0.0 / 10
$i = 10$	wAOBF	33.3 / 0.0 / 6	100.0 / 0.0 / 9	100.0 / 0.0 / 10	100.0 / 0.0 / 10	100.0 / 0.0 / 10
	wR-AOBF	0.0 / 0.0 / 6	44.4 / 0.0 / 9	50.0 / 0.0 / 10	60.0 / 0.0 / 10	60.0 / 0.0 / 10
	wAOBB	33.3 / 0.0 / 6	44.4 / 0.0 / 9	60.0 / 0.0 / 10	80.0 / 0.0 / 10	100.0 / 0.0 / 10
$i = 14$	wAOBF	0 / 0 / 0	71.4 / 0.0 / 7	90.0 / 0.0 / 10	100.0 / 0.0 / 10	100.0 / 0.0 / 10
	wR-AOBF	0 / 0 / 0	57.1 / 0.0 / 7	60.0 / 0.0 / 10	80.0 / 0.0 / 10	60.0 / 0.0 / 10
	wAOBB	0 / 0 / 0	71.4 / 0.0 / 7	80.0 / 0.0 / 10	90.0 / 0.0 / 10	100.0 / 0.0 / 10
$i = 18$	wAOBF	0 / 0 / 0	0 / 0 / 0	100.0 / 0.0 / 1	100.0 / 0.0 / 9	88.9 / 11.1 / 9
	wR-AOBF	0 / 0 / 0	0 / 0 / 0	100.0 / 0.0 / 1	55.6 / 0.0 / 9	44.4 / 11.1 / 9
	wAOBB	0 / 0 / 0	0 / 0 / 0	100.0 / 0.0 / 1	88.9 / 0.0 / 9	88.9 / 0.0 / 9

Table 4.5: X% - percentage of instances for which each algorithm is the better than BRAOBB at a specific time bound, Y% - percentage of instances for which algorithm ties with BRAOBB, N - number of instances for which at least one of algorithms found a solution. # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo-tree height. 4 GB memory, 1 hour time limit, MBE heuristic.

MBE heuristic calculation time, which is the same for all schemes.

We show results with weak heuristic because it yields greater diversity in solution accuracy by wAOBF and BRAOBB. Typically, for strong heuristics the algorithms yields solutions having similar costs. Appendix C.1 includes additional scatter diagrams showing results for 3 i-bounds and 3 time bounds per benchmark, along with scatter plots comparing wAOBF and wR-AOBF.

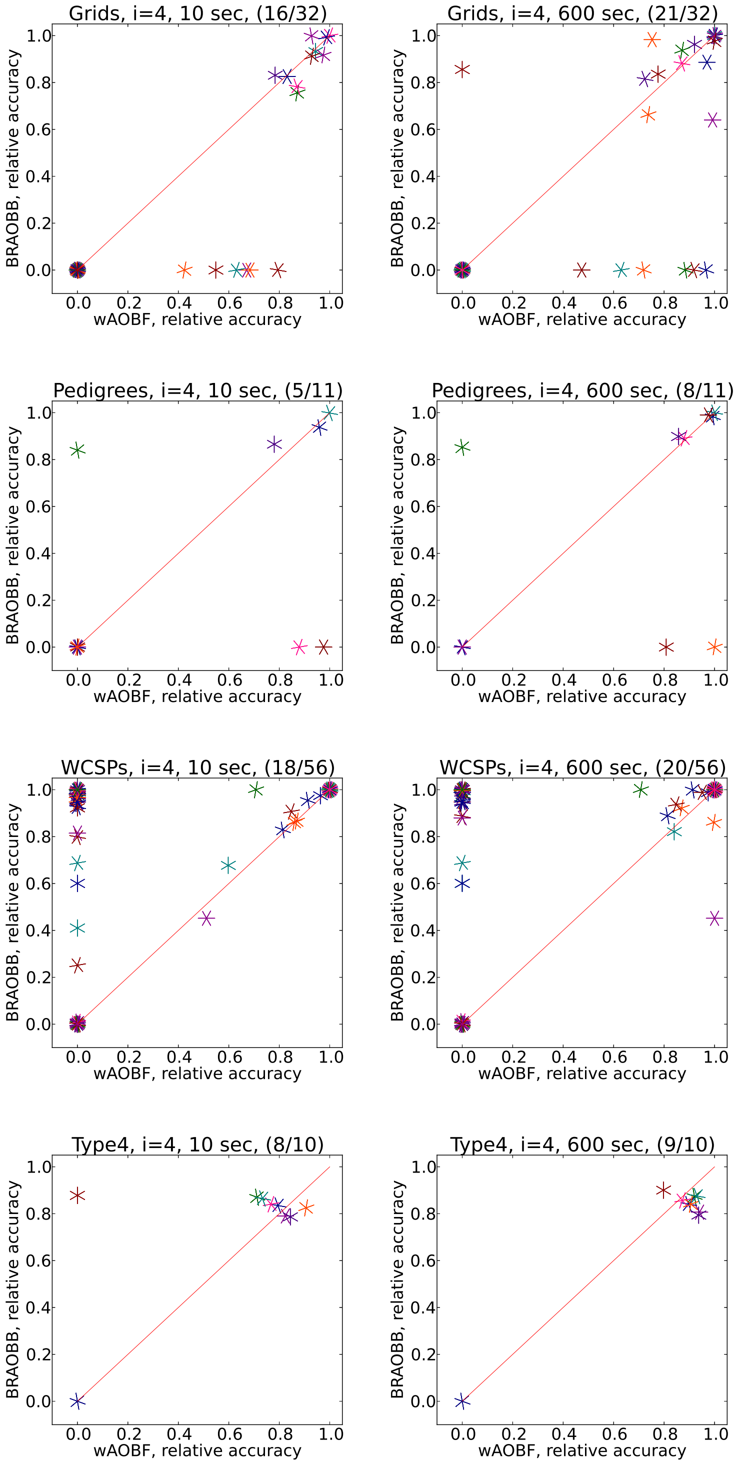


Figure 4.13: wAOBF vs BRAOBB: comparison of relative accuracy at times 10, 3600 sec. Each row shows a single time bound. Each marker represents a single instance. Memory limit 4 GB, time limit 1 hour. In parenthesis (X/Y): X - # instances, for which at least one algorithm found a solution, Y - total # instances, MBE heuristic.

Algorithm 19: AOBB(h_i, w_0, $UB = \infty$) exploring AND/OR search tree [69]

```
Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$, weight $w_0$ (default value 1), pseudo-tree $\mathcal{T}$ rooted at $X_1$, heuristic $h_i$;
Output: Optimal solution to $\mathcal{M}$
create root OR node $s$ labelled by $X_1$ and let stack of created but not expanded nodes OPEN = $\{s\}$;
initialize $v(s) = \infty$ and best partial solution tree rooted in $s$ $T^*(s) = \emptyset$; $UB = \infty$;
while $OPEN \neq \emptyset$ do
    select top node $n$ on OPEN.
    //EXPAND
    if $n$ is OR node labelled $X_i$ then
        foreach $x_i \in D(X_i)$ do
            //Expand node $n$:
            add AND child $n' = \langle X_i, x_i \rangle$ to list of successors of $n$;
            initialize $v(n') = 0$, best partial solution tree rooted in $n$ $T^*(n') = \emptyset$;
    if $n$ is AND node labelled $\langle X_i, x_i \rangle$ then
        foreach OR ancestor $k$ of $n$ do
            recursively evaluate the cost of the partial solution tree rooted in $k$, based on heuristic $h_i$ and weight
            $w_0$, assign its cost to $f(k)$; // see evalPartialSolutionTree($T_n^*$, $h_i(n)$, $w_0$) in Algorithm 20
            if evaluated partial solution is not better than current upper bound at $k$ (e.g., $f(k) \geq v(k)$ for
            minimization then
                prune the subtree below the current tip node $n$;
            else
                foreach successor $X_j$ of $X_i \in \mathcal{T}$ do
                    add OR child $n' = X_j$ to list of successors of $n$;
                    initialize $v(n') = \infty$, best partial solution tree rooted in $n$ $T^*(n') = \emptyset$;
    add successors of $n$ on top of OPEN;
    //PROPAGATE
    //Only propagate if all children are evaluated and the final $v$ are determined
    while list of successors of node $n$ is empty do
        if node $n$ is the root node then
            return solution: $v(n), T^*(n)$ ;
        else
            //update ancestors of $n$, AND and OR nodes $p$, bottom up:
            if $p$ is AND node then
                $v(p) = v(p) + v(n)$, $T^*(p) = T^*(p) \cup T^*(n)$;
            else if $p$ is OR node then
                if the new value of better than the old one, e.g., $v(p) > (c(p,n) + v(n))$ for minimization then
                    $v(p) = c(p,n) + v(n)$, $T^*(p) = T^*(p) \cup \langle x_i, X_i \rangle$;
        remove $n$ from the list of successors of $p$;
        move one level up: $n \leftarrow p$;
```

Figure 4.13 confirms our previous conclusions of superiority of wAOBF over BRAOBB on Grids and its lack of success on WCSPs. On Type4 for small time limits BRAOBB and wAOBF are on average equally good (e.g., for 10 sec), but when given more time, wAOBF produces better solutions (e.g., for 600 sec). For Pedigrees there is no clear dominance.

Algorithm 20: Recursive computation of the heuristic evaluation function [69]

```
function evalPartialSolutionTree($T'_n$, $h(n)$, $w$)
Input: Partial solution subtree $T'_n$ rooted at node $n$, heuristic function $h(n)$;
Output: Heuristic evaluation function $f(T'_n)$;
if $succ(n) == \emptyset$ then
    return $h(n) \cdot w$;
else
    if $n$ is an AND node then
        let $k_1, \dots, k_l$ be the OR children of $n$;
        return $\sum_{i=1}^{l}$ evalPartialSolutionTree($T'_{k_i}$, $h(k_i)$, $w$);
    else if $n$ is an OR node then
        let $k$ be the AND child of $n$;
        return $c(n, k)$ + evalPartialSolutionTree($T'_k$, $h(k)$, $w$);
```

■ 4.6 Weighted Heuristic Depth-First BB for Graphical Models

The primary reason for using weighted heuristics in the context of best-first search is to convert it into memory effective anytime scheme and to get a solution with some bounded guarantee. Since depth-first branch and bound schemes are already inherently anytime, the idea of using weighted heuristic search may seem irrelevant. However, branch and bound schemes do not provide any guarantees when terminating early. So a desire to have beneficial w-optimality bounds on solutions and to possibly improve the anytime performance intrigued us into exploring the principle of weighted heuristic search for depth-first search schemes as well. Specifically, weighted heuristic may guide the traversal of the search space in a richer manner and may lead to larger and more effective pruning of the space.

Therefore, in this section we extend the depth-first branch and bound algorithms AOBB and BRAOBB to weighted anytime schemes, yielding wAOBB and wBRAOBB, and evaluate their performance in much the same way we did for the weighted heuristic best-first search algorithms.

■ 4.6.1 Introducing the Weighted Branch and Bound Schemes

The extension of AOBB to weighted heuristic search is straightforward. Just multiply the heuristic value by the weight $w > 1$ and conduct AOBB as usual. We denote by AOBB(h_i,w_0, UB) a weighted version of AOBB that uses the mini-bucket heuristic h_i having i-bound equal to i, multiplied by the weight w_0, and an initial upper bound equal to UB, as shown in Algorithm 19. It is easy to show that:

Theorem 4.7. *Algorithm AOBB(h_i,$w_0 > 1$,UB$= \infty$) (or BRAOBB) terminates with a solution π, whose cost C_π is a factor w_0 away from the optimal cost C^*. Namely, $C_\pi \leq w_0 \cdot C^*$.*

Proof. By definition, due to pruning, AOBB generates solutions in order of decreasing costs: $C_1 \geq \cdots \geq C_i \cdots \geq C_\pi$, where C_π is the returned solution. If π is not optimal (otherwise the claim is trivially proved), there exists an optimal solution π^* which must have been pruned by the algorithm. Let n be the last node on π^* that was generated and which was pruned. Since the heuristic h is admissible, the un-weighted evaluation function of f along π^* satisfies that

$$f_{\pi^*}(n) = g(n) + h(n) \leq g(n) + h^*(n) = C^* \tag{4.3}$$

Let C_i be the solution cost used to prune n (namely it pruned relative to the weighted evaluation function). Therefore,

$$C_i \leq g(n) + w \cdot h(n)$$

Therefore (as $w \geq 1$) and from Equation 4.3

$$C_i \leq w \cdot (g(n) + h(n)) \leq w \cdot C^*$$

and since $C_\pi \leq C_i$, we get

$$C_\pi \leq w \cdot C^*.$$

Algorithm 21: wAOBB(w_0, h_i)

```
Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$; heuristic $h_i$ obtained with i-bound $i$; initial weight $w_0$
Output: $\mathcal{C}$ - a set of suboptimal solutions $C_w$, each with a bound $w$
Initialize $j = 1$, $UB_j = \infty$, $w_j = w_0$, weight update schedule $S$ and let $\mathcal{C} \leftarrow \emptyset$;
while $w_j >= 1$ do
    while $AOBB(h_i,w_j, UB_j)$ not terminated do
        run AOBB($h_i$,$w_j$, $UB_j$)
        if AOBB found an intermediate solution $C'_j$ then
            output the solution bounded by the weight of previous iteration: $\mathcal{C} \leftarrow \mathcal{C} \cup \{\langle w_{j-1}, C'_j \rangle\}$
    output the solution with which AOBB terminated, bounded by the current weight:
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{\langle w_j, C_j \rangle\}$
    Decrease weight $w$ according to schedule $S$;
    $UB \leftarrow C_j$
    return $\mathcal{C}$
```

□

We present two iterative weighted heuristic branch and bound schemes denoted wAOBB and wBRAOBB. Similar to wAOBF, these algorithms iteratively execute the corresponding base algorithm with the weighted heuristic, AOBB(h_i,w_0,UB) and BRAOBB(h_i,w_0,UB). However, there are some inherent differences between these two schemes and wAOBF, explained next.

Iterative Weighted AOBB (wAOBB). At the first iteration (Algorithm 21) wAOBB executes AOBB(h_i,w_0,$UB = \infty$), namely AOBB with heuristic $h_i \cdot w_0$ and with default upper bound of infinity. The algorithm does no pruning until it discovers its first solution. Then the upper bound is set to the current best cost. At termination of the first iteration, the algorithm returns the final solution and its cost C_1 with the corresponding weight $w_1 = w_0$. During each subsequent iteration $j \geq 2$ wAOBB executes AOBB(h_i,w_j,UB_j) to completion. The weight w_j is decreased according to the weight policy. The input upper bound UB_j is the cost of the solution returned in iteration $j - 1$, i.e., $UB_j = C_{j-1}$. We denote by C'_j the costs of the intermediate solutions wAOBB generates during iteration j, until it terminates with the final solution, having cost C_j.

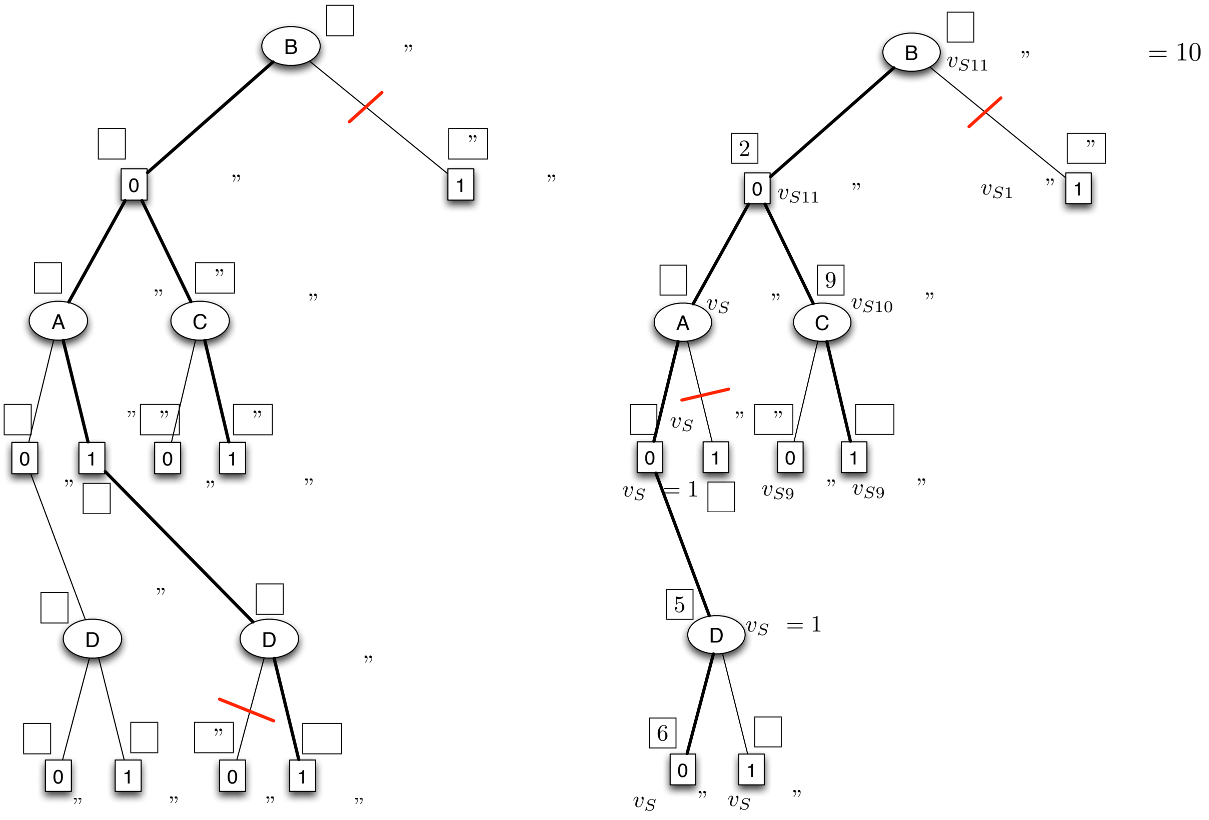


Figure 4.14: Search graphs explored by (a) AOBB, (b) First iteration of wAOBB, w=10. Boxed numbers indicate the order in which the nodes are expanded. v_{SN} indicates that value v was last assigned during step N, i.e., while expanding the N^{th} node.

Proposition 4.4. *At each iteration $j > 0$ the cost of the solution of AOBB(h_i,w_j,UB_j) C_j is guaranteed to be within the factor w_j from the optimal cost C^*. Moreover, for iterations $j \geq 1$ all the intermediate solutions generated by AOBB(h_i,w_j,UB_j) are guaranteed to have costs within the factor of w_{j-1} from the optimal.*

Proof. The solution cost C_j with which AOBB(h_i,w_j,UB_j) terminates at iteration j is bounded: $C_{w_j} \leq w_j \cdot C^*$. The upper bound used for pruning at iteration $j > 1$ is equal to the cost of the solution on the previous iteration ($UB_j = C_{j-1}$). No intermediate solutions worse than this upper bound are ever explored. Thus the costs C'_j of all solutions generated at iteration j prior to its termination are bounded by the upper bound $C'_j \leq C_{j-1}$.

Since C_{j-1} is bounded by a factor of w_{j-1} from the optimal, it follows $C'_j \leq w_j \cdot C^*$. □

Figure 4.14 shows the search space explored by AOBB (on the left) and by the first iteration of wAOBB with weight $w = 10$ (on the right), when solving the problem from Figure 4.2. In this example AOBB explores 15 nodes in order to find an optimal solution. wAOBB explores 12 nodes, discovering a suboptimal solution with cost $C = 8$ and assignment $\mathbf{x} = \{A = 0, B = 0, C = 1, D = 0\}$.

Iterative Weighted BRAOBB (wBRAOBB): extends BRAOBB(h_i,w_0, UB) to a weighted iterative scheme in the same manner. Clearly, for both schemes the sequence of the solution costs is non-increasing.

Theorem 4.8. *Worst case time and space complexity of each iteration of wAOBB and wBRAOBB that uses caching is bounded by $O(n \cdot k^{w^*})$. Number of iterations varies based on weight policy and start value w_0.*

Proof. At each iteration wAOBB and wBRAOBB execute AOBB and BRAOBB respectively, exploring at most the entire context-minimal AND/OR search graph. The precise number of expanded nodes depends on the pruning and is hard to characterize in general, as is always the case for branch and bound search. □

■ 4.6.2 Empirical Evaluation of Weighted Heuristic BB

We carry out an empirical evaluation of the two weighted heuristic depth-first branch and bound schemes: wAOBB and wBRAOBB, described above. The algorithms were implemented in C++ and the experiments were conducted in the same setting as before.

We compared the two weighted heuristic branch and bound schemes against each other and against wAOBF, the superior of the weighted heuristic best-first schemes. We also compared against the state-of-the-art anytime BRAOBB. All algorithms use the same MBE heuristic. We use the same $sqrt(1.0)$ policy and starting weight equal to 64 as we did for the weighted heuristic best-first schemes in Section 4.5.

■ 4.6.2.1 Weighted Heuristic BB as Approximation

In **Table 4.6** we report the entire runtime required to find a first solution for a particular weight level (e.g., $w = 2.8284$) for each algorithm. In this sense these tables are different from Table 4.2, where we only reported the time it took the scheme to find the w-optimal solution starting from a particular weight, e.g., $w = 2.8284$, not starting with initial weight $w_0 = 64$. The difference is due to the fact that wAOBB and wBRAOBB use the results of previous iterations as upper bounds and their iterations are not completely independent runs of AOBB(h_i,w_j,UB_j) and BRAOBB(h_i,w_j,UB_j), respectively.

We also report the runtime and the cost by BRAOBB at termination. The time equal to 3600 seconds for BRAOBB signifies that it failed to report the optimal solution within the time bound and we then report the best solution found.

Comparing with weighted heuristic best-first search. Based on Table 4.6 we see that time-wise none of the schemes dominates for a given weight. For example, for grid 75-22-5 for weight $w = 1.033$ wAOBB reports the solution the fastest, while for type4b_130_21 wAOBF reaches a 1.2968-optimal solution almost ten seconds before either wAOBB or wBRAOBB.

Time saving for w-bounded sub-optimality. Comparing pairs of columns, in particular column 2 (exact results for BRAOBB) and columns 4-5 (1.2968- and 1.0330-optimal solutions) in Table 4.6, we observe remarkable time savings of the weighted heuristic schemes compared with BRAOBB.

Overall, based on these instances, which are quite representative, we see as before that

Instance	BRAOBB	Weights			
		2.8284	1.2968	1.0330	1.000
		wAOBF	wAOBF	wAOBF	wAOBF
		wAOBB	wAOBB	wAOBB	wAOBB
		wBRAOBB	wBRAOBB	wBRAOBB	wBRAOBB
	time / cost	time / cost	time / cost	time / cost	time / cost
Grids, I-bound=18					
75-22-5 (484, 2, 30, 107)	115.45 / -15.605	5.92 / -15.72	6.66 / -15.72	59.81 / -15.61	423.76 / -15.61
		5.87 / -19.08	6.1 / -17.55	52.46 / -15.65	— / —
		5.91 / -19.08	6.22 / -17.55	79.91 / -15.7	— / —
75-25-5 (625, 2, 34, 122)	3582.08 / -20.836	8.0 / -23.38	9.77 / -21.7	— / —	— / —
		8.08 / -31.0	8.34 / -22.55	— / —	— / —
		8.14 / -31.0	8.56 / -22.84	— / —	— / —
Pedigrees, I-bound=18					
pedigree9 (935, 7, 27, 100)	220.34 / -122.904	26.75 / -129.55	26.96 / -123.06	33.56 / -122.9	— / —
		12.44 / -129.76	12.47 / -128.56	13.63 / -123.2	— / —
		12.59 / -129.76	12.61 / -128.56	13.74 / -123.2	— / —
pedigree51 (871, 5, 39, 98)	3600 / -111.55	120.94 / -119.16	— / —	— / —	— / —
		29.01 / -121.77	31.15 / -121.77	— / —	— / —
		26.41 / -121.77	28.36 / -121.77	3035.48 / -109.83	— / —
WCSP, I-bound=6					
capmo2.wcsp (200, 100, 100, 100)	3600 / -0.28	26.81 / -0.31	— / —	— / —	— / —
		22.43 / -0.31	— / —	— / —	— / —
		22.47 / -0.31	— / —	— / —	— / —
myciel5g_3.wcsp (47, 3, 19, 24)	12.93 / -64.0	2.52 / -72.0	50.47 / -64.0	— / —	— / —
		0.96 / -72.0	7.8 / -64.0	37.63 / -64.0	— / —
		0.9 / -72.0	7.37 / -64.0	35.63 / -64.0	— / —
Type4, I-bound=18					
type4b_120_17 (4072, 5, 24, 319)	3600 / -1332.18	80.49 / -1354.93	81.24 / -1329.59	84.98 / -1327.6	— / —
		49.79 / -1353.83	49.97 / -1337.37	50.25 / -1334.85	— / —
		54.91 / -1353.83	55.12 / -1337.37	55.44 / -1334.85	— / —
type4b_130_21 (4874, 5, 29, 416)	3600 / -1383.74	86.21 / -1438.24	88.89 / -1386.34	— / —	— / —
		58.12 / -1414.3	97.66 / -1489.57	— / —	— / —
		96.61 / -1512.32	96.93 / -1489.57	— / —	— / —

Table 4.6: Runtime (sec) and cost (on logarithmic scale) obtained by wAOBF, wAOBB and wBRAOBB for selected w, and by BRAOBB (that finds C^* - optimal cost). Instance parameters: n - number of variables, k - max domain size, w^* - induced width, h_T - pseudo-tree height. "time out" - running out of time, "—" - running our of memory. 4 GB memory limit, 1 hour time limit, MBE heuristic.

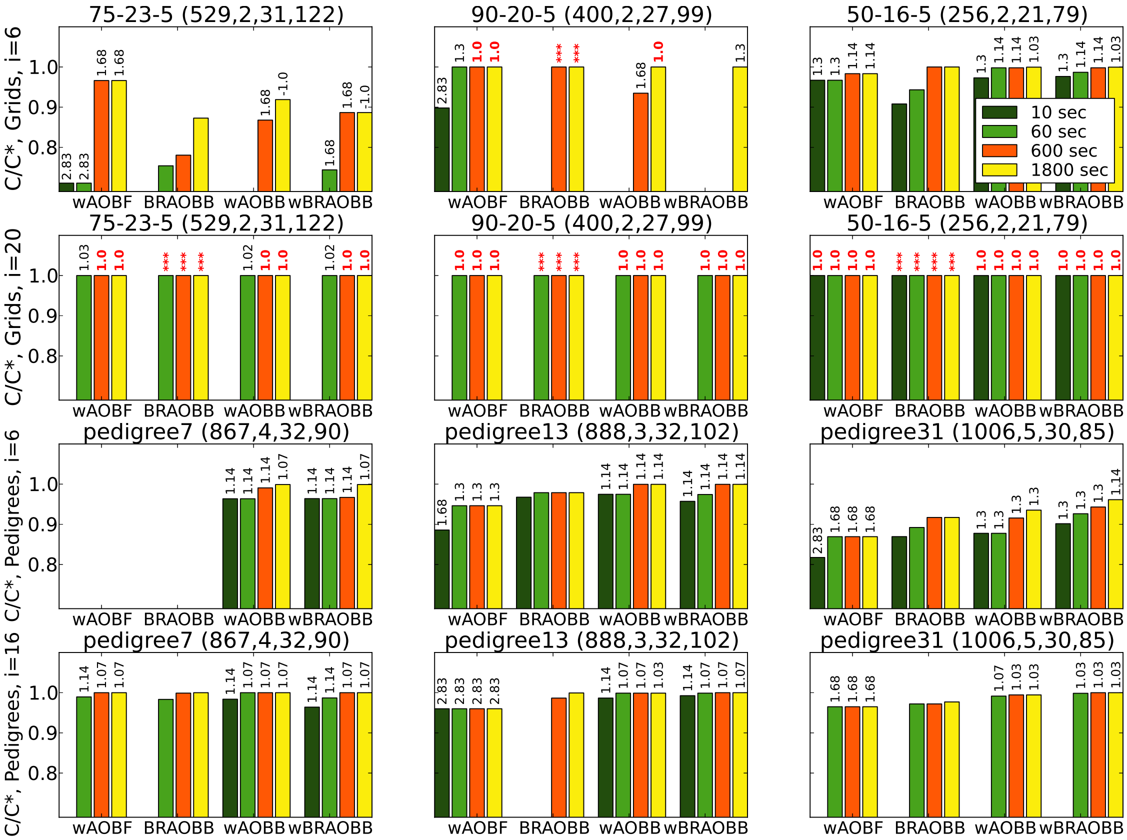


Figure 4.15: Ratio of the cost obtained by some time point (10, 60, 600 and 1800 sec) and max cost. Max. cost = optimal, if known, otherwise = best cost found for the problem. Corresponding weight - above the bars. The cases where BRAOBB proved solution optimality is indicated by '***' above bars. In red - optimal solutions. Instance parameters are in format (n,k,w^*,h_T), where n - number of variables, k - max. domain size, w^* - induced width, h_T - pseudo-tree height. Grids and Pedigrees. Memory limit 4 GB, time limit 1 hour, MBE heuristic.

the weighted heuristic schemes can often provide good approximate solutions with tight suboptimality bounds, yielding significant time savings compared to finding optimal solutions by competing BRAOBB. The weighted branch and bound schemes are more memory efficient than wAOBF. However, on the instances feasible for all three weighted heuristic schemes there is no clear winner.

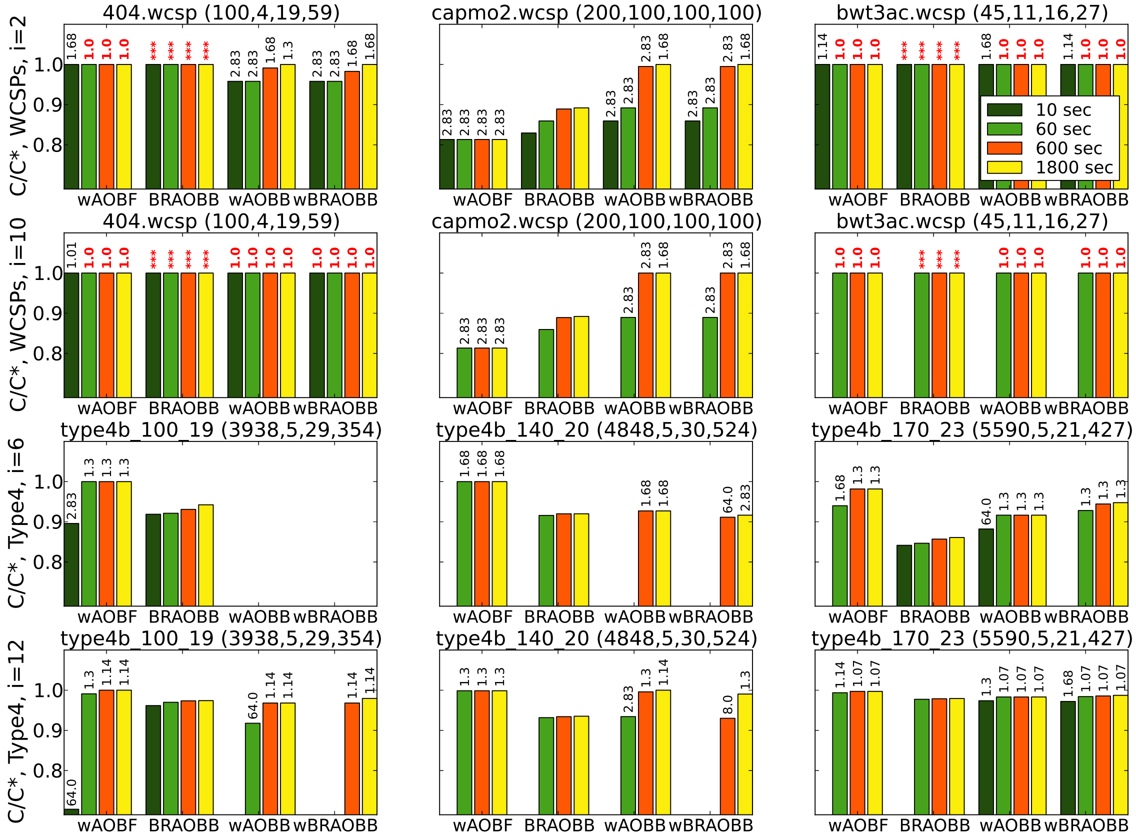


Figure 4.16: Ratio of the cost obtained by some time point (10, 60, 600 and 1800 sec) and max cost. Max. cost = optimal, if known, otherwise = best cost found for the problem. Corresponding weight - above the bars. The cases where BRAOBB proved solution optimality is indicated by '***' above bars. In red - optimal solutions. Instance parameters are in format (n,k,w^*,h_T), where n - number of variables, k - max. domain size, w^* - induced width, h_T - pseudo-tree height. Type4 and WCSPs. Memory limit 4 GB, time limit 1 hour, MBE heuristic.

■ 4.6.2.2 Anytime Performance Comparison

Figures 4.15 and **4.16** display the anytime behavior of the schemes for typical instances from each benchmark using bar charts that show the ratio between the cost available at a particular time point (at 10, 60, 600 and 1800 sec) and the optimal (if known) or best cost found (similarly to Figures 4.7 and 4.8 in the previous section). Figure 4.15 shows the results for Grids and Pedigrees, while Figure 4.16 presents WCSPs and Type4 instances.

Grids (Figure 4.15): we observe that the weighted heuristic branch and bound schemes are typically inferior to wAOBF, finding solutions slower and of lower accuracy. For example, wAOBF is the only scheme to return a solution within 10 seconds on grid 75-23-5, $i = 6$. However, there are exceptions (e.g., grid 50-16-5, $i = 6$).

Pedigrees: both weighted heuristic branch and bound schemes are often the best (e.g., pedigree13, i=6). For some problems they even provide solutions with accuracy approaching 1.0 while the other schemes fail to find any solution, e.g., pedigree7, $i = 6$.

WCSPs (Figure 4.16): The wAOBB and wBRAOBB perform better than BRAOBB and wAOBF on such instances (e.g., capmo2.wcsp for all i-bounds), except for a number of problems (not explicitly shown), for which BRAOBB is the only scheme to return solutions.

Type4: wAOBF mostly dominates over the branch and bound schemes, included the weighted heuristic ones. However, for larger i-bounds on Type4 weighted heuristic branch and bound schemes can find good solutions, sometimes even providing tighter sub-optimality guarantees than wAOBF. For example, for type4b_140_20, $i = 12$, for 1800 sec the bound by wAOBF is $w = 1.3$ while for wAOBB it is $w = 1.14$. BRAOBB is inferior for this benchmark.

We turn now to Figures 4.9-4.12 in order to summarize the performance of weighted heuristic depth-first branch and bound search algorithms, concentrating on wAOBB as a slightly better of the two. From these bar charts we see that wAOBB is more successful than BRAOBB when the heuristics are weak. For example, for Grids, 60 second, for $i = 10$, wAOBB finds solutions of higher accuracy than BRAOBB on 23.1% of instances, while for $i = 18$, superiority is 16.7%. The two schemes are tied on 82.1% of instances for Grids, $i = 18$, 3600 seconds, and in general, when the i-bound is high, the difference between the solution costs reported by various algorithms diminishes.

We conclude from these figures that the wAOBB can definitely be superior to wAOBF on 2 out of 4 benchmarks (Pedigrees and WCSPs) for many problems, and can find solution of better accuracy than BRAOBB on a number of instances from all four benchmarks.

■ 4.7 On the Weighted Heuristic Strengthened by Cost-Shifting

We conducted a set of experiments with Mini-Bucket with the Moment-Matching (MBE-MM) and Join Graph Linear Programming (JGLP) heuristics, known to be overall superior to MBE [48]. These more sophisticated heuristics will be discussed in more details in Chapter 5. For now we treat them as black-boxes, providing heuristic evaluations in the same format as MBE, though, in most cases, more accurate. The accuracy of both algorithms is controlled by i-bound parameter, similar to MBE. MBE-MM is a single-pass algorithm. JGLP is an iterative scheme that has been run for 60 seconds in our experiments.

Figures 4.17 and 4.18 show the anytime performance of wAOBF, wR-AOBF and BROABB for MBE-MM heuristic in a form of bar charts, similar to Figures 4.7-4.8. We show the ratio of the cost obtained by some time point by each algorithm and max cost available for 10, 65, 300, 600 and 3600 seconds. Each two rows correspond to a particular benchmark, upper row showing smaller i-bound, the lower row showing the higher i-bound. Figures 4.19 and 4.20 present the results for JGLP heuristic. Note that none of the weighted heuristic search algorithms report any solution before 60 second time bound when employing JGLP, since the calculation of heuristic always requires at least a minute.

The behavior of the algorithms using the MBE-MM and JGLP heuristics is consistent with our previous observations. Again wAOBF and wR-AOBF are quite successful on many grids, pedigrees and type4 instances. On WCSP instances the calculation of the advanced heuristics (both MBE-MM and JGLP) fails due to memory limitations on a significant portion of instances for medium and large i-bounds, for example, on 41 problems out of 56, for JGLP, $i = 10$, as opposed to 33 instances out of 56 for MBE, same i-bound.

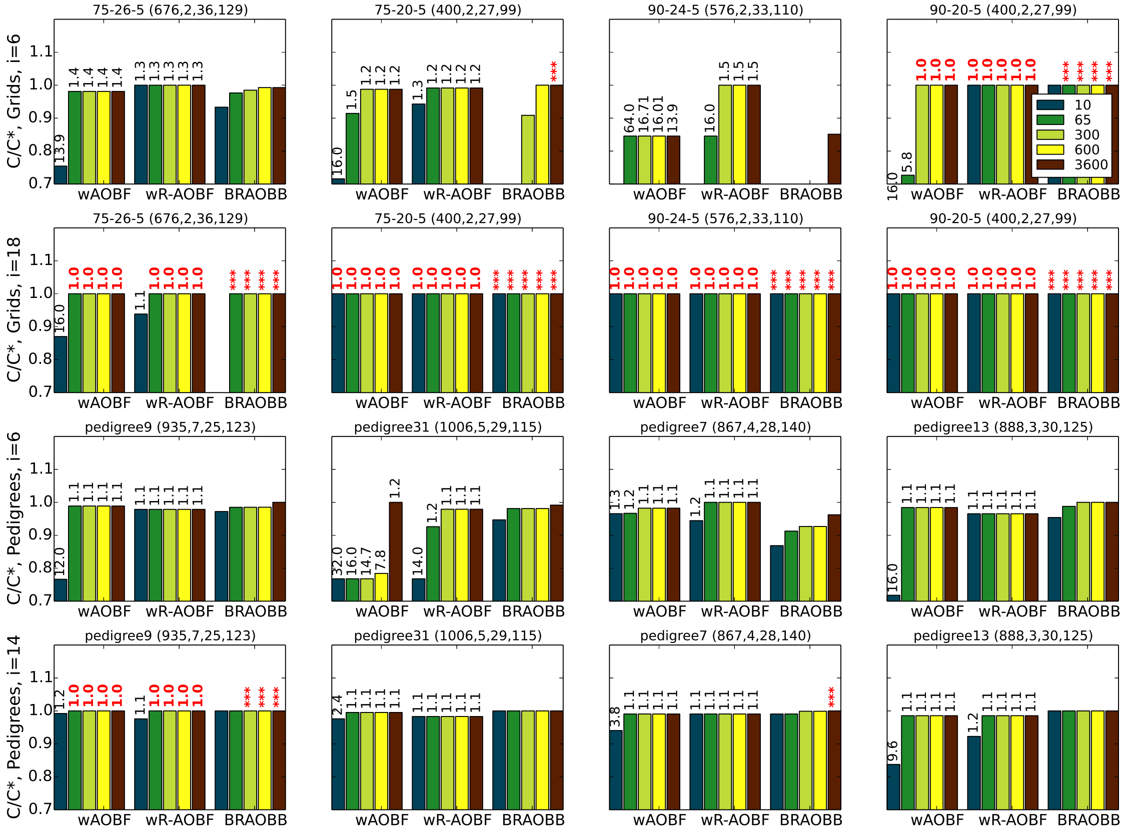


Figure 4.17: Ratio of the cost obtained by some time point (10, 65, 300, 600 and 3600 sec) and max cost (optimal, if known, otherwise - best cost found for the problem by any of the schemes). Corresponding weight - above the bars. '***' indicated proven solution optimality. Instance parameters are in format (n,k,w^*,h_T), where n - number of variables, k - max. domain size, w^* - induced width, h_T - pseudo-tree height. Memory limit 4 GB, time limit 1 hour. Grids and Pedigrees benchmarks, MBE-MM heuristic.

Figures 4.21-4.24 show the summary of the experiments with MBE-MM and JGLP. The figures show the percentage of instances for which wAOBF and wR-AOBF respectively find strictly better solutions than BRAOBB (leftmost plot in each row), the percentage for which they find the solution of equal cost with BRAOBB (center plot), as was done in Figures 4.9-4.12. Additionally, on the right the weight at termination averaged over the instances for which the weighted scheme has found any (possibly suboptimal) solution is presented. This metric allows to estimate how tight, on average, is the w-optimality bound by each algorithm. We show the results for a relatively small and large i-bounds, for two time bounds each.

Appendix C.2 presents supplementary summaries in table form, including results for wAOBB,

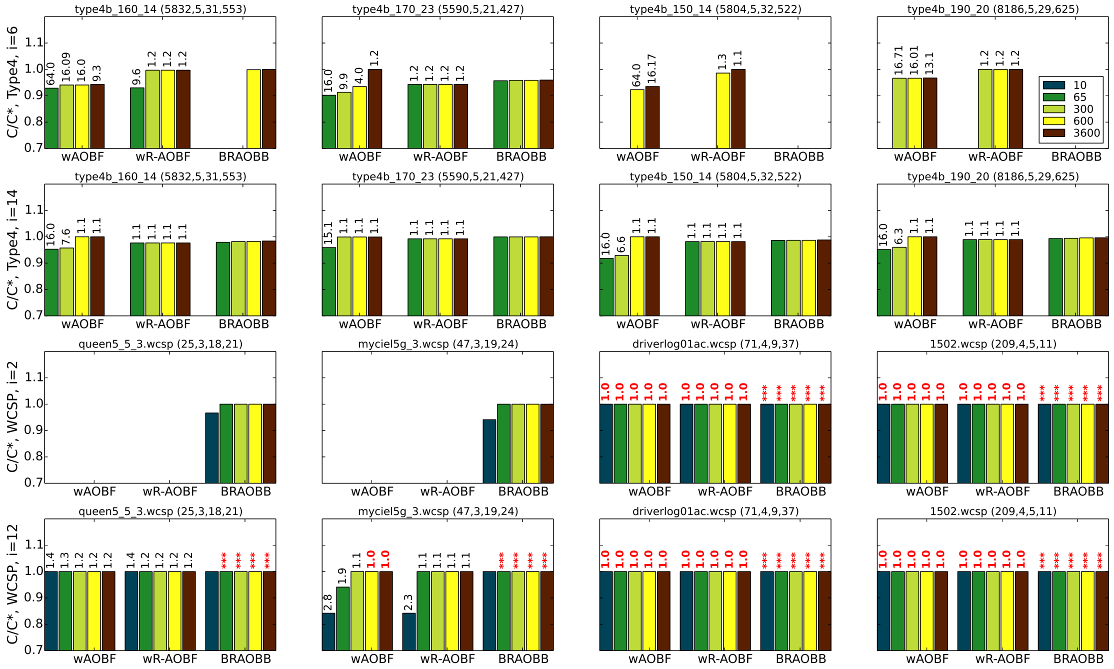


Figure 4.18: Ratio of the cost obtained by some time point (10, 65, 300, 600 and 3600 sec) and max cost (optimal, if known, otherwise - best cost found for the problem by any of the schemes). Corresponding weight - above the bars. '***' indicated proven solution optimality. Instance parameters are in format (n,k,w^*,h_T), where n - number of variables, k - max. domain size, w^* - induced width, h_T - pseudo-tree height. Memory limit 4 GB, time limit 1 hour. Type4 and WCSP benchmarks, MBE-MM heuristic.

in addition to wAOBF and wR-AOBF, for a large variety of i-bounds and time points.

From the results in Figures 4.21-4.24 we once again observe many trends also evident in our previous experiments when MBE heuristics were used.

1. Anytime weighted heuristic best-first schemes are superior to BRAOBB in quite a few cases. However, their performance varies a lot across benchmarks. wAOBF and wR-AOBF yield better solutions than BRAOBB on a large percentage of instances on Grids (e.g., 58.3% at 120 sec for wR-AOBF with MBE, $i = 6$) and Type4 (e.g., at least 80.0% for wAOBF for MBE, both time bounds). They are also quite effective on Pedigrees (better than BRAOBB on up to 33% instances for certain time bounds and heuristic strength). On WCSP weighted heuristic BF schemes are mostly inferior, often not producing a single solution more accurate than BRAOBB. One outlier is

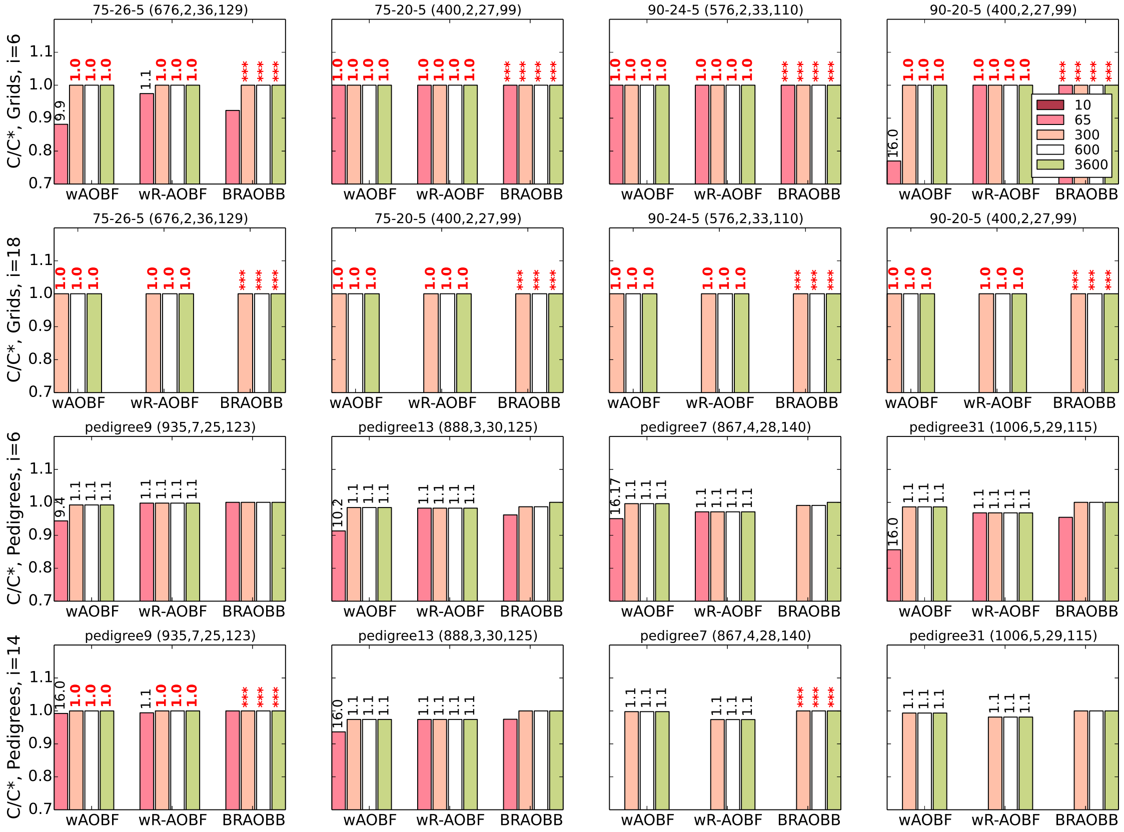


Figure 4.19: Ratio of the cost obtained by some time point (10, 65, 300, 600 and 3600 sec) and max cost (optimal, if known, otherwise - best cost found for the problem by any of the schemes). Corresponding weight - above the bars. '***' indicated proven solution optimality. Instance parameters are in format (n,k,w^*,h_T), where n - number of variables, k - max. domain size, w^* - induced width, h_T - pseudo-tree height. Memory limit 4 GB, time limit 1 hour. Grids and Pedigrees benchmarks, JGLP heuristic.

$i = 6$ for JGLP, where wAOBF and wR-AOBF are better than BRAOBB in up to 50% cases. However, these results pertain to just the 4 easiest instances of the benchmark and thus are not very conclusive.

2. Weighted heuristic best-first algorithms tend to be superior when the i-bound is small (e.g., $i = 6$). When it is large (i.e., strong heuristics) their dominance is usually less pronounced. This ability to produce good solutions when there is a large gap between the i-bound and the problem's induced width should be especially beneficial when solving hard problems, for which calculation of accurate heuristics is infeasible.

3. Same trends are seen when comparing different heuristics schemes. Weighted heuristic

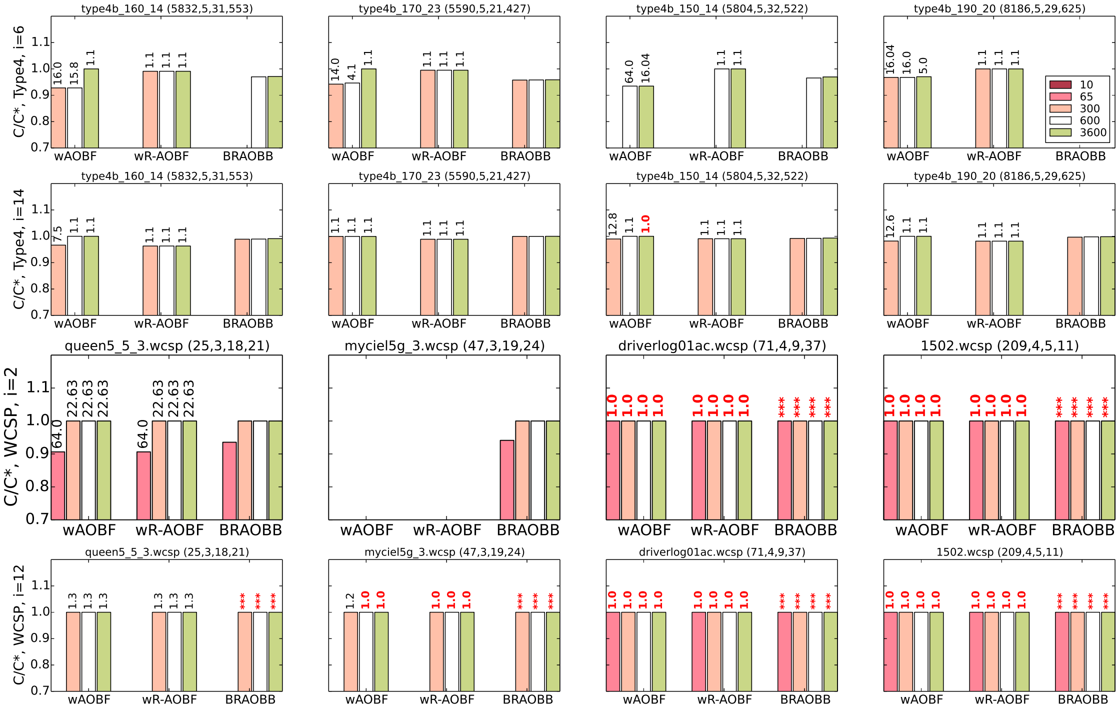


Figure 4.20: Ratio of the cost obtained by some time point (10, 65, 300, 600 and 3600 sec) and max cost (optimal, if known, otherwise - best cost found for the problem by any of the schemes). Corresponding weight - above the bars. '***' indicated proven solution optimality. Instance parameters are in format (n,k,w^*,h_T), where n - number of variables, k - max. domain size, w^* - induced width, h_T - pseudo-tree height. Memory limit 4 GB, time limit 1 hour. Type and WCSP benchmarks, JGLP heuristic.

BF algorithms are more successful than BRAOBB when heuristic is relatively weaker, as in case of using MBE algorithm, as opposed to stronger heuristic yielded by JGLP.

4. Though JGLP generally provides more accurate heuristics, it is known to have higher time and space requirements [48], which is why on some benchmarks, e.g., Pedigrees and Type4, it can be infeasible for higher i-bounds on many problems and yields less solved instances.

5. wAOBF and wR-AOBF almost always demonstrate better performance for short time limits. For example, for Pedigrees, $i = 6$, MBE, wAOBF is superior to BRAOBB on 33.3% of problems for 120 seconds, but only on 18.8% for 3600 seconds.

6. Even on benchmarks where weighted heuristic BF schemes are inferior to BRAOBB,

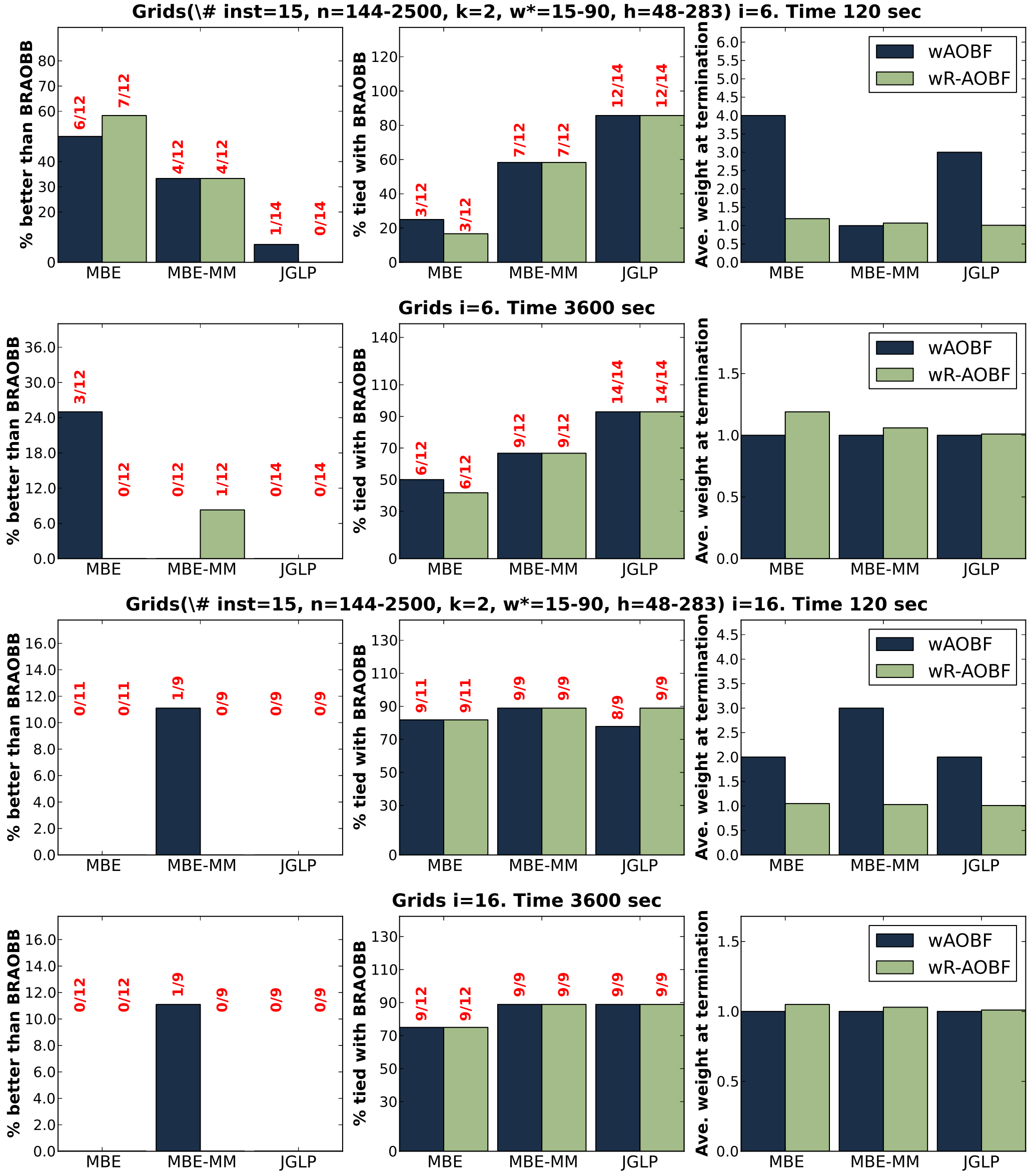
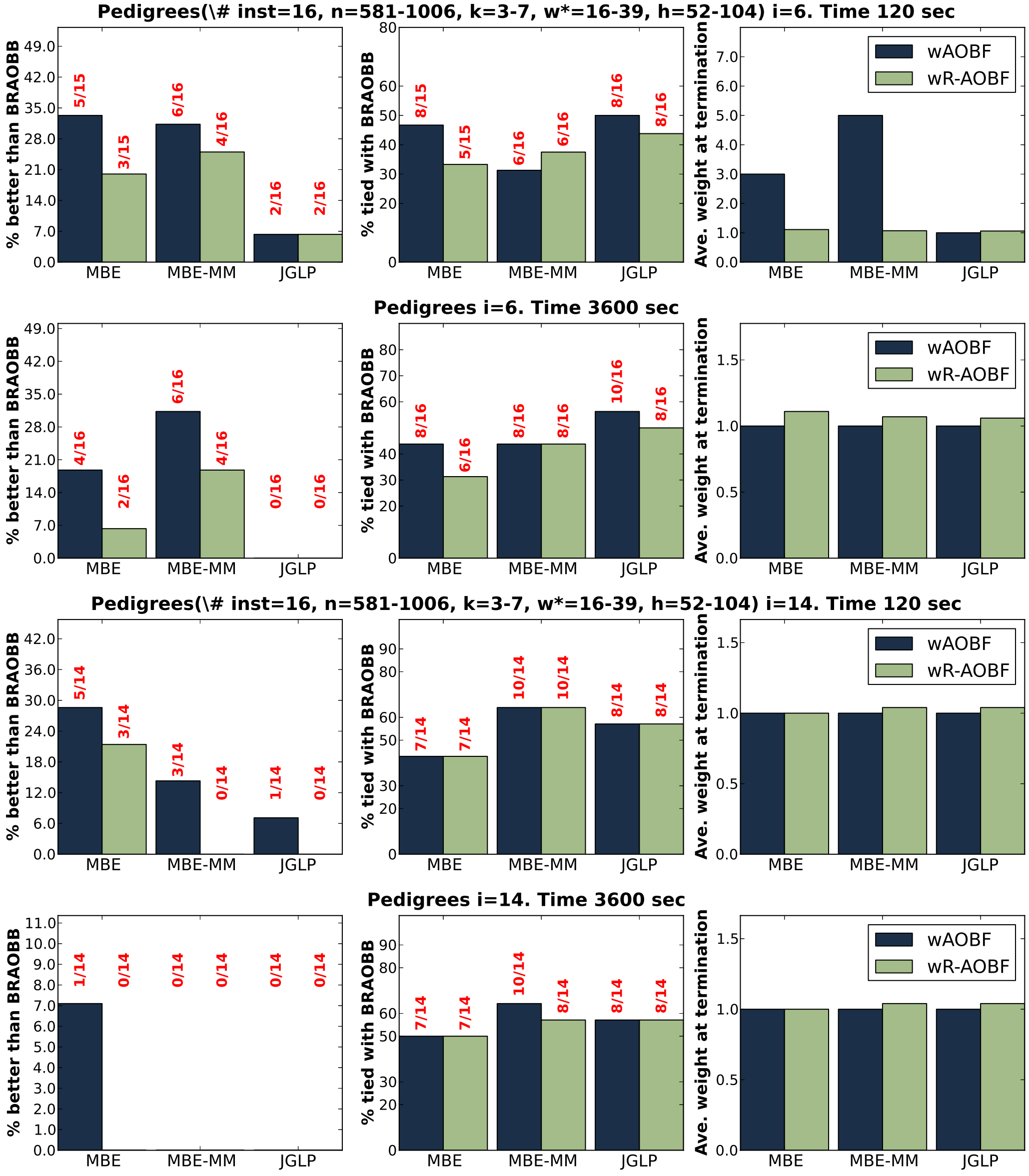


Figure 4.21: Grids: percentage of instances for which each algorithm found solutions of more accurate cost than BRAOBB at a specific time bound (left), percentage of instances for which algorithm is tied with BRAOBB, i.e., found solution of equal cost (center), average weight at termination (right). Above bars - number of instances where algorithm is better than BRAOBB (tied with BRAOBB) and number of instances solved by the algorithm. Parameters: # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo-tree height. 4 GB memory, 1 hour time limit.



Pedigrees(\# inst=16, n=581-1006, k=3-7, w*=16-39, h=52-104) i=6. Time 120 sec

Pedigrees i=6. Time 3600 sec

Pedigrees(\# inst=16, n=581-1006, k=3-7, w*=16-39, h=52-104) i=14. Time 120 sec

Pedigrees i=14. Time 3600 sec

Figure 4.22: Pedigrees: percentage of instances for which each algorithm found solutions of more accurate cost than BRAOBB at a specific time bound (left), percentage of instances for which algorithm is tied with BRAOBB, i.e., found solution of equal cost (center), average weight at termination (right). Above bars - number of instances where algorithm is better than BRAOBB (tied with BRAOBB) and number of instances solved by the algorithm. Parameters: # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo-tree height. 4 GB memory, 1 hour time limit.

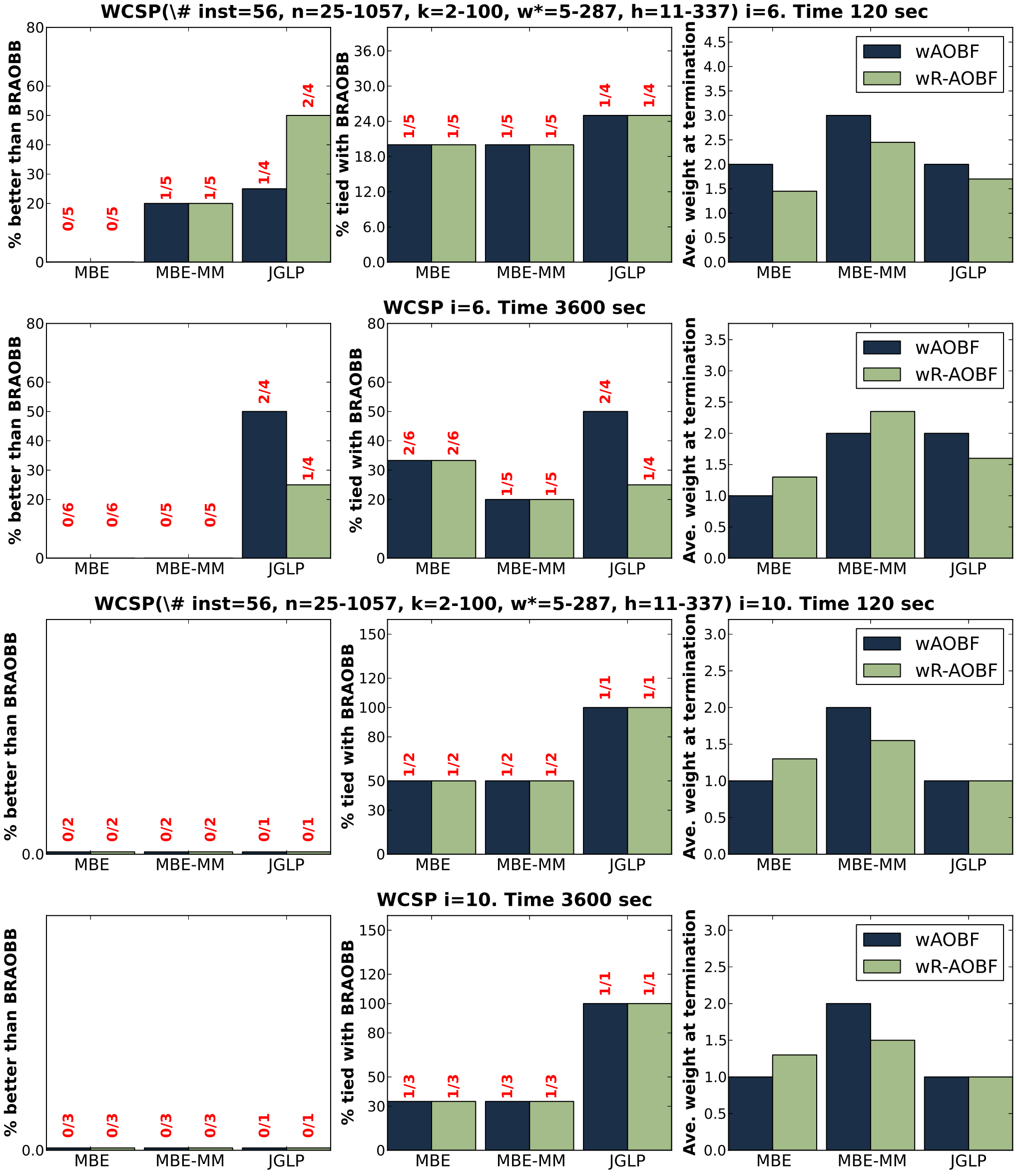


Figure 4.23: WCSP: percentage of instances for which each algorithm found solutions of more accurate cost than BRAOBB at a specific time bound (left), percentage of instances for which algorithm is tied with BRAOBB, i.e., found solution of equal cost (center), average weight at termination (right). Above bars - number of instances where algorithm is better than BRAOBB (tied with BRAOBB) and number of instances solved by the algorithm. Parameters: # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo-tree height. 4 GB memory, 1 hour time limit.

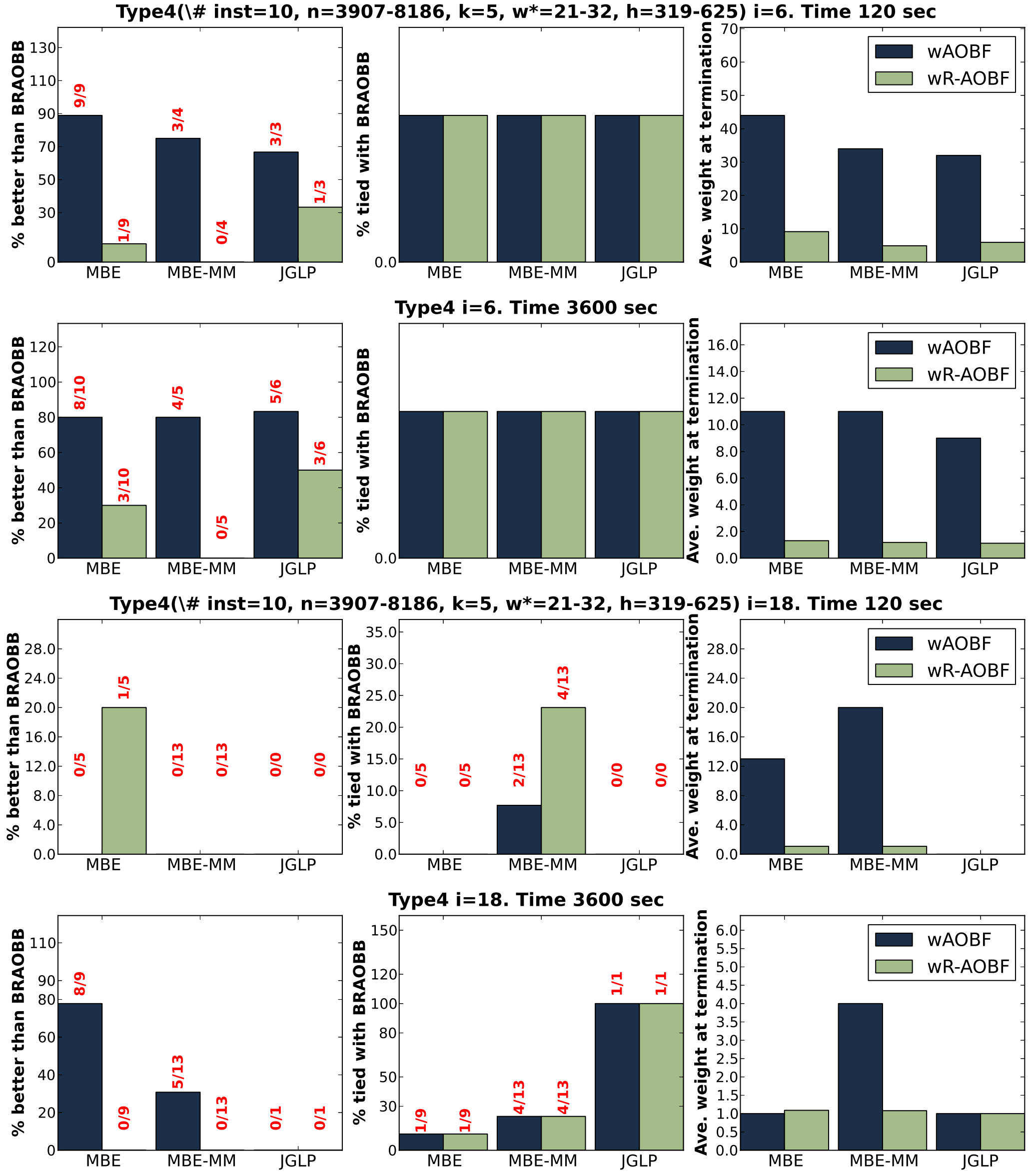


Figure 4.24: Type4: percentage of instances for which each algorithm found solutions of more accurate cost than BRAOBB at a specific time bound (left), percentage of instances for which algorithm is tied with BRAOBB, i.e., found solution of equal cost (center), average weight at termination (right). Above bars - number of instances where algorithm is better than BRAOBB (tied with BRAOBB) and number of instances solved by the algorithm. Parameters: # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo-tree height. 4 GB memory, 1 hour time limit.

e.g., WCSPs, they sometime provide tight w-optimality bounds, as indicated by the average weights being close to 1.0. For example, the average weight for wR-AOBF on Type4, $i = 18$, 120 sec for all heuristics is at most 1.08. Such bounds are especially valuable for instances where optimal solution can not be obtained within the time and memory limit, as is the case for majority of Type4 problems.

7. wAOBF seems somewhat superior to wR-AOBF in terms of accuracy, dominating BRAOBB more often, especially on Type4 benchmark. wR-AOBF typically provides tighter w-optimality bounds, i.e., has lower average weight.

Overall, we see that, as we noticed when analyzing different i-bounds, a stronger heuristic improves the power of BRAOBB compared with the weighted heuristic schemes. The advantage of the latter remains throughout to be the generation of an optimality guarantee.

■ 4.8 Summary and Concluding Remarks

The chapter provides a study of weighted heuristic best-first and weighted heuristic depth-first branch and bound search for graphical models. These algorithms are distinguished by their ability to provide a w-optimality guarantee (namely they can guarantee solutions that are at most a w factor away from the optimal cost, for a given w). Alternatively, when run in anytime fashion, whenever stopped they generate the best solution encountered thus far and a weight w bounding its sub-optimality.

The idea of weighted heuristic best-first search is widespread in the path-finding and planning communities. In this chapter we extended this idea to graphical models optimization tasks, specifically, to AND/OR best-first search scheme AOBF and AND/OR depth-first branch and bound scheme AOBB (see [69]). This resulted in two anytime weighted heuristic best-first schemes, wAOBF and wR-AOBF, and two weighted depth-first, wAOBB and wBRAOBB. We evaluated these algorithms against each other and against several competing schemes, and especially against the state of the art BRAOBB [76], on a large variety of instances from 4 benchmarks. We evaluated the algorithms for varying heuristic strength and for different time bounds. The heuristic functions were generated by the mini-bucket elimination scheme [25], whose strength is controlled by an i-bound parameter.

Our experiments revealed the following primary trends:

- **On the anytime performance of weighted heuristic schemes:** First and foremost we showed that the weighted schemes can perform better than BRAOBB in many cases. In addition the schemes provide the useful w-optimality guarantee. We saw that overall wAOBF had a better anytime behavior than wR-AOBF and produced more accurate solutions in a comparable time on many instances (see Figures 11-15).
- **Performance varied per benchmark:** On two out of four benchmarks wAOBF and wR-AOBF dominated over BRAOBB, more often finding costs of better accuracy within the time limit. This good behavior on Grids and Type4 benchmarks was mostly consistent across heuristic strengths and time bounds. The weighted heuristic DFS branch and bound schemes wAOBB and wBRAOBB were better than wAOBF and BRAOBB on Pedigrees and, especially, WCSPs. This dominance often corresponded to cases where wAOBF ran out of memory, since branch and bound schemes are more memory efficient. Therefore, together the weighted schemes had an effective performance on instances across all benchmarks.
- **On the impact of the heuristic strength:** the weighted heuristic schemes were more powerful compared with BRAOBB for weak heuristics, namely, when the i-bound characterizing the heuristic strength was far smaller than the problem's induced width. One explanation is that the weight may make the weak admissible heuristic more accurate (a weak lower bound becomes stronger lower bound, closer to the actual

optimal cost, when multiplied by a constant).

- **On w-optimality in practice:** In quite a few cases the weighted schemes (e.g., wAOBF, wR-AOBF, wAOBB and wBRAOBB) reported solutions orders of magnitude faster, even for small w, than the time required to generate an exact solution by the unweighted schemes BRAOBB or AOBF. Moreover, in some cases the weighted heuristic schemes generated w-optimal solutions for a small w even for some hard instances that were infeasible (within the time limit) for the baseline algorithms.

Selection and combination of algorithms. Clearly, however, due to the fact that no algorithm is always superior, the question of algorithm selection requires further investigation. We aim to identify problem features that could be used to predict which scheme is best suited for solving a particular instance, and to combine the algorithms within a portfolio framework, known to be successful for such solvers as SATzilla [104] and PbP [38].

In that context, however it is important to note that the weighted heuristic schemes can be valuable anyway by supplying any approximation with w-optimality guarantees. For example, it can be used alongside incomplete approximate schemes, such as Stochastic Local Search. If the solution by SLS has a cost better or equal to that of the generated w-optimal solution, it yields a w-optimal bound on the SLS solution.

The potential impact of weights on various heuristics. While we evaluated the weighted schemes relative to the mini-bucket heuristics only, these ideas are orthogonal to the heuristic type and they are likely to similarly boost the anytime behavior with any other heuristics. We have recently provided some initial empirical evaluation of the impact of the weighted schemes for cost-shifting relaxation schemes that augment the mini-bucket scheme, introduced in [48]. Our initial findings are presented in [86].

Chapter 5

Cost-Shifting for Better Approximation

■ 5.1 Introduction[1]

Finding the exact solutions to combinatorial optimization tasks over graphical models is often infeasible, since these problems are typically NP-hard. Various approximation approaches have been suggested over the years.

Mini-bucket elimination (MBE), described in Section 1.2.2.2 of Chapter 1, is a popular bounding scheme that generates upper and lower bounds by applying the exact bucket elimination algorithm (Section 1.2.2.1) to a simplified (or relaxed) problem obtained by duplicating variables. The relaxation view of MBE is closely related to a family of iterative approximation techniques based on linear programming (LP). These include “re-weighted” max-product [98], Max-Product Linear Programming (MPLP) [40], dual decomposition [58], and soft arc consistency [83, 8]. These algorithms simplify the graphical model into independent components and tighten the resulting bound via iterative cost-shifting updates. They can be thought of as “re-parametrizing” the original functions without changing the global model. Most of the schemes operate on the original factors of the model, although some works tighten the approximations by introducing larger clusters [92].

[1] Part of this work has already been published in Alexander Ihler, Natalia Flerova, Rina Dechter, and Lars Otten. ”Join-graph based cost-shifting schemes” in Proceedings of UAI, 2012.

Contribution. We combine the ideas of MBE and re-parametrization to define two new hybrid schemes. One algorithm, called mini-bucket elimination with max-marginal matching (MBE-MM), is a non-iterative algorithm that applies a single pass of cost-shifting during the mini-bucket construction bucket by bucket. It is closely related to horizontal mini-bucket elimination (h-MBE) by Rollon and Larossa [82]; the two methods differ primarily in the form of the cost-shifting update within each bucket. However, our update is motivated by its connection to a globally applicable tightening algorithm, while the horizontal MBE seems somewhat *ad hoc*.

The second approach, Join Graph Linear Programming (JGLP), iteratively applies cost-shifting updates to the full mini-bucket join-graph. Our empirical evaluation demonstrates the increased power of these hybrid approximation schemes over their individual components. From an LP perspective, JGLP is a variant of generalized Max-Product Linear Programming (MPLP) [91]. Its main novelty is in showing how the mini-bucket elimination approach can facilitate effective construction of a join graph. JGLP works “top down”, creating large clusters immediately, compared with other methods, e.g., [91, 6] that work “bottom up” (gradually including triplets of variables, then clusters over four variables and so forth).

One of the primary uses of bounding algorithms is in generating heuristics for best-first and branch and bound search (Section 1.2.4.4). Our hybrid schemes, when used as heuristics to guide search, help to efficiently prune the search space explored by AND/OR Branch and Bound, thus significantly increasing its power. This is evident in both our empirical evaluation and the results of the 2011 Probabilistic Inference Challenge[2], where our algorithm won first place in all optimization categories. Our experiments also demonstrate the benefit of the improved new heuristics for other anytime search schemes, using as examples weighted anytime AND/OR Best First (previously introduced in Section 4.4) and weighted AND/OR Branch and Bound (Section 4.6).

[2] http://www.cs.huji.ac.il/project/PASCAL/

The remainder of the chapter is organized as follows. Section 5.2 provides the background information on the Linear Programming methods. Section 5.3 describes Join Graph Linear Programming (JGLP) and Section 5.4 presents the mini-bucket elimination with max-marginal matching (MBE-MM). In Section 5.5 we describe how to use the new bounding schemes to provide heuristics for the search algorithms. Section 5.6 presents empirical evaluation and Section 5.7 concludes.

5.2 Background: Linear Programming Methods

We discussed the mini-bucket elimination algorithm in Section 1.2.2.2. Here we provide background on the alternative approximation approach for solving optimization tasks, using existing LP-relaxation methods. Note that throughout we describe algorithms in terms of a maximization task, unless specified otherwise.

Wainwright, et al., [98] established the connections between LP relaxations of integer programming problems and approximate dynamic programming methods using message passing in the max-product algebra. Subsequent improvements in algorithms such as Max-Product Linear Programming (MPLP) include coordinate-descent updates that ensure convergence [40, 91]. We next introduce some of these ideas in more details.

Coordinate descent is a popular optimization approach [45, 101, 17], widely used for solving LP-relaxation tasks. Consider a general minimization task: $\min_{x_j} f(x_1, \ldots, x_n)$. A simple way to bound the solution of this problem is by iteratively minimizing the objective function with respect to a single variable X_j, while fixing the values of all other variables:

$$x_i^k j \leftarrow \arg\min_{x_j} f(x_1^k, x_2^k, \ldots, x_j, \ldots, x_n^{k-1})$$

where x_j^k is the value of variable X_j during k^{th} iteration.

Bounding the solution to a max-sum problem. To match the style and notation of LP-relaxation literature, we assume that the network consists only of pairwise functions $f_{ij}(X_i, X_j)$ and that the problem is max-sum, i.e., to compute $C^* = \max_{\mathbf{X}} \sum_{f_{ij}\in F} f_{ij}(X_i, X_j)$. A simple bound on the max-sum objective is then given by maxima of the individual functions, exchanging the sum and max operators:

$$C^* = \max_{\mathbf{X}} \sum_{f_{ij}\in F} f_{ij}(X_i, X_j) \leq \sum_{f_{ij}\in F} \max_{\mathbf{X}} f_{ij}(X_i, X_j) = \sum_{f_{ij}\in F} \max_{X_i, X_j} f_{ij}(X_i, X_j) \tag{5.1}$$

One can interpret this operation as making an individual copy of each variable for each function, and optimizing over them separately. See, for example, the problem in Figure 5.1(a) with 3 variables and 3 functions. Figure 5.1(e) shows the modified problem with each variable duplicated, so that each function can be maximized over independently. Note that the notation of the bucket in this figure is slightly different from the one used previously: by $\mathbf{B}_p$ we denote the bucket of variable X_p.

We can also derive a bound on the optimal cost by introducing a collection of functions $\{\lambda_{ij}(X_i), \lambda_{ji}(X_j)\}$ for each edge (ij) and requiring

$$\lambda \in \Lambda \quad \Leftrightarrow \quad \forall i, \quad \sum_j \lambda_{ij}(X_i) = 0 \tag{5.2}$$

Then, we have

$$\begin{aligned} C^* &= \max_{\mathbf{X}} \sum_{f_{ij}\in F} f_{ij}(X_i, X_j) \\ &= \max_{\mathbf{X}} \sum_{f_{ij}\in F} f_{ij}(X_i, X_j) + \sum_i \sum_j \lambda_{ij}(X_i) \\ &\leq \min_{\lambda\in\Lambda} \sum_{f_{ij}\in F} \max_{X_i, X_j} \big(f_{ij}(X_i, X_j) + \lambda_{ij}(X_i) + \lambda_{ji}(X_j)\big) \end{aligned} \tag{5.3}$$

The last expression is obtained by distributing each λ_{ij} to its associated factor and applying the inequality (5.1).

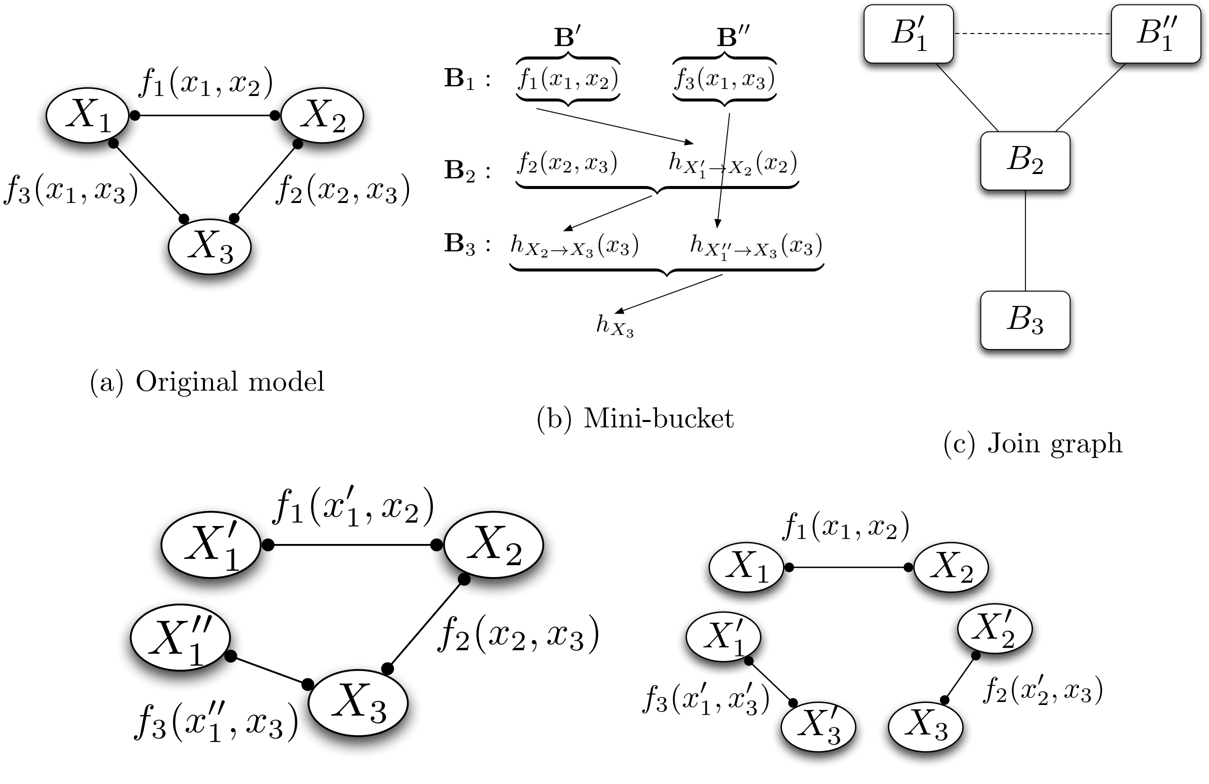
The new functions $\tilde{f}_{ij} = f_{ij}(X_i, X_j) + \lambda_{ij}(X_i) + \lambda_{ji}(X_j)$ define a *re-parametrization* of the original distribution, i.e., they change the individual functions without modifying the global function $F(\mathbf{X}) = \sum_{\mathbf{X}} f_{ij} = \sum_{\mathbf{X}} \tilde{f}_{ij}$. Depending on the literature, the λ_{ij} are interpreted as "cost-shifting" operations that transfer cost from one function to another while preserving the overall cost [83, 82], or as Lagrange multipliers enforcing consistency among the copies of X_i [107, 98]. In the former interpretation, the updates are called "soft arc-consistency" due to their similarity to arc-consistency for constraint satisfaction [14]. Under the latter view, the bound corresponds to a *dual decomposition* solver for a linear programming (LP) relaxation of the original problem [57, 90].

The main distinguishing feature among such dual decomposition approaches is the way in which the bound is tightened by updating the functions λ. This is done either by sub-gradient or gradient approaches [57, 49] or by coordinate descent updates that can be interpreted as "message passing" [40, 91]. Without going into the details of these approaches, we refer to these iterative bound improvement updates as "LP-tightening" updates, although technically we are tightening the decomposition bound (5.3) which is the dual of the LP. This is in contrast to literature that uses "tightening" to mean the inclusion of additional constraints (higher-order consistency), e.g., [92].

LP-tightening algorithm. Next we will show a derivation of a particular simple, yet effective scheme that minimizes over λs the expression

$$\sum_{f_{ij} \in F} \max_{X_i, X_j} \left(f_{ij}(X_i, X_j) + \lambda_{ij}(X_i) + \lambda_{ji}(X_j) \right) \tag{5.4}$$

tightening the upper bound on C^* (Eq. 5.3). This scheme, though formulated by us, is closely related to the "tree-block" coordinate descent updates derived by Sontag and Jaakkola [91].



(a) Original model

(b) Mini-bucket

(c) Join graph

(d) Model with duplicated variables in a mini-bucket

(e) Model with all variables duplicated

Figure 5.1: The mini-bucket procedure for a simple graph. (a) Primal graph; (b) the buckets and messages computed in MBE; (c) join-graph, dotted line corresponds to an edge absent from a mini-bucket tree; (d) interpreting MBE as variable duplication, X_1 is duplicated in each of two mini-buckets $q_1^1 = \{f_1(X_1, X_2)\}$ and $q_1^2 = \{f_3(X_1, X_3)\}$; (e) The graph on which FGLP runs with all variables duplicated and each factor processed separately.

The intuition behind the iterative updates used in this scheme is very similar to the ones employed by our algorithms JGLP (Section 5.3) and MBE-MM (Section 5.4).

The LP-tightening scheme is initialized with all $\lambda_{ij}(X_i) = 0$. In the spirit of well-known coordinate descent approaches we iteratively minimize expression 5.4 with respect to each $\lambda_{ij}(X_i)$ separately, while fixing the values of all other λs. For simplicity of derivation, without loss of generality, we assume that each variable X_i appears in scope of at most two functions, $f_{ij}(X_i, X_j)$ and $f_{ik}(X_i, X_k)$. Identifying only the terms relevant to variable X_i, we obtain:

$$\begin{aligned}
&\min_{\lambda_{ij}(X_i),\lambda_{ik}(X_i)} \Big[\big[\max_{X_i,X_j} f_{ij}(X_i,X_j)+\lambda_{ij}(X_i)\big]+\big[\max_{X_i,X_k} f_{ik}(X_i,X_k)+\lambda_{ik}(X_i)\big]\Big]\\
&=\min_{\lambda_{ij}(X_i),\lambda_{ik}(X_i)} \Big[\max_{X_i}\big[\max_{X_j} f_{ij}(X_i,X_j)+\lambda_{ij}(X_i)\big]+\max_{X_i}\big[\max_{X_k} f_{ik}(X_i,X_k)+\lambda_{ik}(X_i)\big]\Big]
\end{aligned}$$

Let us define the so-called "max-marginals" $\gamma_{ij}(X_i) = \max_{X_j} f_{ij}(X_i, X_j)$ and re-arrange the terms in the above expression, yielding the following bound:

$$\begin{aligned}
&\min_{\lambda_{ij}(X_i),\lambda_{ik}(X_i)} \Big[\max_{X_i}\big[\max_{X_j} f_{ij}(X_i,X_j)+\lambda_{ij}(X_i)\big]+\max_{X_i}\big[\max_{X_k} f_{ik}(X_i,X_k)+\lambda_{ik}(X_i)\big]\Big]\\
&=\min_{\lambda_{ij}(X_i),\lambda_{ik}(X_i)} \Big[\max_{X_i}\big[\gamma_{ij}(X_i)+\lambda_{ij}(X_i)\big]+\max_{X_i}\big[\gamma_{ik}(X_i)+\lambda_{ik}(X_i)\big]\Big] \qquad (5.5)\\
&\geq \min_{\lambda_{ij}(X_i),\lambda_{ik}(X_i)} \Big[\max_{X_i}\big[\gamma_{ij}(X_i)+\gamma_{ik}(X_i)+\lambda_{ij}(X_i)+\lambda_{ik}(X_i)\big]\Big]
\end{aligned}$$

We require that $\lambda_{ij}(X_i) + \lambda_{ik}(X_i) = 0$ to make sure that the total cost of the problem is not changed (Equation 5.2).

Many choices of λ_{ij} are known to achieve the minimum of the right-hand side of expression 5.5. We chose to use the following one:

$$\lambda_{ij}(X_i) = \frac{1}{2}\big(\gamma_{ik}(X_i) - \gamma_{ij}(X_i)\big) = \frac{1}{2}\big(\max_{X_k} f_{ik}(X_i,X_k) - \max_{X_j} f_{ij}(X_i,X_j)\big) \qquad (5.6)$$

Iterating over all functions, while updating each function $\forall i,j, f_{ij}(X_i,X_j) \leftarrow f_{ij}(X_i,X_j) + \lambda_{ij}(X_i)$ and recalculating λ_{ij} at each step until convergence, yields a minimization procedure that can be interpreted as a *max-marginal* or *moment-matching* procedure on the functions

Algorithm 22: LP-tightening (based on [91])

```
Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$, where $f_{\mathbf{S}_i}$ is a potential defined on variables $\mathbf{S}_i$
Output: Upper bound on the optimum value of $\max_{\mathbf{X}} \sum \mathbf{F}$
while NOT converged do
    for any pair of function scopes $\boldsymbol{S}_i$, $\boldsymbol{S}_j$ such that $\boldsymbol{S}_{ij} = \boldsymbol{S}_i \cap \boldsymbol{S}_j \neq \emptyset$ do
        Compute max-marginals:
        $\gamma_{\mathbf{S}_i}(\mathbf{S}_{ij}) = \max_{\mathbf{S}_i \setminus \mathbf{S}_{ij}} f_{\mathbf{S}_i}$;
        $\gamma_{\mathbf{S}_j}(\mathbf{S}_{ij}) = \max_{\mathbf{S}_j \setminus \mathbf{S}_{ij}} f_{\mathbf{S}_j}$;
        Update parametrization:
        $f_{\mathbf{S}_i} \leftarrow f_{\mathbf{S}_i} + \frac{1}{2}\big(\gamma_{\mathbf{S}_j}(\mathbf{S}_{ij}) - \gamma_{\mathbf{S}_i}(\mathbf{S}_{ij})\big)$;
        $f_{\mathbf{S}_j} \leftarrow f_{\mathbf{S}_j} + \frac{1}{2}\big(\gamma_{\mathbf{S}_i}(\mathbf{S}_{ij}) - \gamma_{\mathbf{S}_j}(\mathbf{S}_{ij})\big)$;
```

$f_{ij}(X_i, X_j)$. Intuitively, we would like the updates to diminish in magnitude and converge to zero as fast as possible. To achieve that, taking into account Equation 5.6, we would like to make the max-marginals $\gamma_{ij}(X_i)$ and $\gamma_{ik}(X_i)$ of each variable X_i to be equal. Algorithm 22 generalizes this update to higher-order functions $f_{\mathbf{S}_i}$ over scopes of variables $\mathbf{S}_i \subseteq \mathbf{X}$.

A well-known algorithm quite similar to our LP-tightening in Algorithm 22 is a message-passing scheme called Max-Product Linear Programming (MPLP) [40]. Algorithm 23 presents a version of MPLP that we call Factor Graph Linear Programming (FGLP). At each step FGLP simultaneously updates all functions $f_{ij}(X_i, X_j)$ involving a single variable X_i. The messages sent by the algorithms on a factor graph are schematically illustrated in Figure 5.2. In this example variable X_i is in the scope of three functions: $f_{\mathbf{S}_t}(X_i, X_k, X_m)$, $f_{\mathbf{S}_p}(X_i, X_j)$ and $f_{\mathbf{S}_q}(X_i, X_n)$, where $\mathbf{S}_t = \{X_i, X_k, X_m\}$, $\mathbf{S}_p = \{X_i, X_j\}$ and $\mathbf{S}_q = \{X_i, X_n\}$. Note that in the figure we only show the messages involving X_i. The max-marginals are: $\gamma_{\mathbf{S}_q}(X_i) = \max_{X_n} f_{\mathbf{S}_q}(X_i, X_n)$, $\gamma_{\mathbf{S}_p}(X_i) = \max_{X_j} f_{\mathbf{S}_p}(X_i, X_j)$ and $\gamma_{\mathbf{S}_t}(X_i) = \max_{X_k, X_m} f_{\mathbf{S}_t}(X_i, X_k, X_m)$.

```
Algorithm 23: Factor Graph Linear Programming (FGLP, based on [40])
Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$, variable ordering $o$
Output: Upper bound on the optimum value of MPE cost
while NOT converged do
    for each variable $X_i$ do
        Get factors $F_i = f_{\mathbf{S}_k} : X_i \in \mathbf{S}_k$ with $X_i$ in their scope;
        //for each function compute max-marginals $\gamma$ marginalizing out all variables except for $X_i$:
        $\forall f_{\mathbf{S}_k}\ \gamma_{\mathbf{S}_k}(X_i) = \max_{\mathbf{S}_k \setminus X_i} f_{\mathbf{S}_k}$;
        // compute messages $\beta_{\mathbf{S}_k}(X_i)$ from $X_i$ back to a function $f_{\mathbf{S}_k}$ correcting for the function's
        own max-marginal $\gamma_{\mathbf{S}_k}$:
        $\forall f_{\mathbf{S}_k}\ \beta_{\mathbf{S}_k} = \frac{1}{|F_i|} \sum_{\{\mathbf{S}_j | f_{\mathbf{S}_j} \in F_i\}} \gamma_{\mathbf{S}_j}(X_i) - \gamma_{\mathbf{S}_k}(X_i)$
        // update (re-parametrize) each function:
        $\forall f_{\mathbf{S}_k},\ f_{\mathbf{S}_k} \leftarrow f_{\mathbf{S}_k} + \beta_{\mathbf{S}_k}$;
```

The update messages from variable X_i back to the functions are:

$$\beta_{\mathbf{S}_q}(X_i) = \frac{1}{3}(\gamma_{\mathbf{S}_q}(X_i) + \gamma_{\mathbf{S}_p}(X_i) + \gamma_{\mathbf{S}_t}(X_i)) - \gamma_{\mathbf{S}_q}(X_i)$$
$$\beta_{\mathbf{S}_p}(X_i) = \frac{1}{3}(\gamma_{\mathbf{S}_q}(X_i) + \gamma_{\mathbf{S}_p}(X_i) + \gamma_{\mathbf{S}_t}(X_i)) - \gamma_{\mathbf{S}_p}(X_i)$$
$$\beta_{\mathbf{S}_t}(X_i) = \frac{1}{3}(\gamma_{\mathbf{S}_q}(X_i) + \gamma_{\mathbf{S}_p}(X_i) + \gamma_{\mathbf{S}_t}(X_i)) - \gamma_{\mathbf{S}_t}(X_i)$$

The benefit of messages passing is that it can be performed asynchronously. The issue of efficient message scheduling has been extensively studied [e.g., 28, 94, 95].

Theorem 5.1 (Complexity of FGLP). *The total time complexity of a single iteration of FGLP is $O(n \cdot Q \cdot k^{Sc})$, where n is the number of variables in the problem, k is the largest domain size, $|\boldsymbol{F}|$ is the number of functions, Sc bounds the largest scope of the original functions, Q is the largest number of functions having the same variable X_j in their scopes. The space complexity is $O(|\boldsymbol{F}| \cdot k^{Sc})$.*

Proof. Given a variable X_j, in a single iteration the algorithm:

1. computes max-marginals of all the functions (Q of them) that have X_j in their scopes

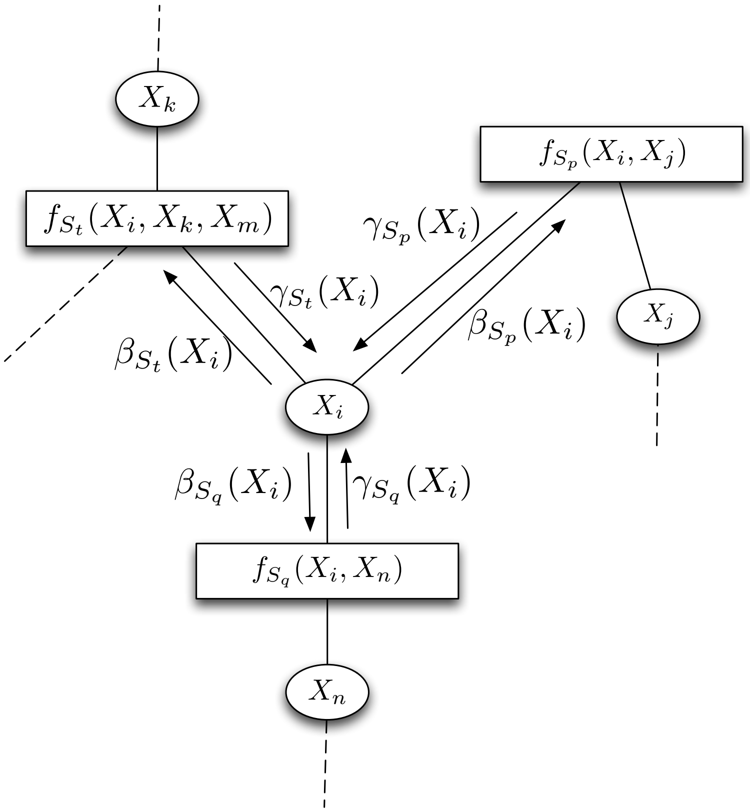


Figure 5.2: FGLP example: local messages involving variable X_i.

by marginalizing out all variables in the scope, except X_j (line 4). The required time is $O(Q \cdot k^{Sc-1})$.

2. computes messages from X_j back to the factor nodes based on max-marginals (line 5), requiring $O(Q \cdot k^{Sc})$ time.
3. updates the functions (line 6). It takes $O(Q \cdot k^{Sc})$ time.

The space complexity is bounded by the size of the input factor graph. □

The main distinction between MPLP and FGLP lies in the fact that MPLP is formulated as an update to a single multi-variate factor (that we can refer to as an "edge", since it corresponds to a graph edge in the case of pairwise functions, or a hyperedge in the problem hypergraph otherwise), along with its corresponding singleton factors over variables, in such a way that as much "cost" as possible is placed into the singleton factors, with the "edge" being left with best-cost zero, similar to cost-shifting schemes in constrain programming domain [83, 82]. Conversely, FGLP updates all edges adjacent to a particular variable, spreading the cost equally between the edges.

■ 5.3 Join Graph Linear Programming

In this section we introduce a way of using the well-known mini-bucket elimination scheme to construct a join-graph with a particular cluster size. Moreover, we present a new bounding scheme called Join Graph Linear Programming.

Join-Graph MBE Structuring. The mini-bucket procedure defines a mini-bucket tree, as discussed in Section 1.2.4.4. Each mini-bucket defines a cluster. Two mini-buckets are connected if there exists a message between them. Once the mini-buckets of the same variables are connected, the mini-bucket tree yields a join-graph, where each cluster has at most $i+1$ variables, where i is the i-bound [70].

Join-Graph MBE Structuring (Algorithm 24) constructs a join-graph using the mini-bucket elimination. Note that the separators between the clusters corresponding to the mini-buckets of the same variable may be over more than a single variable, so the resulting graph may not be a join-graph in a strict sense, not satisfying the induced tree property. However, this minor point does not impact the use of the graph by our algorithms.

As usual, the original functions $f_{\mathbf{S}_k}$ are assigned to the buckets of the highest-index variable in their scope $\mathbf{S}_k$, based on ordering o (lines 1-2). Each bucket $\mathbf{B}_k$, whose scope is larger than i variables, is split into mini-buckets $Q_k = \{q_k^1, \ldots, q_k^p\}$ (line 4). Note that Join-Graph MBE Structuring algorithm does not compute the actual messages between the mini-buckets, but only creates sets of variables corresponding to the messages' scopes and places them into

Algorithm 24: Join-Graph MBE Structuring(i)

```
   Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$, variable ordering $o$, parameter i-bound $i$
   Output: Join-graph with cluster size $\leq i+1$
   //Initialize:
1  Order the variables from $X_1$ to $X_n$ according to ordering $o$;
2  Generate an ordered partition of functions $\mathbf{F} = \{f_{\mathbf{S}_i}\}$ into buckets $\mathbf{B}_{X_1}, \ldots, \mathbf{B}_{X_n}$, where $\mathbf{B}_{X_k}$ is a
   bucket of variable $X_k$;
   //Processing bucket $\mathbf{B}_{X_k}$
3  for $k \leftarrow n$ down to 1 do
4      Partition functions in $\mathbf{B}_{X_k}$ into mini-buckets $Q_{X_k} = \{q_k^1, \ldots, q_k^p\}$; //Each $q_k^p$ has no more than
       $i+1$ variables
       //emulate sending messages between mini-buckets without computing actual functions
5      For each mini-bucket $q_k^p$ create a new set of variables $S_k^p = \{X | X \in q_k^p\} - X_k$ and place it in the
       bucket of its highest variable in the ordering;
6      Maintain an arc between $q_k^p$ and the mini-bucket that includes $S_k^p$;
7  Associate each resulting mini-bucket with a node in the join-graph;
8  Creating arcs: keep the arcs created in step 6 and also connect the mini-bucket clusters belonging
   to the same bucket (for example, in a chain);
9  return A set of functions and variables, corresponding to graph nodes, and the edges between them
```

appropriate buckets, to define the edges of the graph (lines 5-6).

Figure 5.1 presents an example of a problem with 3 variables, whose primal graph is shown in Figure 5.1(a). In Figure 5.1(b) we see the trace of MBE on the problem, namely the buckets and the messages computed. Figure 5.1(c) shows the join-graph created by Join-Graph MBE Structuring. The dotted line corresponds to an edge absent from a mini-bucket tree and added during step at line 8.

The join-graph created by mini-bucket procedure can be used by an LP-tightening algorithm. Join-Graph MBE Structuring is a “top-down” approach, in which i is set to the induced width and reduced until the computational resource constraints are met. In contrast, most existing generalized LP solvers work in a “bottom-up” fashion, running the LP to convergence on the original graph, then proposing slightly larger cliques (for example, from among fully connected triplets of variables [92]) based on some greedy heuristic and running the LP again. Clearly, when representation of large clusters ($i = 15$ to 25) is feasible, the top-down MBE-like approach is far more effective

Algorithm 25: Algorithm JGLP

```
  Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$, variable order $o = \{X_1, \ldots, X_n\}$, parameter $i$.
  Output: Upper bound on the optimum value of MPE cost
  //Initialize: Partition the functions in $\mathbf{F}$ into $\mathbf{B}_{X_1}, \ldots, \mathbf{B}_{X_n}$, where $\mathbf{B}_{X_p}$ contains all functions $f_{S_j}$
  whose highest variable is $X_p$;
1 Build mini-bucket join graph; // (Algorithm 24)
2 Find the function of each mini-bucket $q_k^p$:
  $F_k^p = \sum_{f_{\mathbf{S}_j} \in q_k^p} f_{\mathbf{S}_j}$
3 while NOT converged OR NOT time limit reached do
4     for all pairs of mini-buckets $q_k^p$, $q_k^l$ connected by an edge do
5         Find the separators $S = Scope(q_k^p) \cap Scope(q_k^l)$ ;
6         Find the max-marginals of each mini-bucket
          $q_k^p$: $\gamma_k^p = \max_{Scope(q_k^p)-S}(F_k^p)$;
          $q_k^j$: $\gamma_k^l = \max_{Scope(q_k^l)-S}(F_k^l)$;
7         Update functions in both mini-buckets
          $q_k^p$: $F_k^p \leftarrow F_k^p - \frac{1}{2}(\gamma_k^p - \gamma_k^l)$
          $q_k^l$: $F_k^l \leftarrow F_k^l + \frac{1}{2}(\gamma_k^p - \gamma_k^l)$;
```

Join-Graph Linear Programming. From the perspective of linear programming relaxation (Section 5.2) mini-bucket elimination can be interpreted as running a single pass of LP-tightening, sending messages top down only along edges of the spanning tree of the mini-bucket join graph. A straightforward extension of MBE can be an iterative procedure that repeatedly performs LP-tightening along all of the join graph edges. Algorithm 25 shows the resulting Join-Graph Linear Programming (JGLP) scheme. It constructs the join graph (line 1), calculated mini-bucket functions F_k^p (line 2) and performs re-parametrization updates to F_k^p (as in Algorithm 22) until convergence (lines 2-7). Note that once JGLP converges, performing mini-bucket elimination on the resulting graph will not change the bound value.

Theorem 5.2 (**Complexity of JGLP**). *Given a problem with n variables with largest domains of size k, where at most Q functions have the same variable in their scope, and i-bound i, the time complexity of a single iteration of JGLP is $O(n \cdot Q \cdot k^i)$. The time complexity of join-graph construction step is $O(n \cdot k^{i+1})$. The overall space complexity is $O(n \cdot k^{i+1} + n \cdot k^i)$.*

Proof. The complexity of constructing a join-tree is the same as the complexity of running full mini-bucket elimination algorithm, namely $O(n \cdot k^{i+1})$. The time complexity of a single iteration of JGLP consists of performing for each edge the following steps:

1. compute max-marginals of a pair of clique functions, requires time equal to $O(2 \cdot k^{i+1-|S|})$, where $|S|$ is the size of a separator between the mini-buckets of the same variable.
2. compute the mean, which takes $O(2 \cdot k^{|S|})$ time.
3. update the clique functions, requiring $O(2 \cdot k^{i+1})$ time.

In the worst case there are $O((n-1) \cdot Q + 1)$ edges. Thus the total complexity of a single iteration is $O\Big(2 \cdot \big((n-1) \cdot Q + 1\big) \cdot \big(k^{i+1-|S|} + k^{S} + k^{i+1}\big)\Big) = O(n \cdot Q \cdot k^{i+1})$.

The space complexity of the algorithm is dominated by the necessity of storing in memory the join-graph, whose size is bounded by $O(n \cdot k^{i+1})$ and the messages between the clusters of size $O(n \cdot k^{i})$, yielding the overall space complexity of $O(n \cdot k^{i+1})$. □

■ 5.4 MBE-MM

While the iterative nature of JGLP yields more accurate bounds, in practice it can have a significant additional time and space overhead compared to MBE. The latter scheme does not need to store the entire functions computed in mini-buckets, requiring space of $O(nk^{(i+1)} + nk^{i})$, only the messages between them, reducing the space complexity to $O(nk^{i})$ [25]. The difference may seem insignificant worst-case wise, but makes a practical impact, as we will see in the empirical section. Moreover, the non-iterative nature of MBE makes it easier to estimate the runtime.

There are two orthogonal ways for increasing the accuracy of MBE while keeping the value of parameter i fixed. The first one, extensively studied in [81], involves choosing a partitioning

Algorithm 26: Algorithm MBE-MM

```
Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$, variable order $o = \{X_1, \ldots, X_n\}$, i-bound parameter $i$
Output: Upper bound on the optimum value of MPE cost
//Initialize:
Partition the functions in $\mathbf{F}$ into $\mathbf{B}_{X_1}, \ldots, \mathbf{B}_{X_n}$, where $\mathbf{B}_{X_k}$ contains all functions $f_j$ whose highest variable is $X_k$;
//processing bucket $\mathbf{B}_{X_k}$
for $k \leftarrow n$ down to 1 do
    Partition functions $g$ (both original and messages generated in previous buckets) in $\mathbf{B}_{X_k}$ into the mini-buckets defined $Q_{X_k} = \{q_k^1, \ldots, q_k^t\}$, where each $q_k^p$ has no more than $i+1$ variables;
    Find the set of variables common to all the mini-buckets of variable $X_k$:
    $\mathbf{S}_k = Scope(q_k^1) \cap \cdots \cap Scope(q_k^t)$;
    Find the function of each mini-bucket
    $q_k^p$: $F_k^p \leftarrow \prod_{g \in q_k^p} g$;
    Find the max-marginals of each mini-bucket
    $q_k^p$: $\gamma_{X_k}^p = \max_{Scope(q_k^p) \setminus \mathbf{S}_k} (F_k^p)$;
    Update functions of each mini-bucket
    $F_k^p \leftarrow F_k^p - \gamma_{X_k}^p + \frac{1}{t} \sum_{j=1}^t \gamma_{X_k}^j$;
    Generate messages $h_{X_k \to X_m}^p = \max_{X_k} F_k^p$ and place each in the bucket of highest in the ordering $o$ variable $X_m$ in $Scope(q_k^p)$;
return All the buckets and the cost bound from $\boldsymbol{B}_1$;
```

of the bucket into mini-buckets in a way that introduces the least possible error for a fixed complexity. In this work, however, we do not focus on this issue, assuming in most cases a given partitioning that only takes into account the sizes of function scopes.

We take a different approach, attempting to increase the accuracy of MBE by defining a non-iterative scheme that performs re-parametrization between the mini-buckets of the same variable only. The algorithm mini-bucket elimination with max-marginal matching (MBE-MM, Algorithm 26) proceeds by following the standard mini-bucket downward pass. When each mini-bucket $q_k^p \in Q_k$ is processed, before eliminating variable X_k, we first perform an LP-tightening update (that is, a re-parametrization) to the mini-bucket functions $f_{q_k^p}$. For storage and computational efficiency reasons, we perform a single update on all mini-buckets of the same variable simultaneously, matching their max-marginals on their joint intersection. Alternatively, the updates can be done between all possible pairs of mini-buckets.

Although any max-marginal matching step strictly improves the bound within each bucket

(Equation 5.3), it is not guaranteed to increase the overall accuracy. However, it is reasonable to expect that the update will help, and in practice we find that the bounds are almost always significantly improved (see the experiments, Section 5.6).

Theorem 5.3 (Complexity of MBE-MM). *Given a problem with n variables having domain of size k and an i-bound i, the worst-case time complexity of MBE-MM is $O(n \cdot Q \cdot k^{i+1})$ and its space complexity is $O(n \cdot k^i)$, where Q bounds the number of functions having the same variable X_i in their scopes.*

Proof. Assuming that $|S|$ bounds the size of separator between the mini-buckets of variable X_j, the complexity of processing a *single bucket* comprises of:

1. computing functions of the mini-bucket, requiring time $O(Q \cdot k^{i+1})$
2. marginalizing out the variables that are not in the separator: $O(Q \cdot k^{i+1-|S|})$
3. calculating the geometric mean: $O(Q \cdot k^{|S|})$
4. updating the mini-bucket functions: $O(Q \cdot k^{i+1})$
5. generating the message: $O(Q \cdot k^i)$

Steps 2-4 contribute to the overhead of MBE-MM compared with MBE by $O\big(Q \cdot (k^{i+1-|S|} + k^{|S|} + k^{i+1})\big)$ per bucket. However, the overall complexity is dominated by $O(Q \cdot k^{i+1})$ per bucket and $O(n \cdot Q \cdot k^{i+1})$ overall. The space complexity is dominated by the size of the messages stored: $O(n \cdot k^i)$. □

Interestingly, our algorithms are related to known methods in constraint satisfaction. MBE(i) and JGLP(i), parametrized by an i-bound, are analogous to directional i-consistency and full i-consistency, respectively. Our algorithm MBE-MM represents an intermediate step between these two, and is analogous to an improvement of directional i-consistency with full iterative relational consistency schemes within each bucket [20].

■ 5.5 Heuristics for AND/OR search

The mini-bucket scheme yields a powerful heuristic for informed search algorithms, as discussed in Section 1.2.4.4. The intermediate messages h recorded by MBE (see Figure 5.1(b)) are used to express upper bounds on the best extension of any partial assignment, and so can be used as admissible heuristics guiding best-first or branch and bound search.

Algorithms JGLP and FGLP do not produce heuristic functions directly; we obtain one by applying MBE to the modified (re-parametrized) functions output by the iterative algorithms. As noted earlier, for JGLP constructed with the same i-bound i, this additional pass does not change the value of the bound. In contrast, applying MBE to the (much smaller) functions re-parametrized by FGLP forms new clusters and typically tightens the bound, yielding a hybrid heuristic generator "FGLP+MBE". MBE-MM algorithm directly yields a heuristic function suitable for informed search.

■ 5.6 Empirical Evaluation

We investigate the impact of single-pass and iterative LP-tightening for the task of finding the most probable explanation (MPE) over Bayesian networks. Specifically, we evaluate the performance of MBE-MM, FGLP (applied to the original functions) and JGLP (applied to the mini-bucket-based join graph). We compare these three algorithms against each other and against "pure" MBE, both as stand-alone bounding schemes (Section 5.6.3) and as heuristic evaluation function generators (Section 5.6.4) for three AND/OR search schemes. The first one is AND/OR Branch and Bound (AOBB, Section 1.2.4.3). We also use two weighted heuristic search schemes discussed in Chapter 4: weighted anytime AND/OR Best First search (wAOBF) and weighted AND/OR Branch and Bound (wAOBB).

Benchmark	# inst	n	k	w^*	h_T
Pedigrees	10	581-1006	3-7	16-39	52-104
Grids	32	144-2500	2	15-90	48-283
LargeFam	40	863-1400	17-58	33-111	
Type4	10	3907-8186	5-5	21-32	319-625
WCSP	56	25-1057	2-100	5-287	11-337

Table 5.1: Benchmark parameters: # inst - number of instances, n - number of variables, k - domain size, w^* - induced width, h_T - pseudo-tree height.

5.6.1 Experimental Settings

Our benchmark problems include three sets of instances from genetic linkage analysis networks [32] (denoted Pedigrees, Type4 and LargeFam) and Grid networks from the UAI 2008 competition [16]. In total we evaluated 10 Pedigrees, 10 Type4, 40 LargeFam and 32 Grid networks. We also ran some experiments (Section 5.6.4.2) on the WCSP instances. We previously discussed all these benchmarks, except for LargeFam, in Sections 2.6.1 and 3.6.1, which can be consulted for the details on the datasets. The LargeFam instances come from the same domain of haplotyping networks as the Pedigrees, but are larger and more difficult. All instances are available online[3]. Table 5.1 shows the benchmark parameters. Unless specified differently, each algorithm was run for 24 hours, with memory limit of 3 GB, with i-bounds equal to 10, 15 and 20. The algorithms were implemented in C++ (64-bit). Four schemes were evaluated: plain MBE with no re-parametrization (MBE), MBE with Max-Marginal-Matching (MBE-MM), FGLP followed by MBE (FGLP) and JGLP, that uses cluster size equal to i-bound divided by two, followed by MBE (JGLP). The duration of the run of the two latter heuristic schemes was chosen arbitrarily to be equal to 30 seconds (the configuring pre-processing time requires further exploration).

Our empirical evaluation aims to assess: 1. the impact of moment-matching on the performance of MBE; 2. the value of LP-relaxation and the mini-bucket algorithms as approximation schemes; 3. their effectiveness as search heuristics.

[3] http://graphmod.ics.uci.edu/

5.6.2 Impact of Moment-Matching on the Performance of MBE

We compare MBE and MBE-MM on the Pedigree benchmark when employing various heuristics for splitting a bucket of a variable into several mini-buckets. We use a scope-based heuristic and six content-based partitioning heuristics, some of which were suggested by Rollon, et al., [81]. Note that L1, L2 and Linf errors are calculated in relation to the logarithms of the factors, not the original functions.

- L1: integrated absolute error
- L2: integrated squared error
- Linf: maximum absolute error
- KL: Kullback-Leibler divergence
- HPM: Hilberts projective metric, $HPM(f_1, f_2) = \max(\log \frac{f_1}{f_2}) - \min(\log \frac{f_1}{f_2})$
- MAS: $MAS(f_1, f_2) = \max(\max(\frac{\log f_1}{\log f_2}), \frac{1}{\min(\frac{\log f_1}{\log f_2})}) - 1$

Table 5.2 displays the percentage of Pedigree problems for which the version with the particular partitioning heuristic (with or without max-marginal-matching) found the best solution. Here we mainly focus on the accuracy of the solution, using the runtime only as a tie-breaker, whenever more than one scheme found the same best solution. We considered small i-bounds equal to 5 and 10, in order to highlight the impact of moment-matching and different partitioning strategies when the heuristic is weak. There is no clear winner between the partitioning heuristics, though there is a definite loser: KL distance measure never yields the best solution for either MBE or MBE-MM. The scope-based heuristic fares very well for both values of i-bound. There is no doubt that overall the use of moment-matching almost always yields more accurate results. It yields better performance for all partitioning strategies and both i-bounds. The sole exception is the Linf heuristic for $i = 10$.

Algorithm	% best, i-bound=5	% best, i-bound=10
Scope-based heuristic + no MM	0.0	0.0
Scope-based heuristic + MM	21.4	21.4
L1 heuristic + no MM	7.1	7.1
L1 heuristic + MM	14.3	7.1
L2 heuristic + no MM	0.0	0.0
L2 heuristic + MM	14.3	21.4
Linf heuristic + no MM	0.0	14.3
Linf heuristic + MM	7.1	14.3
KL heuristic + no MM	0.0	0.0
KL heuristic + MM	0.0	0.0
HPM heuristic + no MM	7.1	0.0
HPM heuristic + MM	14.3	14.3
MAS heuristic + no MM	0.0	0.0
MAS heuristic + MM	14.3	0.0

Table 5.2: The percentage of the instances for which each algorithm found the best bound, for i-bounds equal to 5 and 10. Pedigrees (total 14 problems). Ties broken based on runtime.

				MBE-MM		FGLP		
Instance	n	k	w	L2	Linf	5 iter	500 iter	1500 iter
1502.uai	209	4	6	**-2.8954**	**-2.8954**	-2.6753	-2.6886	-2.6886
29.uai	82	4	14	**-3.6906**	-3.6888	-3.2006	-3.2259	-3.2259
404.uai	100	4	19	**-5.2229**	-5.0545	-3.5222	-3.7092	-3.7432
408.uai	200	4	35	-3.1147	-3.1177	-3.6974	-3.9934	**-4.0735**
42.uai	190	4	26	**-3.1872**	-3.0472	-2.1092	-2.3906	-2.5227
503.uai	143	4	9	-3.1872	-3.1872	-2.9683	-3.2905	**-3.4497**
505.uai	240	4	22	-1.1207	-2.1888	-2.7076	-3.0725	**-3.2433**
54.uai	67	4	11	**-3.0701**	-2.9848	-1.8466	-2.0719	-2.0812

Table 5.3: The upper bounds on the log(MPE) for the select WCSP instances by MBE-MM with two content-based heuristics using L2 and Linf distance measures with $i = 10$ and FGLP ran for 5, 500 and 1500 iterations. For each instance we report the number of variables n, the largest domain size k and the induced width along the ordering used w. The best bounds are shown in bold. Memory limit 3 GB, time limit 24h.

5.6.3 LP-tightening Algorithms as Bounding Schemes

We next compare the single-pass schemes MBE and MBE-MM against the iterative FGLP and JGLP schemes. In Table 5.3 we see the upper bounds produced by MBE-MM and FGLP for select WCSP instances. MBE-MM used two content-based heuristics using L2 and Linf distance measures and $i = 10$. FGLP ran for 5, 500 and 1500 iterations. We show the upper bounds on the log scale (lower values are better). We see that even for a large number of

instance name	i	MBE	MBE-MM	FGLP time cut-offs			JGLP time cut-offs		
(n, k, w^*)				5	300	3600	5	300	3600
		UB/time	UB/time	UB	UB	UB	UB	UB	UB
Grids									
75-25-5	10	-15.4553/1	-18.4089/1	-16.6853	-16.6854	-16.6854	-20.0289	**-20.8364**	**-20.8364**
(625, 2, 34)	20	-17.4417/4	-20.0576/4				-20.0576	-20.1278	**-20.7067**
90-30-5	10	-8.2481/1	-10.2597/1	-10.2450	-10.2705	-10.2705	-11.8469	-12.9594	**-13.015**
(900, 2, 42)	20	-9.7424/7	-11.6004/7				-11.6004	-11.6942	**-12.5259**
90-34-5	10	-8.42007/1	-10.3708/1	-9.65003	-9.69458	-9.69458	-12.3469	-13.2262	**-13.2883**
(1156, 2, 48)	20	-9.58332/8	-12.3670/9				-12.3670	-12.5621	**-13.1538**
90-42-5	10	-12.7401/1	-15.9680/1	-15.2480	-15.3653	-15.3653	-18.4100	-20.7714	**-20.8136**
(1764, 2, 60)	20	-14.6136/13	-18.5487/14				-18.5487	-18.7679	-13.2883
LargeFam									
largeFam4_11_51	10	-201.136/1	-211.656/1	-201.582	**-201.673**	**-201.673**	-211.671	-216.500	**-217.176**
(1002, 4, 40)	20	OOM	OOM				OOM	OOM	OOM
largeFam4_11_55	10	-229.43/1	-242.489/1	-226.075	**-226.328**	**-226.328**	-242.657	-249.551	-250.453
(1114, 4, 38)	20	OOM	OOM				OOM	OOM	OOM
largeFam4_12_51	10	-218.229/2	-239.896/3	-217.564	**-217.740**	**-217.740**	-239.896	-245.900	-253.153
(1461, 4, 56)	20	OOM	OOM				OOM	OOM	OOM
Pedigrees									
pedigree7	10	-105.854/1	-109.569/1	-110.179	-110.187	-110.187	-109.960	-110.810	**-111.293**
(867, 4, 32)	20	-108.011/33	**-111.120/42**				OOM	OOM	OOM
pedigree13	10	-69.0973/1	-70.0999/1	-71.8561	**-71.8591**	**-71.8591**	-70.4581	-71.9869	**-72.0374**
(888, 3, 32)	20	-69.8890/8	-71.1071/11				-71.1071	-71.1071	-71.3658
pedigree31	10	-125.032/1	-126.629/1	-126.667	**-126.678**	**-126.678**	-126.644	-129.158	**-129.277**
(1006, 5, 30)	20	OOM	OOM				OOM	OOM	OOM
pedigree41	10	-110.156/1	-114.858/1	-114.681	-114.681	-114.681	-115.050	-118.133	**-118.419**
(885, 5, 33)	20	-112.153/29	**-117.638/37**				OOM	OOM	OOM
pedigree51	10	-97.741/1	-103.461/1	-101.927	-101.977	-101.977	-103.791	-106.542	**-107.276**
(871, 5, 39)	20	-102.110/13	-105.734/16				-105.734	-105.734	**-106.619**
Type4									
type4_120_17	10	-1128.22/1	-1203.08/1	-1049.34	-1049.85	-1049.86	-1203.21	-1221.21	**-1223.69**
(4302, 5, 23)	20	-1235.94/18	**-1237.95/21**				-1237.95	OOM	OOM
type4_170_23	10	-1682.9/1	-1747.18/1	-1509.96	-1511.61	-1511.65	-1747.22	-1769.96	**-1772.16**
(6933, 5, 21)	20	-1783.18/7	**-1783.76/7**				**-1783.76**	**-1783.76**	**-1783.76**

Table 5.4: Upper bound (log scale) and runtime (# seconds) for a typical set of instances, $i = 10$ and $i = 20$. Lower values are better. OOM shows that the algorithm ran out of memory (4 GB). We report the number of variables n, largest domain size k, and the induced width w along the ordering used. Memory limit 3 GB, time limit 24h. The best cost for each instance and i-bound are highlighted in bold. Highlighted FGLP solution corresponds to $i = 20$, cases where JGLP runs out of memory.

iterations FGLP does not achieve the same accuracy as MBE-MM ($i = 10$) for more than half of these instances.

In Table 5.4 we present a subset of the results obtained by MBE, MBE-MM, MPLP and JGLP from 4 benchmarks (Pedigrees, Type4, LargeFam and Binary Grids) for the i-bound values of $i = 10$ and $i = 20$. Note that the value of i does not influence the results of FGLP that runs on the original functions.

As before, we observe that for all instances MBE-MM is superior to pure MBE. For most instances the MBE-MM bounds are also tighter than pure FGLP (at the comparable point in time), especially for $i = 20$, but only if the memory required by MBE-MM is not too high, see, for example, instances 75-25-5 and largeFam4_11_51 for both i-bounds. JGLP usually finds the best solution for a given i-bound, even for a small number of iterations, e.g., instances 90-30-5 and pedigree51. However, for larger i-bound it often runs out of memory, including the instances feasible for both MBE schemes, such as pedigree41, $i = 20$. It confirms our earlier observation that, though the worst case space complexity bounds are the same for JGLP and MBE-MM, the former requires more space in practice. Overall, it is clear that, given enough time and memory, JGLP eventually produces the most accurate bounds.

■ 5.6.4 LP-tightening Algorithms as Search Guiding Heuristics

In addition to evaluating the algorithms as approximate bounding schemes, we explore their potential as generators of heuristic evaluation functions for search, as described in Section 1.2.4.4. We consider three anytime algorithms: AOBB (see Section 1.2.4.3 for details), wAOBF (Section 4.4.2) and wAOBB (Section 4.6). For each of them we tested five heuristics generated prior to search by pure MBE, by MBE with Max-Marginal-Matching, by FGLP followed by MBE, by JGLP and by FGLP followed by JGLP. The iterative FGLP and JGLP algorithms were run for 30 seconds each. The total time bound for AOBB with each of the heuristics was set to 24 hours (including the pre-processing), memory limit was 3 GB, and the mini-bucket i-bound was set to $i \in \{10, 15, 20\}$.

■ 5.6.4.1 Heuristics for AOBB

In Tables 5.5-5.7 we show results comparing the heuristics for AOBB on a representative set from the full 92 instances from 4 benchmarks. The table reports the total runtime in seconds and the number of nodes expanded by each of the AOBB with each of the heuristic schemes for finding the optimal solution.

We see that the heuristic generated by MBE-MM yields more efficient search, compared

Instances
(n,k,w^*,h_T)
	AOBB-MBE(i)
	AOBB-MBE-MM(i)
	AOBB-FGLP+MBE(i)
	AOBB-JGLP(i)
	i-bound=10
		AOBB-MBE(i)
		AOBB-MBE-MM(i)
		AOBB-FGLP+MBE(i)
		AOBB-JGLP(i)
		i-bound=15
			AOBB-MBE(i)
			AOBB-MBE-MM(i)
			AOBB-FGLP+MBE(i)
			AOBB-JGLP(i)
			i-bound=20

	time / # nodes	time / # nodes	time / # nodes
Pedigrees (# inst=11, n=581-1006, k=3-7, w^*=16-39, h_T=52-104)			
pedigree13	— / —	21992 / 4924235937	2380 / 608940710
(888, 3, 32, 102)	66156 / 11726505961	8150 / 1441111422	704 / 164319080
	5658 / 905160506	**926** / 182970673	357 / 73658489
	3943 / 623204366	1687 / 309201030	**193** / 26959464
pedigree31	— / —	— / —	— / —
(1006, 5, 30, 85)	61382 / 10617627744	3856 / 750931932	— / —
	24896 / 3695993630	1033 / 188749113	— / —
	5711 / 987186989	**374** / 65064202	— / —
pedigree37	2 / 230972	13 / 10236	— / —
(726, 5, 20, 72)	**1** / 26334	**9** / 4086	— / —
	31 / 9108	39 / 2796	— / —
	30 / 3898	54 / 2328	— / —
pedigree38	101 / 19704583	— / —	— / —
(581, 5, 16, 52)	**1** / 105984	— / —	— / —
	31 / 23980	— / —	— / —
	33 / 2760	— / —	— / —
pedigree41	— / —	6592 / 1471067842	1192 / 326841387
(885, 5, 33, 100)	4434 / 784381348	257 / 53398086	**75** / 9235144
	1252 / 214639049	168 / 29612599	88 / 7495210
	833 / 135586985	**72** / 8336890	— / —
pedigree51	— / —	— / —	— / —
(871, 5, 39, 98)	— / —	— / —	3395 / 679214977
	— / —	64554 / 10177661600	**1790** / 338647575
	— / —	**11402** / 1724160339	— / —
pedigree7	76552 / 14504177460	15291 / 3319803362	— / —
(867, 4, 32, 90)	2171 / 348425451	428 / 78953096	— / —
	805 / 140665826	227 / **36619862**	— / —
	531 / 83548121	90 / 10110416	— / —
pedigree9	37198 / 7509543280	14224 / 3161690948	263 / 73694367
(935, 7, 27, 100)	4748 / 521695781	189 / 41599090	**9** / 185567
	685 / 130633536	143 / 25036271	38 / 169196
	650 / 125778651	**34** / 757844	44 / 27751
Type4 linkage (# inst=10, n=3907-8186, k=5, w^*=21-32, h_T=319-625)			
type4b_120_17	— / —	— / —	— / —
(4072, 5, 24, 319)	— / —	— / —	**33** / 720778
	— / —	— / —	71 / 1168656
	— / —	— / —	OOM
type4b_170_23	— / —	— / —	**9** / 121182
(5590, 5, 21, 427)	— / —	— / —	10 / 122064
	— / —	28603 / 2576275134	37 / 45156
	— / —	79917 / 5485723239	OOM

Table 5.5: Pedigree, Type4, search time (seconds) / # nodes expanded for selected instances. FGLP and JGLP ran for 30 seconds. "OOM" indicates that search ran out of memory (3GB) and "— / —" that it ran out of time (24h). In **bold** we highlight the best runtime for each instance. n - number of variables, k - maximum domain size, w^* -induced width, h_T - pseudo-tree height.

Instances	AOBB-MBE(i)	AOBB-MBE(i)	AOBB-MBE(i)
	AOBB-MBE-MM(i)	AOBB-MBE-MM(i)	AOBB-MBE-MM(i)
	AOBB-FGLP+MBE(i)	AOBB-FGLP+MBE(i)	AOBB-FGLP+MBE(i)
	AOBB-JGLP(i)	AOBB-JGLP(i)	AOBB-JGLP(i)
(n,k,w^*,h_T)	i-bound=10	i-bound=15	i-bound=20
	time / # nodes	time / # nodes	time / # nodes
LargeFam (# inst=30, n=863-1400, k=3, w^*=17-58, h_T=33-111)			
largeFam3-haplo_10_51	2 / 380359	2 / 45618	OOM
(863, 3, 21, 48)	**1** / 190092	**0** / 3228	OOM
	31 / 152663	31 / 3472	OOM
	30 / 50125	43 / 2683	OOM
largeFam3-haplo_10_52	OOM	OOM	OOM
(959, 3, 39, 68)	OOM	OOM	528 / 115676769
	OOM	OOM	**262** / 4734126
	OOM	**7505** / 1246219453	OOM
largeFam3-haplo_10_54	11 / 3388405	1 / 156267	13 / 3246
(962, 3, 21, 51)	**1** / 364635	**0** / 5724	13 / 2293
	30 / 149820	31 / 4346	45 / 2292
	31 / 11644	41 / 3041	OOM
largeFam3-haplo_11_50	82 / 20303320	**2** / 346742	23 / 202050
(874, 3, 26, 44)	**4** / 869775	**2** / 132695	14 / 2262
	31 / 305170	31 / 3213	44 / 2265
	31 / 11968	42 / 2434	OOM
largeFam3-haplo_11_51	— / —	— / —	— / —
(1020, 3, 33, 66)	— / —	31043 / 6879525475	— / —
	— / —	**13489** / 3088790310	— / —
	— / —	— / —	— / —
largeFam3_11_53	— / —	— / —	OOM
(1094, 3, 39, 71)	— / —	— / —	OOM
	— / —	44663 / 8080262337	OOM
	— / —	**10292** / *1878168857*	OOM
largeFam3-haplo_11_55	— / —	— / —	— / —
(1133, 3, 40, 66)	— / —	— / —	— / —
	— / —	**34359** / 6896094744	— / —
	— / —	65066 / 10458952821	— / —
largeFam3_11_57	— / —	— / —	— / —
(1128, 3, 39, 77)	— / —	58112 / 9421606282	OOM
	— / —	**4683** / 782421094	OOM
	— / —	10801 / 1715700802	OOM
largeFam3_11_59	— / —	— / —	OOM
(1119, 3, 33, 73)	— / —	— / —	OOM
	— / —	59012 / 8098379409	OOM
	— / —	**22538** / 3025470612	OOM
largeFam3-haplo_12_56	4 / 737612	**0** / 8258	OOM
(1101, 3, 17, 51)	**1** / 196081	1 / 2655	**1** / 2207
	31 / 169153	31 / 2396	30 / 2203
	30 / 63448	31 / 2203	30 / 2203

Table 5.6: LargeFam search time (seconds) / # nodes expanded for selected instances. FGLP and JGLP ran for 30 seconds. "OOM" indicates that search ran out of memory (3GB) and "— / —" that it ran out of time (24h). In **bold** we highlight the best runtime for each instance. n - number of variables, k - maximum domain size, w^* -induced width, h_T - pseudo-tree height.

Instances	AOBB-MBE(i)	AOBB-MBE(i)	AOBB-MBE(i)
	AOBB-MBE-MM(i)	AOBB-MBE-MM(i)	AOBB-MBE-MM(i)
	AOBB-FGLP+MBE(i)	AOBB-FGLP+MBE(i)	AOBB-FGLP+MBE(i)
	AOBB-JGLP(i)	AOBB-JGLP(i)	AOBB-JGLP(i)
(n,k,w^*,h_T)	i-bound=10	i-bound=15	i-bound=20
	time / # nodes	time / # nodes	time / # nodes
Binary Grid (# inst=32, n=144-2500, k=2, w^*=15-90, h_T=48-283)			
50-15-5	1 / 301067	0 / 15251	1 / 451
(225, 2, 19, 76)	**0** / 23525	**0** / 3676	**0** / 451
	30 / 4925	30 / 739	30 / 451
	30 / 2711	30 / 451	31 / 451
50-17-5	17 / 5832680	1 / 102141	2 / 6533
(289, 2, 22, 84)	**0** / 8816	**0** / 1119	**1** / 1096
	30 / 1556	30 / 870	31 / 579
	30 / 579	31 / 722	83 / 579
75-17-5	11 / 3038949	**0** / 76197	2 / 4168
(289, 2, 22, 78)	1 / 233137	**0** / 1845	**1** / 579
	31 / 205205	31 / 1134	32 / 579
	31 / 178737	32 / 698	60 / 579
75-18-5	24 / 7480037	1 / 157973	2 / 2247
(324, 2, 24, 85)	**0** / 80168	**0** / 1688	**1** / 649
	30 / 2309	30 / 1015	32 / 649
	30 / 649	32 / 649	62 / 649
75-19-5	908 / 256088880	3 / 1036278	3 / 39763
(361, 2, 25, 89)	**6** / 1559857	**0** / 11627	**2** / 723
	31 / 228145	31 / 6862	32 / 723
	30 / 14687	31 / 1489	33 / 723
75-20-5	1936 / 422432794	6 / 1686365	7 / 1180807
(400, 2, 27, 99)	**14** / 3340985	**1** / 11539	**2** / 1287
	33 / 577327	30 / 2115	33 / 962
	30 / 4967	30 / 816	34 / 801
90-30-5	— / —	54415 / 10603123693	5853 / 1299094138
(900, 2, 42, 151)	8601 / 1790747055	423 / 97620783	**12** / 1125656
	5928 / 1084067942	337 / 67303699	47 / 2101919
	350 / 62930133	**31** / 28688	48 / 7493
90-34-5	— / —	— / —	— / —
(1156, 2, 48, 186)	— / —	2517 / 396585142	**9** / 413587
	— / —	5439 / 886872519	38 / 174323
	270 / 35270820	**31** / 8445	58 / 4029
90-42-5	— / —	— / —	— / —
(1764, 2, 60, 229)	— / —	62051 / 8399774202	2471 / 340122171
	— / —	17628 / 2349582057	**651** / 93715978
	40 / 1411953	**134** / 13038792	OOM
90-50-5	— / —	— / —	— / —
(2500, 2, 74, 312)	— / —	— / —	— / —
	— / —	— / —	— / —
	— / —	**48781** / 4187198638	OOM

Table 5.7: Grids, search time (seconds) / # nodes expanded for selected instances. FGLP and JGLP ran for 30 seconds. "OOM" indicates that search ran out of memory (3GB) and "— / —" that it ran out of time (24h). In **bold** we highlight the best runtime for each instance, *italics* indicate the smallest search space explored. n - number of variables, k - maximum domain size, w^* -induced width, h_T - pseudo-tree height.

with “pure” MBE, both in terms of runtime and nodes expanded, as we would expect based on our evaluation of MBE-MM as a bounding scheme. The two iterative schemes are even more powerful as heuristic generators. JGLP is the overall best-performing scheme, as long as memory is available. For some memory-intense problems, infeasible for JGLP with large i-bound, such as pedigree13, $i = 15$, FGLP+MBE presents a good balance between accuracy of the heuristic and the runtime.

The i-bound value has a similar impact on the results for all the schemes. As expected, larger i-bounds yield more accurate heuristics, but lead to increased memory requirements. See, for example, pedigree37 and pedigree38, in Table 5.5, where all the schemes run out of memory while attempting to compute heuristics for $i = 20$.

Figures 5.3 and 5.4 illustrate the anytime behavior of AOBB when employing each of the bounding schemes to generate heuristics. We plot solution cost as a function of time (both on log scale) for two instances from Grids, Pedigrees, LargeFam and Type4 benchmarks, chosen to best illustrate prevailing tendencies in algorithms’ behavior, for a relatively high i-bound. Higher values are better. As expected, AOBB-MBE has the least accurate heuristic and is consistently inferior to other schemes. For both AOBB-FGLP-MBE and AOBB-JGLP the choice of the maximum number of iterations (or, equivalently, maximum heuristic computation time) plays an essential role in determining the success as anytime schemes. In our experiments both FGLP and JGLP ran for 30 seconds which caused them to often report the first solution considerably later than AOBB-MBE and AOBB-MBE-MM. See, for example, Figure 5.3, pedigree31. AOBB-MBE-MM usually finds initial solution the fastest. An interesting exception is the instance largeFam3-haplo_10_56 in Figure 5.4. This instance is memory intensive with 1127 variables and induced width equal to 49. JGLP runs out of memory on this instance without reporting any results. FGLP produces a tighter bound than MBE and MBE-MM, yielding search with better anytime behavior.

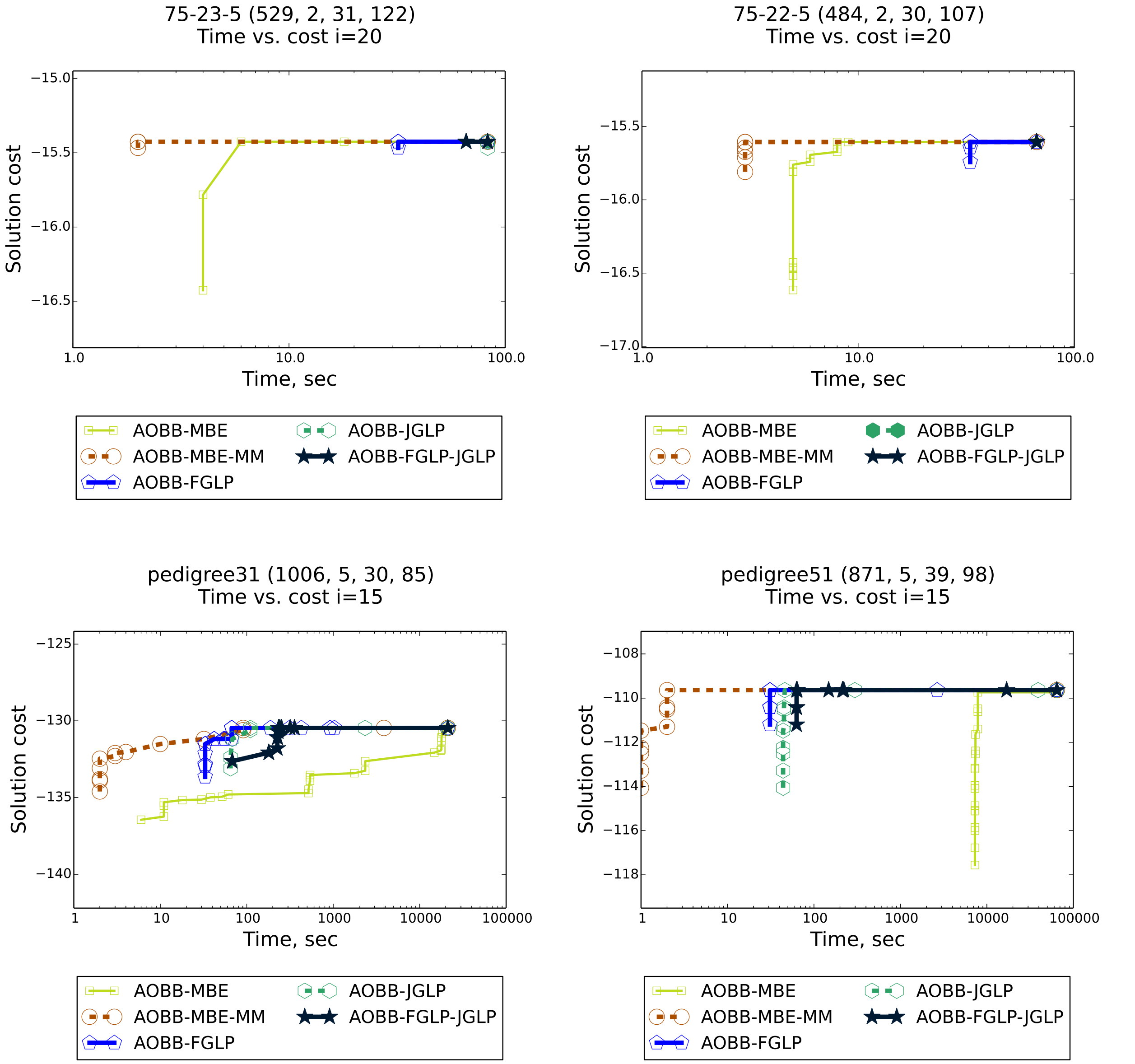


Figure 5.3: Lower bounds by the AOBB as a function of time (sec), memory limit 3 GB, time limit 24h. Grids and Pedigrees. Parameters: (n,k,w^*,h_T), n - number of variables, k - maximum domain size, w^* -induced width, h_T - pseudo-tree height.

■ 5.6.4.2 Heuristics impact on wAOBF and wAOBB

We evaluated the performance of wAOBF and wAOBB with the same 5 heuristic options as AOBB. Each instance was run with i-bounds ranging from 2 to 18. The FGLP and JGLP were ran for 30 sec each. The time limit was 1 hour, the memory limit was 4 GB.

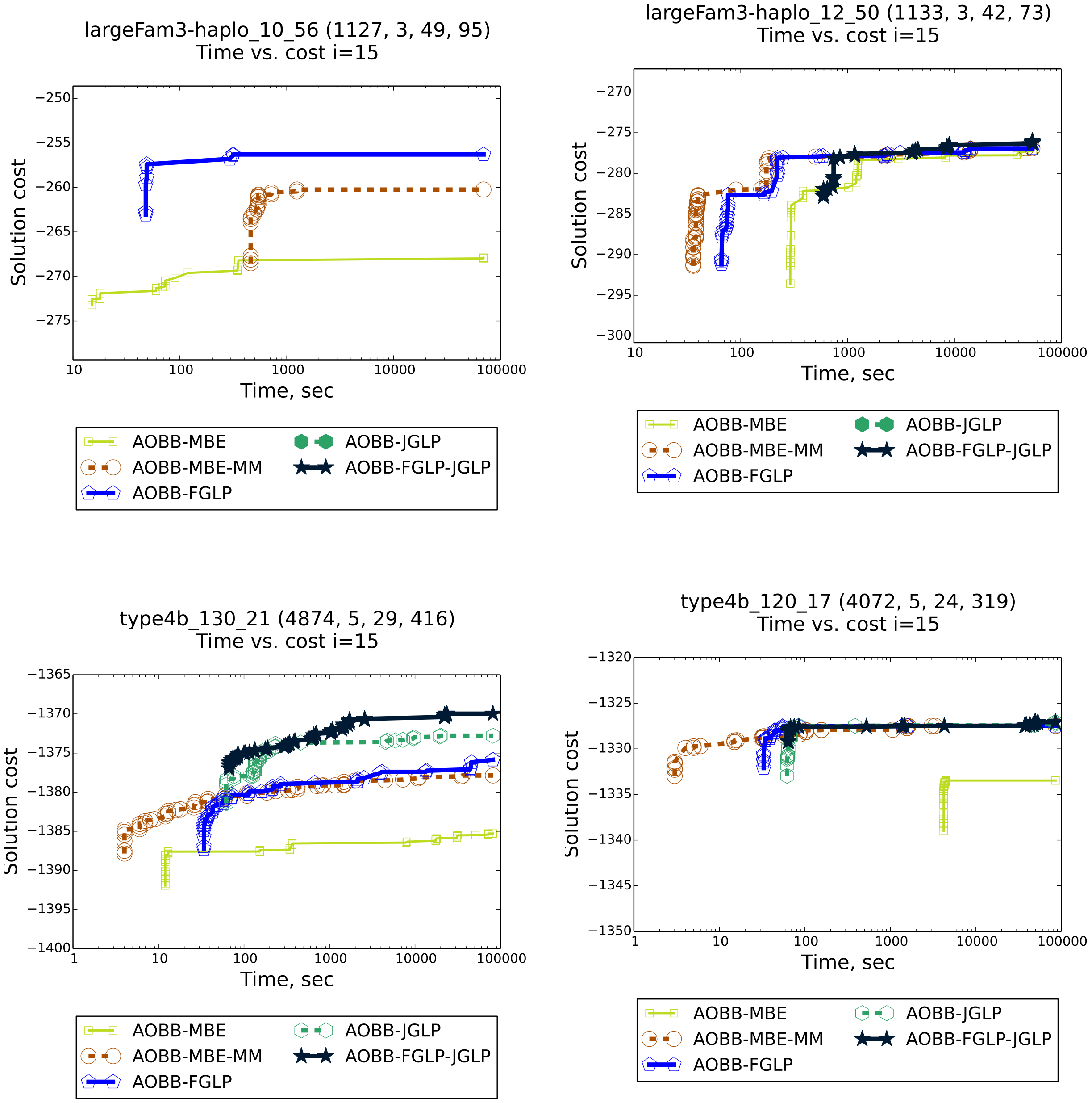


Figure 5.4: Lower bounds by the AOBB as a function of time (sec), memory limit 3 GB, time limit 24h. LargeFam and Type4. Parameters: (n,k,w^*,h_T), n - number of variables, k - maximum domain size, w^* -induced width, h_T - pseudo-tree height.

Tables 5.8 and 5.9 show the comparison between the heuristics for wAOBF and wAOBB respectively. We consider a subset of the hardest instances from all benchmarks. Since the algorithms are anytime, we show the number of problems, for which an algorithm with a

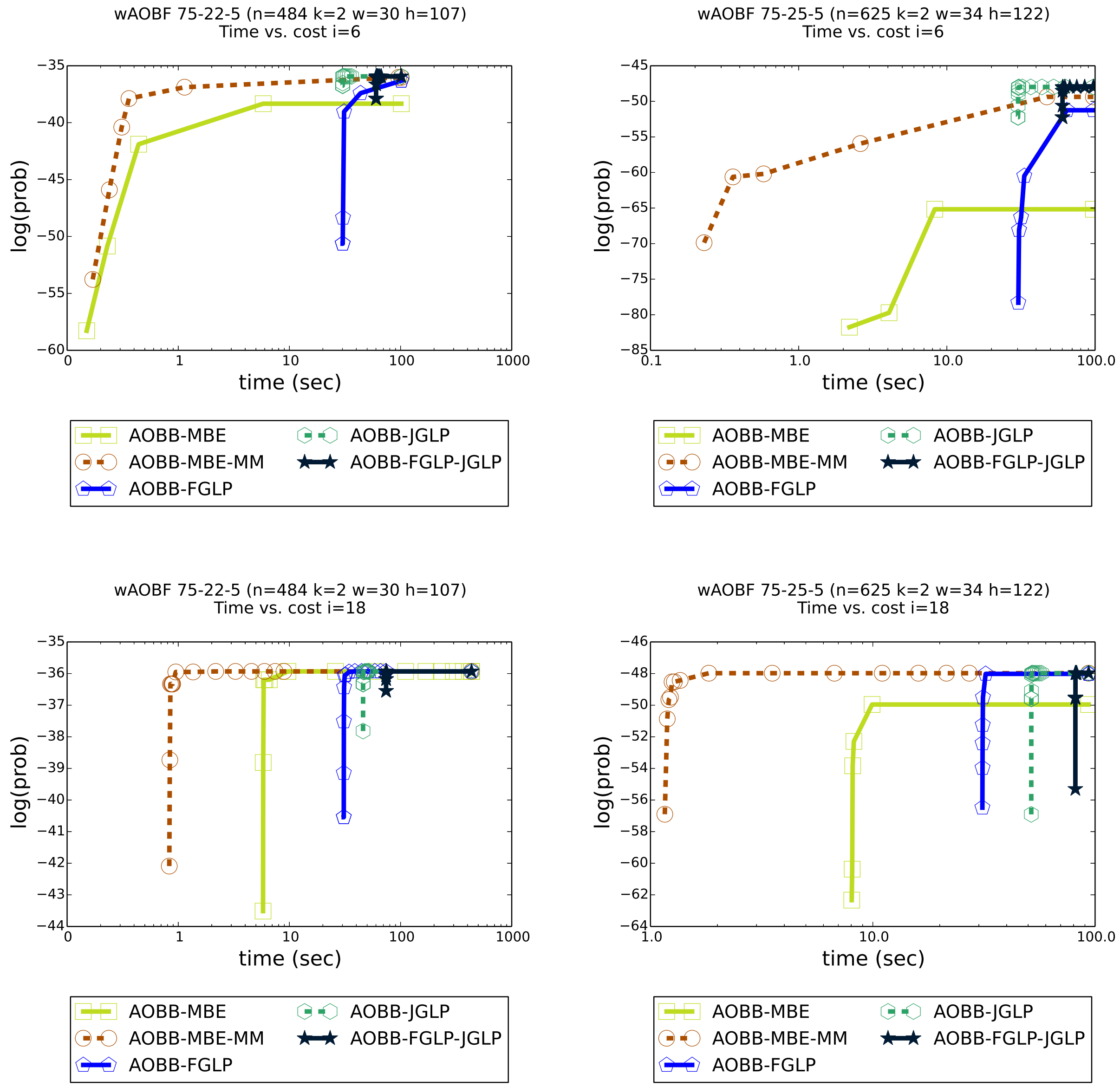


Figure 5.5: Heuristic comparison: Grids, wAOBF. Cost (on log scale) as a function of time (sec). 4 GB memory limit, 1 hour time limit. n - number of variables, k - maximum domain size, w -induced width, h - pseudo-tree height.

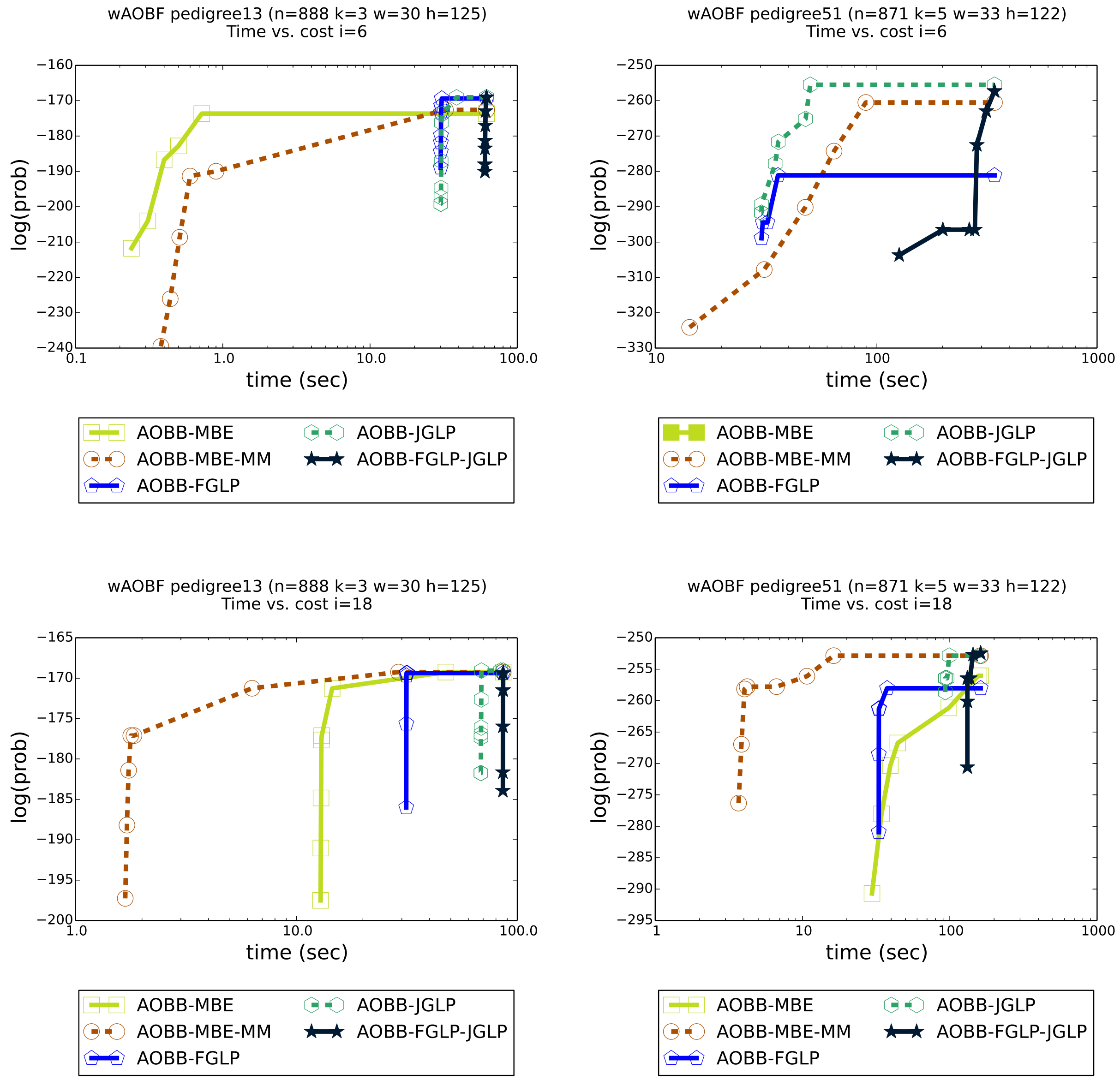


Figure 5.6: Heuristic comparison: Pedigrees, wAOBF. Cost (on log scale) as a function of time (sec). 4 GB memory limit, 1 hour time limit, $i = 6$. n - number of variables, k - maximum domain size, w -induced width, h - pseudo-tree height.

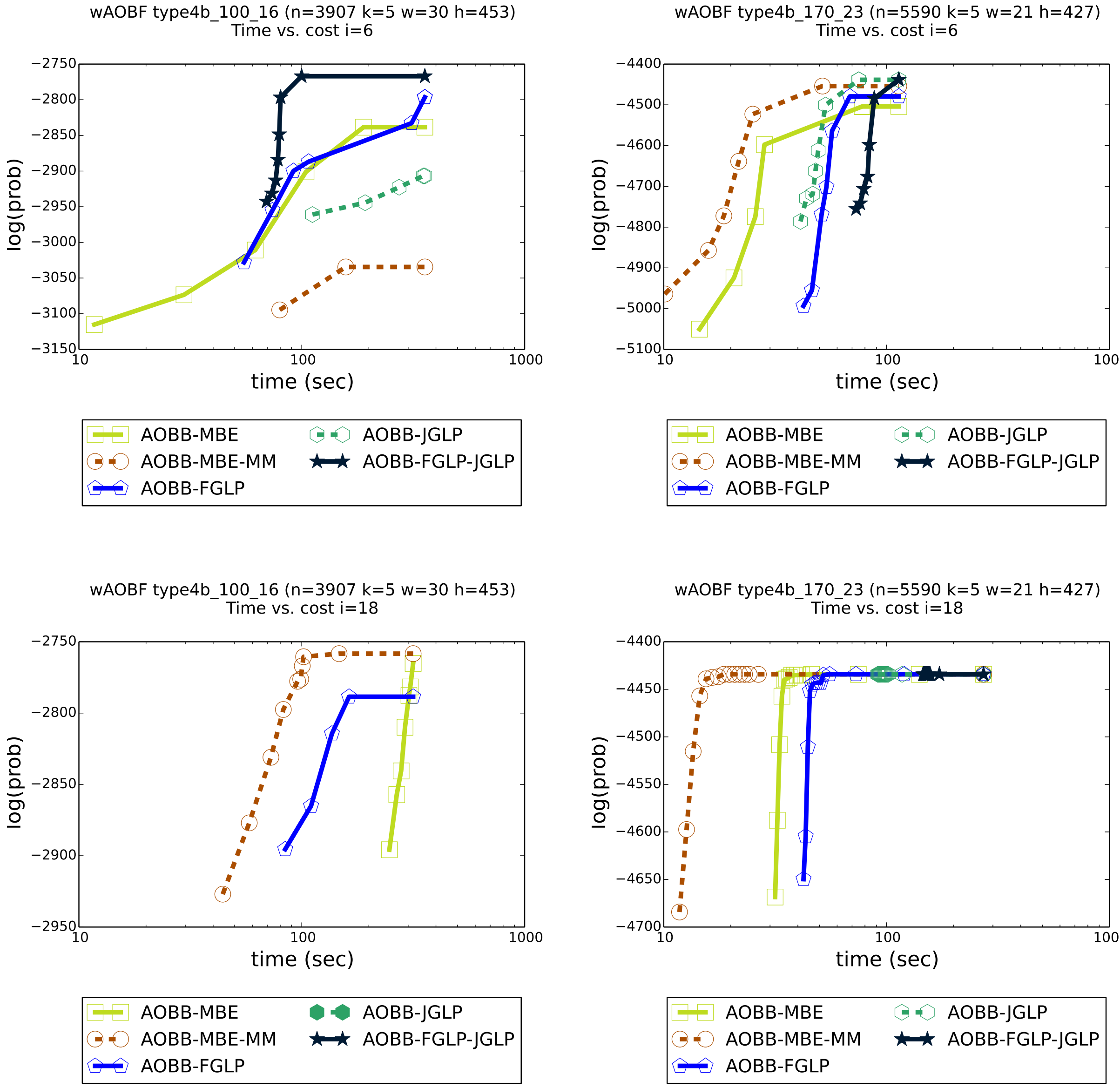


Figure 5.7: Heuristic comparison: Type4, wAOBF. Cost (on log scale) as a function of time (sec). 4 GB memory limit, 1 hour time limit. n - number of variables, k - maximum domain size, w -induced width, h - pseudo-tree height.

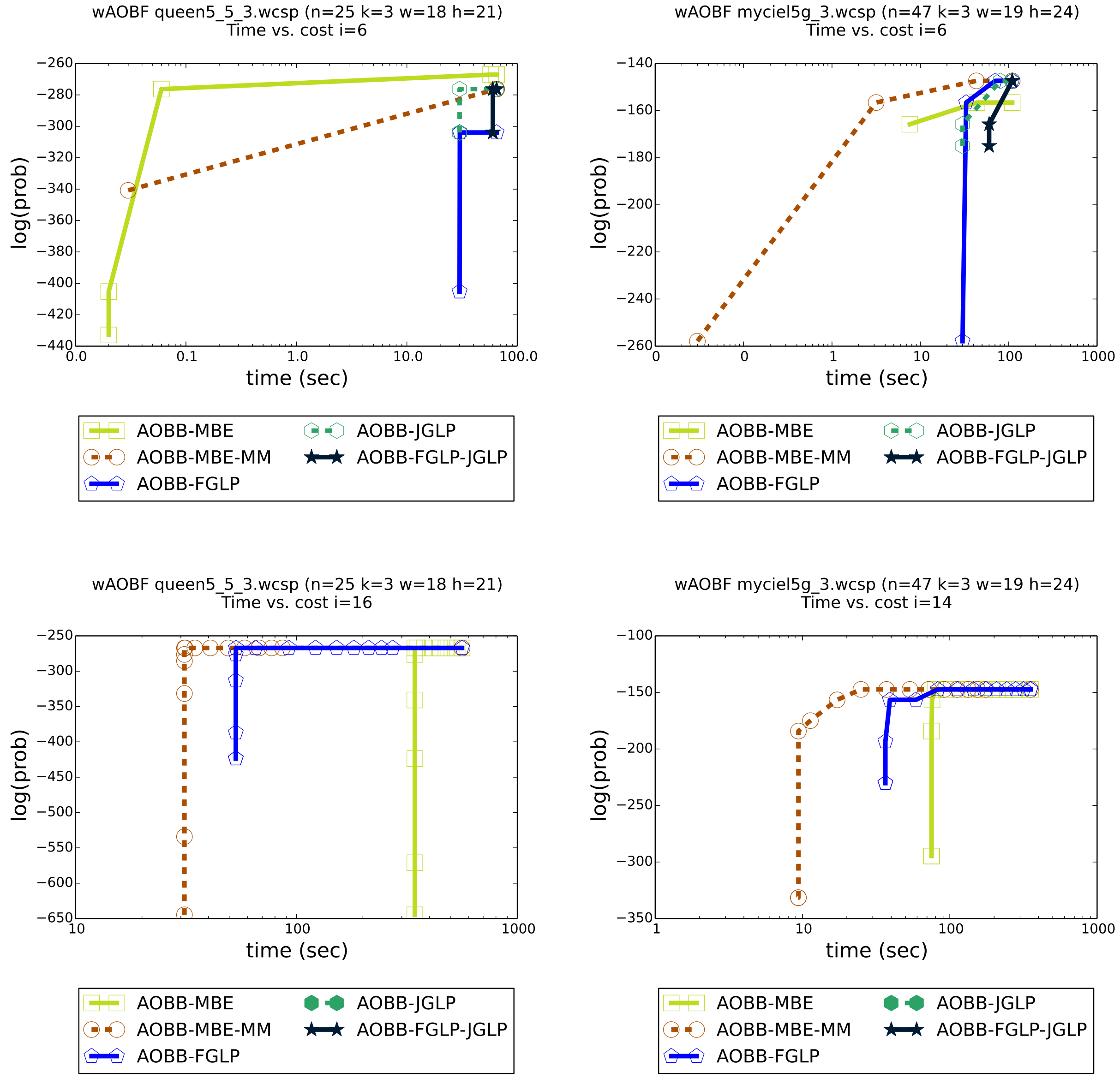


Figure 5.8: Heuristic comparison: WCSPs, wAOBF. Cost (on log scale) as a function of time (sec). 4 GB memory limit, 1 hour time limit. n - number of variables, k - maximum domain size, w -induced width, h - pseudo-tree height.

particular heuristic finds a better solution for a specific time bound. If the solution costs are equal, the smaller weight is preferred (since it corresponds to a tighter bound on the solution accuracy, as we discussed in Chapter 4). The time accounts for pre-processing, i.e., heuristic calculation. The results are consistent for both wAOBF and wAOBB. For shortest time periods MBE-MM is the best heuristic, since it is more powerful than plain MBE and requires much less time to compute than the iterative schemes that we ran for 30 sec, or 60 sec, in the case of FGLP+JGLP. For longer time periods JGLP and FGLP+JGLP are superior, with the latter having a slight edge over the former.

Figures 5.5-5.8 display for wAOBF the plots of a cost on a logarithmic scale as a function of time. For each benchmark we show a subset of instances best representing typical trends in the results, for two i-bounds: relatively low (top row in each figure) and high (bottom row). For most instances (e.g., Figure 5.8, myciel5g_3.wcsp) we see once again that for the short time periods the algorithm using MBE-MM heuristic is superior and reports the initial solutions first, while JGLP and FGLP+JGLP require more time to produce solutions of similar accuracy. However, on other instances, such as pedigree51, $i = 6$ in Figure 5.6, JGLP is often more successful than MBE-MM and finds the initial solutions faster. For higher i-bounds for many instances, such as 75-22-5, $i = 18$ in Figure 5.5, the difference in the solution costs often becomes negligible, given time. However MBE-MM is still clearly the superior, reporting solutions much faster in most cases. It is easy to explain, considering that the high i-bound allows both MBE-MM and JGLP to generate almost equally accurate heuristics, while the calculation of the MBE-MM heuristic is considerably faster, which gives wAOBF with MBE-MM an advantage in terms of the anytime performance.

Figures 5.9 - 5.12 show the plots of the cost as a function of time of wAOBB with various heuristics for some select instances from all four benchmarks. For low i-bounds the dominance between heuristics is not as obvious: there are instances, for which FGLP (e.g., 75-25-5, $i = 2$ in Figure 5.9) and even plain MBE (e.g., 75-23-5, $i = 2$) yield superior anytime performance.

Algorithms	Time bound (sec)						
Assorted, # inst=225, n=25-5590, k=2-100, w^*=15-100, h_T=21-453							
Heuristics	10	30	60	600	1200	2400	3600
MBE	0.0	0.0	1.2	0.0	0.0	0.0	0.0
MBE-MM	**70.6**	**73.1**	22.4	8.7	9.2	9.2	9.2
FGLP+MBE	0.0	0.0	0.0	0.0	0.0	0.0	0.0
JGLP	0.0	0.0	**32.7**	**15.6**	14.5	14.5	14.5
FGLP+JGLP+MBE	0.0	0.0	0.0	15.0	**15.0**	**15.0**	**15.0**

Table 5.8: wAOBF: Moment-matching heuristics, mixed ”difficult” problems: The % of instances, for which an algorithm finds better solution than others. 4 GB memory, 1 hour time limit. # - number of instances, n - number of variables, k - domain size, w^* - induced width, h_T - pseudo-tree height.

Algorithms	Time bound (sec)						
Assorted, # inst=225, n=25-5590, k=2-100, w^*=15-100, h_T=21-453							
Heuristics	10	30	60	600	1200	2400	3600
MBE	0.0	0.0	1.2	1.2	1.2	1.2	1.2
MBE-MM	**68.5**	**73.8**	20.4	9.9	9.9	8.7	8.7
FGLP+MBE	0.0	0.0	0.0	0.0	0.0	0.0	0.0
JGLP	0.0	0.0	**30.9**	**12.3**	12.8	11.0	**12.7**
FGLP+JGLP+MBE	0.0	0.0	0.0	**12.3**	**13.4**	**12.1**	11.0

Table 5.9: wAOBB: Moment-matching heuristics, mixed ”difficult” problems: The % of instances, for which an algorithm finds better solution than others. 4 GB memory, 1 hour time limit. # - number of instances, n - number of variables, k - domain size, w^* - induced width, h_T - pseudo tree height.

On certain other instances (e.g., type4b_120_17, $i = 2$, Figure 5.11) we see behavior similar to that by wAOBF, namely JGLP is quite successful while MBE and FGLP are inferior. On many other instances MBE-MM is superior, especially for large i-bound (e.g., Figures 5.10, pedigree13, $i = 18$ and Figures 5.12, type4b_120_17, $i = 18$).

■ 5.6.4.3 Summary

Based on our empirical evaluation we conclude that LP-tightening can significantly improve the power of the MBE heuristics. The question of instance-based balance, namely tailoring the right level of i-bound and LP-tightening to the problem instance, is clearly a central issue and a direction of future research. In this study we observed that MBE-MM always improves over MBE, using comparable time and memory, while FGLP quickly converges and is less memory-consuming than the other schemes. On the other hand, given sufficient time and memory JGLP produces the tightest bound. The use of FGLP- and JGLP-based heuristics reduces the search space explored for most instances, compared to MBE. For simpler problems runtime may be dominated by the heuristic calculation time and thus faster schemes like MBE and MBE-MM may be cost-effective. None of the three max-marginal matching schemes dominates always. The use of larger clusters can significantly reduce the search space, in some cases enabling quick solutions to problems that were infeasible within the time limit for smaller i-bound, but can increase memory requirements exponentially.

■ 5.7 Conclusion

In this chapter we describe the systematic combination of iterative cost-shifting updates with elimination-order based clustering algorithms and provide extensive empirical evaluation demonstrating its effectiveness. We present Join Graph Linear Programming, a new bounding scheme for optimization tasks in graphical models that combines MBE bounds with LP-based cost-shifting. We discuss the connection between JGLP and previously developed methods: a) shifting costs procedure, e.g., [82, 59, 83]; b) Max-Product Linear Programming [40]. Empirically, JGLP utilizes the available memory to produce better bounds than FGLP that runs on the original clusters. We showed the schemes' ability to improve informed search algorithms; without requiring significantly more computational power than classical MBE (for fixed i) they can drastically reduce the search space. Notably, the algorithm that used as a heuristic generator a sequence FGLP+JGLP+MBE-MM won the first place in all optimization categories in 2011 Pascal2 Probabilistic Inference Challenge[4].

[4] http://www.cs.huji.ac.il/project/PASCAL/realBoard.php

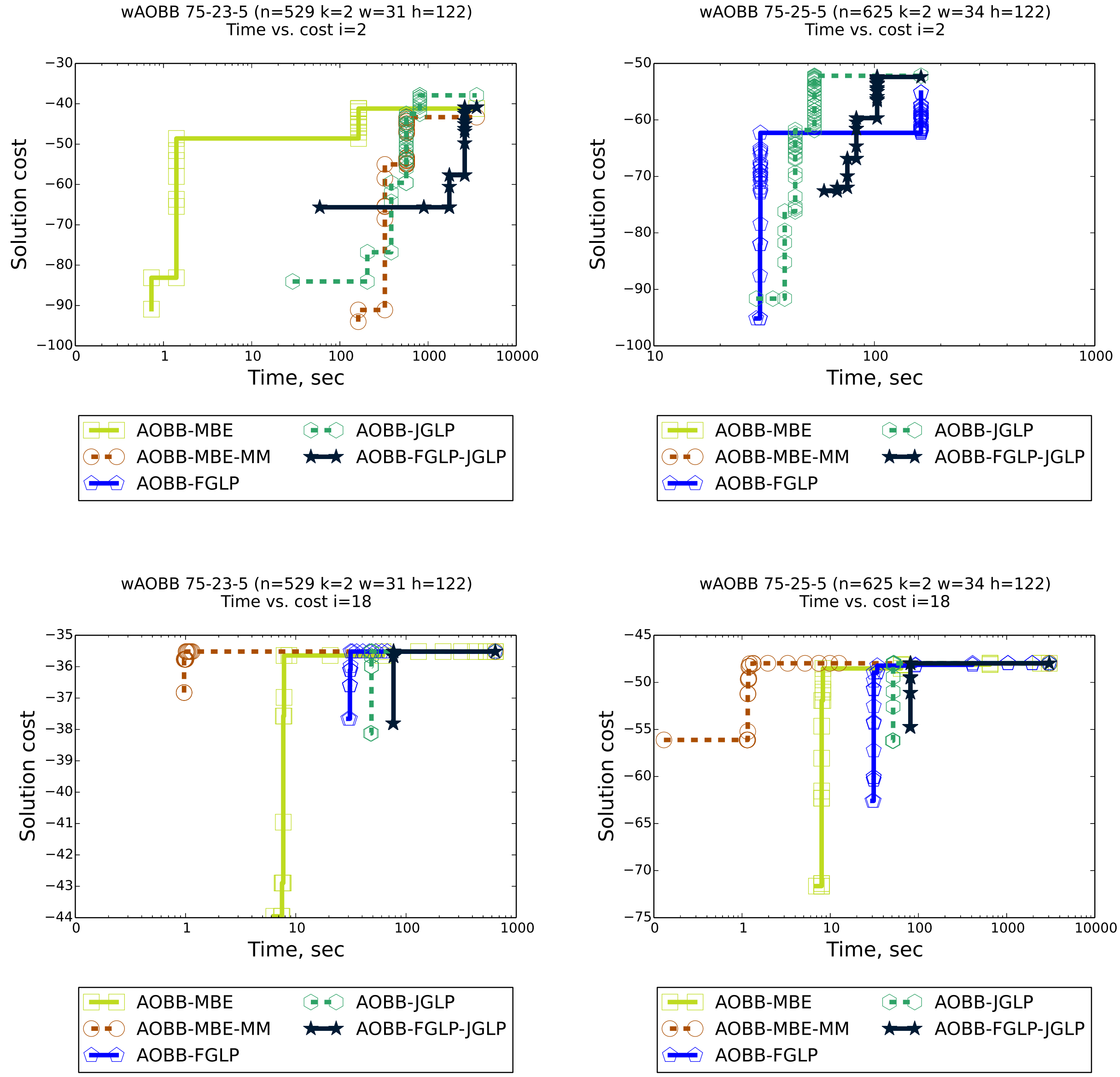


Figure 5.9: Heuristic comparison: wAOBB. Grids. Cost (on log scale) as a function of time (sec). 4 GB memory limit, 1 hour time limit. Low i-bounds. n - number of variables, k - maximum domain size, w -induced width, h - pseudo-tree height.

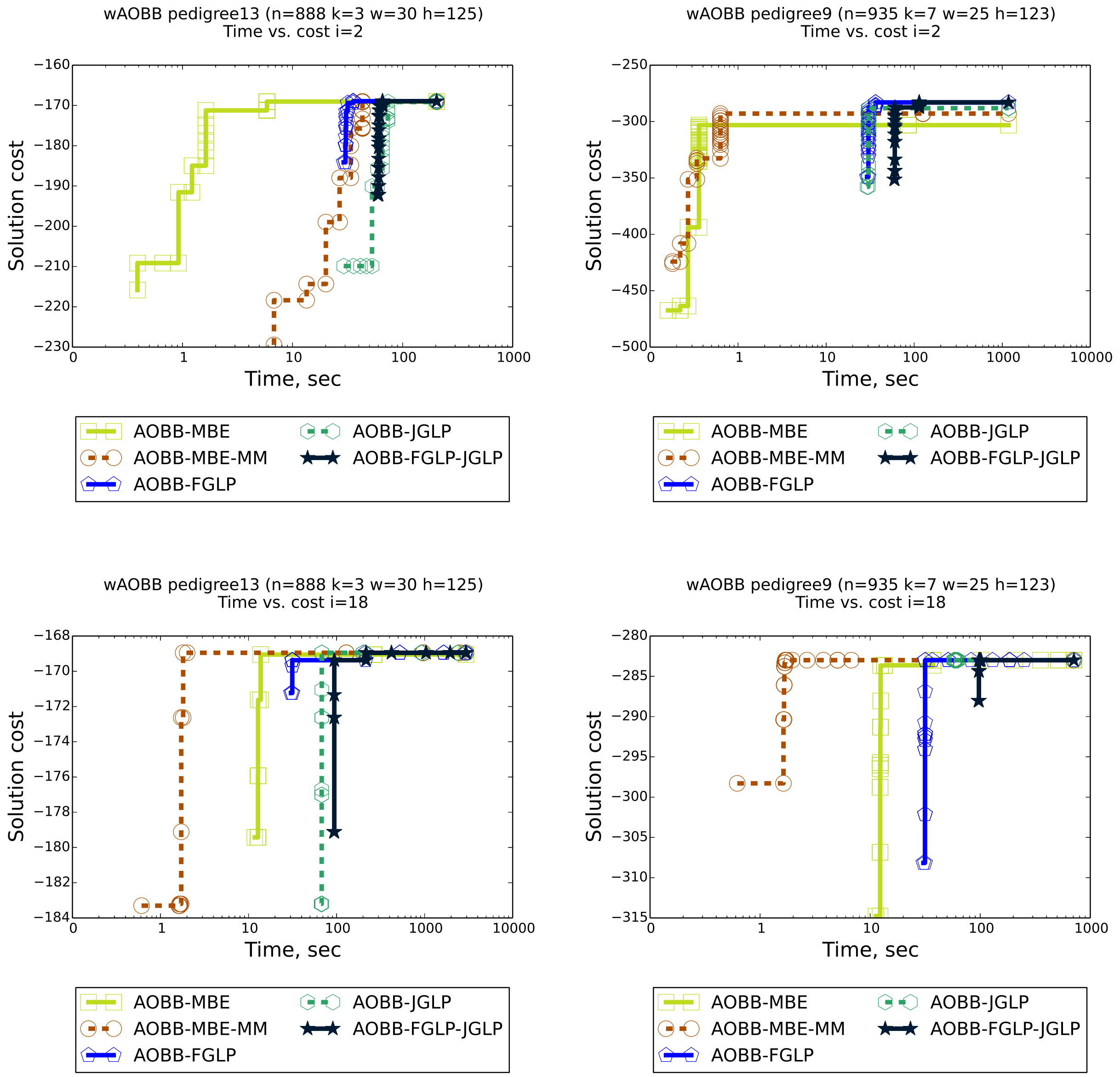


Figure 5.10: Heuristic comparison: wAOBB. Pedigrees. Cost (on log scale) as a function of time (sec). 4 GB memory limit, 1 hour time limit. Low i-bounds. n - number of variables, k - maximum domain size, w -induced width, h - pseudo-tree height.

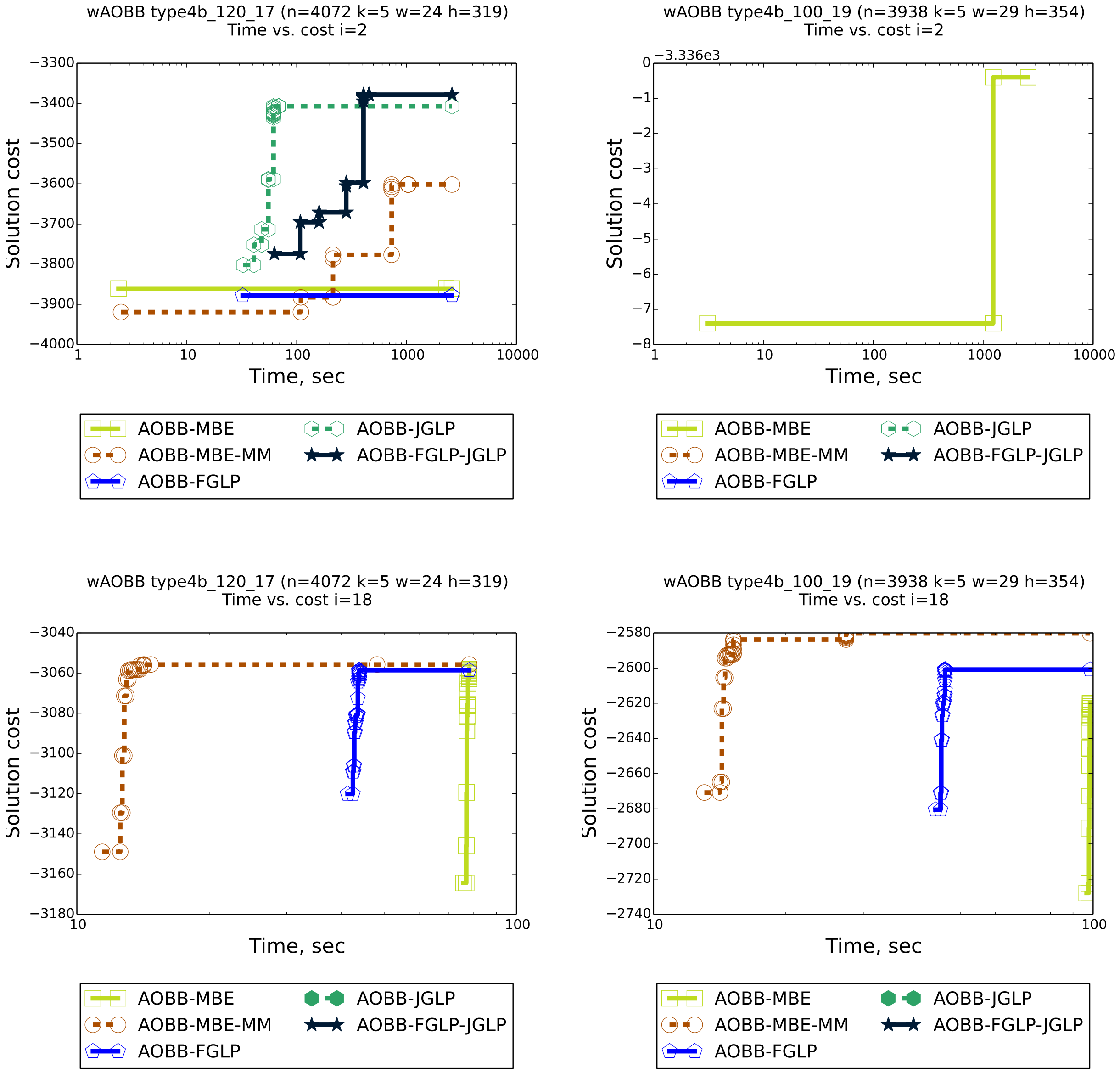


Figure 5.11: Heuristic comparison: wAOBB. Type4. Cost (on log scale) as a function of time (sec). 4 GB memory limit, 1 hour time limit. Low i-bounds. n - number of variables, k - maximum domain size, w -induced width, h - pseudo-tree height.

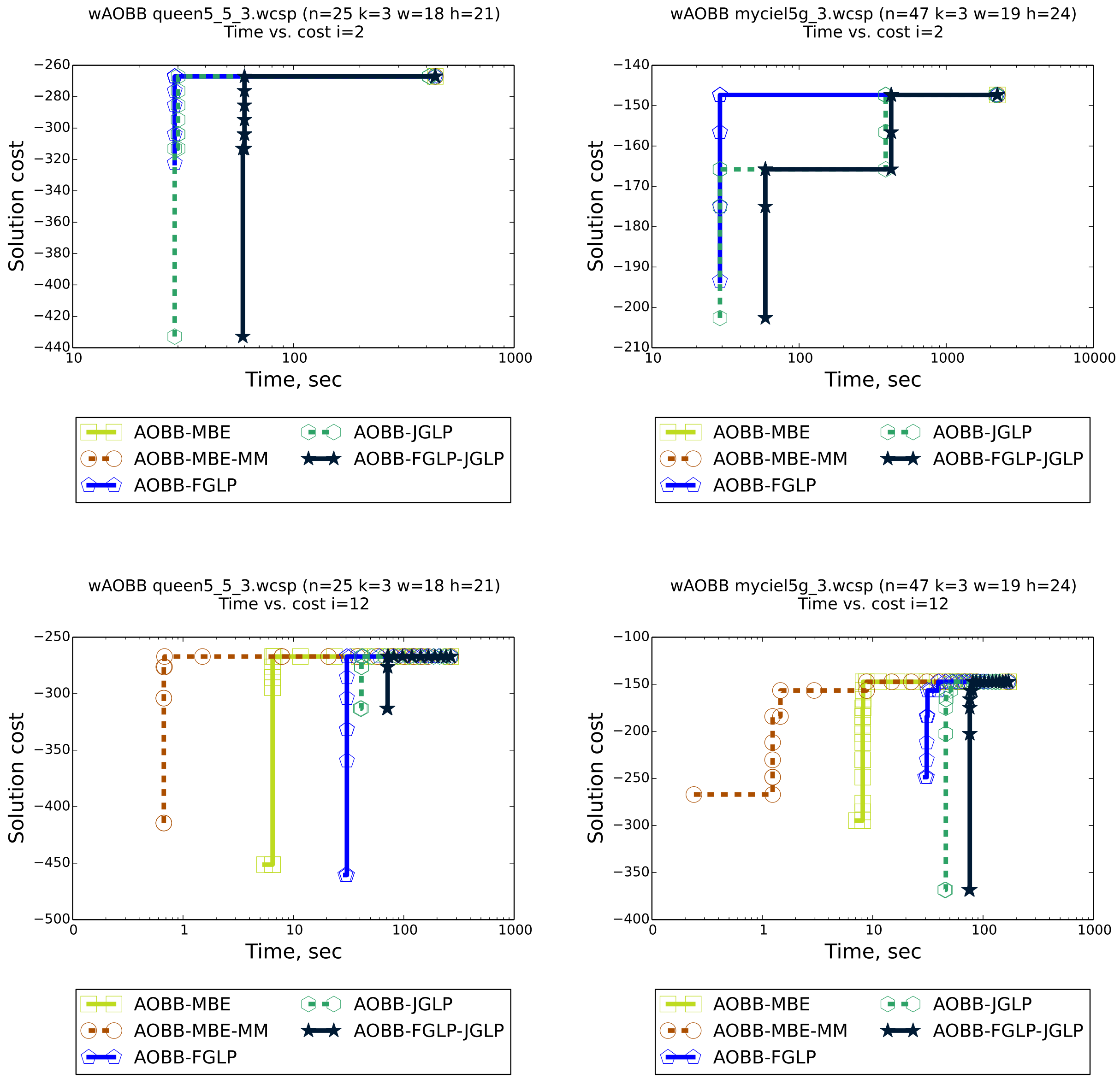


Figure 5.12: Heuristic comparison: wAOBB. WCSP. Cost (on log scale) as a function of time (sec). 4 GB memory limit, 1 hour time limit. Low i-bounds. n - number of variables, k - maximum domain size, w -induced width, h - pseudo-tree height.

Chapter 6

Conclusion

The research presented in this dissertation is focused on advancing the existing schemes for optimization in graphical models in several directions.

The first issue we focused on was finding m-best solutions to the optimization problem. For the first time we formally defined the m-best task in the context of a semiring, thus making existing inference algorithms directly applicable through the newly formulated combination and marginalization operators. As an example of such algorithms we proceeded to adapt the well-known bucket elimination algorithm to the m-best task, analyzing its asymptotic worst case time and space complexity and contrasting it with previously developed schemes. We also presented an approximate extension to the m-best task of the mini-bucket elimination scheme, that provides bounds on the m-best solutions.

We also extended to the m-best task the best-first, in particular A*, and depth-first branch and bound search algorithms, formulating new schemes called m-A* and m-BB. We analyzed the m-A* properties and proved that it is, among other things, sound, complete and optimally efficient. Though the idea behind m-A* is fairly simple and intuitive, this work, to the best of our knowledge, is the first one ever to provide a systematical theoretical analysis of its properties.

Moreover, we presented the m-best versions of popular AND/OR search algorithms, known to efficiently solve optimization task by exploiting problem decomposition. We empirically evaluated the resulting schemes m-AOBF and m-AOBB, contrasting them with m-A* and m-BB and comparing with efficient m-best LP-based schemes, showing that all our algorithms have comparable runtimes for small number of required solutions m, while scaling significantly better as m increases.

The other focus of this dissertation was the application of the ideas of weighted heuristic search, popular in path-finding, to the graphical models domain. We proposed and empirically evaluated two versions of anytime weighted best-first and two versions of anytime depth-first branch and bound search algorithms. Our extensive experiments on a large collection of real-life benchmarks showed the potential of the weighted schemes as anytime algorithms, yielding reasonably accurate solutions while significantly reducing runtime and explored search space. Crucially the weighted heuristic search schemes report a suboptimality bound for each solution equal to the weight used. This bound is often quite tight and is invaluable for solving hard instances, for which the optimal costs are not known.

The final contribution of this dissertation lies in the advancement of the mini-bucket elimination heuristics, used in all the AND/OR search algorithms we mentioned. Presenting the first systematic combination of the ideas of elimination-order based clustering algorithms, such as MBE, and iterative cost-shifting updates, used in such algorithms as MPLP or h-MBE, we propose three new bounding schemes. Among them, there are two iterative algorithms: Factor Graph Linear Programming (FGLP) and Join Graph Linear Programming (JGLP); and a single-pass scheme Mini-Bucket Elimination with Moment-Matching (MBE-MM). Our empirical evaluation demonstrates superiority of the new schemes over ordinary MBE both as approximate schemes and, more importantly, as heuristic generators. The new schemes significantly improve the performance of all tested search algorithms.

■ 6.1 Directions for Future Research

The work presented leaves room for additional improvements that can be pursued in the future, some of which has been already mentioned in the respective chapters.

Inference algorithms for m-best task.

The formulation of the m-best problem as a semiring makes existing inference algorithms applicable to the m-best optimization. In particular, it might be possible, using the proposed combination and optimization operators, to extend a popular loopy belief propagation algorithm, the Pearl polytree algorithm [78]. This could yield an efficient approximate m-best algorithm.

Search algorithms for m-best task.

Our empirical evaluation showed the inferiority of the m-AOBF scheme when exploring AND/OR search graph while using caching. Such poor performance is not suggested by theory and thus leaves potential opening for the improvement of the implementation. Additionally, we have yet to implement and experimentally test a graph version of m-AOBB.

An important way to improve the performance of the m-best search schemes lies in the use of more sophisticated heuristics. In our evaluation we used the regular mini-bucket elimination. We expect that the use of more advanced MBE-MM and JGLP heuristic would considerably improve the efficiency of the schemes, making them even more competitive with the previously developed m-best algorithms.

Weighted anytime schemes.

As we have shown in our empirical evaluation, none of the proposed algorithms is always superior, with the performance varying greatly depending on the benchmark and heuristic strength. Thus one of the very important directions of future work is the issue of automatic algorithm selection and identification of particular problem features, that would allow to predict which algorithm has most potential for a particular instance.

Instead of selecting a single algorithm for a specific class of instances a promising alternative is to combine the algorithms within a portfolio framework. A question of portfolio building and scheduling also requires a further investigation.

The performance of the algorithms can likely be improved by employing dynamic weights, namely the weight that changes its value during a search iteration, for example, based on the current node depth.

Finally, the potential of newly introduced quantity h-weight for bounding the optimal costs remains an open question and requires further exploration.

Bibliography

[1] S. Aji and R. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

[2] H. Aljazzar and S. Leue. K : A heuristic search algorithm for finding the k shortest paths. *Artificial Intelligence*, 175(18):2129–2154, 2011.

[3] D. Allen and A. Darwiche. Rc_link: Genetic linkage analysis using bayesian networks. *International Journal of Approximate Reasoning*, 48(2):499–525, 2008.

[4] S. Arnborg. Efficient algorithms for combinatorial problems with bounded decomposability - a survey. *BIT*, 25(1):2–23, 1985.

[5] D. Batra. An efficient message-passing algorithm for the m-best map problem. *arXiv preprint arXiv:1210.4841*, 2012.

[6] D. Batra, S. Nowozin, and P. Kohli. Tighter relaxations for MAP-MRF inference: A local primal-dual gap based separation algorithm. In *Conference on Uncertainty in Artificial Intelligence (AISTATS)*, 2011.

[7] S. Bistarelli, H. Faxgier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and valued CSPs: Basic properties and comparison. *Over-Constrained Systems*, pages 111–150, 1996.

[8] S. Bistarelli, R. Gennari, and F. Rossi. Constraint propagation for soft constraints: Generalization and termination conditions. *Principles and Practice of Constraint Programming–CP 2000*, pages 83–97, 2000.

[9] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the Association of Computing Machinery*, 44, No. 2:165–201, 1997.

[10] B. Cabon, S. De Givry, and G. Verfaillie. Anytime lower bounds for constraint violation minimization problems. In *Principles and Practice of Constraint ProgrammingCP98*, pages 117–131. Springer, 1998.

[11] P. Chakrabarti, S. Ghose, and S. De Sarkar. Admissibility of AO* when heuristics overestimate. *Artificial Intelligence*, 34(1):97–113, 1987.

[12] E. Charniak and S. Shimony. Cost-based abduction and MAP explanation. *Artificial Intelligence*, 66(2):345–374, 1994.

[13] M. Chavira, A. Darwiche, and M. Jaeger. Compiling relational bayesian networks for exact inference. *International Journal of Approximate Reasoning*, 42(1):4–20, 2006.

[14] M. Cooper and T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154(1):199–227, 2004.

[15] A. Darwiche. Decomposable negation normal form. *Journal of the ACM (JACM)*, 48(4):608–647, 2001.

[16] A. Darwiche, R. Dechter, A. Choi, V. Gogate, and L. Otten. Results from the probablistic inference evaluation of UAI08, a web-report in http://graphmod.ics.uci.edu/uai08/Evaluation/Report. *In: Uncertainty in Artificial Intelligence applications workshop*, 2008.

[17] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *arXiv preprint math/0307152*, 2003.

[18] H. C. Daume. *Practical structured learning techniques for natural language processing.* ProQuest, 2006.

[19] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1):41–85, 1999.

[20] R. Dechter. *Constraint processing.* Morgan Kaufmann, 2003.

[21] R. Dechter. Reasoning with probabilistic and deterministic graphical models: Exact algorithms. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 7(3):1–191, 2013.

[22] R. Dechter, K. Kask, and R. Mateescu. Iterative join-graph propagation. In *UAI02*, pages 128–136. Citeseer, 2002.

[23] R. Dechter and R. Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2-3):73–106, 2007.

[24] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32:506–536, 1985.

[25] R. Dechter and I. Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM*, 50(2):107–153, 2003.

[26] E. Delisle and F. Bacchus. Solving Weighted CSPs by successive relaxations. In *Principles and Practice of Constraint Programming*, pages 273–281. Springer, 2013.

[27] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[28] G. Elidan, I. McGraw, and D. Koller. Residual belief propagation: Informed scheduling for asynchronous message passing. In *Proc. UAI*, pages 165–173, 2006.

[29] P. Elliott. Extracting the K Best Solutions from a Valued And-Or Acyclic Graph. Master's thesis, Massachusetts Institute of Technology, 2007.

[30] D. Eppstein. Finding the k shortest paths. In *Proceedings 35th Symposium on the Foundations of Computer Science*, pages 154–165. IEEE Comput. Soc. Press, 1994.

[31] P. W. Fieguth, W. C. Karl, and A. S. Willsky. Efficient multiresolution counterparts to variational methods for surface reconstruction. *Computer Vision and Image Understanding*, 70(2):157–176, 1998.

[32] M. Fishelson and D. Geiger. Exact genetic linkage computations for general pedigrees. In *International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 189–198, 2002.

[33] M. Fishelson and D. Geiger. Optimizing exact genetic linkage computations. *Journal of Computational Biology*, 11(2-3):263–275, 2004.

[34] M. a. Fishelson, N. Dovgolevsky, and D. Geiger. Maximum likelihood haplotyping for general pedigrees. *Human Heredity*, 59(1):41–60, 2005.

[35] M. Fontaine, S. Loudni, and P. Boizumault. Exploiting tree decomposition for guiding neighborhoods exploration for vns. *RAIRO-Operations Research*, 47(02):91–123, 2013.

[36] E. C. Freuder and M. J. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. In *IJCAI*, volume 85, pages 1076–1078, 1985.

[37] M. Fromer and A. Globerson. An lp view of the m-best map problem. *Advances in Neural Information Processing Systems*, 22:567–575, 2009.

[38] A. Gerevini, A. Saetti, and M. Vallati. An automatically configurable portfolio-based planner with macro-actions: Pbp. In *ICAPS*, 2009.

[39] P. Ghosh, A. Sharma, P. Chakrabarti, and P. Dasgupta. Algorithms for generating ordered solutions for explicit AND/OR structures. *Journal of Artificial Intelligence (JAIR)*, 44(1):275–333, 2012.

[40] A. Globerson and T. Jaakkola. Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. *Advances in Neural Information Processing Systems*, 21(1.6), 2007.

[41] V. G. Gogate. *Sampling Algorithms for Probabilistic Graphical Models with Determinism DISSERTATION*. PhD thesis, University of California, Irvine, 2009.

[42] H. Hamacher and M. Queyranne. K best solutions to combinatorial optimization problems. *Annals of Operations Research*, 4(1):123–143, 1985.

[43] E. Hansen and R. Zhou. Anytime heuristic search. *Journal of Artificial Intelligence Research*, 28(1):267–297, 2007.

[44] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[45] C. Hildreth. A quadratic programming procedure. *Naval research logistics quarterly*, 4(1):79–85, 1957.

[46] R. A. Howard and J. E. Matheson. Influence diagrams. *Decision Analysis*, 2(3):127–143, 2005.

[47] F. Hutter, H. H. Hoos, and T. Stützle. Efficient stochastic local search for MPE solving. In *International Joint Conference on Artificial Intelligence*, pages 169–174, 2005.

[48] A. T. Ihler, N. Flerova, R. Dechter, and L. Otten. Join-graph based cost-shifting schemes. *arXiv preprint arXiv:1210.4878*, 2012.

[49] V. Jojic, S. Gould, and D. Koller. Accelerated dual decomposition for MAP inference. In *ICML*, 2010.

[50] K. Kask and R. Dechter. Branch and bound with mini-bucket heuristics. In *IJCAI*, volume 99, pages 426–433, 1999.

[51] K. Kask and R. Dechter. Mini-bucket heuristics for improved search. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 314–323. Morgan Kaufmann Publishers Inc., 1999.

[52] K. Kask and R. Dechter. Stochastic local search for bayesian networks. *AI and Statistics*, 1999.

[53] K. Kask and R. Dechter. A general scheme for automatic search heuristics from specification dependencies. *Artificial Intelligence*, pages 91–131, 2001.

[54] K. Kask, R. Dechter, J. Larrosa, and A. Dechter. Unifying cluster-tree decompositions for automated reasoning. *Submitted to the AIJ*, 2003.

[55] U. Kjærulff. Triangulation of graphs–algorithms giving small total state space. *Tech. Report R-90-09*, 1990.

[56] J. Kohlas and N. Wilson. Semiring induced valuation algebras: Exact and approximate local computation algorithms. *Artif. Intell.*, 172(11):1360–1399, 2008.

[57] N. Komodakis and N. Paragios. Beyond loose LP-relaxations: Optimizing MRFs by repairing cycles. *Computer Vision–ECCV 2008*, pages 806–820, 2008.

[58] N. Komodakis, N. Paragios, and G. Tziritas. MRF optimization via dual decomposition: Message-passing revisited. In *ICCV*, Rio de Janeiro, Brazil, Oct. 2007.

[59] V. Kovalevsky and V. Koval. A diffusion algorithm for decreasing energy of max-sum labeling problem. *Unpublished, approx*, 1975.

[60] J. Kupiec. Robust part-of-speech tagging using a hidden markov model. *Computer Speech & Language*, 6(3):225–242, 1992.

[61] E. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18(7):401–405, 1972.

[62] E. Lawler and D. Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

[63] C. Lecoutre, O. Roussel, and D. E. Dehani. WCSP integration of soft neighborhood substitutability. In *Principles and Practice of Constraint Programming*, pages 406–421. Springer, 2012.

[64] M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. *NIPS*, 16, 2003.

[65] R. Marinescu. *AND/OR Search Strategies for Combinatorial Optimization in Graphical Models DISSERTATION*. PhD thesis, Citeseer, 2008.

[66] R. Marinescu and R. Dechter. AND/OR branch-and-bound for graphical models. In *International Joint Conference on Artificial Intelligence*, volume 19, page 224. Lawrence Erlbaum Associates Ltd., 2005.

[67] R. Marinescu and R. Dechter. Best-first AND/OR search for graphical models. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 1171. Citeseer, 2007.

[68] R. Marinescu and R. Dechter. AND/OR Branch-and-Bound search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16-17):1457–1491, 2009.

[69] R. Marinescu and R. Dechter. Memory intensive AND/OR search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16-17):1492–1524, 2009.

[70] R. Mateescu, K. Kask, V. Gogate, and R. Dechter. Join-graph propagation algorithms. *J. Artif. Intell. Res. (JAIR)*, 37:279–328, 2010.

[71] B. Neveu, G. Trombettoni, and F. Glover. Id walk: A candidate list strategy with a simple diversification device. *Principles and Practice of Constraint Programming–CP*, 3258:423–437, 2004.

[72] N. J. Nillson. *Principles of Artificial Intelligence*. Tioga, Palo Alto, CA, 1980.

[73] D. Nilsson. An efficient algorithm for finding the M most probable configurations in probabilistic expert systems. *Statistics and Computing*, 8(2):159–173, 1998.

[74] N. Nilsson. *Principles of artificial intelligence*. Springer Verlag, 1982.

[75] L. Otten and R. Dechter. Toward parallel search for optimization in graphical models. In *ISAIM*, 2010.

[76] L. Otten and R. Dechter. Anytime AND/OR depth first search for combinatorial optimization. In *SOCS*, 2011.

[77] J. Pearl. *Heuristics: Intelligent Search Strategies.* Addison-Wesley, 1984.

[78] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Morgan Kaufmann, 1988.

[79] I. Pohl. Heuristic search viewed as path finding in a graph. *Artif. Intell.*, 1(3-4):193–204, 1970.

[80] S. Richter, J. Thayer, and W. Ruml. The joy of forgetting: Faster anytime search via restarting. In *ICAPS*, pages 137–144, 2010.

[81] E. Rollon and R. Dechter. New mini-bucket partitioning heuristics for bounding the probability of evidence. In *Proceedings of the 24th National Conference on Artificial Intelligence (AAAI2010)*, pages 1199–1204, 2010.

[82] E. Rollon and J. Larrosa. Mini-bucket elimination with bucket propagation. *Principles and Practice of Constraint Programming-CP 2006*, pages 484–498, 2006.

[83] T. Schiex. Arc consistency for soft constraints. *International Conference on Principles and Practice of Constraint Programming (CP)*, pages 411–424, 2000.

[84] B. Seroussi and J. Golmard. An algorithm directly finding the K most probable configurations in Bayesian networks. *International Journal of Approximate Reasoning*, 11(3):205–233, 1994.

[85] G. R. Shafer and P. Shenoy. Probability propagation. *Annals of Mathematics and Artificial Intelligence*, 2:327–352, 1990.

[86] P. Sharma, N. Flerova, and R. Dechter. Empirical evaluation of weighted heuristic search with advanced mini-bucket heuristics for graphical models. Technical report, University of California Irvine, 2014.

[87] P. Shenoy. Binary join trees for computing marginals in the Shenoy-Shafer architecture. *International Journal of Approximate Reasoning*, 17(2-3):239–263, 1997.

[88] P. Shenoy and G. Shafer. Axioms for probability and belief-function propagation. *Uncertainty in Artificial Intelligence*, 4:169–198, 1990.

[89] D. Sontag, D. K. Choe, and Y. Li. Efficiently searching for frustrated cycles in MAP inference. *arXiv preprint arXiv:1210.4902*, 2012.

[90] D. Sontag, A. Globerson, and T. Jaakkola. Introduction to dual decomposition for inference. *Optimization for Machine Learning*, 1:219–254, 2011.

[91] D. Sontag and T. Jaakkola. Tree block coordinate descent for MAP in graphical models. In *AI & Statistics*, pages 544–551. JMLR: W&CP 5, 2009.

[92] D. Sontag, T. Meltzer, A. Globerson, T. Jaakkola, and Y. Weiss. Tightening lp relaxations for map using message passing. *UAI*, 2008.

[93] E. B. Sudderth, A. Torralba, W. T. Freeman, and A. S. Willsky. Learning hierarchical models of scenes, objects, and parts. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1331–1338. IEEE, 2005.

[94] C. Sutton and A. McCallum. Improved dynamic schedules for belief propagation. In *Proc. UAI*, pages 376–383, 2007.

[95] D. Tarlow, D. Batra, P. Kohli, and V. Kolmogorov. Dynamic tree block coordinate ascent. In *ICML*, pages 113–120, June 2011.

[96] J. Thayer and W. Ruml. Anytime heuristic search: Frameworks and algorithms. In *SOCS*, 2010.

[97] J. Van Den Berg, R. Shah, A. Huang, and K. Goldberg. ANA*: Anytime nonparametric A*. In *Proceedings of Twenty-fifth AAAI Conference on Artificial Intelligence (AAAI-11)*, 2011.

[98] M. Wainwright, T. Jaakkola, and A. Willsky. Map estimation via agreement on trees: message-passing and linear programming. *Information Theory, IEEE Transactions on*, 51(11):3697–3717, 2005.

[99] M. J. Wainwright and M. I. Jordan. Variational inference in graphical models: The view from the marginal polytope. In *Proceedings of the Annual Allerton congerence on communication control and computing*, volume 41, pages 961–971. Citeseer, 2003.

[100] H. Wang and K. Daphne. Subproblem-tree calibration: A unified approach to max-product message passing. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 190–198, 2013.

[101] J. Warga. Minimizing certain convex functions. *Journal of the Society for Industrial & Applied Mathematics*, 11(3):588–593, 1963.

[102] B. Wemmenhove, J. M. Mooij, W. Wiegerinck, M. Leisink, H. J. Kappen, and J. P. Neijt. Inference in the promedas medical expert system. In *Artificial intelligence in medicine*, pages 456–460. Springer, 2007.

[103] C. M. Wilt and W. Ruml. When does weighted A* fail? In *SOCS*, 2012.

[104] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Satzilla: Portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research (JAIR)*, 32:565–606, 2008.

[105] C. Yanover, O. Schueler-Furman, and Y. Weiss. Minimizing and learning energy functions for side-chain prediction. *Journal of Computational Biology*, 15(7):899–911, 2008.

[106] C. Yanover and Y. Weiss. Finding the M Most Probable Configurations Using Loopy Belief Propagation. In *Advances in Neural Information Processing Systems 16*. The MIT Press, 2004.

[107] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Constructing free energy approximations and generalized belief propagation algorithms. Technical Report 2004-040, MERL, May 2004.

Appendices

A Bucket Elimination for M-best Optimization Task: Additional Proofs

A.1 Theorem 2.2

Theorem 2.2. *The valuation structure* $(\boldsymbol{A}^m, \otimes^m, sort^m)$ *is a semiring.*

Proof. Let S, T, R be arbitrary elements of $\mathbf{A}^m$. We prove one by one the required conditions. 1. commutativity of $\otimes^m$. By definition, $S \otimes^m T = Sorted^m\{a \otimes b \mid a \in S, b \in T\}$. Since $\otimes$ is commutative, the previous expression is equal to $Sorted^m\{b \otimes a \mid b \in T, a \in S\} = T \otimes^m S$.

2. associativity of $\otimes^m$. We have to prove that $(S \otimes^m T) \otimes^m R = S \otimes^m (T \otimes^m R)$. Suppose that the previous equality does not hold. Then, it would imply that:

i. there may exist an element $a \in (S \otimes^m T) \otimes^m R$, s.t. $a \notin S \otimes^m (T \otimes^m R)$; or

ii. there may exist an element $a \in S \otimes^m (T \otimes^m R)$, s.t. $a \notin (S \otimes^m T) \otimes^m R$.

We show that both cases are impossible.

Consider the first case. Let $\{a_1, \ldots, a_m\} = S \otimes^m (T \otimes^m R)$ where $\forall_{1 \leq i < m}, a_i > a_m$. Since $a \notin S \otimes^m (T \otimes^m R)$, it means that $a_m > a$. Element a comes from the combination of three elements $a = (s \otimes t) \otimes r$. Each element a_i comes from the combination of three elements $a_i = s_{a_i} \otimes (t_{a_i} \otimes r_{a_i})$. By associativity of operator $\otimes$, $a_i = (s_{a_i} \otimes t_{a_i}) \otimes r_{a_i}$. Then,

- If $\forall_{1 \leq i \leq m}, s_{a_i} \otimes t_{a_i} \in S \otimes^m T$, then $(s_{a_i} \otimes t_{a_i}) \otimes r_{a_i} > (s \otimes t) \otimes r$ for all $1 \leq i \leq m$, and

$a \notin (S \otimes^m T) \otimes^m R$, which contradicts the hypothesis.

- If $\exists_{1 \leq j \leq m}, s_{a_j} \otimes t_{a_j} \notin S \otimes^m T$, then there exists an element $s' \otimes t' > s_{a_j} \otimes t_{a_j}$. By monotonicity of $>$, $(s' \otimes t') \otimes r_{a_j} > (s_{a_j} \otimes t_{a_j}) \otimes r_{a_j}$. As a consequence $(s_{a_m} \otimes t_{a_m}) \otimes r_{a_m} \notin (S \otimes^m T) \otimes^m R$. Since $a_m > a$, then $a \notin (S \otimes^m T) \otimes^m R$, which contradicts the hypothesis.

The proof for the second case is the same as above, but interchanging the role of a and $\{a_1, \ldots, a_m\}$, and S and R.

3. commutativity of $sort^m$. By definition, $sort^m\{S, T\} = Sorted^m\{S \cup T\}$. Since set union is commutative, $Sorted^m\{S \cup T\} = Sorted^m\{T \cup S\}$ which is by definition $sort^m\{T, S\}$.

4. associativity of $sort^m$. By definition, $sort^m\{sort^m\{S, T\}, R\} = Sorted^m\{Sorted^m\{S \cup T\} \cup R\}$, and $sort^m\{S, sort^m\{T, R\}\} = Sorted^m\{S \cup Sorted^m\{T \cup R\}\}$. Clearly, the two expressions are equivalent to $Sorted^m\{S \cup T \cup R\}$.

5. $\otimes^m$ distributes over $sort^m$. Let us proceed by induction:

   1. Base case. When $m = 1$, by Proposition 2.2, the valuation structure $(\mathbf{A}^m, \otimes^m, sort^m)$ is a semiring and, as a consequence, $\otimes^m$ distributes over $sort^m$.

   2. Inductive step. Up to m, operator $\otimes^m$ distributes over $sort^m$, and let $\{a_1, \ldots, a_m\}$ be its result. We have to prove that $S \otimes^{m+1} (sort^{m+1}\{T, R\}) = sort^{m+1}\{S \otimes^{m+1} T, S \otimes^{m+1} R\}$. By definition of the operators, the result is the same ordered set of elements $\{a_1, \ldots, a_m\}$ plus one element a_{m+1}. Suppose that $\otimes^{m+1}$ does not distribute over $sort^{m+1}$. Then, it would imply that:

      i. Element $a_{m+1} \in S \otimes^{m+1} (sort^{m+1}\{T, R\})$, but $a_{m+1} \notin sort^{m+1}\{S \otimes^{m+1} T, S \otimes^{m+1} R\}$; or,

      ii. Element $a_{m+1} \notin S \otimes^{m+1} (sort^{m+1}\{T, R\})$, but $a_{m+1} \in sort^{m+1}\{S \otimes^{m+1} T, S \otimes^{m+1} R\}$

   We show that both cases are impossible.

Consider the first case. Since $a_{m+1} \notin sort^{m+1}\{S \otimes^{m+1} T, S \otimes^{m+1} R\}$, it means that $\exists a' \in sort^{m+1}\{S \otimes^{m+1} T, S \otimes^{m+1} R\}$ such that $a' > a_{m+1}$. Element a' comes from the combination of two elements $a' = s' \otimes u'$, where $s' \in S$ and $u' \in T$ or $u' \in R$. Then:

- If $u' \in sort^{m+1}\{T, R\}$, then since $a' > a_{m+1}$, by definition of $\otimes^{m+1}$, $a_{m+1} \notin S \otimes^{m+1} (sort^{m+1}\{T, R\})$, which contradicts the hypothesis.
- If $u' \notin sort^{m+1}\{T, R\}$, then $\exists u'' \in sort^{m+1}\{T, R\}$ such that $u'' > u'$. By monotonicity of the order, $u'' \otimes s' > u' \otimes s'$ and, by transitivity, $u'' \otimes s' > a_{m+1}$. By definition of $\otimes^{m+1}$, $a_{m+1} \notin S \otimes^{m+1} (sort^{m+1}\{T, R\})$, which contradicts the hypothesis.

Consider now the second case. Since $a_{m+1} \notin S \otimes^{m+1} (sort^{m+1}\{T, R\})$, it means that $\exists a' \in S \otimes^{m+1} (sort^{m+1}\{T, R\})$ such that $a' > a_{m+1}$. Element a' comes from the combination of two elements $a' = s' \otimes u'$, where $s' \in S$ and $u' \in T$ or $u' \in R$. Then:

- If $u' \in T$:
  - * and $a' \in S \otimes^{m+1} T$. If $a \in S \otimes^{m+1} T$, since $a' > a_{m+1}$ and by definition of $\otimes^{m+1}$, $a_{m+1} \notin sort^{m+1}\{S \otimes^{m+1} T, S \otimes^{m+1} R\}$, which contradicts the hypothesis. If $a \notin S \otimes^{m+1} T$, since $a' > a_{m+1}$ and by definition of $sort^{m+1}$, $a_{m+1} \notin sort^{m+1}\{S \otimes^{m+1} T, S \otimes^{m+1} R\}$, which contradicts the hypothesis.
  - * and $a' \notin S \otimes^{m+1} T$. Then, $\exists a'' \in S \otimes^{m+1} T$ such that $a'' > a'$. By transitivity of the order, $a'' > a_{m+1}$. Then, either by definition of $\otimes^{m+1}$ or by definition of $sort^{m+1}$, $a_{m+1} \notin sort^{m+1}\{S \otimes^{m+1} T, S \otimes^{m+1} R\}$, which contradicts the hypothesis.
- If $u' \in R$. The reasoning is the same as above, but interchanging the role of T and R.

□

■ B Heuristic Search for M-best Task: Additional Proofs

■ B.1 Theorem 3.11

Theorem 3.11 (Complexity of m-AOBF). *The complexity of algorithm m-AOBF traversing either the AND/OR search tree or the context minimal AND/OR search graph is time and space $O(k^{deg^{h-1}})$, where h is the depth of the underlying pseudo-tree, k is the maximum domain size, and deg bounds the degree of the nodes in the pseudo-tree. If the pseudo-tree is balanced (i.e., each internal node has exactly deg child nodes), then the time and space complexity is $O(k^n)$, where n is the number of variables.*

Proof. It is easy to see that the complexity of m-AOBF is dominated by the time and space required to process the OPEN list (lines 3-29 in Algorithm 12). Therefore, in the worst case, m-AOBF needs to go through the entire OPEN list of partial solution subtrees.

We denote by N the number of partial solution subtrees contained by an AND/OR search tree $S_{\mathcal{T}}$ relative to a pseudo-tree $\mathcal{T}$ with depth h. Let k be the maximum domain size, and let deg be the maximum number of children for any node in the pseudo-tree.

We also define a partial solution subtree T' to be a subtree of $S_{\mathcal{T}}$ such that: (1) T' contains the root s of $S_{\mathcal{T}}$; (2) if a non-terminal OR node n is in T', then T' contains exactly one AND child node m of n; (3) if a non-terminal AND node n is in T' then T' contains all OR child nodes $m_1, \ldots, m_p$ of n; (4) a leaf or tip node of T' doesn't have any successors in T'.

Let N_i^{OR} (respectively T_i^{OR}) be the number of (respectively set of) partial solution subtrees whose leaf nodes correspond to the i^{th} level of OR nodes in $S_{\mathcal{T}}$. Similarly, N_i^{AND} (resp. T_i^{AND}) is the number of (respectively set of) partial solution subtrees whose leaf nodes correspond to the i^{th} level of AND nodes in $S_{\mathcal{T}}$. We assume that the i^{th} level of OR nodes in $S_{\mathcal{T}}$ contains the OR nodes labeled by the variables at depth i in the pseudo-tree $\mathcal{T}$, while the i^{th} level of AND nodes in $S_{\mathcal{T}}$ contains the AND nodes whose parents are the OR nodes of the i^{th} level. For example, the first level ($i = 1$) of OR nodes contains the root OR node s of $S_{\mathcal{T}}$ while the first level of AND nodes contains the children of s.

We can estimate the total number of partial solution subtrees as:

$$N = \sum_{i=1}^{h} (N_i^{OR} + N_i^{AND})$$

Let $T' \in T_i^{OR}$ having m OR leaf nodes. Clearly, T' can be extended to at most k^m partial solution subtrees $\{T'' | T'' \text{ extends } T'\}$ because each of the OR leaf nodes in T' can have exactly one AND child node in T'' and an OR node can have at most k AND child nodes in $S_{\mathcal{T}}$. On the other hand, if $T' \in N_i^{AND}$ has m AND leaf nodes, then T' can be extended to a single partial solution subtree T'' such that each of the m AND leaf nodes in T' has at most deg OR child nodes in T''.

Let m_i^{OR} (respectively m_i^{AND}) be the number of leaf nodes for a partial solution subtree $T' \in T_i^{OR}$ (resp. $T' \in T_i^{AND}$). Then, $m_i^{OR} = m_{i-1}^{AND} \cdot deg$ because a leaf AND node in $T' \in T_{i-1}^{AND}$ can have at most deg OR child nodes in $T'' \in T_i^{OR}$, while $m_i^{AND} = m_i^{OR}$ because a leaf OR node in $T' \in T_i^{OR}$ can have exactly one AND child node in $T'' \in T_i^{AND}$. Since $m_1^{OR} = 1$ and $m_1^{AND} = 1$, we have that $m_i^{OR} = m_i^{AND} = deg^{i-1}$.

It follows that $N_i^{OR} = N_{i-1}^{AND}$ and $N_i^{AND} = N_i^{OR} \cdot k^{m_i^{OR}} = N_i^{OR} \cdot k^{deg^{i-1}}$, where $N_1^{OR} = 1$ and $N_1^{AND} = k$, respectively. Therefore, we can write:

$$
\begin{aligned}
N = {} & (1 + k) \\
& + (k + k^{deg+1}) \\
& + (k^{deg+1} + k^{deg^2+deg+1}) \\
& + ... \\
& + (k^{deg^{h-2}+deg^{h-3}+...+1} + k^{deg^{h-1}+deg^{h-2}+...+1}) \\
& \approx O(k^{\frac{deg^h-1}{deg-1}})
\end{aligned}
$$

Thus, the worst-case number of partial solution subtrees that need to be stored in OPEN is $N \approx O(k^{deg^{h-1}})$. Therefore, the time and space complexity of m-AOBF follows as $O(k^{deg^{h-1}})$. When the pseudo-tree $\mathcal{T}$ is balanced, namely each internal node has exactly deg child nodes, the time and space complexity bound is to $O(k^n)$, since $n \approx O(deg^{h-1})$.

□

C Weighted Heuristic Search: Additional Results

C.1 Scatter Diagrams Summaries: Weighted Heuristic Best-First Search

In this section we include additional scatter plots, summarizing the performance of weighted heuristic search schemes. Figure C.1 shows the comparison between wAOBF and wR-AOBF for two time bound and two levels of heuristic strength and, Figures C.2-C.2 show the comparison between wAOBF and BRAOBB for 3 values of i-bounds and 3 time bounds for each instance. For completeness we include also some of the plots previously presented in Chapter 4.

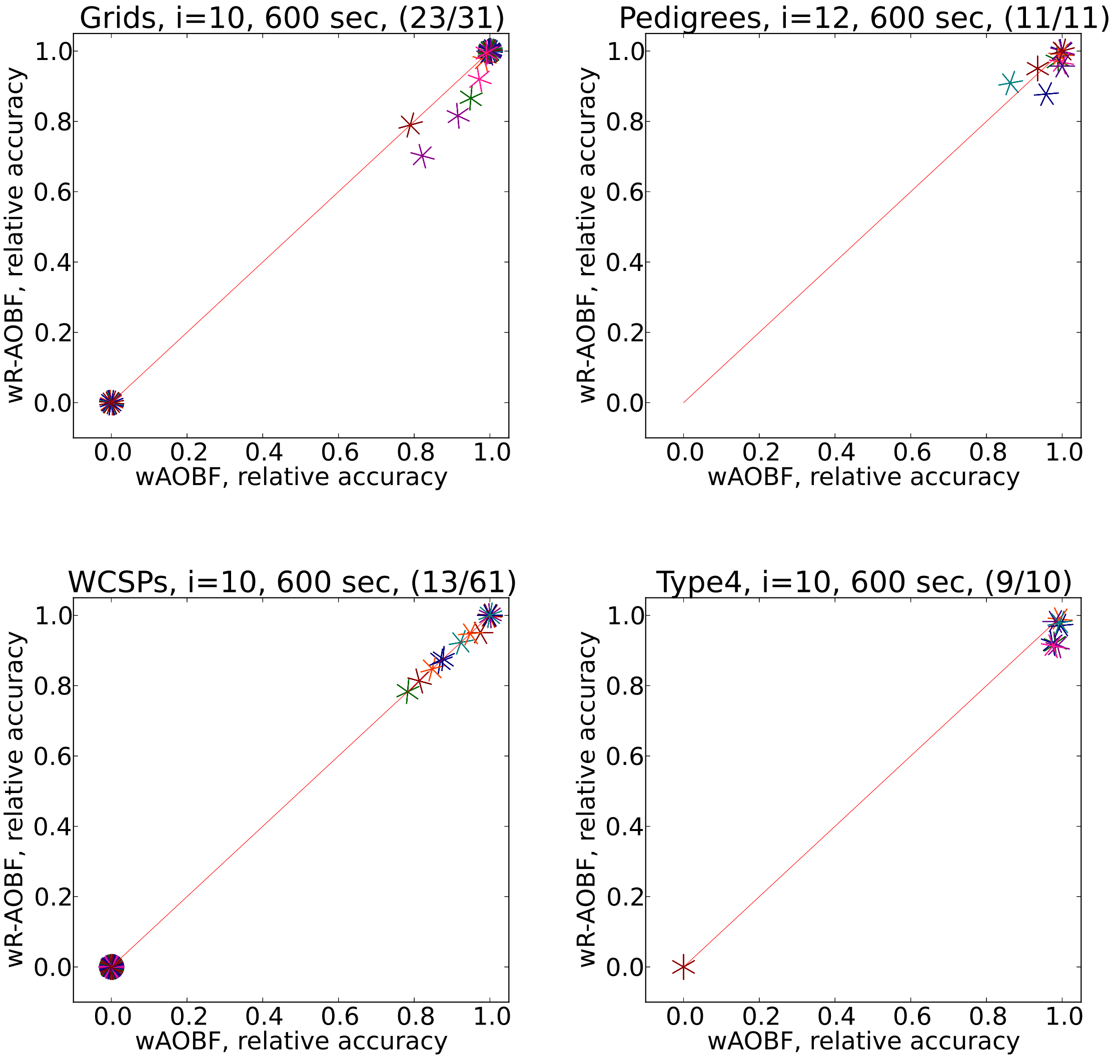


Figure C.1: wAOBF vs wR-AOBF, all benchmarks: comparison of relative accuracy at 600 sec. Each marker represents a single instance. Memory limit 4 GB, time limit 1 hour. In parenthesis (X/Y): X - # instances, for which at least one algorithm found a solution, Y - total # instances, MBE heuristic.

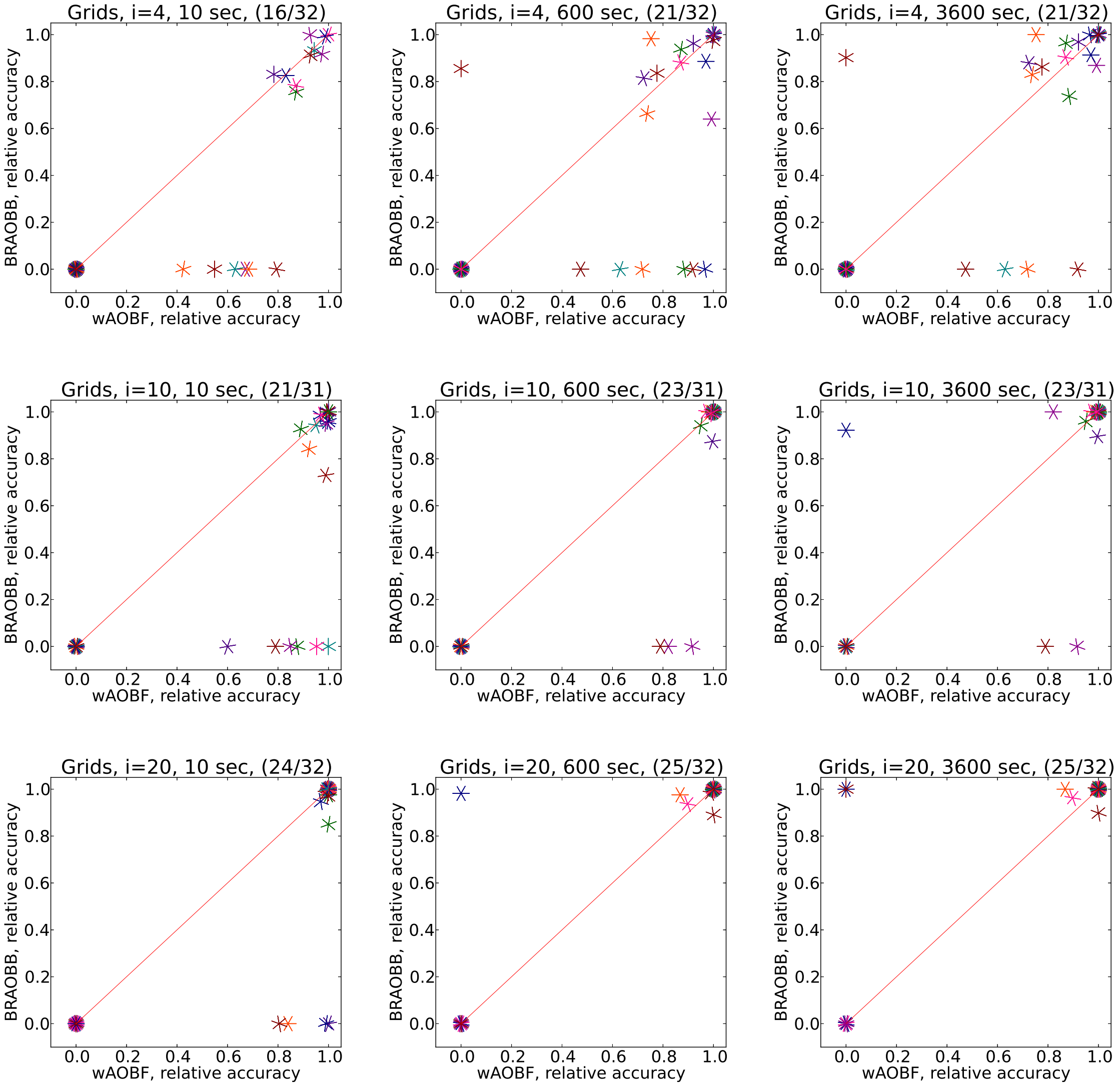


Figure C.2: wAOBF vs BRAOBB on Grids: comparison of relative accuracy at times 10, 600 and 3600 sec. Each row - a single time bound. Each marker represents a single instance. Memory limit 4 GB, time limit 1 hour. In parenthesis (X/Y): X - # instances, for which at least one algorithm found a solution, Y - total # instances, MBE heuristic.

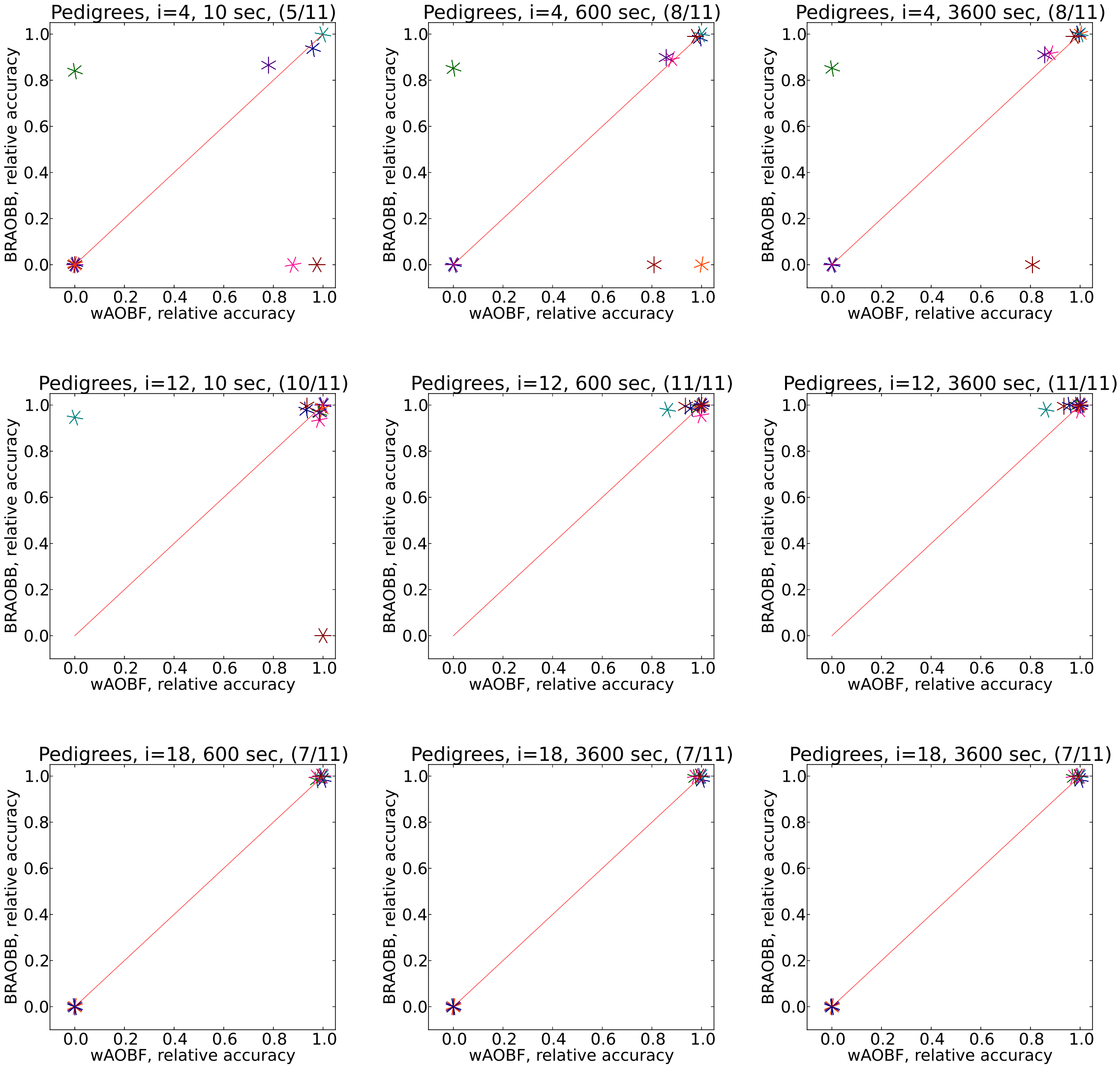
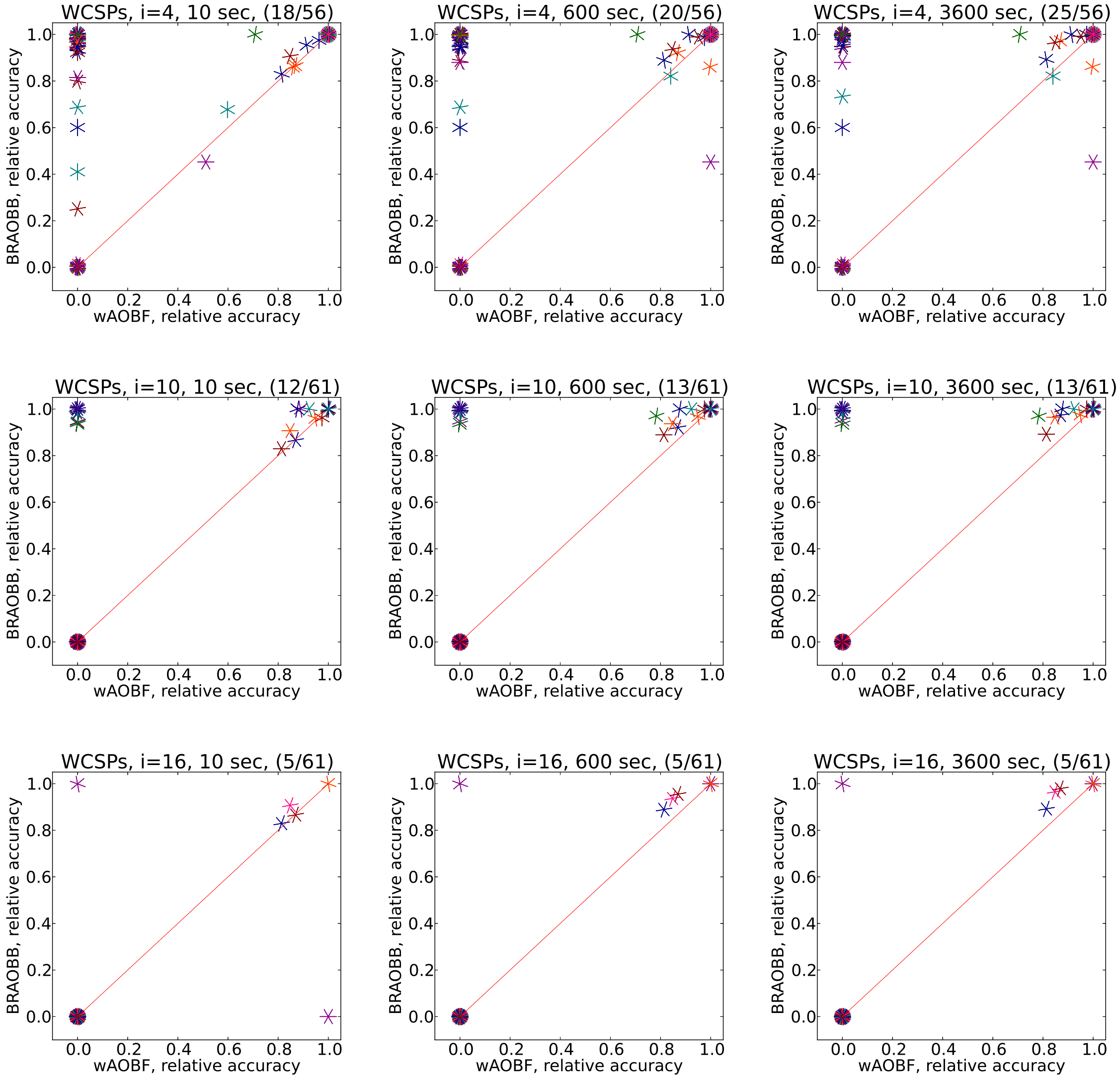


Figure C.3: wAOBF vs BRAOBB on Pedigrees. comparison of relative accuracy at times 10, 600 and 3600 sec. Each row - a single time bound. Each marker represents a single instance. Memory limit 4 GB, time limit 1 hour. In parenthesis (X/Y): X - # instances, for which at least one algorithm found a solution, Y - total # instances, MBE heuristic.


WCSPs, i=4, 10 sec, (18/56)

WCSPs, i=4, 600 sec, (20/56)

WCSPs, i=4, 3600 sec, (25/56)

WCSPs, i=10, 10 sec, (12/61)

WCSPs, i=10, 600 sec, (13/61)

WCSPs, i=10, 3600 sec, (13/61)

WCSPs, i=16, 10 sec, (5/61)

WCSPs, i=16, 600 sec, (5/61)

WCSPs, i=16, 3600 sec, (5/61)

Figure C.4: wAOBF vs BRAOBB on WCSPs: comparison of relative accuracy at times 10, 600 and 3600 sec. Each row - a single time bound. Each marker is a single instance. Memory limit 4 GB, time limit 1 hour. In parenthesis (X/Y): X - # instances, for which at least one algorithm found a solution, Y - total # instances, MBE heuristic.

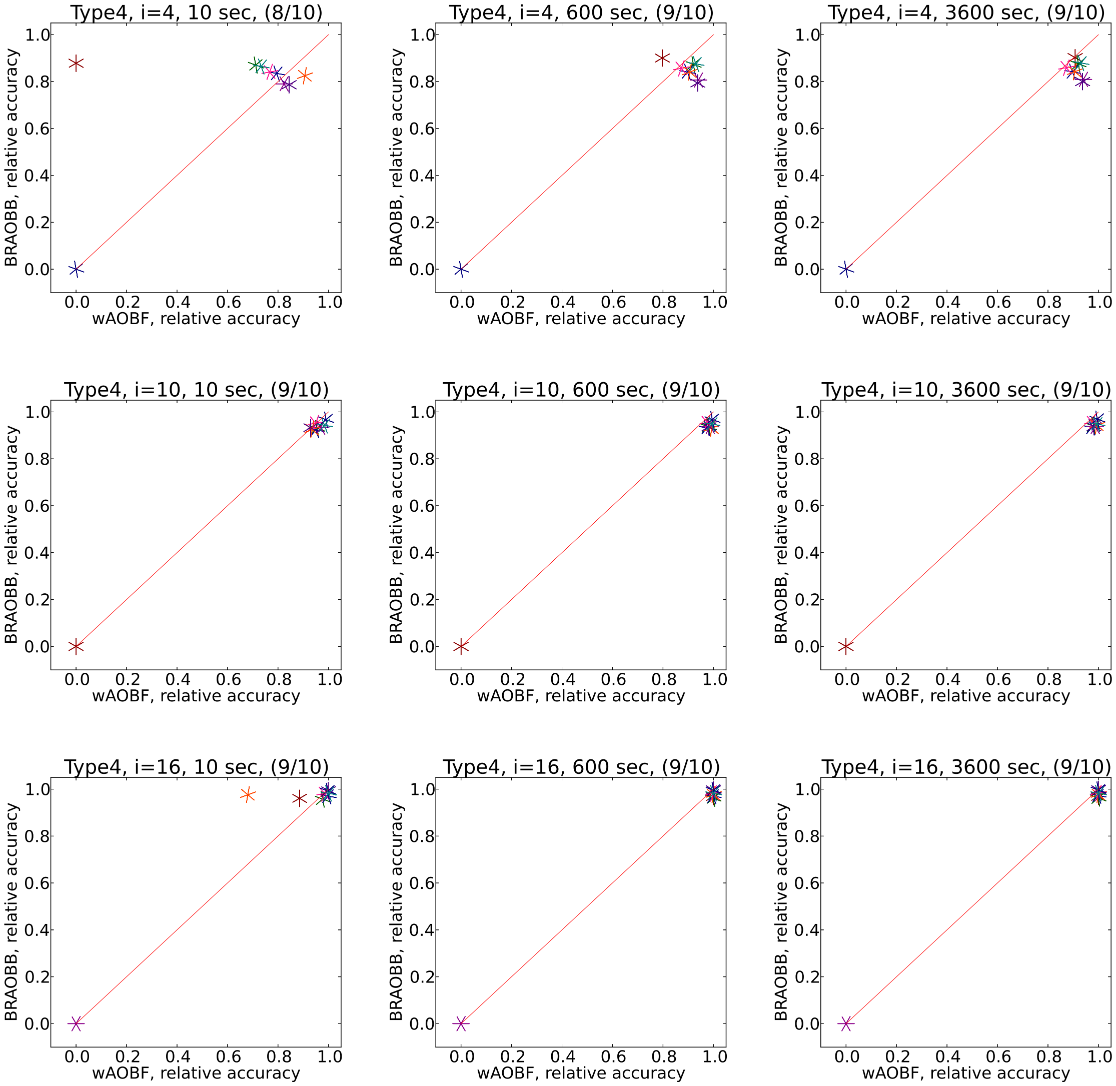


Figure C.5: wAOBF vs BRAOBB on Type4: comparison of relative accuracy at times 10, 600 and 3600 sec. Each row - a single time bound. Each marker is a single instance. Memory limit 4 GB, time limit 1 hour. In parenthesis (X/Y): X - # instances, for which at least one algorithm found a solution, Y - total # instances, MBE heuristic.

I-bound	Algorithm	Time bounds				
		10	30	600	3600	21600
		X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N
Grids (# inst=10, $n = 144 - 2500, k = 2, w^* = 15 - 74, h_T = 48 - 312$						
i=2	wAOBF	56.3 / 6.3 / 16	73.7 / 5.3 / 19	63.2 / 21.1 / 19	47.6 / 19.0 / 21	31.8 / 18.2 / 22
	wR-AOBF	43.8 / 6.3 / 16	42.1 / 5.3 / 19	31.6 / 21.1 / 19	4.8 / 19.0 / 21	9.1 / 18.2 / 22
	wAOBB	18.8 / 112.5 / 16	31.6 / 89.5 / 19	26.3 / 84.2 / 19	38.1 / 71.4 / 21	18.2 / 72.7 / 22
i=4	wAOBF	50.0 / 9.1 / 22	58.3 / 20.8 / 24	37.5 / 29.2 / 24	20.8 / 33.3 / 24	16.0 / 32.0 / 25
	wR-AOBF	27.3 / 9.1 / 22	33.3 / 16.7 / 24	25.0 / 29.2 / 24	8.3 / 29.2 / 24	4.0 / 28.0 / 25
	wAOBB	36.4 / 68.2 / 22	16.7 / 58.3 / 24	8.3 / 79.2 / 24	16.7 / 66.7 / 24	12.0 / 72.0 / 25
i=6	wAOBF	45.8 / 25.0 / 24	46.2 / 46.2 / 26	34.6 / 53.8 / 26	18.5 / 51.9 / 27	10.7 / 50.0 / 28
	wR-AOBF	29.2 / 29.2 / 24	23.1 / 46.2 / 26	23.1 / 53.8 / 26	14.8 / 51.9 / 27	7.1 / 50.0 / 28
	wAOBB	16.7 / 79.2 / 24	15.4 / 76.9 / 26	11.5 / 76.9 / 26	11.1 / 85.2 / 27	7.1 / 85.7 / 28
i=8	wAOBF	40.7 / 25.9 / 27	37.0 / 51.9 / 27	22.2 / 59.3 / 27	11.1 / 63.0 / 27	3.6 / 60.7 / 28
	wR-AOBF	33.3 / 40.7 / 27	25.9 / 51.9 / 27	3.7 / 59.3 / 27	3.7 / 59.3 / 27	3.6 / 57.1 / 28
	wAOBB	29.6 / 59.3 / 27	14.8 / 74.1 / 27	7.4 / 88.9 / 27	11.1 / 96.3 / 27	7.1 / 92.9 / 28
i=10	wAOBF	29.6 / 55.6 / 27	18.5 / 66.7 / 27	14.8 / 81.5 / 27	7.4 / 81.5 / 27	0.0 / 79.3 / 29
	wR-AOBF	18.5 / 59.3 / 27	7.4 / 74.1 / 27	3.7 / 81.5 / 27	0.0 / 81.5 / 27	0.0 / 75.9 / 29
	wAOBB	18.5 / 74.1 / 27	3.7 / 100.0 / 27	0.0 / 111.1 / 27	7.4 / 100.0 / 27	3.4 / 93.1 / 29
i=12	wAOBF	22.7 / 54.5 / 22	4.3 / 78.3 / 23	4.3 / 87.0 / 23	0.0 / 87.5 / 24	0.0 / 91.7 / 24
	wR-AOBF	13.6 / 68.2 / 22	0.0 / 82.6 / 23	0.0 / 87.0 / 23	0.0 / 79.2 / 24	0.0 / 79.2 / 24
	wAOBB	9.1 / 95.5 / 22	0.0 / 95.7 / 23	4.3 / 91.3 / 23	4.2 / 95.8 / 24	0.0 / 104.2 / 24
i=14	wAOBF	10.3 / 75.9 / 29	12.9 / 80.6 / 31	9.4 / 81.3 / 32	9.4 / 84.4 / 32	6.3 / 84.4 / 32
	wR-AOBF	3.4 / 82.8 / 29	6.5 / 80.6 / 31	0.0 / 78.1 / 32	0.0 / 78.1 / 32	0.0 / 78.1 / 32
	wAOBB	3.4 / 93.1 / 29	3.2 / 90.3 / 31	6.3 / 84.4 / 32	0.0 / 87.5 / 32	0.0 / 90.6 / 32
i=18	wAOBF	0.0 / 86.7 / 30	6.5 / 87.1 / 31	9.7 / 87.1 / 31	3.2 / 87.1 / 31	3.2 / 87.1 / 31
	wR-AOBF	10.0 / 90.0 / 30	3.2 / 87.1 / 31	3.2 / 87.1 / 31	0.0 / 87.1 / 31	0.0 / 87.1 / 31
	wAOBB	0.0 / 86.7 / 30	6.5 / 90.3 / 31	6.5 / 93.5 / 31	3.2 / 100.0 / 31	3.2 / 100.0 / 31

Table C.1: X% - percentage of instances for which each algorithm is the better than BRAOBB at a specific time bound, Y% - percentage of instances for which algorithm ties with BRAOBB, N - number of instances for which at least one of algorithms found a solution. # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo-tree height. 4 GB memory, 6 hour time limit, MBE-MM heuristic, Grids.

■ C.2 Impact of Advanced Heuristics on Weighted Heuristic Search

■ C.2.1 MBE-MM heuristic

Tables C.1-C.4 present the summary of relative accuracy of wAOBF, wR-AOBF and wAOBB compared to BRAOBB for several time bounds (10, 30, 600, 3600 and 21600 seconds), using Mini-Bucket Elimination with Max-Marginal-Matching (MBE) heuristic. For each time bound we show X% - percentage of instances, for which each algorithm is the better than BRAOBB at a specific time bound, Y% - percentage of instances, for which algorithm ties with BRAOBB and N - number of instances, for which at least one of algorithms found a solution. The memory limit was 4 GB, time limit 6 hours.

I-bound	Algorithm	Time bounds				
		10	30	600	3600	21600
		X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N
Pedigrees (# inst=16, $n = 298 - 1015, k = 3 - 7, w^* = 15 - 33, h_T = 59 - 140$						
i=2	wAOBF	68.8 / 6.3 / 16	56.3 / 6.3 / 16	62.5 / 6.3 / 16	43.8 / 12.5 / 16	31.3 / 12.5 / 16
	wR-AOBF	25.0 / 6.3 / 16	50.0 / 6.3 / 16	37.5 / 6.3 / 16	25.0 / 6.3 / 16	12.5 / 6.3 / 16
	wAOBB	37.5 / 37.5 / 16	31.3 / 31.3 / 16	31.3 / 31.3 / 16	37.5 / 25.0 / 16	37.5 / 37.5 / 16
i=4	wAOBF	26.7 / 6.7 / 15	31.3 / 25.0 / 16	37.5 / 25.0 / 16	25.0 / 25.0 / 16	18.8 / 25.0 / 16
	wR-AOBF	20.0 / 6.7 / 15	25.0 / 18.8 / 16	25.0 / 18.8 / 16	18.8 / 18.8 / 16	6.3 / 18.8 / 16
	wAOBB	13.3 / 46.7 / 15	6.3 / 62.5 / 16	31.3 / 50.0 / 16	25.0 / 62.5 / 16	12.5 / 62.5 / 16
i=6	wAOBF	18.8 / 18.8 / 16	31.3 / 31.3 / 16	18.8 / 43.8 / 16	31.3 / 43.8 / 16	18.8 / 43.8 / 16
	wR-AOBF	18.8 / 25.0 / 16	25.0 / 37.5 / 16	18.8 / 43.8 / 16	18.8 / 43.8 / 16	12.5 / 43.8 / 16
	wAOBB	25.0 / 43.8 / 16	31.3 / 50.0 / 16	18.8 / 68.8 / 16	18.8 / 75.0 / 16	12.5 / 81.3 / 16
i=8	wAOBF	25.0 / 25.0 / 16	31.3 / 31.3 / 16	25.0 / 37.5 / 16	12.5 / 50.0 / 16	12.5 / 50.0 / 16
	wR-AOBF	12.5 / 25.0 / 16	37.5 / 31.3 / 16	18.8 / 43.8 / 16	12.5 / 43.8 / 16	6.3 / 43.8 / 16
	wAOBB	12.5 / 43.8 / 16	18.8 / 56.3 / 16	25.0 / 62.5 / 16	18.8 / 68.8 / 16	12.5 / 81.3 / 16
i=10	wAOBF	6.3 / 37.5 / 16	12.5 / 62.5 / 16	6.3 / 62.5 / 16	6.3 / 62.5 / 16	6.3 / 62.5 / 16
	wR-AOBF	18.8 / 43.8 / 16	12.5 / 56.3 / 16	6.3 / 56.3 / 16	6.3 / 56.3 / 16	0.0 / 56.3 / 16
	wAOBB	12.5 / 62.5 / 16	6.3 / 75.0 / 16	12.5 / 87.5 / 16	6.3 / 93.8 / 16	0.0 / 93.8 / 16
i=14	wAOBF	14.3 / 50.0 / 14	14.3 / 64.3 / 14	7.1 / 64.3 / 14	0.0 / 64.3 / 14	0.0 / 64.3 / 14
	wR-AOBF	7.1 / 57.1 / 14	0.0 / 64.3 / 14	0.0 / 57.1 / 14	0.0 / 57.1 / 14	0.0 / 57.1 / 14
	wAOBB	0.0 / 92.9 / 14	0.0 / 100.0 / 14	0.0 / 100.0 / 14	0.0 / 92.9 / 14	0.0 / 100.0 / 14
i=18	wAOBF	0.0 / 50.0 / 12	0.0 / 66.7 / 12	0.0 / 66.7 / 12	0.0 / 66.7 / 12	0.0 / 66.7 / 12
	wR-AOBF	0.0 / 58.3 / 12	0.0 / 66.7 / 12	0.0 / 66.7 / 12	0.0 / 66.7 / 12	0.0 / 66.7 / 12
	wAOBB	0.0 / 91.7 / 12	0.0 / 100.0 / 12	0.0 / 100.0 / 12	0.0 / 100.0 / 12	0.0 / 100.0 / 12

Table C.2: X% - percentage of instances for which each algorithm is the better than BRAOBB at a specific time bound, Y% - percentage of instances for which algorithm ties with BRAOBB, N - number of instances for which at least one of algorithms found a solution. # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo-tree height. 4 GB memory, 6 hour time limit, MBE-MM heuristic, Pedigrees.

■ C.2.2 JGLP heuristic

Tables C.5-C.8 show the summary of the result for wAOBF, wR-AOBF and wAOBB using Join-Graph Linear Programming (JGLP) heuristic in the same for as used in Subsection C.2.1.

I-bound	Algorithm	Time bounds				
		10	30	600	3600	21600
		X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N
WCSPs (# inst=8, $n = 100 - 665, k = 2 - 3, w^* = 19 - 89, h_T = 45 - 287$						
i=2	wAOBF	16.7 / 0.0 / 6	0.0 / 12.5 / 8	0.0 / 12.5 / 8	0.0 / 25.0 / 8	0.0 / 25.0 / 8
	wR-AOBF	0.0 / 0.0 / 6	0.0 / 25.0 / 8	0.0 / 25.0 / 8	0.0 / 25.0 / 8	0.0 / 25.0 / 8
	wAOBB	33.3 / 16.7 / 6	12.5 / 25.0 / 8	0.0 / 25.0 / 8	0.0 / 25.0 / 8	25.0 / 37.5 / 8
i=4	wAOBF	33.3 / 16.7 / 6	16.7 / 16.7 / 6	0.0 / 33.3 / 6	16.7 / 33.3 / 6	16.7 / 33.3 / 6
	wR-AOBF	33.3 / 16.7 / 6	16.7 / 33.3 / 6	16.7 / 33.3 / 6	16.7 / 33.3 / 6	16.7 / 33.3 / 6
	wAOBB	33.3 / 33.3 / 6	16.7 / 33.3 / 6	0.0 / 33.3 / 6	0.0 / 50.0 / 6	0.0 / 50.0 / 6
i=6	wAOBF	0.0 / 20.0 / 5	20.0 / 20.0 / 5	0.0 / 20.0 / 5	0.0 / 20.0 / 5	0.0 / 20.0 / 5
	wR-AOBF	20.0 / 20.0 / 5	20.0 / 20.0 / 5	0.0 / 20.0 / 5	0.0 / 20.0 / 5	0.0 / 20.0 / 5
	wAOBB	0.0 / 40.0 / 5	0.0 / 40.0 / 5	0.0 / 40.0 / 5	0.0 / 40.0 / 5	0.0 / 40.0 / 5
i=8	wAOBF	0.0 / 25.0 / 4	0.0 / 25.0 / 4	25.0 / 25.0 / 4	25.0 / 25.0 / 4	25.0 / 25.0 / 4
	wR-AOBF	25.0 / 25.0 / 4	25.0 / 25.0 / 4	25.0 / 25.0 / 4	25.0 / 25.0 / 4	25.0 / 25.0 / 4
	wAOBB	25.0 / 25.0 / 4	25.0 / 25.0 / 4	25.0 / 25.0 / 4	25.0 / 25.0 / 4	25.0 / 25.0 / 4
i=10	wAOBF	100.0 / 0.0 / 1	0.0 / 50.0 / 2	0.0 / 33.3 / 3	0.0 / 33.3 / 3	0.0 / 33.3 / 3
	wR-AOBF	100.0 / 0.0 / 1	0.0 / 50.0 / 2	0.0 / 33.3 / 3	0.0 / 33.3 / 3	0.0 / 33.3 / 3
	wAOBB	100.0 / 0.0 / 1	0.0 / 100.0 / 2	0.0 / 33.3 / 3	0.0 / 33.3 / 3	0.0 / 33.3 / 3
i=14	wAOBF	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1
	wR-AOBF	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1
	wAOBB	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1
i=18	wAOBF	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1
	wR-AOBF	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1
	wAOBB	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1

Table C.3: X% - percentage of instances for which each algorithm is the better than BRAOBB at a specific time bound, Y% - percentage of instances for which algorithm ties with BRAOBB, N - number of instances for which at least one of algorithms found a solution. # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo-tree height. 4 GB memory, 6 hour time limit, MBE-MM heuristic, WCSP.

I-bound	Algorithm	Time bounds				
		10	30	600	3600	21600
		X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N
Type4 (# inst=10, $n = 3938 - 8186, k = 5, w^* = 24 - 32, h_T = 319 - 625$						
i=2	wAOBF	100.0 / 0.0 / 1	66.7 / 0.0 / 3	66.7 / 0.0 / 3	66.7 / 0.0 / 3	100.0 / 0.0 / 3
	wR-AOBF	0.0 / 0.0 / 1	33.3 / 0.0 / 3	33.3 / 0.0 / 3	66.7 / 0.0 / 3	66.7 / 0.0 / 3
	wAOBB	0.0 / 400.0 / 1	33.3 / 66.7 / 3	33.3 / 133.3 / 3	33.3 / 100.0 / 3	33.3 / 166.7 / 3
i=4	wAOBF	100.0 / 0.0 / 1	75.0 / 0.0 / 4	83.3 / 0.0 / 6	50.0 / 0.0 / 6	100.0 / 0.0 / 6
	wR-AOBF	0.0 / 0.0 / 1	25.0 / 0.0 / 4	33.3 / 0.0 / 6	50.0 / 0.0 / 6	50.0 / 0.0 / 6
	wAOBB	100.0 / 200.0 / 1	25.0 / 75.0 / 4	33.3 / 33.3 / 6	16.7 / 33.3 / 6	33.3 / 33.3 / 6
i=6	wAOBF	0.0 / 0.0 / 1	75.0 / 0.0 / 4	60.0 / 0.0 / 5	80.0 / 0.0 / 5	80.0 / 0.0 / 5
	wR-AOBF	0.0 / 0.0 / 1	0.0 / 0.0 / 4	0.0 / 0.0 / 5	0.0 / 0.0 / 5	20.0 / 0.0 / 5
	wAOBB	0.0 / 200.0 / 1	25.0 / 50.0 / 4	20.0 / 20.0 / 5	60.0 / 0.0 / 5	20.0 / 20.0 / 5
i=8	wAOBF	66.7 / 0.0 / 3	50.0 / 0.0 / 6	33.3 / 0.0 / 6	100.0 / 0.0 / 6	100.0 / 0.0 / 6
	wR-AOBF	33.3 / 0.0 / 3	33.3 / 0.0 / 6	66.7 / 0.0 / 6	66.7 / 0.0 / 6	66.7 / 0.0 / 6
	wAOBB	33.3 / 66.7 / 3	50.0 / 0.0 / 6	16.7 / 0.0 / 6	16.7 / 0.0 / 6	50.0 / 0.0 / 6
i=10	wAOBF	25.0 / 0.0 / 8	11.1 / 0.0 / 9	70.0 / 0.0 / 10	80.0 / 0.0 / 10	90.0 / 0.0 / 10
	wR-AOBF	12.5 / 0.0 / 8	33.3 / 0.0 / 9	20.0 / 0.0 / 10	20.0 / 0.0 / 10	20.0 / 0.0 / 10
	wAOBB	25.0 / 0.0 / 8	11.1 / 11.1 / 9	30.0 / 0.0 / 10	20.0 / 0.0 / 10	30.0 / 0.0 / 10
i=12	wAOBF	60.0 / 0.0 / 5	16.7 / 0.0 / 6	66.7 / 0.0 / 6	83.3 / 0.0 / 6	83.3 / 0.0 / 6
	wR-AOBF	20.0 / 0.0 / 5	16.7 / 0.0 / 6	16.7 / 0.0 / 6	16.7 / 0.0 / 6	16.7 / 0.0 / 6
	wAOBB	20.0 / 20.0 / 5	33.3 / 0.0 / 6	16.7 / 0.0 / 6	16.7 / 0.0 / 6	16.7 / 16.7 / 6
i=14	wAOBF	22.2 / 0.0 / 9	30.0 / 0.0 / 10	90.0 / 0.0 / 10	80.0 / 0.0 / 10	70.0 / 0.0 / 10
	wR-AOBF	0.0 / 0.0 / 9	0.0 / 0.0 / 10	0.0 / 0.0 / 10	0.0 / 0.0 / 10	0.0 / 0.0 / 10
	wAOBB	11.1 / 11.1 / 9	10.0 / 10.0 / 10	0.0 / 30.0 / 10	0.0 / 30.0 / 10	0.0 / 30.0 / 10
i=18	wAOBF	0.0 / 0.0 / 4	0.0 / 11.1 / 9	33.3 / 22.2 / 9	44.4 / 22.2 / 9	44.4 / 22.2 / 9
	wR-AOBF	0.0 / 25.0 / 4	0.0 / 22.2 / 9	0.0 / 22.2 / 9	0.0 / 22.2 / 9	0.0 / 22.2 / 9
	wAOBB	0.0 / 0.0 / 4	0.0 / 55.6 / 9	0.0 / 66.7 / 9	0.0 / 66.7 / 9	22.2 / 55.6 / 9

Table C.4: X% - percentage of instances for which each algorithm is the better than BRAOBB at a specific time bound, Y% - percentage of instances for which algorithm ties with BRAOBB, N - number of instances for which at least one of algorithms found a solution. # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo-tree height. 4 GB memory, 6 hour time limit, MBE-MM heuristic, Type4.

I-bound	Algorithm	Time bounds				
		10	30	600	3600	21600
		X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N
Grids (# inst=10, $n = 144 - 2500, k = 2, w^* = 15 - 74, h_T = 48 - 312$						
i=2	wAOBF	0 / 0 / 0	57.1 / 23.8 / 21	59.1 / 31.8 / 22	30.4 / 30.4 / 23	16.7 / 33.3 / 24
	wR-AOBF	0 / 0 / 0	38.1 / 19.0 / 21	31.8 / 22.7 / 22	13.0 / 26.1 / 23	8.3 / 25.0 / 24
	wAOBB	0 / 0 / 0	19.0 / 76.2 / 21	31.8 / 77.3 / 22	39.1 / 73.9 / 23	25.0 / 70.8 / 24
i=4	wAOBF	0 / 0 / 0	18.5 / 40.7 / 27	13.8 / 48.3 / 29	13.8 / 58.6 / 29	6.9 / 62.1 / 29
	wR-AOBF	0 / 0 / 0	25.9 / 37.0 / 27	17.2 / 48.3 / 29	6.9 / 55.2 / 29	6.9 / 55.2 / 29
	wAOBB	0 / 0 / 0	11.1 / 63.0 / 27	10.3 / 65.5 / 29	13.8 / 72.4 / 29	3.4 / 89.7 / 29
i=6	wAOBF	0 / 0 / 0	3.2 / 77.4 / 31	0.0 / 87.1 / 31	0.0 / 90.3 / 31	0.0 / 90.3 / 31
	wR-AOBF	0 / 0 / 0	6.5 / 87.1 / 31	3.2 / 87.1 / 31	0.0 / 90.3 / 31	0.0 / 90.3 / 31
	wAOBB	0 / 0 / 0	3.2 / 80.6 / 31	0.0 / 96.8 / 31	0.0 / 100.0 / 31	0.0 / 103.2 / 31
i=8	wAOBF	0 / 0 / 0	6.5 / 90.3 / 31	3.2 / 90.3 / 31	3.2 / 93.5 / 31	0.0 / 93.5 / 31
	wR-AOBF	0 / 0 / 0	3.2 / 90.3 / 31	3.2 / 93.5 / 31	3.2 / 93.5 / 31	0.0 / 93.5 / 31
	wAOBB	0 / 0 / 0	0.0 / 93.5 / 31	0.0 / 100.0 / 31	6.5 / 93.5 / 31	0.0 / 103.2 / 31
i=10	wAOBF	0 / 0 / 0	3.2 / 93.5 / 31	0.0 / 96.8 / 31	0.0 / 96.8 / 31	0.0 / 96.8 / 31
	wR-AOBF	0 / 0 / 0	0.0 / 96.8 / 31	0.0 / 96.8 / 31	0.0 / 96.8 / 31	0.0 / 96.8 / 31
	wAOBB	0 / 0 / 0	3.2 / 100.0 / 31	0.0 / 100.0 / 31	0.0 / 100.0 / 31	0.0 / 103.2 / 31
i=12	wAOBF	0 / 0 / 0	11.1 / 81.5 / 27	3.7 / 85.2 / 27	0.0 / 88.9 / 27	0.0 / 88.9 / 27
	wR-AOBF	0 / 0 / 0	7.4 / 85.2 / 27	3.7 / 88.9 / 27	0.0 / 88.9 / 27	0.0 / 88.9 / 27
	wAOBB	0 / 0 / 0	3.7 / 88.9 / 27	7.4 / 85.2 / 27	3.7 / 92.6 / 27	3.7 / 96.3 / 27
i=14	wAOBF	0 / 0 / 0	3.1 / 84.4 / 32	3.1 / 87.5 / 32	3.1 / 90.6 / 32	3.1 / 90.6 / 32
	wR-AOBF	0 / 0 / 0	9.4 / 87.5 / 32	3.1 / 90.6 / 32	3.1 / 90.6 / 32	3.1 / 90.6 / 32
	wAOBB	0 / 0 / 0	0.0 / 87.5 / 32	3.1 / 90.6 / 32	3.1 / 96.9 / 32	3.1 / 96.9 / 32
i=18	wAOBF	0 / 0 / 0	0.0 / 89.3 / 28	0.0 / 90.0 / 30	3.3 / 93.3 / 30	0.0 / 93.3 / 30
	wR-AOBF	0 / 0 / 0	3.6 / 96.4 / 28	6.7 / 93.3 / 30	3.3 / 93.3 / 30	0.0 / 93.3 / 30
	wAOBB	0 / 0 / 0	3.6 / 92.9 / 28	0.0 / 86.7 / 30	0.0 / 93.3 / 30	0.0 / 100.0 / 30

Table C.5: X% - percentage of instances for which each algorithm is the better than BRAOBB at a specific time bound, Y% - percentage of instances for which algorithm ties with BRAOBB, N - number of instances for which at least one of algorithms found a solution. # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo-tree height. 4 GB memory, 6 hour time limit, JGLP heuristic, Grids.

I-bound	Algorithm	Time bounds				
		10	30	600	3600	21600
		X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N
Pedigrees (# inst=16, $n = 298 - 1015, k = 3 - 7, w^* = 15 - 33, h_T = 59 - 140$						
i=2	wAOBF	0 / 0 / 0	43.8 / 6.3 / 16	31.3 / 25.0 / 16	37.5 / 25.0 / 16	25.0 / 25.0 / 16
	wR-AOBF	0 / 0 / 0	31.3 / 18.8 / 16	25.0 / 31.3 / 16	25.0 / 31.3 / 16	12.5 / 31.3 / 16
	wAOBB	0 / 0 / 0	56.3 / 31.3 / 16	12.5 / 62.5 / 16	18.8 / 75.0 / 16	25.0 / 68.8 / 16
i=4	wAOBF	0 / 0 / 0	31.3 / 18.8 / 16	12.5 / 37.5 / 16	25.0 / 37.5 / 16	6.3 / 37.5 / 16
	wR-AOBF	0 / 0 / 0	37.5 / 25.0 / 16	12.5 / 37.5 / 16	12.5 / 37.5 / 16	0.0 / 37.5 / 16
	wAOBB	0 / 0 / 0	18.8 / 62.5 / 16	25.0 / 62.5 / 16	25.0 / 75.0 / 16	6.3 / 93.8 / 16
i=6	wAOBF	0 / 0 / 0	6.3 / 50.0 / 16	12.5 / 56.3 / 16	0.0 / 56.3 / 16	0.0 / 56.3 / 16
	wR-AOBF	0 / 0 / 0	6.3 / 43.8 / 16	6.3 / 50.0 / 16	0.0 / 50.0 / 16	0.0 / 50.0 / 16
	wAOBB	0 / 0 / 0	12.5 / 68.8 / 16	12.5 / 87.5 / 16	6.3 / 93.8 / 16	0.0 / 100.0 / 16
i=8	wAOBF	0 / 0 / 0	0.0 / 46.7 / 15	6.3 / 50.0 / 16	0.0 / 50.0 / 16	0.0 / 50.0 / 16
	wR-AOBF	0 / 0 / 0	6.7 / 53.3 / 15	6.3 / 50.0 / 16	0.0 / 50.0 / 16	0.0 / 50.0 / 16
	wAOBB	0 / 0 / 0	6.7 / 80.0 / 15	6.3 / 87.5 / 16	0.0 / 87.5 / 16	0.0 / 87.5 / 16
i=10	wAOBF	0 / 0 / 0	6.3 / 50.0 / 16	0.0 / 62.5 / 16	0.0 / 62.5 / 16	0.0 / 62.5 / 16
	wR-AOBF	0 / 0 / 0	0.0 / 56.3 / 16	0.0 / 62.5 / 16	0.0 / 62.5 / 16	0.0 / 62.5 / 16
	wAOBB	0 / 0 / 0	0.0 / 81.3 / 16	0.0 / 81.3 / 16	0.0 / 87.5 / 16	0.0 / 100.0 / 16
i=12	wAOBF	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
	wR-AOBF	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
	wAOBB	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
i=14	wAOBF	0 / 0 / 0	7.1 / 57.1 / 14	0.0 / 57.1 / 14	0.0 / 57.1 / 14	0.0 / 57.1 / 14
	wR-AOBF	0 / 0 / 0	0.0 / 57.1 / 14	0.0 / 57.1 / 14	0.0 / 57.1 / 14	0.0 / 57.1 / 14
	wAOBB	0 / 0 / 0	0.0 / 92.9 / 14	0.0 / 92.9 / 14	0.0 / 92.9 / 14	0.0 / 100.0 / 14
i=18	wAOBF	0 / 0 / 0	0.0 / 75.0 / 8	0.0 / 77.8 / 9	0.0 / 77.8 / 9	0.0 / 77.8 / 9
	wR-AOBF	0 / 0 / 0	0.0 / 75.0 / 8	0.0 / 77.8 / 9	0.0 / 77.8 / 9	0.0 / 77.8 / 9
	wAOBB	0 / 0 / 0	0.0 / 100.0 / 8	0.0 / 100.0 / 9	0.0 / 100.0 / 9	0.0 / 100.0 / 9

Table C.6: X% - percentage of instances for which each algorithm is the better than BRAOBB at a specific time bound, Y% - percentage of instances for which algorithm ties with BRAOBB, N - number of instances for which at least one of algorithms found a solution. # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo-tree height. 4 GB memory, 6 hour time limit, JGLP heuristic, Pedigrees.

I-bound	Algorithm	**Time bounds**				
		10	30	600	3600	21600
		X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N
WCSPs (# inst=8, $n = 100 - 665, k = 2 - 3, w^* = 19 - 89, h_T = 45 - 287$						
i=2	wAOBF	0 / 0 / 0	25.0 / 37.5 / 8	25.0 / 37.5 / 8	25.0 / 37.5 / 8	25.0 / 37.5 / 8
	wR-AOBF	0 / 0 / 0	25.0 / 37.5 / 8	25.0 / 37.5 / 8	25.0 / 37.5 / 8	25.0 / 37.5 / 8
	wAOBB	0 / 0 / 0	25.0 / 50.0 / 8	25.0 / 50.0 / 8	25.0 / 50.0 / 8	37.5 / 50.0 / 8
i=4	wAOBF	0 / 0 / 0	33.3 / 33.3 / 6	33.3 / 33.3 / 6	33.3 / 33.3 / 6	33.3 / 33.3 / 6
	wR-AOBF	0 / 0 / 0	33.3 / 33.3 / 6	33.3 / 33.3 / 6	33.3 / 33.3 / 6	33.3 / 33.3 / 6
	wAOBB	0 / 0 / 0	33.3 / 33.3 / 6	33.3 / 50.0 / 6	33.3 / 50.0 / 6	33.3 / 50.0 / 6
i=6	wAOBF	0 / 0 / 0	25.0 / 25.0 / 4	50.0 / 50.0 / 4	50.0 / 50.0 / 4	25.0 / 50.0 / 4
	wR-AOBF	0 / 0 / 0	50.0 / 25.0 / 4	25.0 / 25.0 / 4	25.0 / 25.0 / 4	0.0 / 25.0 / 4
	wAOBB	0 / 0 / 0	50.0 / 25.0 / 4	25.0 / 50.0 / 4	25.0 / 50.0 / 4	0.0 / 50.0 / 4
i=8	wAOBF	0 / 0 / 0	66.7 / 33.3 / 3	66.7 / 33.3 / 3	66.7 / 33.3 / 3	66.7 / 33.3 / 3
	wR-AOBF	0 / 0 / 0	33.3 / 33.3 / 3	33.3 / 33.3 / 3	66.7 / 33.3 / 3	66.7 / 33.3 / 3
	wAOBB	0 / 0 / 0	33.3 / 33.3 / 3	33.3 / 33.3 / 3	66.7 / 33.3 / 3	66.7 / 33.3 / 3
i=10	wAOBF	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1
	wR-AOBF	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1
	wAOBB	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1
i=14	wAOBF	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1
	wR-AOBF	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1
	wAOBB	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1
i=18	wAOBF	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1
	wR-AOBF	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1
	wAOBB	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1

Table C.7: X% - percentage of instances for which each algorithm is the better than BRAOBB at a specific time bound, Y% - percentage of instances for which algorithm ties with BRAOBB, N - number of instances for which at least one of algorithms found a solution. # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo-tree height. 4 GB memory, 6 hour time limit, JGLP heuristic, WCSP.

I-bound	Algorithm	Time bounds				
		10	30	600	3600	21600
		X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N	X% / Y% / N
Type4 (# inst=10, $n = 3938 - 8186, k = 5, w^* = 24 - 32, h_T = 319 - 625$						
i=2	wAOBF	0 / 0 / 0	66.7 / 0.0 / 3	33.3 / 0.0 / 3	66.7 / 0.0 / 3	66.7 / 0.0 / 3
	wR-AOBF	0 / 0 / 0	33.3 / 0.0 / 3	66.7 / 0.0 / 3	100.0 / 0.0 / 3	100.0 / 0.0 / 3
	wAOBB	0 / 0 / 0	33.3 / 100.0 / 3	33.3 / 133.3 / 3	33.3 / 166.7 / 3	33.3 / 166.7 / 3
i=4	wAOBF	0 / 0 / 0	50.0 / 0.0 / 2	75.0 / 0.0 / 4	80.0 / 0.0 / 5	60.0 / 0.0 / 5
	wR-AOBF	0 / 0 / 0	50.0 / 0.0 / 2	25.0 / 0.0 / 4	20.0 / 0.0 / 5	20.0 / 0.0 / 5
	wAOBB	0 / 0 / 0	0.0 / 100.0 / 2	50.0 / 50.0 / 4	80.0 / 20.0 / 5	60.0 / 0.0 / 5
i=6	wAOBF	0 / 0 / 0	50.0 / 0.0 / 4	50.0 / 0.0 / 6	83.3 / 0.0 / 6	66.7 / 0.0 / 6
	wR-AOBF	0 / 0 / 0	25.0 / 0.0 / 4	50.0 / 0.0 / 6	50.0 / 0.0 / 6	50.0 / 0.0 / 6
	wAOBB	0 / 0 / 0	50.0 / 25.0 / 4	33.3 / 16.7 / 6	66.7 / 33.3 / 6	50.0 / 33.3 / 6
i=8	wAOBF	0 / 0 / 0	60.0 / 0.0 / 5	50.0 / 0.0 / 6	66.7 / 0.0 / 6	83.3 / 0.0 / 6
	wR-AOBF	0 / 0 / 0	0.0 / 0.0 / 5	16.7 / 0.0 / 6	16.7 / 0.0 / 6	16.7 / 0.0 / 6
	wAOBB	0 / 0 / 0	40.0 / 40.0 / 5	33.3 / 0.0 / 6	0.0 / 0.0 / 6	0.0 / 0.0 / 6
i=10	wAOBF	0 / 0 / 0	33.3 / 0.0 / 9	50.0 / 0.0 / 10	60.0 / 0.0 / 10	70.0 / 0.0 / 10
	wR-AOBF	0 / 0 / 0	0.0 / 0.0 / 9	10.0 / 0.0 / 10	10.0 / 0.0 / 10	10.0 / 0.0 / 10
	wAOBB	0 / 0 / 0	33.3 / 0.0 / 9	10.0 / 0.0 / 10	0.0 / 0.0 / 10	10.0 / 10.0 / 10
i=12	wAOBF	0 / 0 / 0	16.7 / 0.0 / 6	66.7 / 0.0 / 6	83.3 / 0.0 / 6	83.3 / 0.0 / 6
	wR-AOBF	0 / 0 / 0	33.3 / 0.0 / 6	33.3 / 0.0 / 6	33.3 / 0.0 / 6	33.3 / 0.0 / 6
	wAOBB	0 / 0 / 0	0.0 / 0.0 / 6	16.7 / 16.7 / 6	16.7 / 16.7 / 6	16.7 / 16.7 / 6
i=14	wAOBF	0 / 0 / 0	0.0 / 0.0 / 8	66.7 / 0.0 / 9	66.7 / 0.0 / 9	66.7 / 0.0 / 9
	wR-AOBF	0 / 0 / 0	0.0 / 0.0 / 8	0.0 / 0.0 / 9	0.0 / 0.0 / 9	0.0 / 0.0 / 9
	wAOBB	0 / 0 / 0	0.0 / 0.0 / 8	0.0 / 33.3 / 9	11.1 / 33.3 / 9	11.1 / 33.3 / 9
i=18	wAOBF	0 / 0 / 0	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1
	wR-AOBF	0 / 0 / 0	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1
	wAOBB	0 / 0 / 0	0 / 0 / 0	0.0 / 100.0 / 1	0.0 / 100.0 / 1	0.0 / 100.0 / 1

Table C.8: X% - percentage of instances for which each algorithm is the better than BRAOBB at a specific time bound, Y% - percentage of instances for which algorithm ties with BRAOBB, N - number of instances for which at least one of algorithms found a solution. # inst - total number of instances in benchmark, n - number of variables, k - maximum domain size, w^* - induced width, h_T - pseudo-tree height. 4 GB memory, 6 hour time limit, JGLP heuristic, Type4.