A Level-Set Approach for Solving Nonlinear Integer Optimization Problems

by

Ryan Killea

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Data Science

by

May 2023

APPROVED:

Professor Andrew Trapp, MS Thesis Advisor

Professor Fabricio Murai, Reader

Professor Elke Rundensteiner, Data Science Program Director

Abstract

This thesis studies new methods to explore and exploit the structure of nonlinear integer optimization problems to create a level set optimal structure for parameterized integer nonlinear optimization problems, conducive to rapid lookup and queries over a wide range of potential resource vectors. The value function is a function that maps resource vectors to their optimal objective function values. General integer nonlinear optimization problems are a class of optimization problems where a nonlinear function is optimized subject to inequality and equality constraints, with all variables being integers. Important subclasses include quadratic integer programs, as well as programs of higher functional order. These problem subclasses are in general NP-hard and so challenging to solve. Certain critical problem contexts, such as discretized model predictive control, disaster response, robust investment optimization, and dynamic energy market allocation, can both be extremely sensitive to time delays and benefit immensely from rapid retrieval of provably optimal results. Further, even in optimization domains where no such urgency exists, it is important from an applied perspective to be able to reason over the maximal values of an optimization problem to understand how it responds both to resource availability as well as to questions of how to best elicit improvements by changing resource levels. Such competing, compelling, demands of optimality and speed call for new methods that allow for fast lookup of provable, pre-computed optima to avoid longer wait times as new resource vector queries arrive.

This thesis contributes a new general solution approach to nonseparable, nonconvex, polynomial integer programs with separable constraints. Storing the value function in a way that allows its efficient maintenance and search during and after the incremental construction is proposed to accelerate the queries during and after construction. The problem is broken down into subproblems that incrementally add to a Minimal R* Tree(MR*T) data structure new level set optimal vectors one at a time, building up and benefiting

from a value function lower bound. New ways of directly comparing the upper bounds at these subproblem nodes to the existing value function lower bound are considered to improve the performance of the approach. The specific optimization routines used for the value function bounds at these subproblems are also tailored to the task of reasoning over a range of right-hand sides (equivalently, resource levels).

Computational experiments explore solving a frequent problem of the applicable class – quadratic knapsack : nonseparable, nonconvex, polynomial knapsack problems with several linear constraints. For the considered problem class, reasoning over very large range of right-hand sides (on the order of hundreds of thousands), we demonstrate the ability to construct an optimal structure in as little as six to 35 conventional, state of the art (Gurobi) solver solves with a single right-hand side. Thus, for problems that are repeatedly solved over varying resource vectors, this promises substantial savings. Furthermore, sensitivity analysis results, unobtainable via conventional solver solves due to the lack of a strong integer dual, are easily recoverable from the optimal structure at a speed of < 2 milliseconds for even complex directional sensitivity queries. This thesis gives a first step in solving a broader context of optimization problems with a value function approach than previously possible.

Contents

1 Introduction **1**

2 Background **4**

3 Mathematical and Computational Preliminaries **9**

3.1 Mathematical Preliminaries 9

3.1.1 Relevant Mathematical Optimization Problem Classes 10

3.1.2 Nondominance, Level Set Optimality, and An Efficient Frontier . . . 11

3.1.3 Ordering via w-Weighting 12

3.2 Algorithmic Preliminaries 13

3.2.1 Overview of Tree Search Algorithm 13

3.2.2 Variable Fixing Order 15

3.2.3 Bounding Logic 16

3.2.4 Data Structures 17

4 Comprehensive Value Function Construction via Tree Search Algorithm 19

4.1 Tree Search Algorithm (TSA): Foundational Elements 19

4.1.1 Tree Search Algorithm: Main Loop 20

4.1.2 Tree Search Algorithm: Pruning 25

4.1.3 Tree Search Algorithm: z^j-Upper Bounds 26

4.1.4 Tree Search Algorithm: Initialization 31

4.1.5 Tree Search Algorithm: Proof of Correctness 32

4.2 Tree Search Algorithm: Performance Enhancements 33
4.2.1 Enforcing Pareto Monotonicity 33
4.2.2 Cuts and $m-$region Bounds Adjustment 34
4.2.3 $m-$region Bounds Adjustment 37
4.2.4 Extending EPM to Pairs of Variables 38
4.2.5 Reusing Computation 39
4.2.6 Algorithmic Description with All Enhancements 41

5 Computational Studies **45**
5.1 Computational Setup 45
5.2 Computational Experiments 45
5.3 Sensitivity Analysis 46
5.4 Problem Instance Generation 47
5.5 Computational Experiments: Test Instance Parameters 48
5.6 Computational Experiments: Performance Metrics 49

6 Computational Results and Discussion **50**
6.1 Sensitivity Analysis 50
6.2 Overall Performance 51
6.2.1 Performance Tables 53
6.3 Performance on a Single Right-Hand Side Vector 57
6.4 Binary Integer Multi-dimensional Quadratic Knapsack 58
6.5 Integer Multi-dimensional Quadratic Knapsack 60

7 Conclusion **62**

List of Figures

2.1 The relationship of the problem class addressed in this thesis, to less general problems. In the acronyms used, "I" means Integer, "L" means Linear, "Q" means Quadratic, and "P" means Program. Past value function approaches have considered up to Integer Quadratic Programs (IQPs). 6

3.1 How variable ordering relates to variable fixing, independently of the w-ordering. In green are the most-recently incremented variables that remain unfixed (but lower bound their descendants in that variable's dimension), in gray are variables that are fixed for the node and all descendants, and the * symbol indicates subsequent variables in the lexicographic ordering. . 15

4.1 A flowchart capturing the overlying structure of the TSA's main loop . . . 22

4.2 The main loop procedure. 24

4.3 Pseudocode for the Initialization step of our main loop 32

4.4 Illustration of the *Enforcing Pareto Monotonicity* procedure. The space depicted is a $m = 2$-dimensional constraint space. The point in black is the current node, considering a smaller box $\mathcal{B}(min(\ell^j, L), u^j)$ of potentially LSO descendants. Green is a node providing a value function lower bound that is able to change the u^j value through *Enforcing Pareto Monotonicity*, while red is not. Blue illustrates how this might eliminate a solution of the z^j-upper bounding ILP, prompting another ILP solve. 34

4.5 Illustrated above are the weakly-dominated statuses of solutions corresponding to all x-values with coordinates between 0 and 2 depicted in the space of their right-hand sides. Blue indicates nondominated, red indicates dominated. 36

4.6 Illustration of the Extended EPM procedure. The space depicted is a constraint space of dimension $m = 2$. The point in black is the current node, considering a smaller box $\mathcal{B}(min(\ell^j, L), u^j)$ of potentially-optimal descendants. The red points cannot be used in standard EPM but can be used in the Extended EPM. The red dashed lines are the strongest monotonicity-implied nonconvex cut, relaxed to the green dashed lines to successfully cut the region containing the ILP solution in blue. 40

4.7 The main loop procedure with all algorithmic enhancements. 42

4.8 The pruning procedure. 43

5.1 Parameters used for generating problem instances. 49

6.1 A comparison of the number of dominated nodes that are still considered and not pruned across n values for various levels of L and U. 52

6.2 A comparison of the size of the data structure that stores a superset of S^{opt} across U-values and n values. 53

6.3 A comparison of the runtimes of the proposed approach to Gurobi for solving all instances of the smallest size, n=40. 54

6.4 A comparison of the runtimes of the proposed approach to Gurobi for solving all instances of the medium size, n=60. 55

6.5 A comparison of the runtimes of the proposed approach to Gurobi for solving all instances of the large size, n=70, outside of a single instance for $L = 0, U = 60$, denoted by the asterisk. 56

6.6 A comparison of the runtimes of the proposed approach to Gurobi for solving the same problem. Although tested on a small number of instances, the TSA outperforms on average across all problem sizes. 57
6.7 A comparison of the performance ratios of the proposed approach to Gurobi for solving problems with a given upper bound magnitude (20) across various magnitudes of lower bound for the binary case. 58
6.8 A comparison of the performance ratios of the proposed approach to Gurobi for solving problems with a given upper bound magnitude (40) across various magnitudes of lower bound for the binary case. 59
6.9 A comparison of the performance ratios of the proposed approach to Gurobi for solving problems with a given upper bound magnitude (60) across various magnitudes of lower bound for the binary case. 59
6.10 A comparison of the performance ratios of the proposed approach to Gurobi for solving problems with a given upper bound magnitude (20) across various magnitudes of lower bound for the integer case. 60
6.11 A comparison of the performance ratios of the proposed approach to Gurobi for solving problems with a given upper bound magnitude (40) across various magnitudes of lower bound for the integer case. 60
6.12 A comparison of the performance ratios of the proposed approach to Gurobi for solving problems with a given upper bound magnitude (60) across various magnitudes of lower bound for the integer case. 61

List of Tables

2.1 Comparison of value-function based solution methods for bilevel and stochastic programs. MIP indicates support for continuous and discrete variables, Linear, Quadratic, and Nonlinear indicates support for nonquadratic nonlinear objectives. Exact indicates that the value functions returned are exact. 7

3.1 Parameters used in the Overview of the Tree Search Algorithm. 14

4.1 Additional Parameters used in the Tree Search Algorithm. 20

Acknowledgements

I would like to acknowledge the people and organizations who helped me on my journey to complete this thesis. To my amazing advisor, Prof. Trapp, who has known me as an undergrad and taught and mentored me through it all. To my family, Mom, Dad, and Phoebe, and all my cousins as well I am blessed to have you in my life. To Trivani for the unwavering support and love. And to ARCHES, DS Council, GSG and all my graduate school friends, thank you for making my time at WPI really special.

I would also like to acknowledge HIAS for the amazing work they do, and the RUTH team in particular. Lastly, I would like to acknowledge the NSF for the RAPIDS grant which supported my studies in Fall 2022.

Chapter 1

Introduction

Exceedingly challenging integer nonlinear optimization problems that would have been prohibitive even 10 to 20 years ago are increasingly being tackled. While difficult, integer nonlinear programs appear in a diverse array of contexts, and are sometimes treated with specialized solvers tailored to their specific application areas. Other problems have a latent polynomial nonseparable discrete substructure, and as a result they can be phrased as problems that are able to be handled by solvers for integer polynomial problems. Model predictive control for robots, problems in compiler design, and hedge fund investment-specific problems are being solved using increasingly complicated models [1–4]. In these domains, it would be extremely helpful if an algorithm could respond to changes in resource constraints.

The ability to quickly reason around the superoptimal feasible region is even more compelling, yet to date remains out of reach for problem classes lacking a tractable strong dual. Instead, these (polynomial) Mixed Integer Nonlinear Programs (MINLPs) are solved with modern solvers such as Gurobi and BARON. Solving MINLPs to provable optimality is challenging and almost all tractable problems in this class have deep structure that can bring computational advantages when properly understood and employed.

Though difficult to optimize, MINLPs appear in a diverse array of contexts. The empirical results demonstrated in this thesis exist in the domain of nonlinear knapsack

problems, a subset of integer nonlinear programs. Consider the use case where there is a range of resource values that could be expected to see in practice, whether at a later time with uncertainty, as an adversarial action, or during re-optimization. The question this thesis answers is:

> *How can optimality information be intentionally and efficiently generated for the integer nonlinear knapsack problem over all resource vectors in a region of interest, to thereby provide near-instantaneous lookups for any resource vector within the region of interest?*

When there is a lack of clarity with respect to the resource levels that form the right-hand side of knapsack or budget constraints, it can be advantageous to formulate problems in terms of a value function to provide flexibility with varying resource levels. The value function approach associates to each resource vector a corresponding optimal objective function value. By simultaneously leveraging optimality criteria and problem structure over a range of constraint vectors, in this thesis we demonstrate how to construct a unique data structure that efficiently stores optimality information over a large set of possible resource levels, thus enabling efficient query and reasoning capabilities for fast, repeated optimization. Alternatively, our value function approach reduces the overall cost of many distinct solves by sharing computational cost between all solves, eliminating redundancies and avoiding computation that is overly specific to a single solve.

This thesis constructs optimality information for the nonlinear integer programming value function over a region of uncertainty in resource availability. This information is minimally stored in an optimized data structure in such a manner that is extremely efficient to query and reason, enabling subsequent optimization and reoptimization in milliseconds. This data structure also forms the underpinning of a new algorithm that is designed to incrementally construct the value function. A number of enhancements in value function generation are demonstrated such as the inclusion of cuts, explicitly comparing partial solutions, and best, intentional reuses of computation. The approach presented demonstrably works on MINLP problem instances from the literature specifically

integer quadratic knapsack.

The remainder of this thesis outlines and expands upon the aforementioned novel methods addressing this challenging class of problems. Section 2 situates the work in the existing literature on MINLPs. Section 3 covers important details about the algorithm, mathematical notation, variable bounding, and the data structure employed for fast lookups. Section 4 fully describes our foundational tree-search algorithm, complete with a proof of correctness, and the various enhancements we have used to make it competitive. Section 5 explains the details of and motivation for the experimental decisions made. Section 6 contains the numerical and qualitative results and analysis of results for the experiments. Section 7 concludes, summarizing what we have shown and highlighting important next steps. Lastly, Section 8 is our Appendix.

Chapter 2

Background

Gurobi [5] and BARON [6] are two state of the art solvers that can solve to global optimality various classes of integer nonlinear programs (INLPs) for fixed resource vectors. Upon solving to global optimality, standard solver output is a single optimal solution and related optimality information.

Convex optimization problems including linear programs (LPs) have a property known as strong duality that implies the existence of a solution to the bounding dual problem that achieves the same optimal objective function value as the solution to the primal. This fact means that adjustment to resource vectors can be efficiently accounted for using the data from the dual solution. Specifically, another property of LPs that comes from strong duality, Complementary Slackness, that if a linear program is solved to optimality, all slack variables in the primal (those for that strict inequality holds) can be matched to corresponding dual variables that are not slack for all inequality constraints. This means using basis information from a solve, one can quickly reoptimize to find an optimal solution with adjusted constraints. Similarly, when solving Integer Linear Programs (ILPs), another similar property, Integral Complementary Slackness, holds that gives upper bounding knowledge about the value function from the solution of the LP when variables are restricted to integers. Mixed Integer Linear Programs (MILPs or MIPs) contain LPs and ILPs as special cases and can have a mixture of discrete and continuous variables. Unlike

with LPs or ILPs, INLPs do not have a strong duality or integral complementary slackness result that works in full generality because their relaxations are nonlinear. Without these strong results, directly using solver outputs to speed up the process of computing a value function is fraught with problems.

To place the proposed algorithm's problem class within the broader scope of nonconvex optimization problems, it is helpful to start at ILPs and work outward. As can be seen in Figure 2.1, ILP is the least general problem class. From there, a popular class are separable Integer Quadratic Programs (IQPs) that have a quadratic objective that separates like $\sum_i c_i x_i + Q_{i,i} x_i^2$. From there, nonseparable IQPs have a general quadratic objective with symmetric off-diagonal terms and can be expressed as $1/2 x^\top Q x + c^\top x$. While these problems are more complicated than separable IQPs because the variables interact, that same property can make them useful, for instance in the hedge fund application area [4] shows how the modeler can make use of interactions to model correlations across asset risk. Lastly, the most general problem class that this thesis considers (with linear constraints in the experimental study, yet no such restriction need apply) is the nonseparable Integer Polynomial Program class. This class has as an objective function an arbitrary polynomial with interaction terms and can model interaction across more than two variables at a time as well as be used via series expansion as a general approximation of any analytic function. Although the presented algorithms and theory behind the results in the thesis supports this broad class, quadratic objectives are considered to demonstrate its effectiveness. This is because Gurobi can solve them for single right-hand sides without reformulation.

Previous value function approaches have been used in linear stochastic programming [7–9], understanding the integrality gap in Integer Linear Programs (ILPs) [10, 11], and multi-parametric problems [12]. Stochastic Programming is a technique and problem class that models the case where a random process may change the objective function or constraints for a dependent, future optimization problem that is also conditional on an initial stage and uncertain realizations of randomness.

An example of such a problem is the classical farmer's problem [13], where a farmer

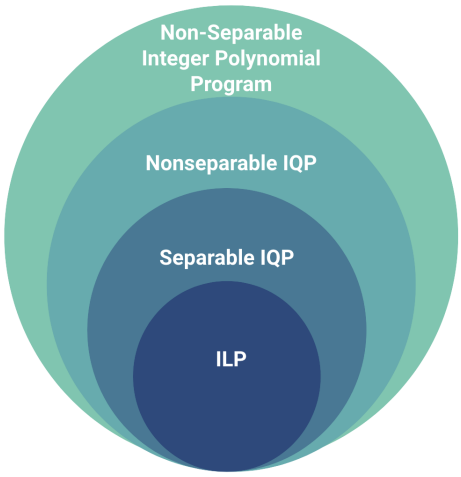

Figure. 2.1: The relationship of the problem class addressed in this thesis, to less general problems. In the acronyms used, "I" means Integer, "L" means Linear, "Q" means Quadratic, and "P" means Program. Past value function approaches have considered up to Integer Quadratic Programs (IQPs).

must make purchasing decisions up front and later choose how best to respond to how good of a harvest year it is. The farmer's initial decision has corresponding variables in the optimization model and is called the first stage problem, and recourse decisions are made in the second-stage after observing the uncertain yields. This separates the problem into a first stage that is summed with a weighted combination of second-stage recourse objectives, with identical structure only varying parametrically, weighted proportionately to the likelihood that each scenario is encountered. In particular, the two-stage stochastic integer programming literature includes Trapp et al. [7], Antley [9], and Özaltın et al. [14] that all apply value function methods to problems of interest to avoid duplicating computation using results related to duality for their respective problem classes. These approaches re-use the information implicit in the solutions to ILPs, along with the structure of the constraint matrix, to formulate a solution to restricted supersets of the necessary right-hand sides to solve the problem under consideration. Trapp et al. [7] uses a restricted integral monoid to obtain the superset of interest, and Antley [9] uses inequalities on the difference between restrictions of the problem to pure integer solutions, pure continuous, and full mixed-integer. This study is closely related to Antley [9], where the authors similarly reuse bounds computation, a space-partitioning data structure, and pursue value

function improvement. A key difference is that they use an ϵ bound alongside the linear problem structure to go further and incorporate continuous variables, avoiding the computation of potentially very many candidate vectors. Bertsimas and Stellato [15] argue for an understanding of optimization problems through solution motifs and study how they change with respect to right-hand side data explicitly. Although Özaltın et al. [14] and Zenarosa et al. [16] study quadratic problem classes, both approaches make additional assumptions on the problem class. Due to implementation difficulties the most similar quadratic approach in Özaltın et al. [14] was unable to be compared with this work.

Author / Year	MIP	Linear	Quadratic	Polynomial	Exact
Özaltın et al. [14] (2012)			Y		Y
Trapp et al. [7] (2013)		Y			Y
Zenarosa et al. [16] (2018)		Y	Y		Y
Bertsimas and Stellato [17] (2019)	Y	Y			
Tavaslıoğlu et al. [18] (2019)		Y			Y
Antley [9] (2022)	Y	Y			
This work (2023)	*[a]	Y	Y	Y	Y

Table 2.1: Comparison of value-function based solution methods for bilevel and stochastic programs. MIP indicates support for continuous and discrete variables, Linear, Quadratic, and Nonlinear indicates support for nonquadratic nonlinear objectives. Exact indicates that the value functions returned are exact.

[a] While not discussed in this thesis, there is a natural extension to the mixed integer case with linear objective terms on continuous variables.

The work in this thesis is primarily concerned with INLP Polynomial Optimization problems, and among them chiefly the quadratic knapsack problem, with varying resource vectors. While less common than ILPs, this problem class has been studied extensively in the INLP literature. A potential reason for this is the deep connection between theory and practice that is present. Lasserre [19] connects polynomial optimization to a complexity hierarchy. On the practical side, these problems are widely applied in Operations Research and Management Science. They have been applied to solve engineering design problems [20, 21], in computational biology [22], project selection [23], the location problem [24], the allocation problem [25–27], and knapsack problems [28, 29]. Lookup approaches have also been used to (at least approximately) solve large sets of similar NP-hard optimization problems in milliseconds [17], by using past searches to approximately guide new searches

for a-priori unknown optima.

Chapter 3

Mathematical and Computational Preliminaries

Foundational concepts are now introduced to establish a mathematical framework for the algorithmic approach.

3.1 Mathematical Preliminaries

The section provides an understanding of the standard formulations and properties for related mathematical optimization problem classes, gives definitions to nondominance relations, defines and characterizes the concept of Level Set Optimality, weak and strong nondominance, and the ordering used by the algorithm to ensure an incremental construction of the value function.

3.1.1 Relevant Mathematical Optimization Problem Classes

The standard *linear program (LP)* is an optimization problem that can be expressed in the following canonical form:

$$\begin{aligned} \text{maximize} \quad & c^\top x && \text{(3.1a)} \\ \text{subject to:} \quad & Ax \leq b, && \text{(3.1b)} \\ & x \in \mathbb{R}^n_+, && \text{(3.1c)} \end{aligned}$$

where c is a vector of known objective function coefficients in $\mathbb{R}^n$, A is an $m \times n$ constraint matrix ($\mathbb{R}^{m,n}$), b is a vector of capacities or resource levels in $\mathbb{R}^m_+$, and x is a variable vector in the positive reals, $\mathbb{R}^n_+$.

The standard (primal) LP has a corresponding *dual* problem:

$$\begin{aligned} \text{minimize} \quad & b^\top y && \text{(3.2a)} \\ \text{subject to:} \quad & A^\top y \geq c, && \text{(3.2b)} \\ & y \in \mathbb{R}^n_+, && \text{(3.2c)} \end{aligned}$$

with the same data elements A, b, c as previously defined, and y a variable vector in the positive reals, $\mathbb{R}^m_+$.

Linear programs, as a type of convex optimization problem [30], possess the property of strong duality, meaning that if there is an optimal solution $x^\star$ to the primal, then there is an optimal solution $y^\star$ to the dual, and at optimality, the optimal objective function value of the primal is equal to that of the dual, that is, $c^\top x^\star = b^\top y^\star$. While the worst case computational complexity of linear programs can be exponential, LPs on average solve very quickly in practice, making them useful for a variety of optimization tasks.

Integer linear programs (ILPs) differ from standard LPs in that the variables are required to be integer valued, $x \in \mathbb{Z}^n_+$. This seemingly simple adjustment causes the ILP to become a computationally challenging problem (NP-hard), generally speaking.

By additionally relaxing linearity in the objective or constraint function(s), we arrive at an integer *nonlinear* program (INLP):

$$\begin{aligned} \text{maximize} \quad & f(x) && \text{(3.3a)} \\ \text{subject to:} \quad & g(x) \leq b, && \text{(3.3b)} \\ & x \in \mathbb{Z}_+^n, && \text{(3.3c)} \end{aligned}$$

where $f(x)$: $\mathbb{Z}_+^n \mapsto \mathbb{R}$ is a general nonlinear function, typically at least twice-differentiable, $g(x)$: $\mathbb{Z}_+^n \mapsto \mathbb{R}^m$ and $b \in \mathbb{R}_+^m$. For the purpose of this thesis, we consider f to be a polynomial function of x, not required to be separable or convex.

When there is uncertainty in the right-hand side resource vector, it can be useful to view INLPs in a parameterized manner, that yields the INLP value function formulation:

$$\begin{aligned} z(\beta) = \text{maximize} \quad & f(x) && \text{(3.4a)} \\ \text{subject to:} \quad & g(x) \leq \beta, && \text{(3.4b)} \\ & x \in \mathbb{Z}_+^n, && \text{(3.4c)} \end{aligned}$$

where $\beta \in \mathbb{R}_+^m$ replaces b as parametric input, and $z(\beta)$ is the *value* function. We further assume β lives in $\mathcal{B}(L,U)$, the set of m-dimensional vectors contained within lower and upper bounding vectors L and U, that is: $\mathcal{B}(L,U) = \{\beta \in \mathbb{R}_+^m \mid L \leq \beta \leq U\}$.

3.1.2 Nondominance, Level Set Optimality, and An Efficient Frontier

Weak and strong nondominance relations are important concepts when algorithmically computing the value function over $\mathcal{B}(L,U)$, particularly with respect to certain vectors $[-f(x), g(x)_1, \cdots, g(x)_m]$ that combine objective and constraint function information.

Definition 1. *(Weak Nondominance) For two distinct vectors $a \in \mathbb{R}_+^{\overline{m}}, b \in \mathbb{R}_+^{\overline{m}}$, b is weakly-LSO-dominated by a if $a_i \leq b_i$ for $i = 1, \ldots, \overline{m}$ and $a \neq b$. If it is not weakly-LSO-*

dominated, it is LSO-nondominated. Denote being LSO-nondominated in relation to two vectors as $\not\preceq$.

Definition 2. *(Strong Nondominance) For two distinct vectors* $a \in \mathbb{R}_+^{\overline{m}}, b \in \mathbb{R}_+^{\overline{m}}$, *b is strongly-LSO-dominated by a if* $a_i < b_i$ *for* $i = 1, \ldots, \overline{m}$. *If it is not strongly-LSO-dominated, it is weakly-LSO nondominated.*

The notion of *Level Set Optimality* is defined as the property of being weakly nondominated as in Definition 1. Trapp et al. [7] proved that the set of all level-set optimal (LSO) vectors (henceforth, Θ) is sufficient to characterize the value function of an integer linear program [7]. This result was used to motivate an algorithm to find a restricted superset of LSO vectors and from that a restricted value function, much as is the goal of this thesis, for a set of right-hand sides $\mathcal{B}(L, U)$. For performing queries on integer vectors β in the region of interest $\mathcal{B}(L, U)$, the performance of their approach depends on the fraction $\rho = |\Theta \cap \mathcal{B}(L, U)|/|\{\beta : \exists x \in \mathbb{Z}_+^n \text{ s.t. } g(x) = \beta \leq U\}|$ of possible β vectors that remain after eliminating from consideration all right-hand sides that are not LSO vectors. As $\rho \to 1$, the importance of only considering β values that could be LSO diminishes. Additionally, a practical consideration is that as $\rho \to 1$, the space required to represent the LSO set becomes a limiting factor. The algorithm detailed in this thesis also benefits from ρ being small, as it iteratively identifies nondominated vectors to construct an *efficient frontier* representing the value function.

Definition 3. *(Efficient Frontier) The Efficient Frontier is defined as the set of weakly nondominated* $\overline{m}$*-vectors considered by the algorithm. So if the algorithm has considered a set of* $\overline{m}$*-vectors* $\mathcal{S}$, *it is the set* $\mathcal{E} = \{a \in \mathcal{S} : \forall\ b \in \mathcal{S}, (a \not\preceq b)\}$.

3.1.3 Ordering via w-Weighting

We employ a w-weighting scheme to induce an ordering on β vectors, defined as follows:

Definition 4. *(w-weighting) The w-weighting of a vector* $\beta \in \mathbb{R}_+^m$ *is induced by another fixed vector* $w \in \mathbb{Z}_+^m$ *as the following weighted sum:* $w(\beta) = w^\top \beta$. *The w vector must have*

the property that it is strictly increasing with respect to x, that is $w(g(x)) < w(g(x + e_i))$ for all values of x and unit vectors e_i.

The order induced by the w-weighting scheme on x vectors and their $g(x)$ transformations is essential for ensuring that lower weight vectors never dominate higher weight vectors. Thus, vectors once believed to be optimal in the algorithmic generation of level-set optimal vectors, are never removed from the search data structure. In the domain of multi-objective optimization, a similar lexicographic ordering exists over objective functions to attain this ordering [1, 31, 32]. The process of w-weighting generalizes this concept to including constraints without a priori determination of a constraint ordering. The w-weighting scheme is also useful for the *Enforcing Pareto Monotonicity* and cuts/$m-region$ bounds adjustment techniques defined in Section 4, in that it produces LSO bounds uniformly over the constraint dimensions.

3.2 Algorithmic Preliminaries

In this section we present an overview of the main contribution of this thesis – *Tree Search Algorithm* (TSA) – including nomenclature to aid in streamlining the exposition, and then provide additional information to explain variable fixing order as well as the data structure used for efficient lookup.

3.2.1 Overview of Tree Search Algorithm

This section introduces a high-level overview of an algorithm that uses a search tree to incrementally construct the value function of an integer nonlinear program of the form (3.4). Table 3.1 details useful notation for the ensuing exposition.

The main algorithm incrementally constructs the value function by using the w-ordering scheme to build a search tree of subproblems stored as nodes with parent-child relationships. The TSA builds the search tree by starting at the root node with every variable unfixed and set to zero. It then incrementally advances to node j, adding to x_i^j for index i a unit vector $e_i, 1 \leq i \leq n$. All nodes have an associated variable vector x^j that

Table 3.1: Parameters used in the Overview of the Tree Search Algorithm.

Notation	Definition
d^j	Set of feasible descendant nodes (after constraints have been applied) of a node j
p^j	Index of the parent node of a node j
x^j	The vector of variable values stored at node j
$\overline{x}^j$	Vector of inferred upper bounds on x in d^j (given x^j and the variable fixing order, see Section 3.2.2)
w	$m-$dimensional positive vector quantity that allows us to find *weights*
ω_i^j	*Weight* of a variable at node j; equal to $w^\top g(x^j + e_i)$ for chosen w
ω^j	Current total *weight* of a node j ($w^\top g(x^j)$)
$m-$region	Region of $g(x)$ values that the node has yet to eliminate from consideration for optimality
S	Constructed set of nodes (indexed by j) that have yet to be pruned
S^{heur}	Set of value function lower bounds determined by the heuristic insertions
S^{opt}	Set corresponding to the notion of a nondominated frontier comprised of LSO vectors
$\overline{m}$	$m+1$, the number of dimensions that $f(x), g(x)$ comparisons occur in.

consists of a fixed prefix of values, a variable index that was incremented from its parent, and a suffix of not yet assigned variables that are set to zero. Hence at each step of the TSA, the tree is a subtree of a lexicographically ordered tree on the variable lower bounds. This way of ordering the choice of increments and variable-fixings allows the algorithm to always only insert nodes in Θ in S^{opt}.

Because the w-weighting implies that strictly greater x values have greater w-weight, and the x^j are incremented in a strictly increasing manner, the variable value at each child node has strictly greater weight (ω^j for node j) than that of its parent. This can be seen from the associated resource consumption at node j, β^j, in relation to its variable lower bounds as follows: $\omega^j = w^\top \beta^j = w^\top g(x^j)$. Figure 3.1 visually depicts the process of moving from a parent to a child node with precise values given to the variables. At every node, the algorithm is designed to prune child nodes through the use of auxiliary

information. Section 4.1 details the additional algorithmic enhancements that make use of this additional information. Specifically, these nodes limit the possible values of x that can be considered in their descendants, quantify upper and lower bounds on the objective function values achieved subject to their constraints, and represent eliminating a single value of x from consideration corresponding to the variable lower bound they have. Because the w-weighting guarantees that all nodes with x^j values that may weakly dominate the current node will have been considered before it, a lookup into the data structure containing all LSO vectors discovered up until this point is sufficient to establish if the current x^j is weakly dominated. In this way, the algorithm proceeds to incrementally add LSO vectors to a set S^{opt} representing the efficient frontier, defined to be the set of points that are weakly nondominated that have been observed thus far in the search.

3.2.2 Variable Fixing Order

Figure 3.1 seeks to illustrate the variable fixing order from the perspective of what the process might look like from a mid-level node in a search tree in practice. The * symbol indicates variables that are not fixed and do not yet have a variable (x_i^j) value that has been modified (and until they are, are treated as zero), the green oval indicates the only variable that is unfixed and does have a lower bound greater than zero imposed, while the grey oval indicates variables that are fixed to their values for that node and any descendants.

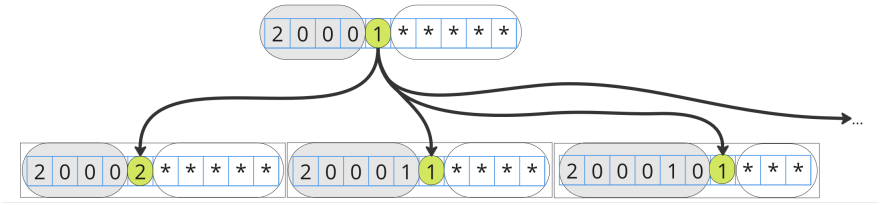

Figure. 3.1: How variable ordering relates to variable fixing, independently of the w-ordering. In green are the most-recently incremented variables that remain unfixed (but lower bound their descendants in that variable's dimension), in gray are variables that are fixed for the node and all descendants, and the * symbol indicates subsequent variables in the lexicographic ordering.

Increments by one are considered in order of decreasing ω_i^j, and all variable indices in the prefix ($i < i'$ where i' is the index being incremented) are fixed to the value they

took in the parent node. Figure 3.1 shows an example of how this ordering dictates the order in which subsets of feasible vectors are considered. The parent node, 20001*****, may branch into children that are lexicographically higher than the parent while leaving 2000 fixed. For instance, the next child to the right has value equal to 2000101***.

This variable ordering commits earliest to the most constraining branches for a given solution that enables many nodes to be pruned early in the search. The lowest w-weight children of a node (the ones created by increasing the green variable) are the least constrained, and each child is exponentially more constrained (by variable fixing) than the last. As a result, a large number of suffixes are likely to be pruned if pruning is applicable and promising subproblems are quickly separated from those that are less likely to have LSO descendants.

3.2.3 Bounding Logic

As the TSA progresses and new nodes are added, the node descendants are associated with three kinds of lower and upper bounds. These bounds are used to establish weak-dominance, as well as to accelerate algorithmic performance through elimination of unnecessary search space.

There are $m-$region bounds for a node j that are associated with $g(x^k)$ vector values $k \in d^j$ that may be LSO. These bounds are established in the course of the upper bounding procedure (Section 4.1.3) tightened during $m-region$ bounds adjustment (Section 4.2.2).

The $n-$region bounds for a node j are associated with lower and upper bounds on the variable values that are taken at each of the n indices across all descendants d^j. These bounds arise naturally as the tree search increments and fixes variable values at nodes as well as by virtue of reduced costs (Section 4.1.3). The bounds are able to tighten approximations to higher-order terms in the upper bounding as well as improve the more advanced techniques in Sections 4.2.2 and 4.2.4.

The scalar objective is also bounded through $z-$region bounds. These bounds quantify the minimum and maximum objective value that any descendant of a node j may have.

The lower bound is achieved by value function lookup in conjunction with the $m-$region lower bound and the problem parameters, while the upper bound is set explicitly by the upper bounding routine detailed in Section 4.1.3.

3.2.4 Data Structures

Another contribution of this thesis is the use of a spatial search data structure that allows for fast querying the incrementally-constructed efficient frontiers ($\mathcal{E}$) of the value function. The data structure, originally based upon the k-d tree data structure [33] and since a dynamic R^*-tree [34], stores the combination of feasible right-hand sides and objective function values necessary to represent the value function over LSO vectors (Θ), that was proven [7] to be sufficient to represent it over the entire set of right hand sides between L and U. The choice of the values to store in the data structure, itself determined by the data structure's current contents, leads to it being judiciously populated, and therefore being a Minimal R^* Tree (MR*T) for the nodes discovered by the TSA at any point in time. The data structure stores all lower and upper bounding information in a nested tree structure with lower and upper bounds on both bounds and objective-function values to efficiently prune branches of the search. This MR*T data structure, being an R^* tree, has strong properties under insertion, not requiring frequent rebalancing like a k-d tree. Additionally the MR*T is designed to support efficient querying of optimal value functions by using a splitting rule that heuristically minimizes the volume of enclosing boxes around points using a quadratic rule. This means that in the process of an orthogonal range query, the primary type of query performed on value functions, the MR*T can often prune large branches without reaching the leaves containing actual full data points. It is imperative that the algorithm have this efficient MR*T data structure because it heavily relies upon lookups to not only determine nondominance for vectors but also to establish all value function lower bounds for nodes as it generates its own search tree. Empirically this approach has been seen to scale approximately logarithmically in time with the number of vectors inserted for both lookup and insertion. The proposed algorithm performs a number

of lookups greater than 1 at each step in its main loop as well as solves a subproblem. This means that if there is a constant-size subproblem solved at every step of the algorithm as well as some number of lookups and insertion (in the worst case), lookups and insertions will eventually come to dominate the performance considerations.

Chapter 4

Comprehensive Value Function Construction via Tree Search Algorithm

Having presented foundational elements in Section 3, we now discuss the specifics of the implementation of the TSA, MR*T, and enhancements.

4.1 Tree Search Algorithm (TSA): Foundational Elements

Building upon the overview presented in Section 3.2.1, the Tree Search Algorithm (TSA) proceeds in two steps. In step 1, precompute valid lower bounds (Section 4.1.4) on $z(\beta)$, used to speed up the global search for level set optima. In step 2, run the main loop function (Section 4.1.1) until there are no more nodes to explore, returning the resulting MR*T data structure as means to efficiently access the value function over $\mathcal{B}(L, U)$. The main loop is central to the TSA, as it is where the set of optimal x values are found by iteratively constructing the search tree. Thus, it is prioritized over the important discussion of algorithm initialization, that is deferred until Section 4.1.4.

Critical to the TSA are bounding strategies associated with each node j. These bounds

apply to the variable vector x^j and the associated $m-$vector $g(x^j)$. As the TSA advances it dynamically reduces the search space through multiple bounding procedures to update respective 1. $m-$regions and 2. $n-$regions, as well as 3. scalar objective $z-$region bounds for each node j. The bounding features are naturally inherited from increasingly more specific subproblems at nodes of the tree structure, and their interaction amplifies their respective effects. These steps are critical for the TSA to complete, in reasonable time, and avoid an exhaustive search of the full set of feasible x values.

4.1.1 Tree Search Algorithm: Main Loop

Several additional concepts extending beyond the high-level overview of the node data discussed in Section 3.2.1 improve the efficiency of the search for LSO vectors. The associated nomenclature is introduced in Table 4.1.

Table 4.1: Additional Parameters used in the Tree Search Algorithm.

Notation	Definition
f^j	Shorthand for $f(x^j)$, the (accrued) objective value for partial variable assignment x^j
$\overline{f}^j$	Upper bound on f of LSO vectors in d^j
$\underline{f}^j$	Lower bound on f of LSO vectors in d^j
$opt(\beta)$	Set of x values in $\mathbb{Z}^n$ that, with respect to β, are maximizers of $z(\beta)$, that is, $z(\beta) = z(g(x)) = f(x)\ \forall\ x \in opt(\beta)$
β^j	The $m-$dimensional mapping of x, that is, $g(x^j)$
z^j	The range of potential value function values that d^j may influence, spanning $\underline{f}^j$ to $\overline{f}^j$
ℓ^j	Local lower bound on β vectors for that descendant nodes d^j have yet to be proven suboptimal
u^j	Local upper bound on β vectors for that descendant nodes d^j have yet to be proven suboptimal

First, each currently-considered node j maintains an associated $m-$region with lower (ℓ^j) and upper (u^j) bounds over that its descendants could possibly yield one or more LSO vectors. This region may not necessarily be contained in $\mathcal{B}(L, U)$ in the overall problem, as $\mathcal{B}(\ell^j, u^j)$ must include all feasible $m-$vectors that could still dominate a value function bound and some of those may occur below L. Node j also keeps track of its current value

function lower and upper bound, $\underline{f}^j$ and $\overline{f}^j$ respectively. Descendants d^j of node j are those nodes resulting from adding a suffix to the x-value at that node (as detailed in Section 3.2.2 and Figure 3.1). The weight of node j is denoted ω^j; because the values assigned to variables for nodes increase in a monotonic fashion, their weights also monotonically increase according to the a-priori-determined w. The weight of variable i at node j is denoted ω_i^j and conveys its contribution to w-weighting when variable i is incremented. The value of $f(x^j)$, denoted f^j, is distinct from the lower bound on descendant nodes $\underline{f}^j$. It is used to find vectors that weakly dominate x^j to improve z^j-bounding, a process that is further described in Section 4.2.2. The upper bounds for values that variables take at prefix indices are set and inherited by descendants according to the variable fixing ordering. For linear constraints, this means a variable index that has a higher resource consumption will be fixed earlier in the variable fixing ordering, induced by the $w-$weight assigned. The variable fixing ordering is determined by the weights at the root node.

The main loop proceeds as follows. A priority queue (P) is used to maintain nodes in nondecreasing order according to their weights. When multiple nodes share the same weight, ties are broken by f value (highest first, ties broken arbitrarily), for the purpose of increasing the likelihood that nodes consider prior nodes that LSO-weakly-dominate them.

To handle many nodes sharing a weight ω^j (that may often be the case for integer weights), a set of nodes with that weight is considered at a time. For ω^j any nodes with the same β are sorted by their $f(\cdot)$ value, regardless of feasibility; the feasible node with the largest $f(\cdot)$ value is checked for nondominance. If it is, it immediately passes to the optimal set in the MR^*T data structure.

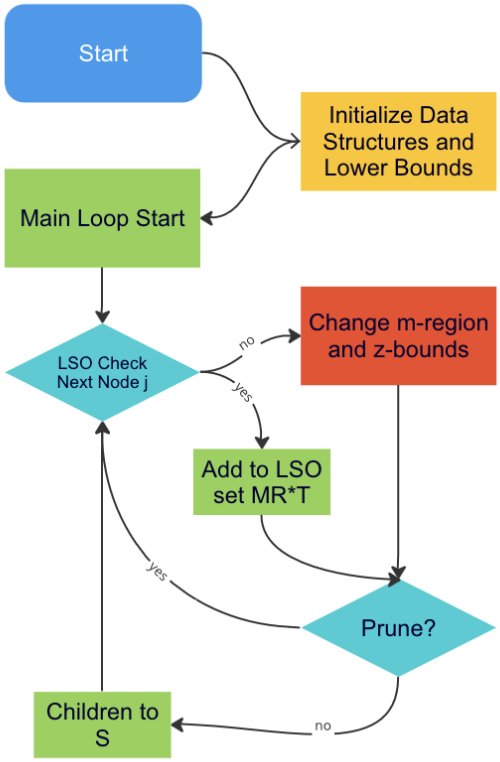

Figure. 4.1: A flowchart capturing the overlying structure of the TSA's main loop

The rest of this section considers each node j separately. An inherited value function upper bound $\overline{f}^j$ is compared against the value function lower bound $\underline{f}^j$ found by lookup against the minimum of ℓ^j and L into the optimal set S^{opt} and heuristic lower bound set S^{heur} to determine whether a given node can be immediately pruned in the event that $\overline{f}^j \leq \underline{f}^j$. All nodes that were not immediately pruned search for all dominating nodes from the optimal set S^{opt}, the heuristic set S^{heur}, and the other nodes sharing the same β^j vector. The query to find dominating nodes from these sets is not against the minimum of ℓ^j and L, but rather against β^j as the objective is to find x values that weakly-dominate the node itself, not a value function lower bound on its descendants. These nodes sharing the same β^j vector are convenient when creating the $m - region$

bounds adjustment constraints defined in Section 4.2.2. After applying these constraints, the TSA computes a new value function upper bound $\overline{f}^j$ and performs *Enforcing Pareto Monotonicity* as defined in section 4.2.1. To find the value function upper bound, an ILP is used for z^j-upper-bounding that reasons over the feasible descendants of the node. The specifics of the implementation and derivation of the z^j-upper-bounding ILP are detailed in Section 4.1.3.

If the z^j-upper-bounding ILP is infeasible or the global lower bound the TSA finds dominates the upper bound after computing the ILP or its LP relaxation, the TSA can safely prune the node j. Further, if the combination of the z^j-upper-bounding ILP value and the z^j-upper-bounding LP value plus reduced costs are sufficient to produce a set of lower bounds (through lookups) that cover the full space from ℓ^j to u^j and dominate either the ILP upper bound or the implied LP-plus-reduced-cost upper bound, the TSA can safely prune the node j. This process is detailed further in Section 4.2.4. After applying a step that attempts to constrain $m-$region upper bounds that we refer to as *Enforcing Pareto Monotonicity* (detailed in Section 4.2.1), the TSA attempts to further tighten the region contained between ℓ^j and u^j through optimization using $m-region$ Bounds Adjustment. If successful in changing ℓ^j or u^j or both, and not pruning the node, the TSA repeats $m-region$ Bounds Adjustment, reusing results that remain feasible after the changes. This is accomplished by simply storing the optimal solutions achieving the results and checking against the new constraints for violation.

After all nodes have undergone this process, remaining nodes have new variable lower and upper bounds computed for all unfixed variables and are inserted into the priority queue for the weight corresponding to their completion. Throughout this process, nodes inherit all three types of bounds ($m-$vectors, $n-$vectors, and z^j-bounds) from their parents. Any logical changes to ℓ^j are then applied, specifically, each coordinate of ℓ^j (being a lower bound on the $m-$region vector values of descendants) is set to a value strictly greater than or equal to the minimum $g(x)$ at that coordinate for descendants.

The pseudocode for this loop is presented in Figure 4.8.

Main Loop
(1) INITIALIZE THE FIRST NODE IN S TO $x^0 = \mathbf{0} \in \mathbb{Z}^n$
(2) **While** S is not empty:
(3) Consider the next node by weight j, breaking ties by larger f^j value, removing it from P
(4) Collect all (heuristic, optimal) nodes dominating $f(x^j), g(x^j)$ into a set D
(5) Perform any problem-specific updates to ℓ^j, u^j
(6) **If** $|D| > 0$:
(7) Create the dominating cuts
(8) Attempt to prune j, if not pruned continue to (13) otherwise go to (2)
(9) **Else if** x^j is feasible:
(10) Add x^j to S^{opt}
(11) **Else if** x^j is not feasible but otherwise nondominated:
(12) Add x^j to S^{heur}
(13) **If** $\beta^j \not\leq min(\ell^j, L)$:
(14) Attempt to prune j, if not pruned continue to (15) otherwise go to (2)
(15) Perform binary search to prune the largest prefix of j possible from consideration at the current node
(16) Add the remaining children of j to S
(17) Add all nondominated feasible nodes in S^{heur} to S^{opt}
(18) Return S^{opt}

Figure. 4.2: The main loop procedure.

Before this routine has run, S^{heur} has already been populated with an initial heuristic lower bound set as described in pseudocode in Figure 4.3. Lines 1 and 2 initialize the root node of the search tree and run the search until all nodes have been considered ($|S| = 0$) respectively. Lines 3 and 4 perform the checks against the data structure to incrementally construct S^{opt} using the current node. Lines 6-8 indicate that if the node is dominated by at least one node in terms of the value function, pruning should be attempted. Lines 9 and 10 establish how nodes enter S^{opt} from the main loop. Lines 11 and 12 are for problems where nodes can be infeasible and nondominated at the same time. Consider cases where some dimensions of the resource consumption vector exceed U, but there exist not-yet-fixed variables that would reduce $g(x)$ in these dimensions if incremented so the node cannot be prematurely pruned. Lines 13 and 14 leverage nodes existing outside of the $m-$region

for their descendants. Pruning can find a value function lower bound on the region where their descendants can exist and compare that against the upper bound for that node. Line 15 is an enhancement that is further described in Section 4.2.5. What sets each child apart from the rest is the simultaneous fixing of variable bounds and incrementing of a variable bound, and the fixing respects an ordering, therefore binary search on the bounds that are fixed allows only a limited subset of children to be considered. Line 16 is the point where the children are added to the search tree. Lastly, lines 17 and 18 return only the proven-optimal feasible nodes encountered during the search, stored in the MR*T data structure.

It should also be noted that the TSA is unlike the more traditional global branch and bound approach that bisects the range of a single variable at a time (thereby eliminating the nonintegral region between the lower and upper branch point). The TSA removes at least one value of x from consideration at all nodes, whereas the bisection-based branch and bound approach only completely eliminates values of x from consideration at leaves (nodes without children) or when nodes are fathomed. The motivation for making such small adjustments is to guarantee that the value function is constructed strictly incrementally and all nodes *see* every potentially dominating x to maximize the benefit from *Enforcing Pareto Monotonicity* and $m - region$ *Bounds Adjustment*. Branch and bound does not need to track β values or the vectors that would dominate the lower bound of a node because it only needs to solve for the value function at a single point, making it easier to discard branches.

4.1.2 Tree Search Algorithm: Pruning

Pruning is important in reducing the rate of growth in computation time as the problem size increases. The process the TSA follows is described in more detail below.

When a node is removed from the set of all nodes (S) because it is the next lowest weight node in step (3) of the main loop, there is a chance that it is pruned without producing its children nodes in steps (8) or (14). To ascertain whether this is possible,

the TSA first checks to see whether one of two prunable node conditions is present.

- If the node is weakly dominated by at least one vector considering $f(x^j)$ and $g(x^j)$, there is the possibility that all descendants of the node are also dominated, motivating the constraints defined in Section 4.2.2.
- If all but one dimension in $g(x^j)$ is less than or equal to $max(\ell^j, L)$, apply *Enforcing Pareto Monotonicity* with the node or heuristic lower bound may show that the node does not have any optimal descendants.

4.1.3 Tree Search Algorithm: z^j-Upper Bounds

It is important to have strong upper bounds on the value function $z(\beta)$ of the parameterized polynomial optimization problem. However, there is a tradeoff between upper bound strength and computation time. The time spent on computing upper bounds should justify the extra computational expense. The TSA uses a combination of upper planes shared among terms in the polynomial ($\pi_i, i \in \mathcal{I}$, $\mathcal{I}$ an index set over shared variable factors) with variable upper bounds to efficiently apply a modification of the technique described in Glover [35] (as lower bounds are only nonzero for a single variable at a node, all products equal 0).

First, a brief introduction to the existing literature places the technique the TSA employs in relation to existing methods. While the TSA treats general integer nonlinear optimization problem, the simplest case is the integer quadratic knapsack problem, which can be formulated as follows:

$$z(\beta) = \text{maximize} \quad \frac{1}{2}x^\top Q x + c^\top x \tag{4.1a}$$

$$\text{subject to:} \quad Ax \leq \beta, \tag{4.1b}$$

$$x \in \mathbb{Z}^n_+, \tag{4.1c}$$

Gallo et al. [36] define upper planes that bound all f values attainable over the feasible

region in terms of a linear function of the assumed variable values. Through this method, they linearize the quadratic knapsack and establish an efficient solution. The strength of these upper planes and others are surveyed in Pisinger et al. [28]. Caprara et al. [37] develops a promising approach based upon a Lagrangean modification of the Gallo et al. [36] upper plane that augments the quadratic matrix with an anti-symmetric matrix to balance the contribution of items and tighten the bound to equality. Unfortunately, the subgradient approach they use relies on having a single-dimensional inequality constraint. In contrast, the TSA uses the upper plane approach without Lagrangean modification to find upper planes for general polynomial programs, and adds secondary inequalities based upon variable bounds to form conditional McCormick relaxations.

To gradually illustrate the way the upper bounds operate, it is instructive to consider the quadratic case. In the quadratic case, both for integer and binary variables, there is a need to bound the contributions of the nonlinear term. A standard upper plane approach would dictate that one decouple the problem of bounding the factor (π) that could be applied to each variable. These subproblems to find π_i for variable i compute maximal factor values one at a time, and usefully are only required to reason over x values that could prove LSO for the final solution. As a result, the π values of the parent node can be used to impose a cut, as can cuts implied by dominating vectors described in Section 4.2.2, as well as node-specific variable upper bounds $\overline{x}^j$.

We use an LP to find π_i that is adapted from the 4th upper plane LP (U^4) described in Gallo et al. [36] with these additions and the modification by Caprara et al. [37] noting that the variable x_i must be nonzero, and therefore can deduct from the resource budget of the subproblem. It can be seen that the upper planes in Gallo et al. [36] are an instance of a partial McCormick relaxation cut to produce an upper-bounding hyperplane. Specifically, consider for each variable x_i, a new variable that is equal to the quadratic contribution of the terms multiplied by that variable. Importantly, extending this to higher orders yields n independent linear subproblems and one independent subproblem of degree 1-less than the original problem degree in the Lasserre hierarchy [19]. Gallo et al. [36] exclusively

focus on the latter inequality to obtain optimization problems defining an upper plane for objective values of nodes in d^j. Recall from Table 3.1 that p^j is the parent node of a node j. Because this LP will have been solved for p^j, there are pre-determined upper-plane values for (π^{p^j}). The single vector of n variables (v) in the LP represents increase from x^j. The mathematical program for factor index i, is therefore:

$$\text{maximize} \quad \frac{1}{2} \sum_{k=1\ldots n} (Q_{i,k} + Q_{k,i}) v_k \tag{4.2a}$$

$$\text{subject to:} \quad v_i \geq 1, \tag{4.2b}$$

$$v_k \leq \overline{x}_k^j - x_k^j, \qquad \forall k = 1 \ldots n \tag{4.2c}$$

$$\ell_k^j \leq g(x^j + v)_k \leq u_k^j, \qquad \forall k = 1 \ldots m \tag{4.2d}$$

$$f(x^{p^j}) + \pi^{p^j \top}(x^j - x^{p^j}) + \sum_{k=1\ldots n} \pi_k^{p^j} v_k \geq \underline{f}^j, \tag{4.2e}$$

$$v \in \mathbb{R}_+^n. \tag{4.2f}$$

The optimal objective function of this LP is a single coefficient corresponding to the quadratic factor of index i, representing the best that coefficient can achieve. This LP can be computed at for each variable where incrementing by 1 remains feasible at the current node, allowing for tighter bounds as the tree of solutions is searched. The upper-plane coefficient terms from a node to its descendants are not necessarily monotonically decreasing, but are whenever all coefficients in the objective are nonnegative.

The difference for polynomials of degree greater than two is that rather than bound the coefficient contributions to individual variables, the approach instead becomes to bound the contributions that subsets of variables will have, still employing linear programs. Unfortunately, this may result in a significant weakening of the optimality cut implied by the π value of the parent node.

After the upper planes have been established, the formulation for the overall upper bound revolves around upper plane coefficients for each variable as well as upper bounds on each. Using McCormick's envelope by introducing a new variable w for the bilinear

term xy, but only considering the upper-bounding constraints, we obtain two inequalities:

$$w \leq x^U y + x y^L - x^U y^L, \tag{4.3a}$$

$$w \leq x y^U + x^L y - x^L y^U. \tag{4.3b}$$

Notationally, the choice of w is because for higher order polynomials, it is an indexed set of variable factors (in the quadratic case simply variables), and y is a set of linear contributions for each, thus the TSA can upper bound the nonlinear terms in a polynomial optimization problem by summing over w. This envelope can be reduced to the nonzero terms as we are reasoning over descendants, so variable lower bounds and factor lower bounds can be assumed to be zero:

$$w \leq z^U y, \tag{4.4a}$$

$$w \leq z y^U. \tag{4.4b}$$

The final z^j-upper-bounding formulation given upper-plane values π is a single integer

linear program (ILP) that, for the quadratic case, becomes:

$$\text{maximize} \quad f(x^j) + \sum_k c_k z_k + w_k \tag{4.5a}$$

$$\text{subject to:} \quad \ell_i^j \leq g(z + x^j)_i \leq u_i^j, \qquad \forall i = 1 \ldots m \tag{4.5b}$$

$$w_i \leq (\overline{x}^j - x^j)_i y_i, \qquad \forall k = 1 \ldots n \tag{4.5c}$$

$$w_i \leq z_i \pi_i^j, \qquad \forall k = 1 \ldots n \tag{4.5d}$$

$$y_i = \sum_{k=1 \ldots n} \frac{Q_{i,k} + Q_{k,i}}{2} z_j, \quad \forall k = 1 \ldots n \tag{4.5e}$$

$$z_i \leq \overline{x}_i^j - x_i^j, \qquad \forall k = 1 \ldots n \tag{4.5f}$$

$$z \in \mathbb{Z}_+^n, \tag{4.5g}$$

$$w, y \in \mathbb{R}_+^n. \tag{4.5h}$$

To show how this approach works for higher order problems, consider an objective: $x_1x_2x_3 + 2x_2x_3x_4 + x_1$. This objective would be separated into parts for each variable occurring in at least one nonlinear term, and the parts would be factored over the occurrences with at least 1 variable, divided by the number of variables involved. The parts would be $x_2x_3(x_1 + 2x_4)$, $x_1x_2(x_3)$, $x_3x_4(2x_2)$, $x_2x_4(2x_3)$, and $x_1x_3(x_2)$, all divided by 3. In this way, there would be linear terms such that an LP could be computed to determine each π_i. Because upper bounds are only applied on optimal potential values of x for descendants of a certain node, optimality constraints from the π values of the parent node (using past upper planes) and the cuts in Section 4.2.2 are also applied when finding these upper bounds via the above ILP formulation. The upper bound z_i^U applies to the product of variables that z_i is equal to.

The final z^j-upper-bounding formulation for the general polynomial case becomes:

$$\text{maximize} \quad f(x^j) + \sum_k c_k v_k + w_k \tag{4.6a}$$

$$\text{subject to:} \quad \ell_i^j \leq g(x^j + v)_i \leq u_i^j, \quad \forall i = 1 \dots m \tag{4.6b}$$

$$v_i \leq \overline{x}_i^j - x_i^j, \quad \forall i = 1 \dots n \tag{4.6c}$$

$$y_i = \sum_{k \in R_i} \frac{d_k}{|R_i'| + 1} v_j, \quad \forall i \in \mathcal{I} \tag{4.6d}$$

$$w_i \leq s_i^U y_i, \quad \forall i \in \mathcal{I} \tag{4.6e}$$

$$w_i \leq s_i \pi_i^j, \quad \forall i \in \mathcal{I} \tag{4.6f}$$

$$s_i = \prod_{r \in R_i'} v_j, \quad \forall i \in \mathcal{I} \tag{4.6g}$$

$$v \in \mathbb{Z}_+^n \tag{4.6h}$$

$$s, w, y \in \mathbb{R}_+^{|\mathcal{I}|}. \tag{4.6i}$$

The index sets R and R' are used to consider all products of variables and linear factors occurring in the nonlinear terms of the objective function respectively, and $\mathcal{I}$ is an index set over these pairs of sets. The vector c contains the coefficient terms for linear variables, d contains the coefficient terms for the linear factors of nonlinear terms, and π contains the coefficient terms for nonlinear factors derived from upper planes.

4.1.4 Tree Search Algorithm: Initialization

As in prior work [7, 10, 38], assume that there is a known $m-$region lower bound on all feasible $g(x)$, and without loss of generality set this bound to be 0 for all x^j. In at least the case of g as a linear operator, $x = \mathbf{0}$ implies $Ax = \mathbf{0}$. To that end, the TSA initializes its data structure with a set of nodes it knows to contain all optimal vectors as descendants, the zero vector in $\mathbb{Z}^n$.

Having access to strong lower bounds across the entire space of right-hand sides is particularly important to the TSA. To start with a strong set of heuristic x values, we

adapt the approach of Fomeni and Letchford [39] for the initialization step of the TSA. Figure 4.3 details the pseudocode for this procedure that takes inspiration from classical exact Dynamic Programming approaches to solving integer knapsack problems. This approach, which exactly solves the problem in the linear version, is stronger than other heuristic techniques because it populates the entire space of right-hand sides up to U with reasonable heuristic value function z^j-lower bounds.

The TSA iterates in a preset order (in our case, according to $w-$weight from low to high) over the variables and attempts to add increments of the variable to the existing heuristic set. While in the process of considering these variables, multiple increments to the same index may be considered, so a set C is used to denote the vectors currently under consideration. It immediately prunes (a valid strategy for linear objectives) if a stored heuristic x weakly dominates the x^ν (ν for new) that it is considering. In the case where $g(x^\nu) = g(a), f(x^\nu) = f(a), a \in S^{heur}$, the TSA proceeds with the vector that has the greater sum $\sum_{\imath \in n} x_n$. Lastly, it makes a pass over all x in the heuristic set and removes the weakly-dominated vectors not in E (the efficient frontier).

Initialization
(1) $S^{heur} \leftarrow \{\mathbf{0} \in \mathbb{Z}^n\}$
(2) **For** $i \leftarrow 1$ **to** n **do**
(3) $\quad C \leftarrow S^{heur}$
(4) $\quad$ **While** $C \neq \emptyset$:
(5) $\quad\quad$ **For** $x \in C$:
(6) $\quad\quad\quad x^\nu = x + e_i$
(7) $\quad\quad\quad$ If x^ν is weakly nondominated by nodes in S^{heur}:
(8) $\quad\quad\quad\quad C \leftarrow C \cup x^\nu, S^{heur} \leftarrow S^{heur} \cup x^\nu$
(9) $\quad\quad\quad C \leftarrow C \setminus x$
(10) Remove dominated solutions from S^{heur}
(11) Return S^{heur}

Figure. 4.3: Pseudocode for the Initialization step of our main loop

4.1.5 Tree Search Algorithm: Proof of Correctness

To prove correctness of the TSA, two elements are required. First, the TSA must terminate in a finite amount of time. This can be seen as it operates without backtracking on the set of integer vectors of finite and bounded w-weight, and at-worst performs exhaustive

breadth first search and finite-time (with NP-hard ILP solves) operations at each node. The main loop increments the minimum w-weight after each pass by a nonzero amount. Every w-weight is always strictly less than $\sum_{i \in 1 \ldots m} U_i$.

Second, upon termination, the TSA must yield the correct result as output. In this case, the correct result implies that the TSA has certified that all vectors in the minimal R^* tree data structure, storing S^{opt}, are (level-set) optimal and that no vectors have been omitted. To certify that the vectors are optimal, the condition for entry into the set S^{opt} is weak nondominance, so initializing with the zero vector as a base case, all vectors returned are optimal by induction. To prove that no vectors that are potentially optimal are omitted or ignored, it suffices to show that all pruning only occurs with a valid independent certificate of weak-dominance. Each potential source of pruning or $m-region$ bounds adjustment is guaranteed to retain points that are not weakly dominated. Because the TSA applies dominance in a total ordering defined by order of entry into the data structure (once entered, no further dominance checks are performed), two nodes mutually pruning each other on the basis of equality is impossible.

4.2 Tree Search Algorithm: Performance Enhancements

Several useful performance enhancements, while not strictly necessary for correctness, are employed to improve the run time efficiency of the TSA to solve problems of nontrivial size.

4.2.1 Enforcing Pareto Monotonicity

The *Enforcing Pareto Monotonicity* (EPM) method is how the TSA uses existing optimal vector and heuristic lower bound information to reduce the range of right-hand sides and potentially prune nodes in the search process. By noting that the existing feasible solution x^j provides a lower bound on the value function for $\beta \geq \beta^j$, the TSA can remove portions of u^k for a node k that are guaranteed to be dominated given the value function upper bound $\overline{f}^k$ of the node. Specifically, if $\overline{f}^k \leq f^j$ and $\beta^j \leq \beta^k$ in all but one dimension, the

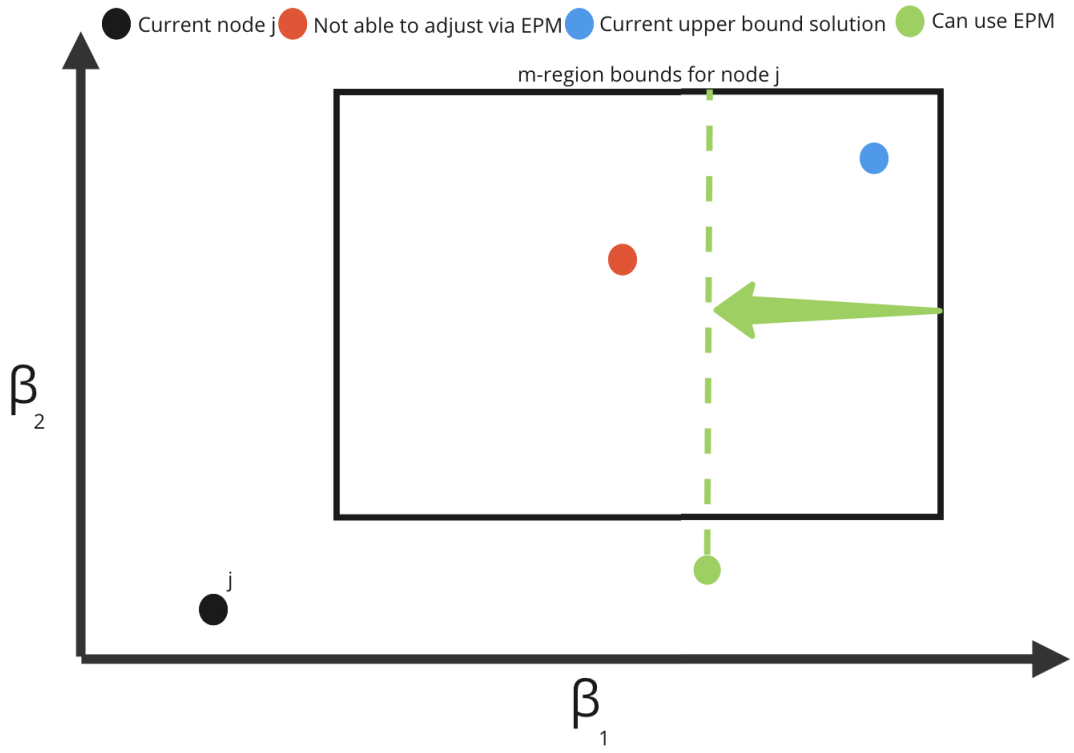

Figure. 4.4: Illustration of the *Enforcing Pareto Monotonicity* procedure. The space depicted is a $m = 2$-dimensional constraint space. The point in black is the current node, considering a smaller box $\mathcal{B}(min(\ell^j, L), u^j)$ of potentially LSO descendants. Green is a node providing a value function lower bound that is able to change the u^j value through *Enforcing Pareto Monotonicity*, while red is not. Blue illustrates how this might eliminate a solution of the z^j-upper bounding ILP, prompting another ILP solve.

TSA has a certificate that the constraint in that dimension is slack past the attained value in β^j for d^k.

4.2.2 Cuts and $m-$region Bounds Adjustment

This thesis introduces a useful cut for solving INLPs based on dominance. The cut applies in particular to polynomial objectives by repeated application of the binomial theorem that states for scalar x and z:

$$(x+z)^p = \sum_{k=0}^{p} \binom{p}{k} (x)^k z^{p-k} = (x^j)^p + \sum_{k=0}^{p-1} \binom{p}{k} (x)^k z^{p-k}. \tag{4.7}$$

The TSA uses the fact that $\mathbb{Z}_+^n$ is closed under addition to quantify exclusively over integer $x^j + z$ values. By inspecting the leading term one can see the contribution of $(x)^p$

is independent of the value of $z \in \mathbb{Z}_+$ and therefore for a polynomial f, $f(x^j+z)-f(y+z)$ is of order at most one less than f for any pair $x^j \in \mathbb{Z}^n_+, y \in \mathbb{Z}^n_+$. Consider a node with a given value-function lower bound associated with variable values x^j. If $f(x^j) \leq f(y), g(x^j) \geq g(y), x^j \neq y$, and $g(y+z) \leq g(x^j+z)$ then one can add a cut to all mathematical programs relevant to node j that indicates,

$$\text{f(y+z)} \leq f(x^j + z),$$
$$z \leq \overline{x}^j - x^j,$$

$z \in \mathbb{Z}^n_+$. This cut requires that all level-set optimal solutions (in Θ) descending from node j must not be dominated by the same increment applied to node y by which it is currently dominated, so long as those descendants remain below $g(x^j + z)$, logically entailed by the definition of a level-set optimal solution. Notably for linear g the requirement that $g(y + z) \not\leq g(x^j + z)$ always holds.

To illustrate how these cuts work, consider the following example with three variables. Let

$$f(x) = 10x_1 + 5x_2 + 7x_3 + 3x_1x_2 + 4x_1x_3 + 6x_2x_3 + 2$$

and

$$g(x) = \begin{bmatrix} 1x_1 + 2x_2 + 1x_3 \\ 1x_1 + 1x_2 + 2x_3 \end{bmatrix}$$

The dominated and nondominated values $x_i \in [0, 2]\ \forall\ i$, can be seen in Figure 4.5.

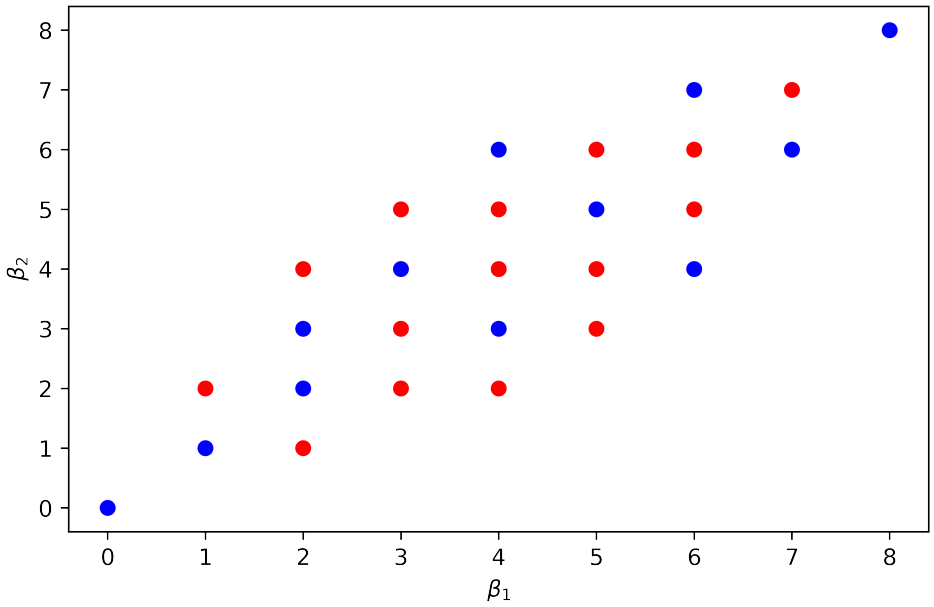

Figure. 4.5: Illustrated above are the weakly-dominated statuses of solutions corresponding to all x-values with coordinates between 0 and 2 depicted in the space of their right-hand sides. Blue indicates nondominated, red indicates dominated.

The right-hand side vectors that are dominated and are therefore not in S^{opt} are shown with a red circle whereas those that are not dominated (LSO) are in S^{opt} and are shown with blue circles. An example cut for the point $a = (0, 0, 1)$ with $f(a) = 9$ would use the nondominated S^{opt} point $b = (1, 0, 0)$ with $f(b) = 12$. The cut would therefore be for the descendants of the node with $x^j = a = (0, 0, 1)$, $f(a+z) - f(b+z) \geq 0$ which is equivalent to $-3 + 4z_1 + 3z_2 - 6z_3 \geq 0$. Note how the linear and constant terms naturally cancel and all that remains is a linear cut based upon the quadratic terms. This cut indicates that at least one of z_1, z_2 must be nonzero to yield a LSO solution. Additionally increasing z_3 requires a higher contribution to z_1 or z_2. If z_1, z_2 are already fixed to zero, the node j can be pruned because the inequality will never be satisfied, making the z^j-bounding ILP infeasible.

For problems that have variable bounds, the conditions to apply the cut comparing a point from $(x^j + z) \in d^j$ to another point $y+z$ may fail to hold if the increment moves $y+z$ outside of the variable bounds. To ensure the cut remains valid, the coefficient applied to the relevant variable indices (where $x_i^j < y_i^j$) can be changed. Specifically, the cut can

be made valid by making the inequality always hold if $z_i + y_i^j$ is greater than the global variable bound in that dimension. Let the coefficient terms applied to z in the inequality generated by the cut be q. The coefficient is changed to the maximum of its current value and the ratio of $f(y) - f(x^j) - \sum_{i \in 1 \ldots n} \overline{x}_i^j min(q_i, 0)$ and the distance to the global variable upper bound at that index for y plus one. This way, when $y + z$ surpasses the global upper bound value, the constraint is always satisfied because it contributes a positive amount to the left-hand side that is guaranteed to be large enough by simple variable bounds logic.

Because these cuts certify that vectors on the infeasible side of the cut are dominated, these cuts will never eliminate an optimal vector, making them valid to apply to all mathematical programs relevant to node j.

4.2.3 $m-$region Bounds Adjustment

When a node is dominated, its lower and upper bounds can also be adjusted by solving an integer linear program in each dimension of β. This adjustment is performed by optimizing (maximizing or minimizing) the value that β_i can attain over d^j for any feasible increment ($z \leq \overline{x}^j - x^j$) while satisfying both the cut constraints and the constraint that $\overline{f}^j \geq \underline{f}^j$. For $m-region$ bounds adjustment, it is important that the procedure be fast, so rather than using the original full z^j-upper-bounding formulation with the McCormick relaxation approach as part of the constraints, the TSA instead just uses the classical single constraint upper plane approach.

The $m - region$ bounds adjustment approach therefore become $2m$ ILPs: one for

maximization, one for minimization per dimension:

$$\begin{aligned}
\text{maximize/minimize} \quad & g(x^j + z)_i && && (4.9a)\\
\text{subject to:} \quad & \ell_i^j \leq g(x^j + z)_i \leq u_i^j && \forall i \in 1 \dots m && (4.9b)\\
& z_k \leq \overline{x}_k^j - x_k^j && \forall k \in 1 \dots n && (4.9c)\\
& w + \sum_{k=1 \dots n} c_k (x^j + z)_k \geq \underline{f}^j && && (4.9d)\\
& w_p = \pi_p \prod_{r \in J'_p} (x^j + z)_r && \forall p \in \mathcal{I} && (4.9e)\\
& z \in \mathbb{Z}_+^n && && (4.9f)\\
& w \in \mathbb{R}^{|\mathcal{I}|} && && (4.9g)
\end{aligned}$$

The cuts implied by dominating vectors are of order one less than the objective for polynomial problems. By optimizing for the minimal and maximal values that the coordinates of the constraint can achieve, the bounds are sequentially adjusted. Because this process does not remove feasible descendants from a shared set, the ordering is immaterial and all solves can be completed in parallel. After a pass of $m - region$ bounds adjustment, the bounds may have changed, which indicates that a new round of bounding can be performed as detailed in Section 4.2.6. If the $m - region$ bounds adjustment problem is infeasible, it indicates that the integrality constraint imposed is sufficient to show that no optimal vectors can be found among the descendants of node j.

4.2.4 Extending EPM to Pairs of Variables

When both ILP and LP forms of the z^j-upper bounding mathematical program are computed, yet the bounds cannot be changed by the other techniques, an additional set of cuts reflecting trade-offs in the value function lower bound between the right-hand side terms can be added and the bounds recomputed. The logic behind this is similar to *Enforcing Pareto Monotonicity*. Consider an LSO solution at coordinate $c \in \Theta$, recalling that Θ is the set of all LSO points in $\overline{m}-$region space. Because the value function is nondecreasing,

if $z(c) \geq \overline{f}^j$, then $min(\ell^j, L) \leq c \leq u^j$, $z(b) \geq f(x^k)$, $\forall\ c \leq b \leq u^j, k \in d^j$ proving nodes in d^j are not LSO by the same logic used in EPM. Because the intersection over the inequalities in right-hand side space are nonconvex for $c_i \geq min(\ell^j, L)_i$ for two or more dimensions, a convexification is necessary for its actual application to the mathematical programs as a linear constraint. This convexification is achieved by considering the convex hull of the union of regions where exactly two dimensions are strictly within $\mathcal{B}(min(\ell^j, L), u^j)$, sorting in one dimension and finding the convex hull. To accomplish this the MR*T data structure is queried for all points from which these cuts may be constructed. The union of the nonconvex regions form a staircase pattern that features a straightforward convex hull. These two-dimensional cuts in $m-$region space are of the form:

$$a_1\beta_i + a_2\beta_j - a_3 \leq 0. \tag{4.10}$$

After generating the cuts, the point solving the z^j upper-bounding ILP is checked for inclusion in the cut region by iterating over each inequality. If it is contained, then the cuts are added to all mathematical programs relevant to node j and the bounds are recomputed. Because this process is expensive, it is only attempted if all other methods of changing the ℓ^j, u^j bounds or otherwise pruning the node fail to succeed. Additionally because of the combinatorial potential for the number of points generated by this process, it is limited to two dimensions to make the number of generated inequalities linear in the number of nonconvex regions. These advanced upper bounding cuts are illustrated visually in Figure 4.6.

4.2.5 Reusing Computation

The initialization step first computes and stores submatrices for fast access to row information for all n variable subsets that will be considered. The TSA uses reduced costs for the variable lower and upper bounds to avoid branching into areas the TSA has proven to be suboptimal.

Lastly, a binary search procedure is used at each node to eliminate entire suffixes that

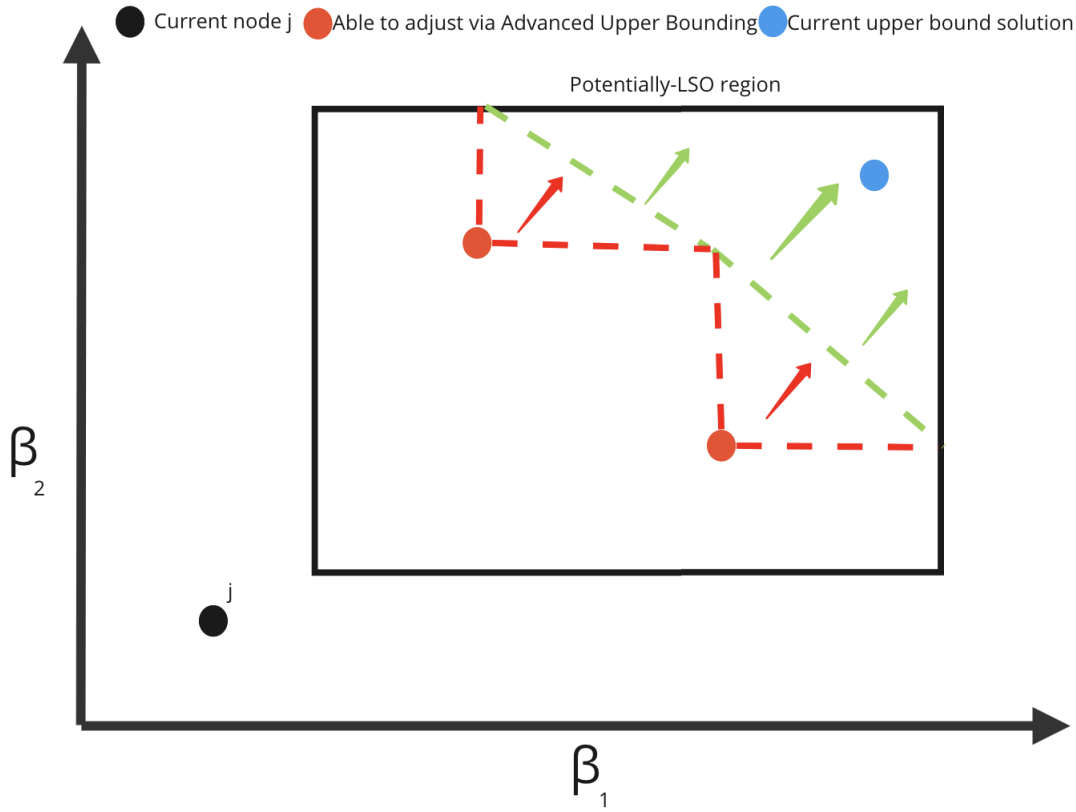

Figure. 4.6: Illustration of the Extended EPM procedure. The space depicted is a constraint space of dimension $m = 2$. The point in black is the current node, considering a smaller box $\mathcal{B}(min(\ell^j, L), u^j)$ of potentially-optimal descendants. The red points cannot be used in standard EPM but can be used in the Extended EPM. The red dashed lines are the strongest monotonicity-implied nonconvex cut, relaxed to the green dashed lines to successfully cut the region containing the ILP solution in blue.

are guaranteed to be immediately prunable without computing upper bounds for each separately before adding the children to the search tree. It does this by noting that each suffix contains all later suffixes. Therefore, if an upper bound is computed on one of these nested sets and it is immediately prunable, all sets that are strictly more constraining than it are also able to be pruned. Because the variables are ordered by least promising nodes first, the variable fixing order is reversed for this binary-search procedure. All branches that are pruned are prefixes of the remaining nodes, and therefore do not have their upper bounds changed to 0 upon branching in the usual way, but are instead deferred for consideration in their descendants nodes.

4.2.6 Algorithmic Description with All Enhancements

The main loop of the TSA with all enhancements is designed so as to efficiently leverage their respective strengths. To show how this works, a new pseudocode description of the routine with all algorithmic enhancements is provided.

There are significant additions to the pruning subroutine that naturally follow from the described enhancements. The subroutine first considers the LP relaxation of the upper bound described in the z^j-Upper Bounds in Section 4.1.3. The LP relaxation of the z^j-upper bounding ILP gives reduced costs (δ_i on variables $\beta_i, \forall i = 1 \dots m$) that, when paired with a value function lower bound $\underline{f}^j$ gives updated bounds on ℓ^j and u^j as:

$$\forall i : \beta_i = u_i^j, \delta_i > 0 \qquad \ell_i^j := max\left\{ \ell_i^j, u_i^j - \frac{\overline{f}^j - \underline{f}^j}{\delta_i} \right\}$$

$$\forall i : \beta_i = \ell_i^j, \delta_i < 0 \qquad u_i^j := min\left\{ u_i^j, \ell_i^j - \frac{\overline{f}^j - \underline{f}^j}{\delta_i} \right\}$$

The reduced cost gives a lower bound on how much the upper bound would decrease if non-slack variable bounds were to change. Therefore, the upper bound would reduce by at least that much, independently of other variable bounds. Combining this fact with the knowledge of a lower bound gives a potentially new variable upper bound.

The same reduced cost approach is used to find new upper bounds for the variables,

Main Loop
(1) INITIALIZE THE FIRST NODE IN S TO $x^0 = \mathbf{0} \in \mathbb{Z}^n$
(2) **While** S is not empty:
(3) Consider the next node by weight j, breaking ties by larger f^j value, removing it from S
(4) Collect all y nodes $f(x^j), g(x^j)$ into a set D
(5) Perform any problem-specific updates to ℓ^j, u^j
(6) **If** $|D| \neq \emptyset$:
(7) Create the dominating cuts from D, x^j
(8) Attempt to prune j, if not pruned continue to (13) otherwise go to (2)
(9) **Else if** x^j is feasible:
(10) Add x^j to S^{opt}
(11) **Else if** x^j is not feasible but otherwise nondominated:
(12) Add x^j to the heuristic set S^{heur}
(13) **If** $\beta^j \not\leq min(\ell^j, L)$:
(14) Attempt to prune j, if not pruned continue to (15) otherwise go to (2)
(15) Perform binary search to prune from consideration the largest possible prefix of j at the current node j
(16) Add the remaining children of j to S
(17) Incorporate optimal feasible heuristic solutions in S^{heur} into S^{opt} by iterating over heuristic solutions and checking against the MR*T
(18) Return S^{opt}

Figure. 4.7: The main loop procedure with all algorithmic enhancements.

leading to tighter heuristic upper bounds by virtue of a tighter McCormick relaxation, the technique underlying the upper bounds calculation. The updated bounds on β are then also updated $m-$region bounds and are used for a lookup in the data structure to find the strongest lower bound for the value of all potential optimal vectors (searching for the maximal $f(x^k)$ with $g(x^k) \leq max(\ell^j, L)$). If the node cannot be immediately pruned, *Enforcing Pareto Monotonicity* is performed. If neither the most recent iteration of relaxed z^j-upper-bounding nor the *Enforcing Pareto Monotonicity* adjusted the bounds, the TSA proceeds either to $m-region\ Bounds\ Adjustment$ for dominated vectors or directly to the strengthened ILP bound computation. After the ILP bound computation followed by *Enforcing Pareto Monotonicity* does not produce an adjusted bound, the pruning

terminates. If at any point a z^j-bounding or $m-region$ bounds adjustment LP or ILP is infeasible or the upper bound is dominated by the lower bound found by lookup in the data structure, the node is pruned.

The pruning pseudocode appears in Figure 4.8.

Pruning
(1) **Do**:
(2) repeatFlag = False
(3) Perform $m-$region Bounds Adjustment if applicable
(4) **If** node j can be pruned by value function lower bound
(5) **Prune and exit**
(6) Compute LPs for the π upper planes if uncomputed or u^j has decreased
(7) Compute LP relaxation for the upper bound
(8) **If** node j can be pruned by value function lower bound
(9) **Prune and exit**
(10) Adjust coefficients and bounds using reduced costs
(11) **If** ℓ^j has changed for node j:
(12) repeatFlag = True
(13) Compute ILP form of the z^j-upper bound
(14) If the ILP yields a nondominated vector, add it to S^{heur}
(15) **If** Enforcing Pareto Monotonicity changes u^j, or u^j has been changed by reduced costs:
(16) repeatFlag = True
(17) **If** repeatFlag = False and the node can be pruned by advanced $m-$upper bounding
(18) **Prune and exit**
(19) **If** repeatFlag = False and advanced upper bounding yields new cuts
(20) Add new cuts to formulations for all mathematical programs for node j
(21) repeatFlag = True
(22) **While** repeatFlag
(23)**Exit without pruning**

Figure. 4.8: The pruning procedure.

The pruning algorithm works iteratively. On lines 1-2 it sets the stage: if it successfully moves the bounds, either the value function lower bound may lead to tighter optimality cuts or the upper bound has made it more difficult to have high x values among its descendants. In either situation, the loop is run again. Line 3 is where the TSA performs

$m - region$ Bounds Adjustment (BA), detailed in Section 4.2.2, which is how the TSA first attempts to reduce the gap between ℓ^j and u^j. Lines 4-5 check to make sure that the BA did not shift the lower bound to a point where the node is now able to be pruned. Line 6-9 are a first attempt at reducing the value function upper bound $\overline{f}^j$, that uses the LP relaxation of the z^j-upper-bounding subproblem (Section 4.1.3) to find a potential way to prune the node. If that fails, line 10 potentially further tightens ℓ^j, u^j and the variable bounds through reduced costs, and if successful indicates that the loop should continue. Lines 13-14 are where the non-relaxed upper bound is used, and if nondominated in the current heuristic set, added to S^{heur}. Lines 15-16 implement *Enforcing Pareto Monotonicity* (Section 4.2.1) that potentially changes the upper bounds and also considers whether the bounds have changed due to reduced costs from the LP relaxation. Lines 18-22 again check the bounds, and if the loop would exit, also recomputes upper planes and the ILP using cuts derived from Section 4.2.4 as a last resort. Line 23 is the failure condition where node j has not been eliminated from consideration by the pruning process. Note that the bounds $(\overline{f}^j, \ell^j, u^j, \overline{x}^j)$ are all inherited by the children of node j, thereby conserving computational effort from earlier in the tree.

Chapter 5

Computational Studies

Having described the TSA, this Section is dedicated to demonstrating the usefulness of the TSA. Computational studies include demonstrating the speed of the advanced types of sensitivity analysis queries possible, as well as gaining a better understanding for performance characteristics.

5.1 Computational Setup

The work for the computational experiments was performed on the WPI Turing cluster using 100GB of RAM and a Intel® Xeon® Processor E5-2695 v4 CPU on Red Hat Enterprise Linux Server 7.3 with kernel version 3.10.0-514.x86. As a solver for linear and quadratic programs, Gurobi 9.5 [40] was used. All problems were solved to optimality with an optimality gap tolerance of $1e^{-10}$ (exact optimality).

5.2 Computational Experiments

The lack of a strong dual for integer (nonlinear) programs, absent special problem knowledge, renders prohibitive optimality insights with respect to sensitivity analysis information on resource level changes. The best that can be accomplished is to resolve the integer (nonlinear) program repeatedly. Even in the case of convex optimization problems with

strong duals, post-optimality analysis is typically limited to a small range of values in a single dimension.

The advantage to the methods in this thesis are lightning fast ($O(log(n))$ in practice) retrieval methods for querying exact optimality information for resource (β) vectors for the associated integer nonlinear optimization problem expressed in (4.9). These extreme performance capabilities on hard exact integer nonlinear optimization problems are available once Algorithm 4.8 completes. Thus, we first demonstrate the performance of the MR*T data structure to query exact optimality results in the context of conducting sensitivity analysis in resource levels. The remainder of the experiments relate to the construction of the MR*T data structure using the methodology of Section 4, that after creation, offers lightning fast (in milliseconds) querying over a range of resource levels. In some cases, the creation of the data structure itself exhibits only a modest increase in run time over a single Gurobi solve.

The overall performance of the TSA is demonstrated through mean and standard deviation values across 3 random experiment instances per run, indicated by a unique combination of n, L, U, Problem Type parameters. Problem Types are classified as either binary or integer variables, respectively. Performance ratios against Gurobi solves are computed by comparing the mean runtime of respective experiments and comparing to a single Gurobi solve against the upper U constraint (often the hardest right-hand side to evaluate).

5.3 Sensitivity Analysis

To demonstrate the utility of the MR*T data structure for the purpose of sensitivity analysis, we experiment with understanding the range of the value function along a certain bounded direction λ emanating from a β vector. The choice of a λ vector is to determine if the constraint were to vary from this resource level of interest (λ) along a direction of some trade off (λ), what is the maximum impact possible? To generate these λ directions, first a random direction is chosen where at least one coordinate is positive and at least one

is negative, important so that the result is more than a simple pair of lookups. A scaling amount is then chosen to extend to the edge of the hypercube of β values considered. All sensitivity analysis experiment queries are performed over the $\mathcal{B}(L, U)$ set, setting $L = \mathbf{0}$.

This experiment is equivalent to solving the following optimization problem for a given λ: maximize/minimize $z(\beta + t\lambda)$, $t \in [-1, 1]$. Quickly solving this optimization problem amounts to traversing the MR*T data structure after its incremental construction. This contrasts with any approach that would require solving one or more optimization problems to obtain bounds on directional sensitivity of the value function to changes in its resource levels.

5.4 Problem Instance Generation

All computational experiments consider test instances from two problem classes:

1. Integer Multi-dimensional Quadratic Knapsack (IMDQK)
2. Binary Integer Multi-dimensional Quadratic Knapsack (BIMDQK)

Test instances for both the IMDQK and BIMDQK problem classes are generated using the procedure outlined in Wang et al. [41], with modest modifications to ensure that the problems are tractable, have more predictable profiling times, and avoid unwanted characteristics that have been previously demonstrated in the literature [42]. The Wang et al. [41] instances have nonseparable positive quadratic and linear objective terms and linear knapsack (positive coefficient) constraints defining the resource consumption for selecting a subset of items. The nonlinear nature of the problem arises from the quadratic terms within the objective, which prompt subsets of items to interact with each other in complex ways. This dynamic can result in variables with lower individual contributions ultimately producing a higher cumulative effect when paired together, compared to options that are stronger on an individual basis. Their original formulation has nonnegative resource consumption (A matrix) parameters that are assigned uniformly within some range, right-hand sides that are assigned uniformly between the largest single value for consuming a

resource and the sum over all consumptions, and quadratic and linear objective terms that are chosen according to a uniform distribution from 1 to 100.

The modifications we make to the formulation of Wang et al. [41] are as follows. We first fix all right-hand side levels to a set value and uniformly generate integer resource consumption levels between $\lceil r/2 \rceil$ and r for a small positive integer parameter r. Fixing all right-hand side values prevents the phenomenon where significant variation in computational performance occurs – both for the described approach, as well as for Gurobi. Limited computational testing revealed that larger budgets are intractable even for modest BIMDQK and IMDQK problem sizes for both the described approach and Gurobi, thus the budget is fixed to a value between $4r$ and $5r$ in every dimension; this is treated as a parameter and described alongside the problem instances in the data. Setting resource consumption levels for items in the range described reduces irregular phenomenon, such as that observed in [42] where the optimal approach quickly becomes to fill the knapsack with the least expensive knapsack items, ignoring interaction. Less variation in resource consumption levels for each item yields considerably more challenging and interesting problem instances. The TSA generates a single right-hand side vector that serves as the upper bound; the full set of considered right-hand side vectors ranges from from 0 to U.

5.5 Computational Experiments: Test Instance Parameters

For all value function and single right-hand side problems this study considers a full-factorial design to understand the impact of the parameters governing the problem instances. The values considered are $n \in \{40, 60, 70\}$, $m \in \{3\}$, and the parameter r to specify the dimensions of the hypercube as $r \in \{5, 10, 15\}$, with $\mathcal{B} = \prod_{i=1}^{m} [a_i, b_i] \cap \mathbb{Z}^m$ where $a \in \{0, 3r, 4r\}$ and $b \in \{4r\}$. This approach to generating problem instances yields a mix of problem sizes that are computationally tractable for both our approach as well as for Gurobi.

Parameter	Symbol	Levels
Number of variables	n	$40, 60, 70$
Upper bound of variable resource consumption	r	$5, 10, 15$
Lower bound of hypercube of interest	L	$0, 3r, 4r$
Upper bound of hypercube of interest	U	$4r$
Problem type		Binary, Integer

Figure. 5.1: Parameters used for generating problem instances.

5.6 Computational Experiments: Performance Metrics

The performance of the TSA across multiple metrics is tested against other solvers that solve for single right-hand sides. One key metric is wall clock time, in seconds, which is set to a maximum of 30 minutes. For this metric it makes sense to compare the performance of the TSA to a single solve of Gurobi. The next is the number of i) linear programs and ii) integer linear programming subproblems solved (4.6) to complete the TSA and find all LSO vectors between L and U. Finally, to understand the performance of MR*T for sensitivity analysis, we consider the worst case time performance over 200 random queries of β vectors in hypercube $\mathcal{B}$.

Chapter 6

Computational Results and Discussion

The results of the computational experiments are now presented, beginning with the performance of the experiments to conduct lookups on sensitivity analysis queries on resource levels, and proceeding to discuss the costs to generate the associated MR*T data structure that enables such lookups. It is first instructive to understand the overall performance of the TSA in relation to the variables studied. To do this, data tables from the aggregate of experiments across both domains are presented and compared. Then the Binary Quadratic Knapsack Problem is considered, showing that the TSA is able in the worst case to incur a cost of no more than 40-times that of a single Gurobi solve, and availing the aforementioned sensitivity analysis information for a variety of resource levels. After this, the Integer Quadratic Knapsack Problem is presented, a domain that reveals more promising results, requiring only around 10 solves to complete the TSA and find all LSO vectors between L and U.

6.1 Sensitivity Analysis

The sensitivity analysis query problem using a given vector is solved on all of the considered computational experiments in less than 0.002 seconds across all $200 \times 3 \times 18$ instances

considered. Because there was no discernable difference in solution time across any choice of problem size, it is likely that constant factors in the performance are dominating the query time and that it would scale to much larger sets of optimal vectors if sufficient time is available for their identification through the TSA. As a result, this can be considered a real-time sensitivity analysis. The main performance factor was actually strictly the number of items in the $S^{opt} \cup S^{heur}$ for the experiment in question, as might be expected from a lookup approach.

6.2 Overall Performance

The overall performance of the TSA with respect to the number of unfathomed nodes is considered as a function of the number of variables n, the lower bound L, and the upper bound U. As discussed in Section 4.1, the TSA and its enhancements seek to efficiently construct the MR*T by iteratively generating nodes and their children, while aggressively eliminating nonpromising search space regions of children wherever possible. Naturally the performance tends to scale at least linearly with the number of generated nodes. One metric to consider is the number of dominated and yet to be pruned nodes. While these nodes would be simple to prune for the linear case, the added complexity of nonlinearity may lead to their retention even after all of the effort placed into attempting their removal by EPM and $m-$region Bounds Adjustment as in Sections **??** and 4.2.3.

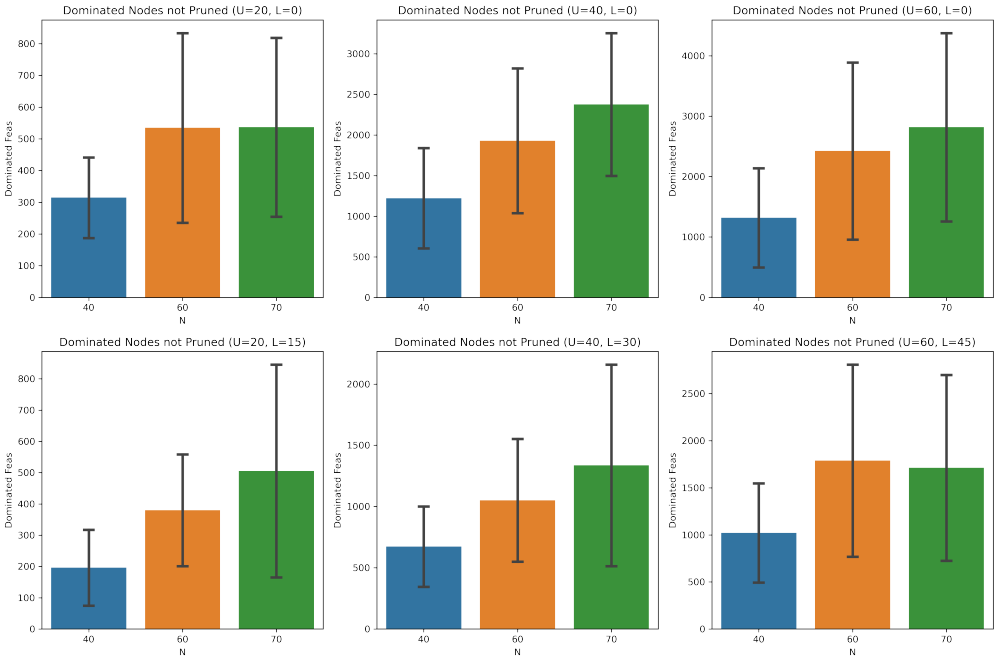

Figure. 6.1: A comparison of the number of dominated nodes that are still considered and not pruned across n values for various levels of L and U.

As can be seen, with added problem complexity, more of these nodes must be considered and as a result it will later be shown that performance increases. Still, the growth is below the cubic amount by which the size of the right-hand side region increases, demonstrating the benefit of sharing computation among these solves. The next most important internal variable is the size of the data structure containing only the points identified to be a minimal superset of Θ, not including infeasible and heuristic points later shown to be suboptimal. The number of points in this structure shows the minimal number of solves a solver would need to complete to represent the value function over $\mathcal{B}(L, U)$.

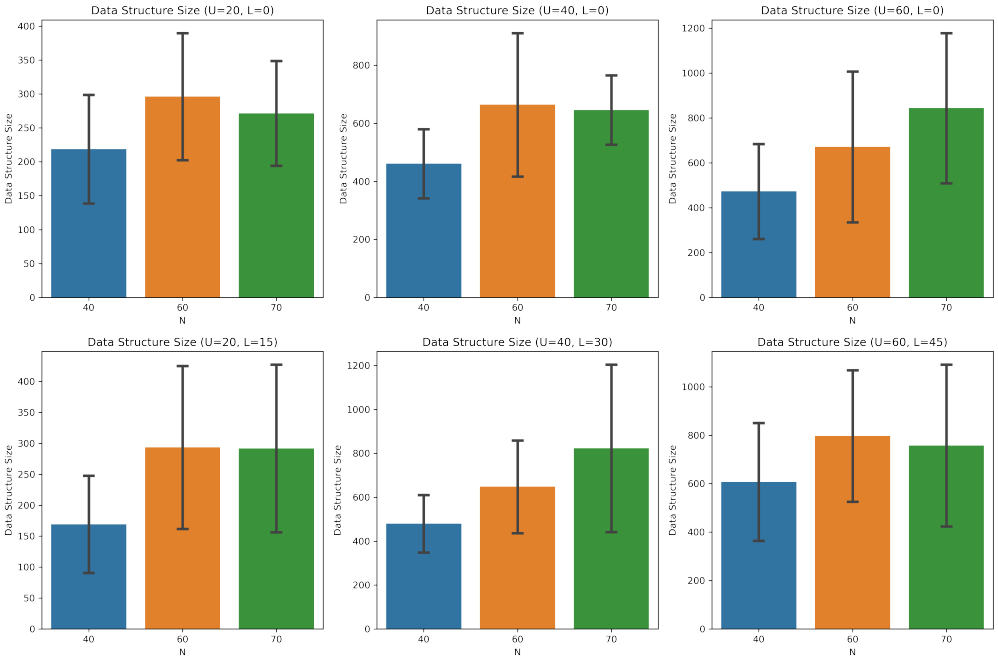

Figure. 6.2: A comparison of the size of the data structure that stores a superset of S^{opt} across U-values and n values.

Of interest is the number of elements in the data structure that are relatively indifferent to the number of variables n and size of $\mathcal{B}(L, U)$. This suggests that a value function lower bound may actually be being constructed for a larger space than just $\mathcal{B}(L, U)$, while certifying $\mathcal{B}(L, U)$ as optimal.

6.2.1 Performance Tables

The performance tables for the problem sizes are now reported. Here, n is the number of variables, m is the number of constraints, time to complete the solve is shown, the number of subproblems (nodes for Gurobi and (I)LP-subproblem solves total for the proposed approach), L, U and problem type are all self-explanatory.

Solver	n	m	L	U	Problem Type	Time (s)± stdev	(I)LP solves±stdev
Gurobi	40	3	20	20	Binary	31.95 ± 20.84	2.85e+04 ± 2.34e+04
Gurobi	40	3	20	20	Integer	8.42 ± 1.04	2.95e+05 ± 5.42e+04
Gurobi	40	3	40	40	Binary	29.88 ± 2.43	3.65e+04 ± 8.77e+03
Gurobi	40	3	40	40	Integer	11.65 ± 4.98	3.87e+05 ± 1.66e+05
Gurobi	40	3	60	60	Binary	17.93 ± 17.08	1.78e+04 ± 7.23e+03
Gurobi	40	3	60	60	Integer	8.35 ± 1.51	4.06e+05 ± 9.63e+04
Ours	40	3	0	20	Binary	53.95 ± 7.07	3.60e+04 ± 2.64e+03
Ours	40	3	0	20	Integer	139.16 ± 28.81	1.40e+05 ± 1.71e+04
Ours	40	3	0	40	Binary	121.08 ± 15.07	1.18e+05 ± 1.15e+04
Ours	40	3	0	40	Integer	509.21 ± 116.96	5.63e+05 ± 1.50e+05
Ours	40	3	0	60	Binary	59.90 ± 10.81	9.97e+04 ± 7.74e+03
Ours	40	3	0	60	Integer	279.82 ± 49.35	6.20e+05 ± 1.26e+05
Ours	40	3	15	20	Binary	23.83 ± 5.84	2.78e+04 ± 3.87e+03
Ours	40	3	15	20	Integer	57.52 ± 27.26	1.02e+05 ± 3.81e+04
Ours	40	3	20	20	Binary	8.67 ± 1.34	8.10e+01 ± 0.00e+00
Ours	40	3	20	20	Integer	3.26 ± 0.64	4.06e+03 ± 1.24e+03
Ours	40	3	30	40	Binary	61.54 ± 13.73	6.67e+04 ± 1.71e+04
Ours	40	3	30	40	Integer	241.59 ± 42.80	3.23e+05 ± 5.87e+04
Ours	40	3	40	40	Binary	8.29 ± 1.98	1.53e+02 ± 1.25e+02
Ours	40	3	40	40	Integer	3.38 ± 0.95	3.26e+03 ± 1.83e+03
Ours	40	3	45	60	Binary	63.74 ± 9.73	8.35e+04 ± 2.06e+04
Ours	40	3	45	60	Integer	227.82 ± 58.60	4.55e+05 ± 1.27e+05
Ours	40	3	60	60	Binary	9.25 ± 1.90	8.10e+01 ± 0.00e+00
Ours	40	3	60	60	Integer	3.39 ± 0.90	3.08e+03 ± 1.04e+03
Ours	40	4	0	20	Binary	35.50 ± 4.04	7.10e+04 ± 1.18e+04

Figure. 6.3: A comparison of the runtimes of the proposed approach to Gurobi for solving all instances of the smallest size, n=40.

For the smallest problems ($n = 40$) the performance table shows a trend that continues for later tables: the number of subproblems is similar but typically smaller for the proposed approach than for Gurobi. This may mean that the value function information is being used effectively to reduce this number to more reasonable amounts. As can be seen in the time column, on the high end just over nine minutes can be expected to solve these small subproblems that take Gurobi on the order of 30 seconds for a single resource level for the binary case and just under ten seconds for the integer case. The setting of U appears to have mixed impact for these problems.

Solver	n	m	L	U	Problem Type	Time (s)± stdev	(I)LP solves±stdev
Gurobi	60	3	20	20	Binary	30.24 ± 4.11	1.93e+06 ± 2.78e+05
Gurobi	60	3	20	20	Integer	62.79 ± 5.25	2.07e+06 ± 2.30e+05
Gurobi	60	3	40	40	Binary	58.76 ± 1.89	3.13e+06 ± 1.90e+05
Gurobi	60	3	40	40	Integer	132.50 ± 18.07	4.18e+06 ± 6.11e+05
Gurobi	60	3	60	60	Binary	66.67 ± 8.10	2.57e+06 ± 9.83e+04
Gurobi	60	3	60	60	Integer	95.53 ± 14.80	3.37e+06 ± 5.46e+05
Ours	60	3	0	20	Binary	120.54 ± 16.07	1.34e+05 ± 7.57e+03
Ours	60	3	0	20	Integer	431.66 ± 75.02	5.91e+05 ± 5.27e+04
Ours	60	3	0	40	Binary	372.01 ± 72.60	4.65e+05 ± 6.53e+04
Ours	60	3	0	40	Integer	1551.68 ± 308.78	1.98e+06 ± 2.42e+05
Ours	60	3	0	60	Binary	253.84 ± 40.50	4.19e+05 ± 2.32e+04
Ours	60	3	0	60	Integer	1659.25 ± 196.51	2.43e+06 ± 1.61e+05
Ours	60	3	15	20	Binary	162.26 ± 99.41	1.19e+05 ± 5.10e+04
Ours	60	3	15	20	Integer	352.44 ± 57.22	4.52e+05 ± 3.16e+04
Ours	60	3	20	20	Binary	20.99 ± 3.96	1.21e+02 ± 0.00e+00
Ours	60	3	20	20	Integer	4.20 ± 0.26	3.39e+03 ± 1.99e+02
Ours	60	3	30	40	Binary	283.75 ± 47.46	2.62e+05 ± 6.31e+04
Ours	60	3	30	40	Integer	840.35 ± 102.80	1.18e+06 ± 2.69e+05
Ours	60	3	40	40	Binary	22.34 ± 1.71	1.21e+02 ± 0.00e+00
Ours	60	3	40	40	Integer	7.37 ± 2.93	8.74e+03 ± 5.05e+03
Ours	60	3	45	60	Binary	396.09 ± 68.62	3.56e+05 ± 7.51e+04
Ours	60	3	45	60	Integer	1059.59 ± 75.16	1.93e+06 ± 4.55e+05
Ours	60	3	60	60	Binary	15.39 ± 3.96	1.21e+02 ± 0.00e+00
Ours	60	3	60	60	Integer	11.10 ± 4.13	1.21e+04 ± 8.51e+03

Figure. 6.4: A comparison of the runtimes of the proposed approach to Gurobi for solving all instances of the medium size, n=60.

For the medium-sized problems ($n = 60$) the performance table shows that Gurobi actually scales quite well to larger problem sizes, as is the proposed approach. While this could simply be due to only 3 replicates being sampled per run causing a relatively large variation in results, it is worth noting. Nevertheless, this batch of problems is interesting because it shows that even with these "easy" problems that are larger than the smaller problems considered before, the proposed approach remains competitive in relative terms to Gurobi. For the full value function reconstruction problem at $U = 40$ the solution of integer problems is challenging for the proposed approach at over 26 minutes.

Solver	n	m	L	U	Problem Type	Time (s)± stdev	(I)LP solves±stdev
Gurobi	70	3	20	20	Binary	105.63 ± 5.99	3.59e+06 ± 2.77e+05
Gurobi	70	3	20	20	Integer	266.03 ± 69.48	5.13e+06 ± 1.27e+06
Gurobi	70	3	40	40	Binary	141.57 ± 25.29	7.69e+06 ± 1.04e+06
Gurobi	70	3	40	40	Integer	238.22 ± 43.03	8.20e+06 ± 1.47e+06
Gurobi	70	3	60	60	Binary	130.71 ± 21.54	5.82e+06 ± 1.02e+06
Gurobi	70	3	60	60	Integer	649.49 ± 99.37	6.94e+06 ± 9.81e+05
Ours	70	3	0	20	Binary	294.35 ± 21.29	1.81e+05 ± 5.72e+03
Ours	70	3	0	20	Integer	1251.96 ± 220.63	8.52e+05 ± 9.38e+04
Ours	70	3	0	40	Binary	628.52 ± 80.13	9.02e+05 ± 1.21e+05
Ours	70	3	0	40	Integer	2484.88 ± 382.10	3.29e+06 ± 3.43e+05
Ours	70	3	0	60	Binary	624.72 ± 52.11	8.31e+05 ± 1.13e+05
Ours*	70	3	0	60	Integer	3803.14 ± 972.26	4.38e+06 ± 7.63e+05
Ours	70	3	15	20	Binary	241.71 ± 15.85	1.47e+05 ± 3.67e+04
Ours	70	3	15	20	Integer	1285.93 ± 165.47	8.49e+05 ± 1.04e+05
Ours	70	3	20	20	Binary	21.70 ± 2.14	1.41e+02 ± 0.00e+00
Ours	70	3	20	20	Integer	15.04 ± 7.96	1.17e+04 ± 7.42e+03
Ours	70	3	30	40	Binary	318.75 ± 96.85	3.60e+05 ± 1.44e+05
Ours	70	3	30	40	Integer	1642.95 ± 514.06	2.27e+06 ± 6.68e+05
Ours	70	3	40	40	Binary	25.33 ± 5.24	1.41e+02 ± 0.00e+00
Ours	70	3	40	40	Integer	15.17 ± 2.58	2.03e+04 ± 7.78e+03
Ours	70	3	45	60	Binary	541.70 ± 21.01	4.01e+05 ± 2.15e+04
Ours	70	3	45	60	Integer	2921.76 ± 767.62	2.77e+06 ± 1.01e+06
Ours	70	3	60	60	Binary	30.56 ± 1.89	1.41e+02 ± 0.00e+00
Ours	70	3	60	60	Integer	16.76 ± 8.50	7.50e+03 ± 3.16e+03

Figure. 6.5: A comparison of the runtimes of the proposed approach to Gurobi for solving all instances of the large size, n=70, outside of a single instance for $L = 0, U = 60$, denoted by the asterisk.

Lastly, the large problems ($n = 70$) are considered. These problems are notably taking longer than the $n = 60$ and lower cases, lasting over ten minutes with Gurobito solve for the most challenging integer instances. These are the most promising for the proposed approach, indicating that it may actually scale quite well to even higher numbers of variables.

A single instance was unable to complete in under 90 minutes for the TSA, showing the increasing difficulty particularly on the integer problem types. The most challenging solved instance took just under 80 minutes to solve for all right hand sides up to $U = 60$.

6.3 Performance on a Single Right-Hand Side Vector

Because the TSA computes the value function over a range of right-hand side vectors, it is important to study its performance on a single right-hand side range to upper bound the potential performance. If it cannot find the solution for a single right-hand side vector quickly for any problem, there is no chance that making the problem harder would improve the situation. For a single right-hand side, the BIMDQKP problem becomes equivalent to a relaxation of the linear reformulation by [35] (modified with cuts) followed by the checks necessary to ensure that no further branches can achieve a higher value.

Thus a very similar approach is already well-studied in [41], specifically under the name LIN2 with a key difference being that the upper plane coefficient values are computed using noninteger relaxations. However, to understand how the single right-hand side case performed against the current version of Gurobi on the problems under consideration it remains worthwhile to have computational results.

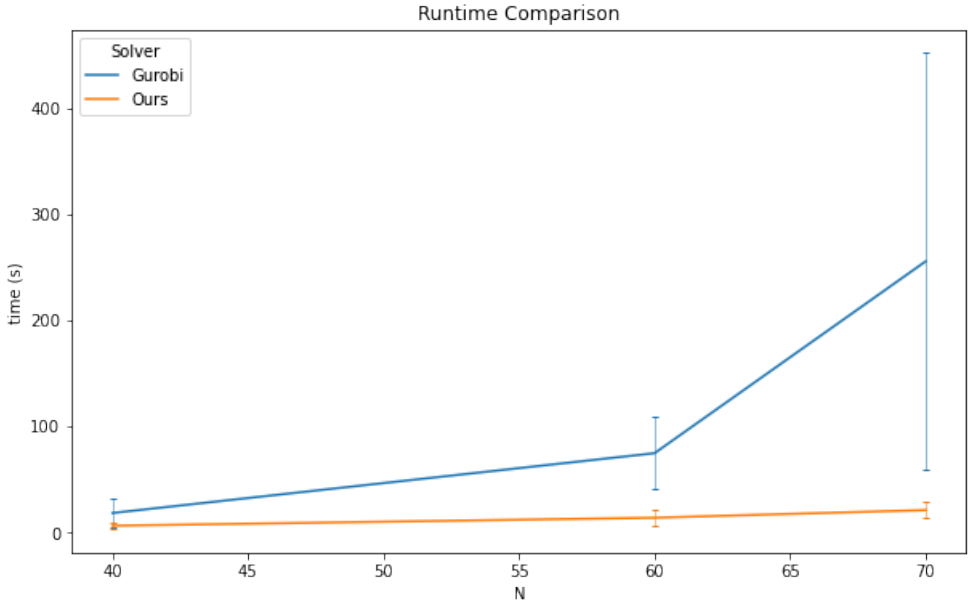

Figure. 6.6: A comparison of the runtimes of the proposed approach to Gurobi for solving the same problem. Although tested on a small number of instances, the TSA outperforms on average across all problem sizes.

6.4 Binary Integer Multi-dimensional Quadratic Knapsack

To understand the performance on specifically the binary problems, performance ratios are presented that indicate how many solves it takes Gurobi to equal the amount of time the proposed approach. Presenting the difference between the U values shows how the performance varies with respect to the volume of the right-hand side region that must be considered.

L	U	Problem Type	Performance Ratio
0	20	Binary	1.69
15	20	Binary	0.75
20	20	Binary	0.27
0	20	Binary	3.99
15	20	Binary	5.37
20	20	Binary	0.69
0	20	Binary	2.79
15	20	Binary	2.29
20	20	Binary	0.21

Figure. 6.7: A comparison of the performance ratios of the proposed approach to Gurobi for solving problems with a given upper bound magnitude (20) across various magnitudes of lower bound for the binary case.

For the problems with a right-hand side upper bound of 20, the performance scales from less than the cost of a single extra solve to find just one right-hand side (direct competition with Gurobi) to just over 5 for all right-hand sides. The intermediate points with $L = 3/4U$ are interesting here and in future comparisons because they show how much of a benefit reducing a majority yet not all of the $\mathcal{B}(L, U)$ region is to performance. Typically the benefit is somewhat understated compared to the shrinkage in volume of the region that it represents, while the feature being available means that there are some gains to be had if some portion of the space can be a priori ignored.

L	U	Problem Type	Performance Ratio
0	40	Binary	4.05
30	40	Binary	2.06
40	40	Binary	0.28
0	40	Binary	6.33
30	40	Binary	4.83
40	40	Binary	0.38
0	40	Binary	4.44
30	40	Binary	2.25
40	40	Binary	0.18

Figure. 6.8: A comparison of the performance ratios of the proposed approach to Gurobi for solving problems with a given upper bound magnitude (40) across various magnitudes of lower bound for the binary case.

For the $U = 40$ case, a jump to just over 6 solves is necessary for parity for the $n = 40$ setting can be seen and this is the worst performance the TSA exhibits over all parameter combinations.

L	U	Problem Type	Performance Ratio
0	60	Binary	3.34
45	60	Binary	3.56
60	60	Binary	0.52
0	60	Binary	3.81
45	60	Binary	5.94
60	60	Binary	0.23
0	60	Binary	4.78
45	60	Binary	4.14
60	60	Binary	0.23

Figure. 6.9: A comparison of the performance ratios of the proposed approach to Gurobi for solving problems with a given upper bound magnitude (60) across various magnitudes of lower bound for the binary case.

For the $U = 60$ problem case, the number of solves does not increase as one would expect if the approach truly struggled with larger volume right-hand side regions. This may be because it only has to consider right-hand sides that are feasible (ρ is small), which suggests that the value function approach is performing as intended.

6.5 Integer Multi-dimensional Quadratic Knapsack

For the Integer Multi-dimensional QKP, the performance ratios are substantially worse. Across all n, U combinations, nearly 44-times as many solves may be necessary to achieve the benefit of running the TSA to find all LSO vectors less than U. While that still places the TSA as a reasonable choice, it is no longer as strong a proposition as for the binary case.

L	U	Problem Type	Performance Ratio
0	20	Integer	16.53
15	20	Integer	6.83
20	20	Integer	0.39
0	20	Integer	6.87
15	20	Integer	5.61
20	20	Integer	0.07
0	20	Integer	4.71
15	20	Integer	4.83
20	20	Integer	0.06

Figure. 6.10: A comparison of the performance ratios of the proposed approach to Gurobi for solving problems with a given upper bound magnitude (20) across various magnitudes of lower bound for the integer case.

For the $U = 20$ case, at worst 16.53 solves are required, but interestingly for these problems (a trend that continues with larger U for the integer problem class), the value function approach outperforms even for a single right-hand side. Remembering that there are roughly 900 solves necessary even if the locations are ex-ante known, this is a considerable performance benefit when sharing the computational effort.

L	U	Problem Type	Performance Ratio
0	40	Integer	43.73
30	40	Integer	20.75
40	40	Integer	0.29
0	40	Integer	11.71
30	40	Integer	6.34
40	40	Integer	0.06
0	40	Integer	10.43
30	40	Integer	6.90
40	40	Integer	0.06

Figure. 6.11: A comparison of the performance ratios of the proposed approach to Gurobi for solving problems with a given upper bound magnitude (40) across various magnitudes of lower bound for the integer case.

For the $U = 40$ case, at worst 43.72 solves are necessary to match Gurobi's performance. The improvements in performance ratio for a single right-hand side continue to widen, and in some cases there is real benefit to reducing the size of the right-hand side region considered.

L	U	Problem Type	Performance Ratio
0	60	Integer	33.52
45	60	Integer	27.29
60	60	Integer	0.41
0	60	Integer	17.37
45	60	Integer	11.09
60	60	Integer	0.12
0	60	Integer	5.86
45	60	Integer	4.50
60	60	Integer	0.03

Figure. 6.12: A comparison of the performance ratios of the proposed approach to Gurobi for solving problems with a given upper bound magnitude (60) across various magnitudes of lower bound for the integer case.

For $U = 60$, the performance ratios improve from the $U = 40$ case, indicating that if the problems that are most-challenging for Gurobi are considered, the proposed approach does not degrade in performance as quickly as does Gurobi. At worst 33.52 Gurobi solves are accomplished in the time it takes the TSA to find all roughly $1,000$ solutions to the value function in this case.

Chapter 7

Conclusion

Decision problems involving discrete choices can often be modeled as Integer Nonlinear Programs (INLPs). Some of these discrete choice problems required constrained choices under a capacitated, fixed (knapsack) budget, giving a resource vector interpretation to a right-hand side β as in (3.4). These INLPs are computationally expensive to solve, and especially when there is uncertainty with respect to resource vectors. This thesis demonstrates a novel algorithmic approach to finding the value function of INLPs for polynomial objectives and separable constraints. The computational results from the experiments conducted are promising towards real-world application of the proposed value function approach to general integer nonseparable polynomial problems with linear constraints. We specifically introduce the Tree Search Algorithm (TSA) to generate a Minimal R* Tree data structure (MR*T), enabling the efficient lookup of solutions, post-optimality analysis, and recomputation for realtime applications.

The limitations of this thesis are that while the considered problem class of polynomial integer optimization problems span a large variety of domains, it does not consider non-polynomial programs with special structure, or even-more-generally non-analytic nonlinear problems. Furthermore, because the scope is limited to integer problems, mixed-integer problems have not been explored for the TSA. Given the use of a spatial data structure, the TSA necessarily has an exponential requirement for data with respect to the number

of dimensions of the stored data to see any benefit from the spatial queries. As a result, the TSA as-it-stands is limited to problems with only a small to moderate number of varying resource constraint dimensions (m).

A natural direction for future work would be to consider a generalized Benders decomposition [43, 44] extension of this approach to mixed-integer problems. Generalized Benders decomposition is an approach for solving very large mixed-integer programs that are nonlinear in the real variables. The decomposition does this through solving a master ILP and iteratively solving the resulting NLP while successively fixing the integer variables. It would be interesting to explore whether the ILP assumption in Benders could be relaxed to INLP with access to a fast value function lookup ahead of time.

Additional use could be made of the sub-millisecond lookup times for the value function that result from this approach. One potential direction would be new ways that having the value function of a nonlinear integer program could improve the way that real-time or repeated problems are solved. This could be not only through faster solutions of that problem with new resource constraints but also, in the future, by allowing the extension of the computed value function by additional new constraints.

Finally, the TSA may also have applicability to a more general class of INLPs. Specifically, the distributive law can be applied in the way the TSA uses it to any polynomials on a discrete commutative ring with scalar valuation, opening the possibility of reasoning over more interesting domains. Most nonlinear solvers have a wide array of custom-built approaches for functions including hyperbolic programs, trigonometric functions, exponentials, logarithms, and (fractional) polynomials. This should commensurably expand the variety of real-world, nonlinear programs that could be addressed.

Bibliography

[1] Ali Esmaeel Nezhad, Mohammad Sadegh Javadi, and Ehsan Rahimi. Applying augmented -constraint approach and lexicographic optimization to solve multi-objective hydrothermal generation scheduling considering the impacts of pumped-storage units. *International Journal of Electrical Power & Energy Systems*, 55:195–204, 2014.

[2] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N Pistikopoulos. The explicit solution of model predictive control via multiparametric quadratic programming. In *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No. 00CH36334)*, volume 2, pages 872–876. IEEE, 2000.

[3] Ellis L Johnson, Anuj Mehrotra, and George L Nemhauser. Min-cut clustering. *Mathematical programming*, 62(1-3):133–151, 1993.

[4] Alessandro Baldo, Edoardo Fadda, Matteo Boffa, Lorenzo Cascioli, Arianna Ravera, and Chiara Lanza. The polynomial robust knapsack problem. *European Journal of Operational Research*, 305, 06 2022. doi: 10.1016/j.ejor.2022.06.029.

[5] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL `https://www.gurobi.com`.

[6] Nikolaos V. Sahinidis. BARON: A general purpose global optimization software package. *Journal of Global Optimization*, 8:201–205, 1996.

[7] Andrew C Trapp, Oleg A Prokopyev, and Andrew J Schaefer. On a level-set characterization of the value function of an integer program and its application to stochastic programming. *Operations Research*, 61(2):498–511, 2013.

[8] Andrew C Trapp and Oleg A Prokopyev. A note on constraint aggregation and value functions for two-stage stochastic integer programs. *Discrete Optimization*, 15:37–45, 2015.

[9] Eric M Antley. Integrated value function global optimization approaches for two-stage stochastic programs. 2021.

[10] Temitayo Ajayi, Christopher Thomas, and Andrew J Schaefer. The gap function: Evaluating integer programming models over multiple right-hand sides. *Operations Research*, 2021.

[11] Laurence A Wolsey. Integer programming duality: Price functions and sensitivity analysis. *Mathematical Programming*, 20(1):173–195, 1981.

[12] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. Geometric algorithm for multiparametric linear programming. *Journal of optimization theory and applications*, 118:515–540, 2003.

[13] John R Birge and Francois Louveaux. *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

[14] Osman Y Özaltın, Oleg A Prokopyev, and Andrew J Schaefer. Two-stage quadratic integer programs with stochastic right-hand sides. *Mathematical programming*, 133 (1):121–158, 2012.

[15] Dimitris Bertsimas and Bartolomeo Stellato. The voice of optimization. *Machine Learning*, 110(2):249–277, 2021.

[16] Gabriel Lopez Zenarosa, Oleg A Prokopyev, and Eduardo L Pasiliao. On exact solution approaches for bilevel quadratic 0–1 knapsack problem. *Annals of Operations Research*, 298(1):555–572, 2021.

[17] Dimitris Bertsimas and Bartolomeo Stellato. Online mixed-integer optimization in milliseconds. *arXiv preprint arXiv:1907.02206*, 2019.

[18] Onur Tavaslıoğlu, Oleg A Prokopyev, and Andrew J Schaefer. Solving stochastic and bilevel mixed-integer programs via a generalized value function. *Operations Research*, 67(6):1659–1677, 2019.

[19] Jean B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001. doi: 10.1137/S1052623400366802. URL `https://doi.org/10.1137/S1052623400366802`.

[20] E Sandgren. Nonlinear integer and discrete programming in mechanical design optimization. 1990.

[21] Jung-Fa Tsai, Han-Lin Li, and Nian-Ze Hu. Global optimization for signomial discrete programming problems in engineering design. *Engineering Optimization*, 34(6):613–622, 2002.

[22] Srinath Sridhar, Fumei Lam, Guy E Blelloch, Ramamoorthi Ravi, and Russell Schwartz. Mixed integer linear programming for maximum-parsimony phylogeny inference. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5 (3):323–331, 2008.

[23] Radhika Santhanam and George J Kyparisis. A decision model for interdependent information system project selection. *European Journal of Operational Research*, 89 (2):380–399, 1996.

[24] Rajdeep Grewal, Gary L Lilien, and Girish Mallapragada. Location, location, location: How network embeddedness affects project success in open source systems. *Management Science*, 52(7):1043–1056, 2006.

[25] Andreas Ernst, Houyuan Jiang, and Mohan Krishnamoorthy. Exact solutions to task allocation problems. *Management Science*, 52(10):1634–1646, 2006.

[26] Nan Kong, Andrew J Schaefer, Brady Hunsaker, and Mark S Roberts. Maximizing the efficiency of the US liver allocation system through region design. *Management Science*, 56(12):2111–2122, 2010.

[27] Mustafa Akan, Oguzhan Alagoz, Baris Ata, Fatih Safa Erenay, and Adnan Said. A broader view of designing the liver allocation system. *Operations Research*, 60(4): 757–770, 2012.

[28] David Pisinger, Anders Bo Rasmussen, and Rune Sandvik. Solution of large quadratic knapsack problems through aggressive reduction. *INFORMS Journal on Computing*, 19:280–290, 2007.

[29] Dimitris Bertsimas and Robert Weismantel. *Optimization over integers*, volume 13. Dynamic Ideas Belmont, 2005.

[30] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[31] Marianna De Santis, Gabriele Eichfelder, Julia Niebling, and Stefan Rocktaschel. Solving multiobjective mixed integer convex optimization problems. *SIAM Journal on Optimization*, 30(4):3122–3145, 2020.

[32] Samira Fallah, Ted K Ralphs, and Natashia L Boland. On the relationship between the value function and the efficient frontier of a mixed integer linear optimization problem. *arXiv preprint arXiv:2303.00785*, 2023.

[33] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[34] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pages 322–331, 1990.

[35] Fred Glover. Improved linear integer programming formulations of nonlinear integer problems. *Management science*, 22(4):455–460, 1975.

[36] Giorgio Gallo, Peter L. Hammer, and Bruno Simeone. Quadratic knapsack problems. 1980.

[37] Alberto Caprara, David Pisinger, and Paolo Toth. Exact solution of the quadratic knapsack problem. *INFORMS Journal on Computing*, 11(2):125–137, 1999.

[38] Seth Brown, Wenxin Zhang, Temitayo Ajayi, and Andrew J Schaefer. A Gilmore-Gomory construction of integer programming value functions. *Operations Research Letters*, 2021.

[39] Franklin Djeumou Fomeni and Adam N Letchford. A dynamic programming heuristic for the quadratic knapsack problem. *INFORMS Journal on Computing*, 26(1):173–182, 2014.

[40] LLC Gurobi Optimization. Gurobi optimizer version 9.5.2, 2022.

[41] Haibo Wang, Gary Kochenberger, and Fred Glover. A computational study on the quadratic knapsack problem with multiple constraints. *Computers & Operations Research*, 39(1):3–11, 2012.

[42] Joachim Schauer. Asymptotic behavior of the quadratic knapsack problem. *European Journal of Operational Research*, 255(2):357–363, 2016.

[43] Arthur M Geoffrion. Elements of large-scale mathematical programming Part I: Concepts. *Management Science*, 16(11):652–675, 1970.

[44] John F Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik*, 4(1):238–252, 1962.