A Binary Search-Based Criterion Space Algorithm for Solving Bi-Objective Integer Programs: The Quadtree Search Method

Lagnajita Basu[1], Aadya Bhattarai[1], Tapas Das[1], Kimia Keshanian[2], and Hadi Charkhgard[1]

[1]Department of Industrial and Management Systems Engineering, University of South Florida, Tampa, Florida, USA
[2]Information and Technology Management Department, University of Tampa, Tampa, Florida, USA

Abstract

We propose an exact binary search-based branch-and-bound algorithm, termed the *Quadtree Search Method*, for solving bi-objective integer programs. The existing literature on criterion space search methods for multi-objective optimization predominantly assumes that subproblems can be solved to optimality, an assumption that becomes computationally prohibitive for hard instances. In contrast, our approach departs from this assumption by prioritizing feasibility checks over optimality at each subproblem, while still guaranteeing eventual generation of the entire nondominated frontier. The proposed method extends binary search by introducing an additional *undecided* state for time-limited subproblems, thereby preventing the search from stalling on computationally challenging nodes. We further incorporate several enhancements, including initial bounds, warm starts, dynamic upper bound tightening, and the selective skipping of redundant feasibility checks. We demonstrate the efficacy of the proposed method on bi-objective assignment, set covering, and traveling salesman problems. The results show that the Quadtree Search Method is particularly effective on difficult instances, providing provable quality guarantees for the approximated nondominated frontier even when terminated prematurely.

Keywords: Bi-Objective Integer Programming, Criterion Space Search Method, Branch and Bound, Binary Search, Approximation

1 Introduction

Bi-objective integer programs (BOIPs) arise in many real-world decision-making scenarios in which decision variables are discrete, and two competing objectives must be optimized simultaneously. Examples include balancing cost and reliability in logistics, return and risk in finance, and weight and strength in engineering design. Unlike single-objective optimization problems (SOOPs), which yield a unique optimal objective value (assuming the existence of an optimal solution), BOIPs often involve conflicting objectives, meaning that no single solution simultaneously optimizes both objectives. As a result, the goal in BOIPs is to generate the set of *nondominated* points in the criterion space, often referred to as the nondominated frontier. This frontier characterizes the set of attainable optimal trade-offs available to a decision maker.

Current solution frameworks for BOIPs can be divided broadly into decision space search and criterion space search, each using fundamentally different strategies to generate the nondominated

frontier. Decision space search algorithms try to modify the algorithms designed for SOOPs to accommodate multiple objectives. For example, see Parragh and Tricoire [2019], Belotti et al. [2016], Gadegaard et al. [2019], Forget and Parragh [2023], Stidsen et al. [2014], Stidsen and Andersen [2018], and De Santis et al. [2020]. They search in the space of feasible solutions, generate outcomes, and use improvement mechanisms to guide the search toward better solutions. On the other hand, criterion space search algorithms search in the space of the objective values. They explore this space by solving a sequence of SOOPs. For example, see Perini et al. [2020], Boland et al. [2015], Santis et al. [2020], Soylu [2018], Lokman and Köksalan [2013], and Charkhgard et al. [2025]. They solve each SOOP subproblem using one of the existing off-the-shelf solvers. A clear advantage of these methods is that when the off-the-shelf solvers improve, it automatically enhances the criterion space search methods.

The focus of this study is on the criterion space search methods. While these approaches offer the key advantage of benefiting directly from improvements in off-the-shelf solvers, as discussed earlier, their requirement to solve every SOOP to optimality remains a significant bottleneck. If the algorithm encounters a computationally intensive subproblem, the entire process may stall, preventing the discovery of additional nondominated points and resulting in a poor, if any, approximation of the nondominated frontier. Moreover, prematurely terminating a SOOP can compromise the mathematical correctness of these methods, as they rely on guaranteed optimality conditions. Conversely, waiting for full optimality on difficult instances can be computationally prohibitive, often leading to indefinite execution times. This limitation represents a key research gap in the criterion space search literature that we aim to address. To overcome these challenges, we propose an approach that shifts the focus from costly optimality proofs to feasibility checks.

The proposed approach, termed the *Quadtree Search Method*, is motivated by the binary search framework commonly used in the single-objective optimization literature. In that context, the key idea behind binary search is to iteratively reduce the domain containing the optimal objective value by approximately half at each iteration. Specifically, for single-objective minimization problems, the method begins with an interval defined by lower and upper bounds of the objective value. At each iteration, it evaluates a midpoint by checking whether a feasible solution exists with an objective value less than or equal to that point while satisfying all constraints. If such a solution exists, the search is restricted to the lower half of the interval by updating the upper bound; otherwise, it proceeds to the upper half by increasing the lower bound. This feasibility-based branching rule systematically eliminates around half of the remaining search space at each iteration, leading to rapid convergence to the optimal objective value.

Our main contribution is to extend the binary search framework to solve BOIPs. To achieve this, the Quadtree Search Method introduces an additional *undecided* state alongside the traditional feasible and infeasible states. A subproblem is classified as undecided if it cannot be solved within a pre-specified time limit. By incorporating this state, the proposed method avoids stalling on computationally difficult subproblems. To accommodate the undecided state while ensuring eventual convergence to the nondominated frontier, the Quadtree Search Method integrates a bi-objective branch-and-bound scheme within the binary search process. An important feature of the proposed approach is its flexibility in feasibility checking, allowing users to employ any suitable solver for subproblem evaluation. In our implementation, we utilize constraint programming solvers due to their effectiveness in detecting infeasibility [Bockmayr and Pisaruk, 2006]. Another key property of the proposed method is that, even if terminated prematurely, it can still provide an approximation of the nondominated frontier with a provable quality guarantee. Specifically, we introduce an easy-to-compute performance metric, denoted by Δ, which quantifies how far the current approximation is from the complete nondominated frontier at any point during the search. This measure is defined as the percentage of the unexplored nondominated area in the criterion space relative to the initial

unexplored area at the beginning of the search.

Another contribution of this study is the development of several algorithmic enhancements that can significantly accelerate the Quadtree Search Method. These enhancements include computing tighter initial bounds tailored to each problem class, warm-starting the search using solutions from weighted scalarizations to enable early pruning, dynamically tightening upper bounds at each iteration based on discovered incumbents, and selectively skipping redundant feasibility checks in regions already proven to be feasible. To evaluate the impact of these enhancements, we conduct a set of numerical experiments that assess both their individual and combined effects on solution time. The experiments also examine the primary promise of the proposed approach, namely its ability to outperform state-of-the-art criterion space search methods on challenging BOIPs, where existing methods often struggle due to their reliance on solving subproblems to optimality.

We test the proposed method on three problem classes: the bi-objective assignment problem (BOAP), the bi-objective set covering problem (BOSCP), and the bi-objective traveling salesman problem (BOTSP). BOAP instances are relatively tractable for existing criterion space search methods, which can quickly generate the nondominated frontier even for large instances. Although we do not expect the proposed method to outperform existing approaches on this class, it serves as a useful benchmark to isolate and evaluate the effects of the proposed enhancements. BOSCP instances present a moderate level of difficulty. While state-of-the-art methods can still handle reasonably large instances, this class provides a meaningful setting to demonstrate the benefits of the proposed method as problem size increases. In particular, we show that for large BOSCP instances, our approach achieves improved approximations of the nondominated frontier, with an average gap of $\Delta = 0.43\%$. Finally, we evaluate the method on BOTSP instances, which are particularly challenging for criterion space search methods. The existing literature typically considers only small instances of such problems. In contrast, we test instances with up to 160 cities and show that our method can produce high-quality approximations, achieving an average gap of $\Delta = 8.64\%$ even at this scale.

The remainder of this paper is organized as follows. Section 2 introduces the foundational concepts of BOIPs and formally defines the problem under study. Section 3 presents how the binary search framework for SOOPs can be adapted to the bi-objective setting. Section 4 provides a detailed description of the proposed Quadtree Search Method. Section 5 introduces several algorithmic enhancements designed to accelerate the search process. Section 6 presents the proposed quality metric along with a postprocessing procedure aimed at obtaining the most accurate estimate of this metric upon algorithm termination. Section 7 reports a comprehensive numerical analysis and performance evaluation on three benchmark problem classes (BOAP, BOSCP, and BOTSP). Finally, Section 8 concludes the paper and outlines directions for future research.

2 Foundational Concepts and Problem Description

Let $\mathcal{X} \subseteq \mathbb{R}^n$ be a nonempty feasible set in the decision space and $\mathbf{f} : \mathcal{X} \to \mathbb{R}^2_+$ a vector-valued objective function with two components $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}))^\top$. A bi-objective optimization problem (BOOP) is defined by

$$\min_{\mathbf{x}\in\mathcal{X}} \big(f_1(\mathbf{x}), f_2(\mathbf{x})\big).$$

The image $\mathcal{Y} := \mathbf{f}(\mathcal{X}) \subseteq \mathbb{R}^2_+$ represents the feasible set in the criterion space and is assumed to be bounded. Without loss of generality, we assume that $f_1(\mathbf{x}), f_2(\mathbf{x}) \geq 0$ for all $\mathbf{x} \in \mathcal{X}$, as any objective function with a finite lower bound can be translated into the non-negative orthant by adding a sufficiently large positive constant. Also, throughout this paper, we adopt the convention of denoting all the vectors in boldface.

Definition 1. *A feasible solution of a BOOP $\bar{\mathbf{x}} \in \mathcal{X}$ is called efficient (or Pareto optimal) if there is no $\mathbf{x} \in \mathcal{X}$ such that $f_i(\mathbf{x}) \leq f_i(\bar{\mathbf{x}})$ for all $i = 1, 2$ and $f_j(\mathbf{x}) < f_j(\bar{\mathbf{x}})$ for at least one index $j \in \{1, 2\}$. The set of all possible efficient solutions is denoted by $\mathcal{X}_{\mathrm{E}}$, and $\mathcal{Y}_N = f(\mathcal{X}_{\mathrm{E}})$ is called the nondominated frontier.*

A BOOP with only integer decision variables is called a bi-objective integer program (BOIP) and can be stated as

$$\min_{\mathbf{x} \in \mathcal{X}} \big(f_1(\mathbf{x}), f_2(\mathbf{x})\big), \qquad \mathcal{X} \subseteq \mathbb{Z}^n. \tag{1}$$

In this study, we assume that $\mathcal{Y} \subseteq \mathbb{Z}^2_+$ but we do not impose any assumptions on the linearity of the objective functions or constraints of BOIPs, as the proposed method is designed to be general. However, in our computational experiments, we evaluate the method on three classes of BOIPs (namely the BOAP, BOTSP, and BOSCP) which happen to have linear objective functions and constraints and are well suited for benchmarking against state-of-the-art criterion space search methods.

Definition 2. *A nondominated point $\mathbf{y} \in \mathcal{Y}_N$ is called **supported** if it can be obtained as an optimal solution to a weighted-sum scalarization defined by $\min_{\mathbf{x} \in \mathcal{X}}(w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x}))$ for some scalars $w_1, w_2 > 0$. Otherwise, $\mathbf{y}$ is called an **unsupported** nondominated point.*

BOIPs are typically challenging because a significant portion of the nondominated frontier consists of unsupported points. Consequently, weighted sum methods that vary the weights to generate nondominated points are unable to capture these unsupported points. Instead, alternative scalarization techniques, such as the ϵ-constraint method [Chankong and Haimes, 1983], are required to compute the nondominated frontier. In the ϵ-constraint method, at each iteration t, the first objective is optimized while the second objective is constrained to be an $\epsilon > 0$ amount smaller than the value obtained in the previous iteration $t-1$. Thus, at iteration t, we solve:

$$\begin{aligned} \min \quad & f_1(\mathbf{x}) + \alpha f_2(\mathbf{x}) \\ \text{s.t.} \quad & f_2(\mathbf{x}) \leq y_2^{t-1} - \epsilon, \\ & \mathbf{x} \in \mathcal{X}. \end{aligned}$$

where $\mathbf{y}^{t-1} = (y_1^{t-1}, y_2^{t-1})$ is the nondominated point found in the previous iteration $(t-1)$. Here $\epsilon > 0$ forces strict improvement in the second objective. Also, α is typically a small positive weight for tie-breaking. The algorithm terminates with an infeasible subproblem, indicating that the entire nondominated frontier has been generated. Figure 1 illustrates the ϵ-constraint method, where the blue points represent nondominated points, and the dashed horizontal line shows the bound used in the current iteration t. If the method had found a nondominated point $\mathbf{y}^{t-1}$ in the previous iteration, then it tightens the bound on the second objective to search the lower region of the objective space and obtain another point y_t. Readers can go through [Boland et al., 2015] for more details.

If the ϵ-constraint method encounters a computationally difficult subproblem that requires excessive (or effectively indefinite) time to reach optimality, the algorithm cannot make further progress. Moreover, if the subproblem is terminated prematurely, its correctness is compromised, as optimality is a critical requirement for guaranteeing the eventual generation of the entire nondominated frontier.

Challenge 2 is not unique to the ϵ-constraint method. Existing criterion space search algorithms (e.g., [Kirlik and Sayin, 2014, Dai and Charkhgard, 2018, Dächert et al., 2024, Soylu and Yıldız,

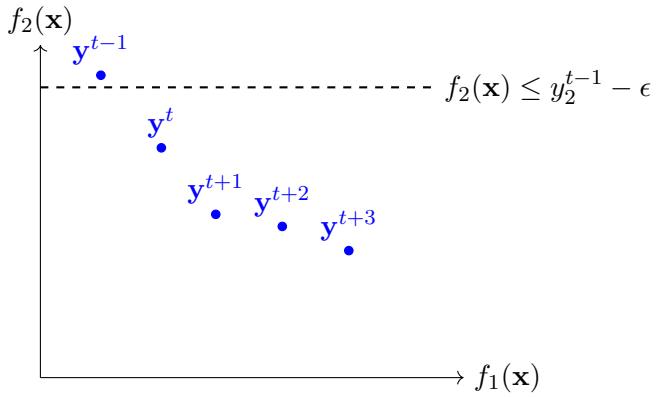


Figure 1: An illustration of the ϵ-constraint method for BOIPs

2016, Mesquita-Cunha et al., 2023]) operate in a similar manner by transforming the problem into a sequence of SOOPs that must be solved to optimality. These SOOPs, commonly referred to as scalarizations [Bazgan et al., 2022, Doğan et al., 2021], are generated during the search using more sophisticated decomposition strategies than those employed in the ϵ-constraint method. However, these existing methods remain susceptible to Challenge 2 due to their reliance on solving each scalarized subproblem to optimality. Consequently, the goal of this paper is to develop a method that avoids this limitation by not requiring the generation of a nondominated point at every iteration. Instead, our approach focuses on identifying feasible points in the criterion space, thereby improving robustness in the presence of computationally difficult subproblems.

3 Binary Search and the Proposed Binary-encoded Reformulation

To address Challenge 2, in this study, we extend the binary search framework to BOIPs. To this end, in this section, we first formally describe the binary search mechanism for SOOPs, followed by a reformulation that facilitates its implementation. We then provide a high-level overview of how this framework can be adapted to the bi-objective setting.

3.1 Binary Search in Single-Objective Optimization

The foundation of our study is inspired by binary search, which works by transforming a single-objective optimization problem into a sequence of feasibility queries. Consider a SOOP of the form $\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})$ where $\mathcal{X} \subseteq \mathbb{Z}^n$. The binary search procedure begins by initializing a lower bound $L \in \mathbb{Z}$ and an upper bound $U \in \mathbb{Z}$ on the objective value such that

$$L \leq f(\mathbf{x}) \leq U \quad \forall\, \mathbf{x} \in \mathcal{X}.$$

At each iteration, the binary search procedure evaluates whether there exists a feasible solution $\mathbf{x} \in \mathcal{X}$ such that $f(\mathbf{x}) \leq V$, where the target value V is typically defined as $V = \lfloor \frac{L+U}{2} \rfloor$. By evaluating this subproblem, the algorithm systematically narrows the search interval. A positive response confirms a feasible point at or below V, allowing the algorithm to update the upper bound to V. Conversely, if the solver provides a certificate of infeasibility, it is proved that no solution exists at or below V, and the lower bound is shifted to $V+1$. This iterative process continues until the lower and upper bounds converge ($L = U$), at which point the remaining value is identified as the global minimum.

In the context of implementing binary search for SOOPs, we propose using binary-encoded reformulations, as they can be readily adapted to the bi-objective setting. Let B be the maximum number of bits required to represent each feasible objective value $f(\mathbf{x})$ as a binary vector. That is, the base-10 value is expressed as

$$(f(\mathbf{x}))_{10} = (b_{B-1}b_{B-2}\cdots b_1 b_0)_2.$$

Using this transformation, we introduce a binary decision variable $b_k \in \{0,1\}$ for each bit index $k \in \{0,\ldots,B-1\}$. In this representation, each bit is assigned a specific weight 2^k that grows exponentially with the index k, allowing the objective value to be encoded as $f(\mathbf{x}) = \sum_{k=0}^{B-1} 2^k b_k$. So, a binary-encoded reformulation of SOOP can be stated as

$$\min\Big\{ \sum_{k=0}^{B-1} 2^k\, b_k : \mathbf{x} \in \mathcal{X},\ f(\mathbf{x}) = \sum_{k=0}^{B-1} 2^k b_k,\ \text{and } b_0,\ldots,b_{B-1} \in \{0,1\}\Big\}$$

Definition 3 (MSB and LSB)**.** *We define the Most Significant Bit (MSB) as b_{B-1} with weight 2^{B-1}, and the Least Significant Bit (LSB) as b_0 with weight 2^0 .*

We notice that if MSB $b_{B-1} = 1$, its contribution to the objective value is 2^{B-1}, whereas the aggregated contribution of all lower-order bits $(b_0,\ldots,b_{B-2})$ is bounded by:

$$\sum_{k=0}^{B-2} 2^k = 2^0 + 2^1 + \cdots + 2^{B-2} = 2^{B-1} - 1 < 2^{B-1}.$$

Therefore, we observe that trivial lower and upper bounds for the objective value are given by $L = 0$ and $U = 2^B - 1$. By setting the most significant bit (MSB) $b_{B-1} = 0$, we bisect the search space, mimicking the behavior of a binary search mechanism. In this case, the lower interval $[L, 2^{B-1} - 1]$ is retained as the active domain, while the upper half $[2^{B-1}, U]$ is eliminated, as illustrated in Figure 2. Similarly, when $b_{B-1} = 1$, the upper interval $[2^{B-1}, U]$ becomes the active domain, and the lower half $[L, 2^{B-1} - 1]$ is discarded.

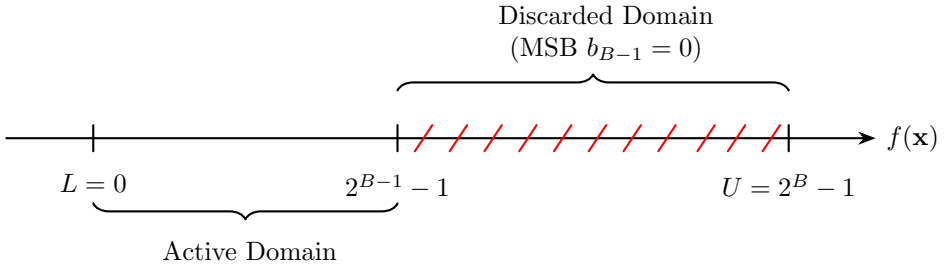


Figure 2: Reduction of objective domain after fixing MSB $b_{B-1} = 0$

Using binary-encoded reformulations, the binary search process can be interpreted as a structured hierarchy of logical decisions [Saghand et al., 2023]. By prioritizing bits according to their weight, proceeding from the most significant (i.e., MSB) to the least significant (i.e., LSB), we eliminate approximately half of the search space at each iteration, discarding regions that are proved to not contain the optimal objective value. To formalize this ordering of exploration, we introduce the following definition:

Definition 4 (Most Significant Unexplored Bit)**.** *The **MSUB** is defined as the bit b_k with the highest weight 2^k among all bits that have not yet been assigned a fixed value $\{0,1\}$.*

Based on Definition 4, fixing the MSUB at any iteration enables systematic pruning of the search space. To faithfully mimic a binary search procedure, we exploit the fact that the SOOP is formulated as a minimization problem. At each iteration, we first test whether the MSUB can be set to zero. If a feasible solution exists under this assignment, we fix the MSUB to zero and proceed to the next iteration (i.e., bit); otherwise, we set it to one and continue. Once the final bit (i.e., LSB) is fixed, the resulting value corresponds to the optimal objective value.

3.2 Extension to Bi-objective Integer Programming

The criterion space in BOIPs is two-dimensional, whereas in SOOPs it is one-dimensional. Consequently, extending binary search to BOIPs leads to a different partitioning behavior: at each iteration, the criterion space is divided into four sub-regions rather than two, since the bisection occurs simultaneously along both objective dimensions. To formally establish this binary search framework, we extend the binary-encoded reformulation proposed for SOOPs to the bi-objective case. Let B denote the maximum number of bits required to represent any feasible value of either objective function. While B can be computed explicitly, we may also adopt a default value; for instance, $B = 30$ is sufficient to represent objective values up to $2^{30} - 1 \approx 1.07 \times 10^9$, which is significantly larger than typical values encountered in most combinatorial optimization problems.

Using B, we introduce binary decision variables $b_{ik} \in \{0, 1\}$, where $i \in \{1, 2\}$ indexes the objective function and $k \in \{0, \dots, B-1\}$ indexes the bit position. The resulting binary-encoded reformulation of the BOIP is given by

$$\begin{aligned} \min_{\mathbf{x}\in\mathcal{X}} \quad & \Big(\sum_{k=0}^{B-1} 2^k \, b_{1k} \; , \; \sum_{k=0}^{B-1} 2^k \, b_{2k} \Big) \\ \text{s.t.} \quad & f_i(\mathbf{x}) \;=\; \sum_{k=0}^{B-1} 2^k \, b_{ik}, \qquad \forall \; i \in \{1, 2\}, \\ & b_{ik} \in \{0, 1\}, \qquad \forall \; i \in \{1, 2\}, \; \forall \; k \in \{0, \dots, B-1\}. \end{aligned} \tag{2}$$

From the discussion in the previous subsection, it follows that for a given B, a trivial lower bound for both objective functions is 0, while a trivial upper bound is $2^B - 1$. These bounds define a bounding *rectangle* (or box) in the criterion space that contains all feasible objective vectors (see Figure 3). Observe that for each bit index $k \in \{0, \dots, B-1\}$, the pair (b_{1k}, b_{2k}) can take four possible values:

$$(b_{1k}, b_{2k}) \in \{(0, 0), (0, 1), (1, 0), (1, 1)\}.$$

During the binary search process, the index k corresponds to the MSUB at each iteration, as the search progresses from the MSB toward the LSB. For a given k, the four possible values of the pair (b_{1k}, b_{2k}) partition the search space into four distinct regions. Once a specific value of this pair is fixed, the corresponding region becomes the active region, and the search proceeds recursively within it. For example, in Figure 3, all four regions are illustrated for the case $k = B - 1$ (i.e., the MSB). The dotted region represents the active region corresponding to the choice $(b_{1k}, b_{2k}) = (0, 0)$. Building upon this binary-encoded reformulation, the following section introduces the Quadtree Search Method. The proposed approach navigates the criterion space by fixing these bit pairs at each level, utilizing the four-way partitioning of the MSUB pair to ensure a hierarchical exploration of the criterion space.

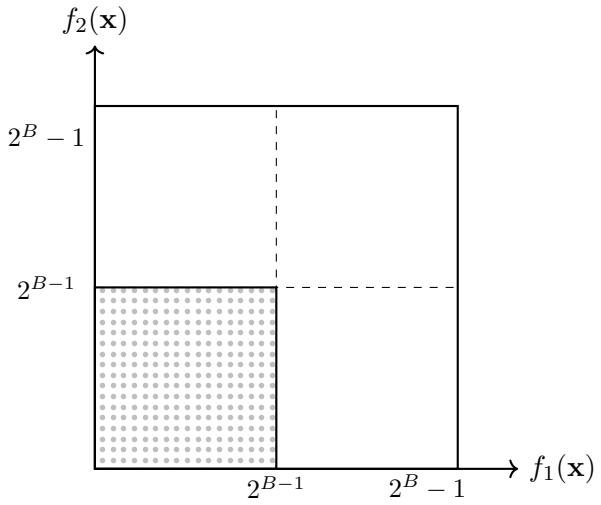


Figure 3: Reduction of objective domain after fixing MSB $(b_{1,B-1}, b_{2,B-1}) = (0, 0)$

4 The Quadtree Search Method

The proposed approach is based on a criterion-space search framework that branches by fixing the MSUB pair (b_{1k}, b_{2k}) at each iteration. As the search progresses, the index k of the MSUB pair decreases from the MSB, $(b_{1,B-1}, b_{2,B-1})$, toward the LSB, $(b_{1,0}, b_{2,0})$. At each level, the number of free bits is reduced, thereby tightening the attainable range of objective values. When k reaches the LSB, all bits are fixed and the objective values collapse to a single point in the criterion space.

The Quadtree Search Method maintains a priority queue, denoted by $\mathcal{L}$, whose elements are nodes. Each node consists of a set of fixed bits and a set of free (unexplored) bits. Geometrically, each node corresponds to a rectangular region in the criterion space that bounds the attainable values of the objective functions. Let $(\hat{b}_{1k}, \hat{b}_{2k})$ denote the value of the MSUB pair k when it is fixed. The bounds of a node are obtained by varying the remaining free bits. For each objective function $i \in \{1, 2\}$, the lower bound LB_i is obtained by setting all free bits to zero, whereas the upper bound UB_i is obtained by setting all free bits to one. Formally,

$$\mathrm{LB}_i = \sum_{k \in \text{fixed}} 2^k \hat{b}_{ik} \quad \leq f_i(\mathbf{x}) \leq \quad \sum_{k \in \text{fixed}} 2^k \hat{b}_{ik} + \sum_{k \notin \text{fixed}} 2^k = \mathrm{UB}_i.$$

We denote the rectangular region associated with a node by $[LB_1, UB_1] \times [LB_2, UB_2]$. The nodes are stored in $\mathcal{L}$ with different priorities order to improve computational efficiency. Specifically, the queue is ordered in ascending order of $LB_1 + LB_2$, with ties broken by the node's creation time. To illustrate, consider four nodes that differ only in the assignment of the current MSUB pair. The four possible assignments are $(0, 0)$, $(0, 1)$, $(1, 0)$, and $(1, 1)$. Among these, the node corresponding to $(0, 0)$ has the highest priority in $\mathcal{L}$, while $(1, 1)$ has the lowest, with the remaining two ordered in between. Consequently, the algorithm prioritizes the exploration of sub-regions closest to the south-west corner of the criterion space, which correspond to high-quality solutions for a minimization problem. In contrast, regions near the north-east corner, which correspond to large objective values, are naturally deferred and are often pruned by solutions identified earlier in the search. This prioritization strategy enhances the effectiveness of pruning mechanisms (see Section 4.2) and facilitates the dynamic tightening of upper bounds, thereby improving overall computational performance (see Section 5.3).

The proposed method also maintains a set of mutually nondominated feasible points in the criterion space, denoted by $\tilde{\mathcal{Y}}_N$. These points are stored in ascending order of the first objective value and are updated dynamically using the following procedure. Whenever a new feasible point $\mathbf{y}$ is discovered, it is inserted into the list at the appropriate position to preserve the ordering, provided that it is not dominated by any existing point. Upon insertion, any points in $\tilde{\mathcal{Y}}_N$ that become dominated by $\mathbf{y}$ are identified and removed. The method returns this set as an approximate nondominated frontier if it is terminated prematurely, or as the exact nondominated frontier if it is allowed to run to completion without time constraints.

Algorithm 1: The Quadtree Search Method

Input: A binary-encoded reformulation of a BOIP instance, and time limit τ
Output: $\tilde{\mathcal{Y}}_N$
$\tilde{\mathcal{Y}}_N \leftarrow \emptyset$
$\mathcal{L} \leftarrow$ Create 4 nodes $\{(0,0),(0,1),(1,0),(1,1)\}$ by fixing MSUB
while $\mathcal{L} \neq \emptyset$ **do**
  Pop out the front element of $\mathcal{L}$ and denote it by d
  if *d can be pruned* **then**
    Discard d
  else
    $(status, \mathbf{y}) \leftarrow$ FeasibilityCheck(d, τ)
    if *status = infeasible* **then**
      Discard d
    else if *status = feasible* **then**
      update $\tilde{\mathcal{Y}}_N$ with $\mathbf{y}$
      foreach *pair in* $\{(0,0),(0,1),(1,0),(1,1)\}$ **do**
        Fix the MSUB pair accordingly
        Push corresponding child node into $\mathcal{L}$
    else if *status = timeout* **then**
      if *d is not a leaf* **then**
        foreach *pair in* $\{(0,0),(0,1),(1,0),(1,1)\}$ **do**
          Fix the MSUB pair accordingly
          Push corresponding child node into $\mathcal{L}$
      else
        Requeue d with lowest priority and set $\tau_d = \infty$ for its next visit
return $\tilde{\mathcal{Y}}_N$

When exploring a node, and prior to invoking a solver for feasibility checking, we first assess whether the node has the potential to yield any feasible point that is nondominated with respect to the current set $\tilde{\mathcal{Y}}_N$. If it can be established that no such point exists, the node is pruned. The details of this pruning mechanism are provided in Section 4.2. If the node cannot be pruned, we proceed to solve the following feasibility problem:

$$
\begin{aligned}
\min_{\mathbf{x}\in\mathcal{X}} \quad & 0 \\
\text{s.t.} \quad & f_i(\mathbf{x}) \;=\; \sum_{k=0}^{B-1} 2^k\, b_{ik}, \qquad i \in \{1,2\}, \\
& (b_{1k}, b_{2k}) = (\hat{b}_{1k}, \hat{b}_{2k}), \qquad \forall\, k \in \text{fixed}, \\
& \mathrm{LB}_i \le f_i(\mathbf{x}) \le \mathrm{UB}_i, \qquad \forall\, i \in \{1,2\}, \\
& b_{ik} \in \{0,1\}, \qquad \forall\, i \in \{1,2\},\ \forall\, k \in \{0,\ldots,B-1\}.
\end{aligned}
\tag{3}
$$

Note that the constraints $\mathrm{LB}_i \le f_i(\mathbf{x}) \le \mathrm{UB}_i$ are redundant in the formulation above. Indeed, as discussed earlier,

$$
\mathrm{LB}_i = \sum_{k\in\text{fixed}} 2^k \hat{b}_{ik}, \quad \text{and} \quad \mathrm{UB}_i = \sum_{k\in\text{fixed}} 2^k \hat{b}_{ik} + \sum_{k\notin\text{fixed}} 2^k,
$$

which are implicitly enforced by the binary representation and fixed constraints. Nevertheless, we retain these bounds because, in later enhancements of the algorithm (see Section 5), they are dynamically tightened during the search. In particular, at each node, improved lower and upper bounds can significantly reduce the feasible region and enable faster detection of feasibility or infeasibility.

In light of the above, during the feasibility-checking phase, if the solver identifies a feasible point, the node is marked as feasible. The set $\tilde{\mathcal{Y}}_N$ is then updated accordingly, and the algorithm branches on this node to explore sub-regions that may contain additional nondominated points. Branching is performed by fixing the value of the MSUB pair at the current level (see Section 4.1). If the node is proven infeasible, it is removed from the search tree. In this case, no branching is required, as all descendant nodes would also be infeasible. If the solver is unable to reach a conclusion within a pre-specified time limit τ, the algorithm adopts a conservative strategy: the node is treated as potentially feasible, and branching proceeds to avoid missing any nondominated solutions. In our implementation, the default time limit is defined as

$$
\tau = \beta \log(\text{size}),
$$

where β is a small constant (in seconds), e.g., $\beta = 5$, which can be tuned based on the difficulty of the problem class, and size denotes a measure of the specific instance under consideration. For example, in BOAP, size can be taken as the number of jobs; in BOSCP, it can be defined as the sum of the number of variables and constraints; and in BOTSP, it can be taken as the number of cities.

The search along each branch terminates at a leaf node ($k = 0$), where all bits in the binary decomposition have been fixed. Leaf nodes require special handling. Initially, they are explored using the same time limit imposed on other nodes. If a time limit is reached without a definitive conclusion, the node is reinserted into the priority queue $\mathcal{L}$ with the lowest priority. This ensures that it will be revisited after all other nodes have been processed, as it may still contain a nondominated point that was not identified due to the time constraint. When such a node is revisited, it is explored without a time limit so that its feasibility status can be determined exactly. Assigning these nodes the lowest priority delays potentially time-consuming computations while allowing the algorithm to make progress on more promising regions of the search space. The pseudocode of the complete algorithm is provided in Algorithm 1.

Table 1: Child Nodes Generated by Branching on the MSB Pair $(b_{1,B-1}, b_{2,B-1})$

Nodes	$[\mathrm{LB}_1, \mathrm{UB}_1]$	$[\mathrm{LB}_2, \mathrm{UB}_2]$	**Rectangle**
$(0,0)$	$[0, 2^{B-1}-1]$	$[0, 2^{B-1}-1]$	$[0, 2^{B-1}-1] \times [0, 2^{B-1}-1]$
$(1,0)$	$[2^{B-1}, 2^B-1]$	$[0, 2^{B-1}-1]$	$[2^{B-1}, 2^B-1] \times [0, 2^{B-1}-1]$
$(0,1)$	$[0, 2^{B-1}-1]$	$[2^{B-1}, 2^B-1]$	$[0, 2^{B-1}-1] \times [2^{B-1}, 2^B-1]$
$(1,1)$	$[2^{B-1}, 2^B-1]$	$[2^{B-1}, 2^B-1]$	$[2^{B-1}, 2^B-1] \times [2^{B-1}, 2^B-1]$

4.1 Branching Mechanism

As noted earlier, in the binary-encoded reformulation each objective value is uniquely represented by a bit string of length B. The branching mechanism exploits this structure by branching on the MSUB, indexed by k. For any node, the preceding bits $(b_{i,k+1}, \ldots, , b_{i,B-1})$ are already fixed for each $i \in \{1, 2\}$. Thus, branching at level k is performed on the pair $(b_{1k}, b_{2k}) \in \{0, 1\}^2$, yielding four possible assignments: $\{(0,0), (1,0), (0,1), (1,1)\}$. This induces a quadtree-like search structure, which motivates the name of the proposed method.

Observation 1. *Branching on a node with MSUB index k generates four child nodes corresponding to $(b_{1k}, b_{2k}) \in \{0, 1\}^2$. These nodes partition the parent node's search region in the criterion space into four pairwise disjoint sub-rectangles whose union exactly recovers the original region.*

To illustrate Observation 1, note that the priority queue is initialized with four nodes obtained by branching on the most significant bit ($k = B - 1$), which is initially the MSUB, as shown in Algorithm 1. Table 1 summarizes the regions associated with each child node generated from branching on $(b_{1,B-1}, b_{2,B-1})$, confirming the disjointness and completeness of the resulting partition. A geometric illustration is provided in Figure 4.

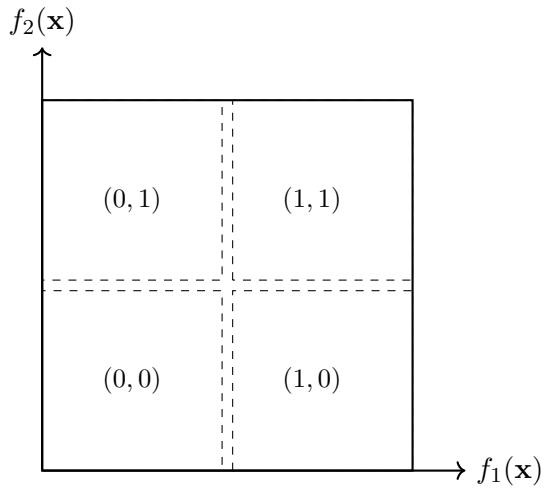


Figure 4: Illustration of the four disjoint sub-rectangles generated by branching on the MSB pair $(b_{1,B-1}, b_{2,B-1})$

4.2 Pruning Mechanisms

As discussed earlier, if the solver determines during the feasibility-checking phase that a node is infeasible, the node is pruned. However, in many cases, it is unnecessary to invoke the solver, as

infeasibility or dominance can be detected directly through simple checks.

Consider an active node associated with the rectangle $[\mathrm{LB}_1, \mathrm{UB}_1] \times [\mathrm{LB}_2, \mathrm{UB}_2]$ in the criterion space. A straightforward yet effective test is to verify whether there exists an index $i \in \{1, 2\}$ such that $\mathrm{UB}_i < \mathrm{LB}_i$. If this condition holds, the rectangle is empty and the node can be immediately pruned. This situation typically arises after applying the bound-updating procedures described in Sections 5.1 and 5.3, which update the lower and upper bounds of each objective. In addition to this bound-based pruning, we employ a dominance-based pruning mechanism. Specifically, a node associated with the rectangle $[\mathrm{LB}_1, \mathrm{UB}_1] \times [\mathrm{LB}_2, \mathrm{UB}_2]$ can be pruned if there exists a point in $\tilde{\mathcal{Y}}_N$ that dominates the entire region. Formally, pruning is valid if there exists $\mathbf{y}^\star = (y_1^\star, y_2^\star) \in \tilde{\mathcal{Y}}_N$ such that

$$y_1^\star \leq \mathrm{LB}_1 \quad \text{and} \quad y_2^\star \leq \mathrm{LB}_2.$$

In this case, every point within the rectangle is dominated by $\mathbf{y}^\star$, and the node can be safely discarded without further exploration.

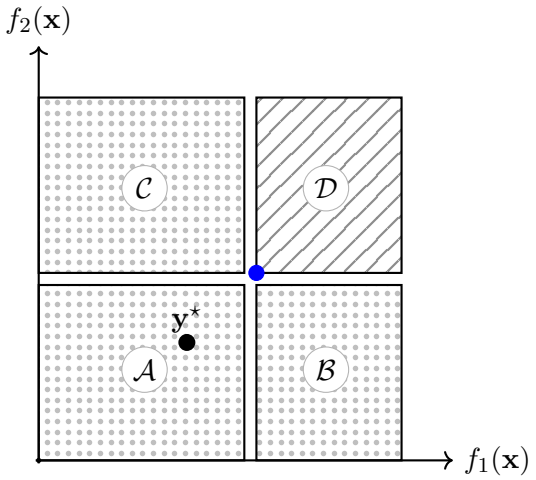


Figure 5: Geometric illustration of dominance-based pruning in the criterion space; the dotted region remains active, while the hatched region is pruned

A geometric illustration of this dominance-based pruning is provided in Figure 5. Let $\mathbf{y}^* \in \tilde{\mathcal{Y}}_N$ denote a feasible point. Nodes $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ remain active since they are not dominated by $\mathbf{y}^\star$. In contrast, node $\mathcal{D}$ is pruned because its lower bound $(LB_1^{\mathcal{D}}, \mathrm{LB}_2^{\mathcal{D}})$, shown as the blue point, is dominated by $\mathbf{y}^\star$. We note that in the proposed Quadtree Search Method, the priority queue maintains nodes in nondecreasing order of the aggregate lower bound $\mathrm{LB}_1 + \mathrm{LB}_2$. As a result, dominance-based pruning is less frequent in the early stages, since the algorithm prioritizes nodes that are more likely to yield improved feasible solutions. As the search progresses and nodes with larger values of $\mathrm{LB}_1 + \mathrm{LB}_2$ are explored, pruning becomes increasingly effective.

5 Algorithmic Enhancement Techniques

5.1 Initial Global Bounds

At initialization, the criterion space is given by $[0, 2^B - 1] \times [0, 2^B - 1]$. Reducing this search space can be achieved by computing valid global lower and upper bounds for each objective function, which can significantly improve overall computational performance. Tight global upper bounds

help focus the search on regions where the nondominated frontier is likely to lie. In particular, if U is a valid global upper bound for both objective functions, then the number of bits that may take the value of one is

$$B' = \lceil \log_2(\mathrm{U} + 1) \rceil .$$

Consequently, all bits with indices greater than B' can be safely fixed to zero, and the corresponding regions of the search space do not need to be explored. Similarly, tight global lower bounds reduce the search space by eliminating regions that are provably infeasible. In particular, tighter global lower bounds help avoid generating and exploring nodes that cannot contain feasible solutions. In terms of implementation, let GLB_i and GUB_i denote the global lower and upper bounds for each objective function $i \in \{1, 2\}$. Consider an active node whose criterion-space region is defined by the rectangle

$$[\mathrm{LB}_1, \mathrm{UB}_1] \times [\mathrm{LB}_2, \mathrm{UB}_2].$$

We update the bounds of this node as follows:

$$\begin{aligned} \mathrm{LB}_i &\leftarrow \max(\mathrm{LB}_i, \mathrm{GLB}_i), \qquad \forall i \in \{1, 2\}, \\ \mathrm{UB}_i &\leftarrow \min(\mathrm{UB}_i, \mathrm{GUB}_i), \qquad \forall i \in \{1, 2\}. \end{aligned}$$

The remaining question is how to compute tight global bounds. The tightest global lower bound for each objective is defined as

$$\mathrm{GLB}_i = \min_{\mathbf{x} \in \mathcal{X}} f_i(\mathbf{x}),$$

which is equivalent to $\min_{\mathbf{x} \in \mathcal{X}_E} f_i(\mathbf{x})$. For some problems, this value can be computed efficiently. For example, in the BOAP, the global lower bound can be obtained using the Hungarian method. However, for more challenging problems such as the BOTSP and BOSCP, computing this value exactly is difficult. To address this, we employ a relaxation of the feasible set, denoted by $R(\mathcal{X})$, to obtain a valid and tractable lower bound:

$$\mathrm{GLB}_i = \min_{\mathbf{x} \in R(\mathcal{X})} f_i(\mathbf{x}).$$

For the BOTSP, initial global lower bounds are obtained by solving the assignment relaxation, which omits the subtour elimination constraints. For the BOSCP, the global lower bounds are computed using the LP relaxation, where integrality constraints are relaxed. In contrast, computing tight global upper bounds is generally more challenging. In multi-objective optimization, it is well known that

$$\max_{\mathbf{x} \in \mathcal{X}_E} f_i(\mathbf{x}) \leq \max_{\mathbf{x} \in \mathcal{X}} f_i(\mathbf{x}),$$

which implies that global upper bounds derived in the same manner as global lower bounds are valid but potentially weak. While one may define

$$\mathrm{GUB}_i = \max_{\mathbf{x} \in \mathcal{X}} f_i(\mathbf{x}),$$

this quantity is typically computationally intractable for hard combinatorial problems. Therefore, for practical purposes, we instead compute

$$\mathrm{GUB}_i = \max_{\mathbf{x} \in R(\mathcal{X})} f_i(\mathbf{x}).$$

For example, in the BOAP, initial global upper bounds are obtained by summing the maximum cost in each row. In the BOTSP, global upper bounds are computed by selecting the largest outgoing cost from each city and summing them. For the BOSCP, global upper bounds are derived by summing the costs of all sets with positive coefficients.

5.2 Warm Start

As explained earlier, the Quadtree Search Method maintains an approximation of the feasible points that are nondominated with each other $\tilde{\mathcal{Y}}_N$ and uses that to prune dominated regions. When this set is empty or very sparse, dominance-based pruning cannot be used from the very beginning. Therefore, we populate the $\tilde{\mathcal{Y}}_N$ with points before the first branching that can immediately be used for the dominance checks described in Section 4.2. One method for achieving an approximate nondominated frontier is applying heuristic approaches such as [Pal and Charkhgard, 2019, Soylu, 2015]. In this paper, to keep the generality and simplicity, we propose a straightforward approach.

To make the initial $\tilde{\mathcal{Y}}_N$ well-spread across the nondominated frontier, we use N uniformly spaced weights and generate feasible points from a family of weighted-sum scalarizations. For a given weight $\omega \in [0, 1]$, we explore the following single-objective problem:

$$\min_{\mathbf{x} \in \mathcal{X}} \quad \omega f_1(\mathbf{x}) + (1 - \omega) f_2(\mathbf{x})$$

where $\omega_i = \frac{i}{N-1}$ for $i = 0, \ldots, N-1$. In our computational experiments, the default value is set to $N = 20$. We emphasize that, although this approach is inspired by the classical weighted-sum method in multi-objective optimization, it differs in a key aspect: each scalarized subproblem is solved under a fixed and tight time budget rather than to optimality. Specifically, we impose a default time limit of 1.5 seconds per subproblem and 4 seconds for the largest instance sizes. This design choice is motivated by the computational difficulty of solving some instances exactly. We then collect the best feasible solutions obtained within the time limit and retain only those that are mutually nondominated, which are added to $\tilde{\mathcal{Y}}_N$.

We note that an important feature of our approach, compared to existing criterion space search methods, is that the proposed warm-start procedure provides a practical and effective mechanism that is naturally tailored to the framework. In this warm-start procedure, optimality is not required for any of the feasible points in $\tilde{\mathcal{Y}}_N$, and the proposed approach can naturally exploit these points to prune the search space. In contrast, in existing criterion space search frameworks, incorporating non-optimal feasible points into the decomposition mechanism is non-trivial, as the search space is typically partitioned only after identifying exact nondominated points.

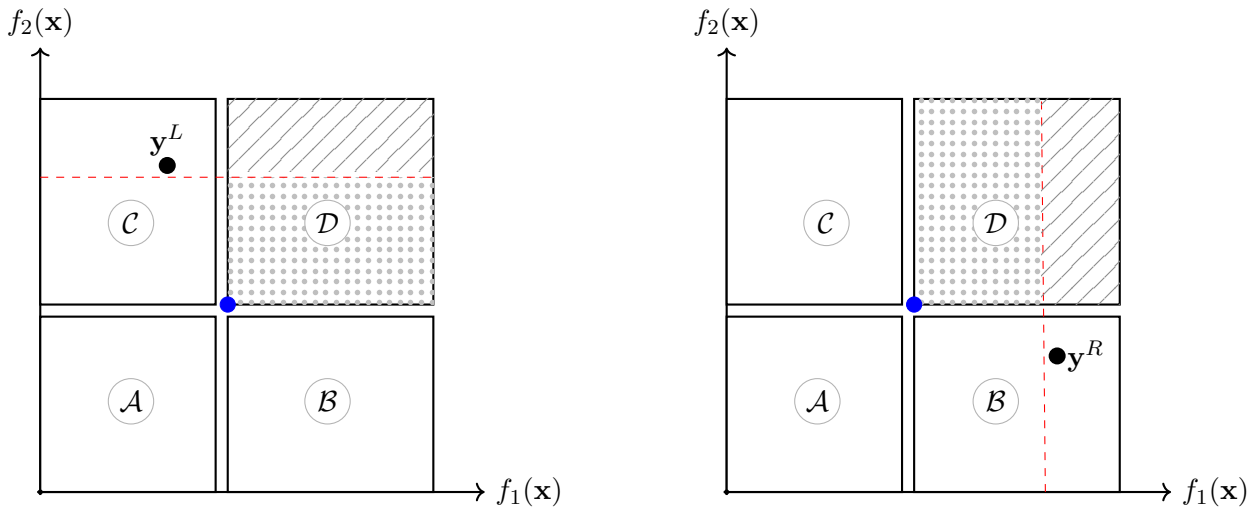
5.3 Upper Bound Tightening

Consider an active node that cannot be pruned by the pruning mechanisms and is therefore selected for a feasibility check. If the node is feasible, discovering a new feasible point is only useful if it is not dominated by the current set of approximate nondominated points in $\tilde{\mathcal{Y}}_N$, i.e., if it improves upon the existing set in at least one objective. This observation implies that certain portions of the rectangular search region associated with an active node may be discarded even when the entire node cannot be pruned.

Motivated by this observation, we introduce an upper bound tightening procedure. By dynamically leveraging the points in $\tilde{\mathcal{Y}}_N$, this procedure imposes tighter upper bounds on the objective values of an active node, thereby reducing the search space prior to each feasibility check. Consider an active node whose criterion-space search region is represented by the rectangle

$$[\mathrm{LB}_1, \mathrm{UB}_1] \times [\mathrm{LB}_2, \mathrm{UB}_2].$$

We analyze the relative positions of the current nondominated feasible points $\mathbf{y} \in \tilde{\mathcal{Y}}_N$ with respect to this rectangle to determine whether the region can be further tightened. Specifically, we identify neighboring nondominated points that lie immediately outside the rectangle but induce dominance



(a) Tightening UB_2 for Rectangle $\mathcal{D}$

(b) Tightening UB_1 for Rectangle $\mathcal{D}$

Figure 6: Geometrical representation of Upper Bound Tightening. Dotted region is active and hatched region is dominated.

restrictions on its interior. Let the bottom-left corner of the rectangle be given by $(\mathrm{LB}_1, \mathrm{LB}_2)$, see the blue circle in Figure 6. Suppose there exists a left neighbor $\mathbf{y}^L = (y_1^L, y_2^L) \in \tilde{\mathcal{Y}}_N$ such that

$$y_1^L \leq \mathrm{LB}_1 \quad \text{and} \quad \mathrm{LB}_2 < y_2^L \leq \mathrm{UB}_2.$$

Then, any feasible point within the node must strictly improve upon y_2^L in the second objective to remain nondominated with respect to $\mathbf{y}^L$. Since the objective values are integer-valued, the upper bound can be tightened as

$$\mathrm{UB}_2 \leftarrow y_2^L - 1.$$

Geometrically, this corresponds to a horizontal cut of the search rectangle (see Figure 6(a)). Similarly, suppose there exists a lower neighbor $\mathbf{y}^R = (y_1^R, y_2^R) \in \tilde{\mathcal{Y}}_N$ such that

$$y_2^R \leq \mathrm{LB}_2 \quad \text{and} \quad \mathrm{LB}_1 < y_1^R \leq \mathrm{UB}_1.$$

Then, any feasible point within the node must strictly improve upon y_1^R in the first objective to remain nondominated with respect to $\mathbf{y}^R$. This leads to the tightening

$$\mathrm{UB}_1 \leftarrow y_1^R - 1.$$

Geometrically, this corresponds to a vertical cut of the search rectangle (see Figure 6(b)).

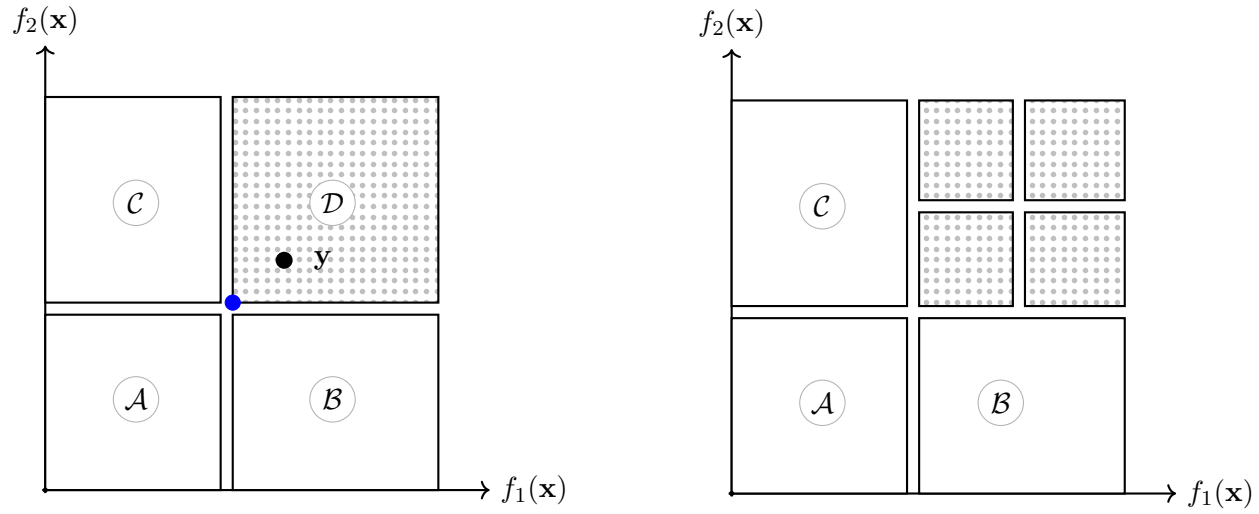
Observe that, to tighten the upper bounds to their maximum extent, it is sufficient to consider only the two *nearest* nondominated neighbors of the bottom-left corner of the rectangle: one restricting the second objective and one restricting the first objective, provided such points exist. Since the points in $\tilde{\mathcal{Y}}_N$ are maintained in increasing order of their first objective values, identifying these critical neighboring points can be done efficiently. In our implementation, we exploit this observation to perform upper bound tightening with minimal overhead while maximizing the reduction of the search region.

5.4 Skip Feasibility Checking

Consider an active node whose criterion-space region is defined by $[\mathrm{LB}_1, \mathrm{UB}_1] \times [\mathrm{LB}_2, \mathrm{UB}_2]$. Suppose that this node cannot be pruned by the dominance-based pruning mechanism and is therefore selected for a feasibility check. However, it is possible that there already exists a point $\mathbf{y} = (y_1, y_2) \in \tilde{\mathcal{Y}}_N$ that lies within this region, i.e.,

$$\mathrm{LB}_1 \leq y_1 \leq \mathrm{UB}_1 \quad \text{and} \quad \mathrm{LB}_2 \leq y_2 \leq \mathrm{UB}_2.$$

In such a case, the node is guaranteed to be feasible. Therefore, the feasibility check can be safely skipped, saving computational effort. Instead, the node is directly marked as feasible, and the algorithm proceeds immediately to the branching step. Figure 7 illustrates the geometric intuition behind this mechanism. As shown in the figure, when an active node $\mathcal{D}$ (represented by the dotted region) contains a previously identified nondominated point $\mathbf{y} \in \tilde{\mathcal{Y}}_N$, the algorithm bypasses the feasibility check and directly decomposes the node through branching.



(a) Point $\mathbf{y} \in \tilde{\mathcal{Y}}_N$ is feasible for Rectangle $\mathcal{D}$

(b) Rectangle $\mathcal{D}$ after branching operation

Figure 7: Geometrical representation of Skip Feasibility Checking. Dotted region is active.

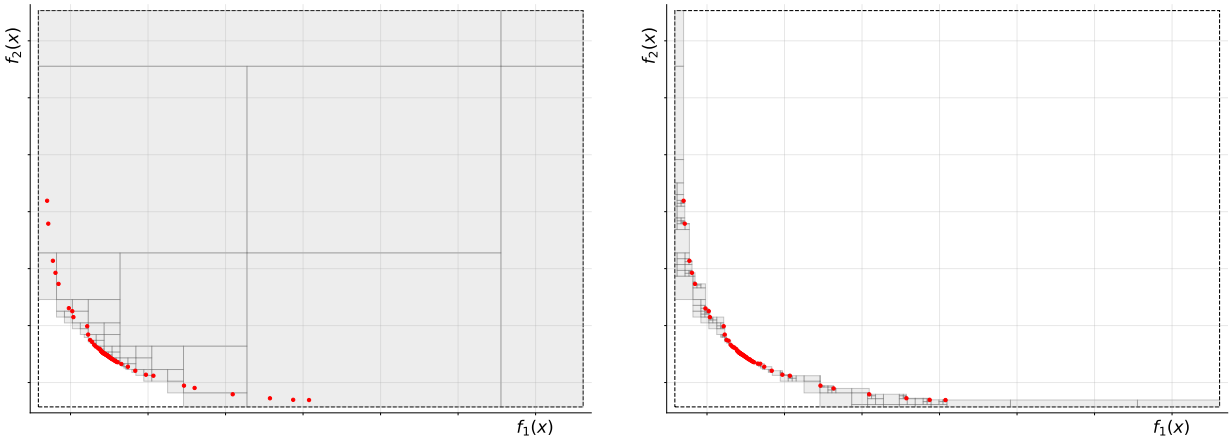
6 Quality Metric and Postprocessing

The proposed Quadtree Search method is guaranteed to generate the complete nondominated frontier if no global time limit is imposed. However, when the algorithm is terminated prematurely, the resulting set $\tilde{\mathcal{Y}}_N$ represents only an approximation of the true nondominated frontier. This raises the question of how to assess the quality of such an approximation.

A key advantage of the proposed method is that it naturally provides a quantitative measure of solution quality by leveraging the set of open nodes remaining in the priority queue $\mathcal{L}$. Each open node corresponds to a rectangular region in the criterion space, and its area can be computed easily. Based on this observation, we define a performance metric Δ that represents the percentage of the unexplored area relative to the initial search region. Specifically,

$$\Delta = 100 \times \frac{\sum_{d \in \mathcal{L}} (\mathrm{UB}_1^d - \mathrm{LB}_1^d)(\mathrm{UB}_2^d - \mathrm{LB}_2^d)}{(\mathrm{GUB}_1 - \mathrm{GLB}_1)(\mathrm{GUB}_2 - \mathrm{GLB}_2)},$$

where the numerator represents the total (disjoint) area of the remaining open rectangles, and the denominator corresponds to the area of the initial criterion space. Here, GLB_i and GUB_i denote the global lower and upper bounds for objective $i \in \{1, 2\}$, as defined in Section 5.1. The metric Δ provides a rigorous quality guarantee for $\tilde{\mathcal{Y}}_N$. In particular, $\Delta = 0\%$ certifies that the nondominated frontier has been fully recovered, whereas $\Delta > 0$ quantifies the remaining unexplored region that may still contain nondominated points.



(a) Without Postprocessing ($\Delta = 96.3\%$) (b) With Postprocessing ($\Delta = 3.7\%$)

Figure 8: Comparison of solution quality with and without postprocessing.

To ensure that Δ is as tight and informative as possible, a postprocessing step is applied to the open nodes in $\mathcal{L}$. This is necessary because, under premature termination, some nodes may not have been fully processed, e.g., they may still be prunable or partially dominated, but the algorithm did not have sufficient time to detect this. Without refinement, these nodes may artificially inflate the value of Δ. We therefore propose a simple and efficient postprocessing procedure that refines the remaining nodes using previously introduced techniques. Specifically, the following steps are applied iteratively:

1. **Pruning:** If the lower-bound vector $(\mathrm{LB}_1^d, \mathrm{LB}_2^d)$ of a node $d \in \mathcal{L}$ is dominated by any $\mathbf{y} \in \tilde{\mathcal{Y}}_N$, the node is pruned (see Section 4.2).

2. **Upper Bound Tightening:** Apply the upper bound tightening procedure (Section 5.3) using $\tilde{\mathcal{Y}}_N$ to potentially reduce $(\mathrm{UB}_1^d, \mathrm{UB}_2^d)$. If the resulting rectangle becomes empty, the node is discarded.

3. **Node Splitting:** If the refined rectangle contains a known solution $\mathbf{y} \in \tilde{\mathcal{Y}}_N$, the node is split into its four children and added back to $\mathcal{L}$ (see Section 5.4).

This process is repeated until no further pruning, tightening, or splitting is possible for any open node in $\mathcal{L}$. At termination, the remaining nodes in $\mathcal{L}$ are disjoint and correspond to regions of the criterion space whose union exactly represents the unexplored area. The effectiveness of this postprocessing step is illustrated in Figure 8. The figure presents results for a BOTSP instance with 80 cities, where the algorithm is terminated after one hour. The shaded regions indicate the unexplored portion of the criterion space. Without postprocessing, the remaining open nodes suggest an unexplored area of 96.3% (Figure 8(a)). After applying the proposed postprocessing

procedure, this value is reduced to 3.7% (Figure 8(b)), demonstrating a substantial improvement in the quality assessment. One interesting observation from the figure is that, even for such a relatively large instance, the proposed method is able to prove optimality for the middle section of the nondominated frontier. In practice, decision-makers are often most interested in this region, as it provides a better balance between conflicting objectives. Notably, this portion of the frontier has already been certified as optimal, since no rectangles remain unexplored in that region. This outcome is not coincidental. The proposed method maintains nodes in the priority queue in a specific manner, sorting them in nondecreasing order of $\mathrm{LB}_1 + \mathrm{LB}_2$. As a result, the middle region of the frontier is explored and completed earlier in the search process.

7 Numerical Experiments

The Quadtree Search method is implemented in python 3.12.10 and we used OR-Tools CP solver 9.14.6206 as the black box solver for feasibility checking. For benchmarking, we utilized the MultiObjectiveAlgorithms.jl (MOA) package via the JuMP modeling language executed within the Julia 1.12.4 environment. Gurobi 13.0 is used as the underlying integer programming solver. The computational experiments are conducted on a Dell Pro Max Slim FCS1250 desktop with an Intel Core Ultra 7 265 processor running at 2.40GHz, 32 GB of RAM and the Microsoft Windows 11 Enterprise operating system using a single thread.

7.1 Impact of Enhancements

To evaluate the individual contribution of each algorithmic enhancement, we conduct a sensitivity analysis using BOAP instances originally introduced in the study of Tuyttens et al. [2000]. These instances are publicly available on GitHub (see https://github.com/d-tuyttens/Bicriteria-Assignment-Problem). On the corresponding GitHub repository, one instance per class is provided for 15 problem sizes, with the number of jobs given by

$$\{5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 90, 100\}.$$

To obtain a more reliable assessment of each enhancement, a single instance per class is insufficient. Therefore, we generate 9 additional instances per class using the same experimental setup as in Tuyttens et al. [2000], where the assignment costs are drawn independently and uniformly at random from the discrete interval $[1, 20]$. In total, 10 instances per class are used in this experiment.

Note that in the proposed method, all algorithmic enhancements are enabled by default. Therefore, to measure the impact of each component, we selectively disable one enhancement at a time and record the resulting increase in solution time. This allows us to quantify the marginal contribution of each individual enhancement. For a fair and consistent comparison, we restrict attention to instances that can be solved to optimality within the imposed time limit of 3,600 seconds under all tested configurations. As a result, the largest instance classes with 70, 80, 90, and 100 jobs are excluded from the reported analysis, since not all instances in these classes can be solved within the time limit in every scenario. Table 2 summarizes the computational impact of the four algorithmic enhancements: Initial Bounds (Init. Bounds), Upper Bound Tightening (UB Tight.), Skip Feasibility Checking (Skip Feas.), and Warm Starting (Warm). The Baseline column reports the average execution time (in seconds) with all features enabled. The subsequent columns quantify the percentage increase in solution time when a specific enhancement is disabled.

The analysis reveals that *Upper Bound Tightening* is the most critical enhancement for maintaining scalability. As the problem dimension increases, the percentage increase in computational

Table 2: Impact of Individual Enhancements on Computational Performance

	Baseline	Percentage Increase (%) in Time After Disabling			
Class	Time (s)	Init. Bounds	UB Tight.	Skip Feas.	Warm
5	0.3	28.9	28.6	51.7	−55.7
10	1.8	19.2	39.2	32.8	−3.4
15	5.4	23.1	67.8	36.7	6.7
20	13.5	15.4	64.9	30.4	5.6
25	27.0	18.5	66.8	26.4	7.4
30	52.1	14.0	72.1	27.0	5.2
35	87.3	12.8	62.2	27.2	4.2
40	136.1	13.0	73.8	27.5	5.0
45	229.6	9.4	72.9	23.7	4.7
50	401.8	8.6	103.5	37.4	5.0
60	1,037.9	17.4	73.5	43.8	14.8

time when this enhancement is disabled grows substantially, reaching a peak of 103.45% for Class 50. This highlights its effectiveness in pruning large portions of the criterion space that would otherwise require expensive solver calls. *Skip Feasibility Checking* also provides a significant computational advantage across all instance sizes. When disabled, the runtime increases by between 24% and 52%, confirming the value of bypassing unnecessary solver calls by exploiting previously identified nondominated points. The *Initial Bounds* enhancement is the third most impactful component. Disabling it results in an increase in solution time ranging from 8.6% to 28.9% on average, demonstrating its meaningful contribution to reducing the initial search space and improving overall efficiency. Finally, we observe that the *Warm Start* enhancement is generally effective as well, demonstrating performance gains in 10 out of the 12 problem classes. For the two smallest instance classes (Classes 5 and 10), the negative percentage increases indicate that the overhead associated with generating warm starts outweighs the computational savings. However, as combinatorial complexity increases, warm starts become increasingly beneficial, yielding an efficiency improvement of up to 14.80% for Class 60.

To further validate the results of Table 2, we conducted a performance profiling study of the enhancements following the methodology of [Dolan and Moré, 2002]. A performance profile visually represents relative efficiency by plotting the performance ratio along the horizontal axis. This ratio is defined as the execution time of a specific method on an instance divided by the minimum time achieved by any method for that same instance. The vertical axis displays the cumulative fraction of instances for which a method's ratio is less than or equal to the value on the horizontal axis. In this visualization, a curve positioned toward the top-left indicates superior performance.

Figure 9 illustrates a similar pattern as Table 2. The performance profile shows that removing different enhancements affects runtime differently. Disabling warm start or initial bounds causes moderate slowdowns, with most instances still finishing within about 1.3 times to 1.4 times of the baseline. Disabling skip feasibility checking has a larger effect, requiring roughly 1.6 times the execution time for completion across instances. The largest degradation comes from disabling upper bound tightening: only about 10% of instances finish within 1.5 times of the baseline, and more than 3 times of execution time are needed before nearly all instances finish. This indicates that upper bound tightening is the most critical enhancement execution time.

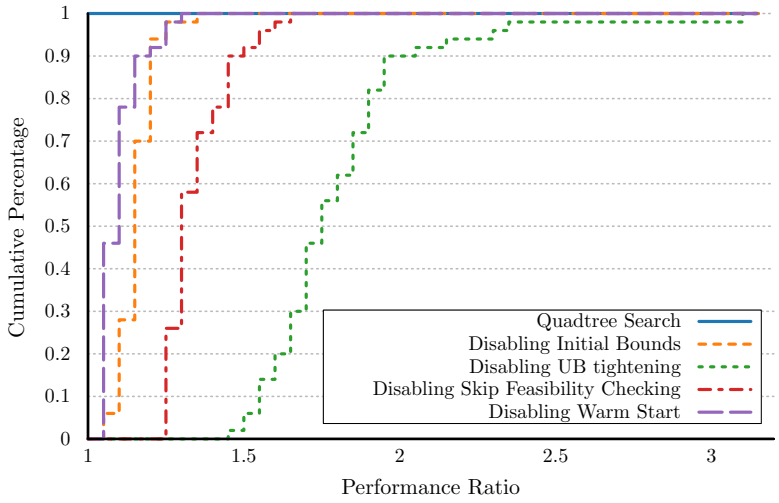


Figure 9: Performance Profiles Comparing the Algorithmic Enhancements

7.2 Comparative Analysis against Benchmark

To evaluate the performance of the proposed method, we benchmark it against the state-of-the-art criterion-space search algorithm by Kirlik and Sayin [2014], available in the *MOA* package in Julia. We refer to this method as the KS algorithm throughout this section.

We intentionally select three distinct BOIPs for this comparison. The primary motivation of our approach is to address Challenge 2. BOAP instances are relatively tractable for existing criterion-space search methods, and therefore Challenge 2 is not particularly pronounced in this setting. As a result, it is reasonable to expect the KS algorithm to perform significantly better on these instances. In contrast, BOSCP instances present a moderate level of difficulty. For sufficiently large instances, the impact of Challenge 2 starts to become evident, allowing the advantages of the proposed method to emerge. Finally, BOTSP instances are significantly more challenging for criterion-space search methods. In this setting, the impact of Challenge 2 manifests rapidly, and our approach is expected to demonstrate substantial performance improvements.

The results of these experiments are summarized in Tables 3, 4, and 5. Each row corresponds to a class of instances, and the reported values represent averages over all instances within that class. For both methods, we report the instance size, total execution time (Time (s)), and the cardinality of the approximated nondominated set ($|\tilde{\mathcal{Y}}_N|$). When an algorithm is not terminated prematurely, this set corresponds to the complete nondominated frontier. For the Quadtree Search method, we additionally report several performance indicators, including the total number of explored nodes (Nodes), the number of feasible and infeasible nodes identified (Feasi and Infeasi), and the number of nodes bypassed via the Skip Feasibility Checking mechanism (Skip). We also report the number of timeout nodes (TO), the number of nodes pruned by dominance (Prun), and the quality metric (Δ). Finally, we report the number of instances in which the proposed algorithm outperforms the KS algorithm (Wins). For example, a value of 7/10 indicates that, out of 10 instances in the corresponding class, the proposed method outperformed the KS algorithm in 7 instances.

7.2.1 Performance Comparison on BOAP Instances

For the performance analysis in this section, we utilize the set of benchmark instances proposed by Tuyttens et al. [2000], which provide one instance per class. This choice is sufficient for the BOAP

setting, as the KS algorithm exhibits strong performance on these instances and the problem structure does not activate Challenge 2. Consequently, increasing the number of instances per class would not meaningfully alter the comparative outcome. Instead, this experiment primarily serves to verify that the proposed method is capable of correctly generating the complete nondominated frontier when it terminates naturally within the imposed time limit of 3,600 seconds.

Table 3: Comparison of the Quadtree Search Method and the KS Algorithm on BOAP Instances.

	Quadtree Search Method									**KS Algorithm**		
Class	Time (s)	$\lvert\tilde{\mathcal{Y}}_N\rvert$	Nodes	Feasi	Infeasi	Skip	TO	Prun	Δ	Time (s)	$\lvert\tilde{\mathcal{Y}}_N\rvert$	**Wins**
5	0.4	8.0	97.0	7.0	41.0	17.0	0.0	32.0	0.0	0.3	8.0	0/1
10	1.8	16.0	205.0	11.0	82.0	41.0	0.0	71.0	0.0	0.6	16.0	0/1
15	5.7	39.0	497.0	37.0	192.0	87.0	0.0	181.0	0.0	1.9	39.0	0/1
20	12.2	55.0	561.0	63.0	214.0	82.0	0.0	202.0	0.0	3.3	55.0	0/1
25	25.2	74.0	777.0	77.0	314.0	119.0	0.0	267.0	0.0	4.6	74.0	0/1
30	45.5	88.0	853.0	101.0	323.0	116.0	0.0	313.0	0.0	6.5	88.0	0/1
35	60.3	81.0	829.0	95.0	295.0	120.0	0.0	321.0	0.0	7.1	81.0	0/1
40	128.7	127.0	1,225.0	155.0	459.0	160.0	0.0	451.0	0.0	11.6	127.0	0/1
45	191.4	114.0	1,189.0	168.0	457.0	142.0	0.0	422.0	0.0	10.5	114.0	0/1
50	465.1	163.0	1,521.0	246.0	543.0	148.0	0.0	584.0	0.0	16.4	163.0	0/1
60	440.3	128.0	1,349.0	171.0	504.0	178.0	1.0	495.0	0.0	17.9	128.0	0/1
70	1,670.0	174.0	1,707.0	227.0	630.0	208.0	1.0	636.0	0.0	29.1	174.0	0/1
80	3,101.8	195.0	1,886.0	256.0	720.0	222.0	29.0	690.0	0.0	42.7	195.0	0/1
90	3,600.0	149.0	1,298.0	189.0	493.0	145.0	7.0	464.0	0.5	47.7	191.0	0/1
100	3,600.0	137.0	990.0	148.0	354.0	121.0	9.0	358.0	0.6	64.3	223.0	0/1

Table 3 illustrates a clear performance trade-off between the Quadtree Search method and the KS algorithm. We observe that, for all instances solved within the time limit, the Quadtree search method successfully identifies the complete nondominated frontier, matching the benchmark in terms of cardinality ($|\tilde{\mathcal{Y}}_N|$). Although the Quadtree search method reaches the time limit for larger instances ($n = 90$ and $n = 100$), it still produces high-quality approximations, with the quality metric (Δ) remaining below 0.6%. Overall, as expected, the KS algorithm significantly outperforms the proposed method on BOAP instances, confirming that this problem class is well-suited for existing criterion-space search techniques.

7.2.2 Performance Comparison on BOSCP Instances

We now compare the Quadtree Search method with the KS algorithm on BOSCP instances introduced by Gandibleux et al. [1998]. These instances are publicly available on GitHub (see https://github.com/vOptSolver/vOptLib/tree/master/SCP/instances). The dataset consists of 11 classes of instances, each characterized by a specific number of constraints and variables. Each class contains two instances (labeled C and D on the GitHub page), resulting in a total of 22 test instances. Compared to BOAP instances, BOSCP instances are more challenging for existing criterion-space search methods. Consequently, the impact of Challenge 2 is expected to become apparent for sufficiently large instances. A time limit of 3,600 seconds is imposed on all instances, except for the largest class, which is substantially more difficult. For this class, we extend the time limit to 7,200 seconds to assess whether the KS algorithm can identify any nondominated points. This is why the largest class is separated by a line in Table 4.

The results in Table 4 show that the proposed method is not competitive with the KS algorithm in 10 out of the 11 classes. Nevertheless, for these 10 classes, the proposed method successfully recovers the complete nondominated frontier for all instances when it terminates naturally within the

Table 4: Comparison of the Quadtree Search Method and the KS Algorithm on BOSCP Instances.

	Quadtree Search Method									KS Algorithm		
Class	Time (s)	$\lvert\tilde{\mathcal{Y}}_N\rvert$	Nodes	Feas	Infeasi	Skip	TO	Prun	Δ	Time (s)	$\lvert\tilde{\mathcal{Y}}_N\rvert$	**Wins**
10×100	1.8	7.5	143.0	6.0	72.5	29.5	0.0	35.0	0.0	0.6	7.5	0/2
40×200	56.3	33.5	745.0	59.0	382.0	127.0	0.0	177.0	0.0	12.1	33.5	0/2
$40 \times 400^{\star}$	608.7	15.0	486.0	17.0	219.0	85.0	89.0	76.0	0.0	4.1	15.0	1/2
40×200	13.1	13.0	299.0	13.0	158.0	61.5	0	66.5	0.0	1.4	13.0	0/2
60×600	1,850.9	47.5	1,507.0	70.5	811.0	288.0	68.0	269.5	0.0	42.0	47.5	0/2
60×600	2,337.8	22.0	640.0	19.0	370.0	105.0	45.0	101.0	0.1	52.1	22.0	0/2
80×800	42.3	13.0	455.0	19.0	226.0	94.5	0	120.5	0.0	3.3	13.0	0/2
80×800	3,600.0	9.0	182.5	1.0	81.5	22.0	78.0	0.0	0.2	249.4	26.0	0/2
$100 \times 1,000$	3,600.0	7.5	202.0	0.0	90.0	15.0	93.5	3.5	0.2	98.6	18.5	0/2
$100 \times 1,000$	3,600.0	8.5	303.0	4.5	191.0	21.0	85.0	1.5	0.1	642.4	15.0	0/2
$200 \times 2,000$	7,200.0	15.5	169.5	0.0	15.0	8.5	144.0	2.0	0.4	7,200.0	0.0	2/2

* The MOA package (KS algorithm) encountered a numerical issue on one instance and terminated prematurely; this instance is counted as a win for the Quadtree Search method, and results are reported for only the remaining instance in that row.

time limit. In cases of premature termination, the method produces high-quality approximations, with the quality metric (Δ) remaining below 0.2%. A notable shift in performance is observed for the largest class, as anticipated due to the activation of Challenge 2. In this case, the KS algorithm fails to generate any nondominated point, and thus provides no approximation. In contrast, the proposed method is able to generate approximate nondominated sets with provable quality guarantees for both instances. On average, the method identifies 15.5 mutually nondominated points (primarily obtained during the warm-start phase) and uses the remaining time to refine the search space and establish quality guarantees. The resulting quality metric (Δ) is approximately 0.4%, providing a strong certificate of the solution quality.

7.2.3 Performance Comparison on BOTSP Instances

We now evaluate the proposed method against the KS algorithm on BOTSP instances, where the impact of Challenge 2 manifests rapidly for existing criterion-space search methods. This is one of the main reasons why, in the existing literature, BOTSP instances have been relatively underexplored or studied only for small problem sizes. Since there is no widely established benchmark set for this class of problems, we generate our own test instances. Specifically, we consider five problem sizes with the number of cities

$$n \in \{10, 20, 40, 80, 160\},$$

and generate 10 instances per class. The instances are constructed following the setup proposed by Filippi and Stevanato [2013], which is consistent with standard generation procedures for the single-objective TSP. In particular, for each objective, we sample n city coordinates uniformly at random from the integer square $[0, 1000]^2$, and define edge costs as the rounded Euclidean distances between cities. Similar to the BOSCP experiments, a time limit of 3,600 seconds is imposed on all instances, except for the largest class ($n = 160$), which is substantially more challenging. For this class, the time limit is extended to 7,200 seconds to assess whether the KS algorithm can identify any nondominated points. For clarity, this largest class is separated by a horizontal line in Table 5.

The results for BOTSP, summarized in Table 5, show that for smaller instances ($n = 10$ and $n = 20$), the KS algorithm performs competitively. However, its performance deteriorates rapidly as the instance size increases and Challenge 2 becomes more pronounced. In particular, the number

Table 5: Comparison of the Quadtree Search Method and the KS Algorithm on BOTSP Instances.

	Quadtree Search Method									**KS Algorithm**		
Class	Time (s)	$\lvert\tilde{\mathcal{Y}}_N\rvert$	Nodes	Feas	Infeasi	Skip	TO	Prun	Δ	Time (s)	$\lvert\tilde{\mathcal{Y}}_N\rvert$	**Wins**
10	10.4	17.1	532.2	13.7	316.9	119.1	0.0	82.5	0.0	1.4	17.1	0/10
20*	955.7	93.6	2,448.0	108.3	1,425.0	501.2	8.8	404.7	0	29.7	93.6	1/10
40	3,600.0	80.5	1,018.0	79.4	571.8	206.7	60.2	99.9	5.5	3,207.8	85.1	7/10
80	3,600.0	81.5	489.5	79.8	269.0	103.0	25.4	12.3	3.7	3,600.0	8.5	10/10
160	7,200.0	40.6	278.5	33.3	143.2	35.1	59.70	0.0	8.6	7,200.0	0.3	10/10

* The MOA package (KS algorithm) encountered a numerical issue on one instance and terminated prematurely; this instance is counted as a win for the Quadtree Search method, and results are reported for only the remaining instances in that row.

of nondominated points identified by the KS algorithm quickly decreases and approaches zero for larger instances, with almost no nondominated points generated for the largest class. In contrast, the Quadtree Search method demonstrates significantly better scalability, producing high-quality approximations with provable guarantees. Even for the largest instances with $n = 160$, the method achieves a quality metric of $\Delta \leq 8.6\%$, indicating a relatively small unexplored portion of the criterion space.

8 Concluding Remarks

In this study, we proposed the Quadtree Search Method, which offers a flexible alternative to traditional criterion-space search methods by prioritizing feasibility over solving each subproblem to optimality. This design is particularly advantageous for challenging instances, where exact subproblem optimization becomes a computational bottleneck. The method is capable of producing high-quality approximations of the nondominated frontier, along with a rigorous quality measure. By allowing the search to proceed even when certain nodes cannot be fully resolved within a prescribed time limit, the method avoids the stagnation commonly encountered in traditional criterion-space search methods. The computational results demonstrate that the proposed method is especially effective on challenging problem classes such as BOSCP and BOTSP, where it delivers approximations with provable quality guarantees within the imposed time limits. Overall, these findings suggest that the Quadtree Search Method is a promising framework for large-scale BOIPs, where both robustness and solution quality are critical.

Future research will focus on extending this framework to multi-objective integer programming problems with an arbitrary number of objectives ($p \geq 2$). While the proposed method can, in principle, be generalized to higher dimensions, a key challenge lies in the exponential growth of the search tree, as each node may generate up to 2^p children. This leads to a 2^p-ary tree structure, which can significantly impact scalability. Addressing this issue will require the development of more refined branching and pruning strategies to control the growth of the search space. In addition, we plan to extend the method to handle mixed-integer programming problems by incorporating continuous decision variables. This extension would broaden the applicability of the framework to a wider range of practical problems, including complex engineering design and resource allocation settings.

References

Sophie N. Parragh and Fabien Tricoire. Branch-and-Bound for Bi-objective Integer Programming. *INFORMS Journal on Computing*, 31(4):805–822, October 2019. doi: 10.1287/ijoc.2018.0856. URL https://ideas.repec.org/a/inm/orijoc/v31y2019i4p805-822.html.

Pietro Belotti, Banu Soylu, and Margaret M. Wiecek. Fathoming rules for biobjective mixed integer linear programs: Review and extensions. *Discrete Optimization*, 22:341–363, 2016. ISSN 1572-5286. doi: https://doi.org/10.1016/j.disopt.2016.09.003. URL https://www.sciencedirect.com/science/article/pii/S1572528616300871.

Sune Gadegaard, Lars Nielsen, and Matthias Ehrgott. Bi-objective Branch-and-Cut Algorithms Based on LP Relaxation and Bound Sets. *INFORMS Journal on Computing*, 31(4):790–804, 06 2019. doi: 10.1287/ijoc.2018.0846.

Nicolas Forget and Sophie N. Parragh. Enhancing branch-and-bound for multiobjective 0-1 programming, 2023. URL https://doi.org/10.1287/ijoc.2022.0299.

Thomas Stidsen, Kim Allan Andersen, and Bernd Dammann. A Branch and Bound Algorithm for a Class of Biobjective Mixed Integer Programs. *Management Science*, 60(4):1009–1032, 2014. doi: 10.1287/mnsc.2013.1802. URL https://doi.org/10.1287/mnsc.2013.1802.

Thomas Stidsen and Kim Allan Andersen. A hybrid approach for biobjective optimization, journal = Discrete Optimization. 28:89–114, 2018. ISSN 1572-5286. doi: https://doi.org/10.1016/j.disopt.2018.02.001. URL https://www.sciencedirect.com/science/article/pii/S1572528617300609.

Marianna De Santis, Gabriele Eichfelder, Julia Niebling, and Stefan Rocktäschel. Solving Multiobjective Mixed Integer Convex Optimization Problems. *SIAM Journal on Optimization*, 30(4): 3122–3145, 2020. doi: 10.1137/19M1264709. URL https://doi.org/10.1137/19M1264709.

Tyler Perini, Natashia Boland, Diego Pecin, and Martin Savelsbergh. A Criterion Space Method for Biobjective Mixed Integer Programming: The Boxed Line Method. *INFORMS Journal on Computing*, 32(1):16–39, 2020. doi: 10.1287/ijoc.2019.0887. URL https://doi.org/10.1287/ijoc.2019.0887.

Natashia Boland, Hadi Charkhgard, and Martin Savelsbergh. A Criterion Space Search Algorithm for Biobjective Integer Programming: The Balanced Box Method. *INFORMS Journal on Computing*, 27(4):735–754, 2015. doi: 10.1287/ijoc.2015.0657. URL https://doi.org/10.1287/ijoc.2015.0657.

Marianna De Santis, Giorgio Grani, and Laura Palagi. Branching with hyperplanes in the criterion space: The frontier partitioner algorithm for biobjective integer programming. *European Journal of Operational Research*, 283(1):57–69, 2020. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2019.10.034. URL https://www.sciencedirect.com/science/article/pii/S0377221719308690.

Banu Soylu. The search-and-remove algorithm for biobjective mixed-integer linear programming problems. *European Journal of Operational Research*, 268(1):281–299, 2018. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2018.01.026. URL https://www.sciencedirect.com/science/article/pii/S037722171830047X.

Banu Lokman and Murat Köksalan. Finding all nondominated points of multi-objective integer programs. *Journal of Global Optimization*, 57(2):347–365, 2013. ISSN 1573-2916. doi: 10.1007/s10898-012-9955-7. URL https://doi.org/10.1007/s10898-012-9955-7.

Hadi Charkhgard, Hanieh Rastegar Moghaddam, Ali Eshragh, Sasan Mahmoudinazlou, and Kimia Keshanian. Solving hard bi-objective knapsack problems using deep reinforcement learning. *Discrete Optimization*, 55:100879, 2025. ISSN 1572-5286. doi: https://doi.org/10.1016/j.disopt.2025.100879. URL https://www.sciencedirect.com/science/article/pii/S1572528625000027.

Alexander Bockmayr and Nicolai Pisaruk. Detecting infeasibility and generating cuts for mixed integer programming using constraint programming. *Computers Operations Research*, 33 (10):2777–2786, 2006. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2005.01.010. URL https://www.sciencedirect.com/science/article/pii/S0305054805000110. Part Special Issue: Constraint Programming.

Vira Chankong and Yacov Y. Haimes. Multiobjective decision making: Theory and methodology. 1983. URL https://api.semanticscholar.org/CorpusID:56746422.

Gokhan Kirlik and Serpil Sayin. A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems. *European Journal of Operational Research*, 232 (3):479–488, None 2014. doi: 10.1016/j.ejor.2013.08.001. URL https://ideas.repec.org/a/eee/ejores/v232y2014i3p479-488.html.

Rui Dai and Hadi Charkhgard. A two-stage approach for bi-objective integer linear programming. *Operations Research Letters*, 46(1):81–87, 2018. ISSN 0167-6377. doi: https://doi.org/10.1016/j.orl.2017.11.011. URL https://www.sciencedirect.com/science/article/pii/S0167637717302250.

Kerstin Dächert, Tino Fleuren, and Kathrin Klamroth. A simple, efficient and versatile objective space algorithm for multiobjective integer programming. *Mathematical Methods of Operations Research*, 100(1):351–384, 2024. ISSN 1432-5217. doi: 10.1007/s00186-023-00841-0. URL https://doi.org/10.1007/s00186-023-00841-0.

Banu Soylu and Gazi Bilal Yıldız. An exact algorithm for biobjective mixed integer linear programming problems. *Comput. Oper. Res.*, 72:204–213, 2016. URL https://api.semanticscholar.org/CorpusID:33750633.

Mariana Mesquita-Cunha, José Rui Figueira, and Ana Paula Barbosa-Póvoa. New ϵ-constraint methods for multi-objective integer linear programming: A Pareto front representation approach. *European Journal of Operational Research*, 306(1):286–307, 2023. ISSN 0377-2217. doi: 10.1016/j.ejor.2022.07.044. URL https://doi.org/10.1016/j.ejor.2022.07.044.

Cristina Bazgan, Stefan Ruzika, Clemens Thielen, and Daniel Vanderpooten. The Power of the Weighted Sum Scalarization for Approximating Multiobjective Optimization Problems. *Theory of Computing Systems*, 66:395–415, 2022. ISSN 1433-0490. doi: 10.1007/s00224-021-10066-5. URL https://doi.org/10.1007/s00224-021-10066-5.

Saliha Ferda Doğan, Özlem Karsu, and Firdevs Ulus. An exact algorithm for biobjective integer programming problems. *Computers Operations Research*, 132:105298, 2021. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2021.105298. URL https://www.sciencedirect.com/science/article/pii/S0305054821000903.

Payman Ghasemi Saghand, Fabian Rigterink, Vahid Mahmoodian, and Hadi Charkhgard. Solving multiplicative programs by binary-encoding the multiplication operation. *Computers & Operations Research*, 159:106340, 2023. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2023.106340. URL https://www.sciencedirect.com/science/article/pii/S0305054823002046.

Aritra Pal and Hadi Charkhgard. A Feasibility Pump and Local Search Based Heuristic for Bi-Objective Pure Integer Linear Programming. *INFORMS Journal on Computing*, 31(1):115–133, February 2019. doi: 10.1287/ijoc.2018.0814. URL https://ideas.repec.org/a/inm/orijoc/v31y2019i1p115-133.html.

Banu Soylu. Heuristic approaches for biobjective mixed 0–1 integer linear programming problems. *European Journal of Operational Research*, 245(3):690–703, 2015. URL https://ideas.repec.org/a/eee/ejores/v245y2015i3p690-703.html.

D. Tuyttens, J. Teghem, Ph. Fortemps, and K. Van Nieuwenhuyze. Performance of the MOSA Method for the Bicriteria Assignment Problem. *Journal of Heuristics*, 6(3):295–310, 2000. doi: 10.1023/A:1009670112978.

Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002. doi: https://doi.org/10.1007/s101070100263.

Xavier Gandibleux, David Vancoppenolle, and Daniel Tuyttens. A first making use of GRASP for solving MOCO problems. In *Proceedings of the 14th International Conference on Multiple Criteria Decision Making (MCDM)*, Charlottesville, USA, June 8–12 1998.

Carlo Filippi and Elisa Stevanato. Approximation schemes for bi-objective combinatorial optimization and their application to the TSP with profits. *Computers & Operations Research*, 40 (10):2418–2428, 2013. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2013.02.011. URL https://www.sciencedirect.com/science/article/pii/S0305054813000518.