On the Design and Anytime Performance of Indicator-based Branch and Bound for Multi-objective Combinatorial Optimization

Alexandre D. Jesus
University of Coimbra, CISUC, DEI, Coimbra, Portugal
Univ. Lille, CNRS, Centrale Lille, Inria,
UMR 9188 - CRIStAL, F-59000 Lille, France
ajesus@dei.uc.pt

Luís Paquete
University of Coimbra, CISUC, DEI, Coimbra, Portugal
paquete@dei.uc.pt

Bilel Derbel
Univ. Lille, CNRS, Centrale Lille, Inria,
UMR 9189 - CRIStAL, F-59000 Lille, France
bilel.derbel@univ-lille.fr

Arnaud Liefooghe
Univ. Lille, CNRS, Centrale Lille, Inria,
UMR 9189 - CRIStAL, F-59000 Lille, France
arnaud.liefooghe@univ-lille.fr

ABSTRACT

In this article, we propose an indicator-based branch and bound (I-BB) approach for multi-objective combinatorial optimization that uses a best-first search strategy. In particular, assuming maximizing objectives, the next node to be processed is chosen with respect to the quality of its upper bound. This quality is given by a binary quality indicator, such as the binary hypervolume or the ε-indicator, with respect to the archive of solutions maintained by the branch and bound algorithm. Although the I-BB will eventually identify the efficient set, we are particularly interested in analyzing its anytime behavior as a heuristic. Our experimental results, conducted on a multi-objective knapsack problem with 2, 3, 5, and 7 objectives, indicate that the I-BB can often outperform the naive depth-first and breadth-first search strategies, both in terms of runtime and anytime performance. The improvement is especially significant when the branching order for the decision variables is random, which suggests that the I-BB is particularly relevant when more favorable (problem-dependent) branching orders are not available.

CCS CONCEPTS

• **Computing methodologies** → **Search methodologies**; • **Theory of computation** → **Branch-and-bound**; • **Mathematics of computing** → *Combinatorial algorithms*;

KEYWORDS

Multi-objective combinatorial optimization, Search methodologies, Branch and bound, Indicator-based approaches

GECCO '21, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8350-9/21/07…$15.00
https://doi.org/10.1145/3449639.3459360

ACM Reference Format:
Alexandre D. Jesus, Luís Paquete, Bilel Derbel, and Arnaud Liefooghe. 2021. On the Design and Anytime Performance of Indicator-based Branch and Bound for Multi-objective Combinatorial Optimization. In *2021 Genetic and Evolutionary Computation Conference (GECCO '21), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3449639.3459360

1 INTRODUCTION

Multi-objective combinatorial optimization (MOCO) deals with the optimization of m objective functions over a combinatorial solution space. Given the often conflicting nature of the objectives, there may not exist a single optimal solution under the notion of *Pareto efficiency*, but rather many *efficient solutions* that describe the trade-offs between the objectives. Although MOCO problems are known to be NP-hard and the efficient set may be exponentially large [9], there are nonetheless problem instances for which the efficient set can be computed in a feasible amount of time [11]. As such, it is desirable to have approaches that can both (i) find the efficient set given enough time and (ii) return an approximate solution set when interrupted before completion.

Multi-objective branch and bound algorithms [19, 20] implicitly traverse the complete solution space of a problem while maintaining an *archive* of the best solutions found. As such, branch and bound algorithms can be seen as both exact and heuristic algorithms depending on whether or not they are executed to completion. A key aspect of a branch and bound approach is to split the original problem into several subproblems at each branching step. Each subproblem consists of assigning values to solution components, given that some other components are already fixed. Assuming maximization, an upper bound for each subproblem is used to show whether the corresponding subproblem contains potential efficient solutions for the original problem. If not, then it is not worthwhile solving that subproblem. Of particular interest to us is to understand which subproblem to choose such that it leads to better performance for MOCO problems both in terms of runtime and anytime behavior. We argue that this selection should be based on the quality of the upper bound with respect to the archive. Up to our knowledge, the idea of using the quality of the upper bound to

guide the search of the branch and bound algorithm has *not* been considered in multi-objective optimization. It has, however, been used in single-objective optimization, e.g. [5].

In this work, we propose using a binary indicator to measure the quality of the upper bound with respect to the archive maintained by the branch and bound. This leads to a new family of branch and bound algorithms for MOCO, namely, indicator-based branch and bound (I-BB). We remark that, in addition to the possible improvements discussed in the previous paragraph, the use of a quality indicator in a multi-objective search technique gives the decision maker a chance to control the search, since an indicator can, for example, be designed or parameterized to benefit regions of the objective space that are considered to be more relevant.

The use of quality indicators to guide the search strategy is commonly used by multi-objective evolutionary and heuristic approaches. Notably, IBEA [23] is a general indicator-based evolutionary algorithm that uses a quality indicator to remove the least promising solutions from a population during the selection process of the evolutionary approach. SMS-EMOA [3] is an evolutionary algorithm that combines ideas from other evolutionary algorithms, such as NSGA-II [7], and that considers the hypervolume quality indicator [24] to remove the least promising solution from the population during selection. HypE [1] is an evolutionary algorithm that similarly uses the hypervolume indicator to remove solutions from the population in the selection step, but that considers a Monte Carlo simulation to approximate the hypervolume due to its increasing computational cost with respect to the number of objectives. SPAM [25] is a heuristic resembling a hill climber that considers a heuristic mutation to the archive of solutions that makes use of a quality indicator to remove the least promising solutions. In [8] the authors consider a Pareto local search [18] variant that uses a quality indicator to select the next solution for processing. However, to our knowledge, a quality indicator has never been used to guide the search of exact algorithms such as branch and bound.

To study the performance of the proposed I-BB approach we analyze its impact on the total runtime of the algorithm, as well as on its anytime behavior in terms of the quality of the archive as a function of CPU time. To measure the quality of an approximate solution set we consider the *hypervolume quality indicator* [24]. To measure the quality of an upper bound with respect to the archive we consider two binary quality indicators: the *binary hypervolume quality indicator* [22], and the ε*-indicator* [26]. For the empirical study we consider the multi-objective binary knapsack problem with 2, 3, 5, and 7 objectives. Our results indicate that the I-BB approach can often improve, or match, the performance of more naïve branch and bound approaches, especially when considering a random branching order.

The rest of this paper is organized as follows. In Section 2 we introduce the necessary definitions related to multi-objective optimization, quality indicators, and anytime performance. In Section 3 we introduce the proposed I-BB approach under a more general branch and bound framework for MOCO. In Section 4 we present the multi-objective binary knapsack problem and discuss the implementation details of the I-BB algorithm for this problem. In Section 5 we present the results of our empirical study. To conclude, in Section 6 we give a summary of the work and discuss possible directions for the future.

2 BACKGROUND

In the following we introduce the necessary definitions related to multi-objective optimization and anytime performance.

2.1 Concepts and Notation for MOCO

Assuming, without loss of generality, that all objective functions are to be maximized, a MOCO problem can be defined by:

$$\underset{x \in X}{\operatorname{argmax}} f(x) = (f_1(x), f_2(x), \ldots, f_m(x)) \tag{1}$$

where x denotes a solution, X denotes the set of all feasible solutions, and $f_i(x) \to \mathbb{R}$ denotes the ith objective function to be maximized.

Given two objective vectors $y, y' \in \mathbb{R}^m$ we say that y *weakly dominates* y' iff $y_i \geq y'_i$ for every $i \in \{1, 2, \ldots, m\}$. This relation is denoted by $y \geq y'$. Moreover, we say that y *dominates* y' iff $y \geq y'$ and $y' \ngeq y$. This relation is denoted by $y > y'$. Then two objective vectors $y, y' \in \mathbb{R}^m$ are said to be *incomparable* or *mutually non-dominated* iff $y \ngeq y'$ and $y' \ngeq y$, i.e. iff neither weakly dominates the other. We describe two solutions $x, x' \in X$ with respect to the relations that hold for their objective vectors, e.g. we say that x *dominates* x' iff $f(x) > f(x')$. The efficient set to a MOCO problem is given by the set of all solutions that are not dominated by any other feasible solution. A set of mutually non-dominated feasible solutions is called an *approximate solution set*.

We extend the relations above to sets of vectors in the objective space [26]. Given two sets $A, B \subset \mathbb{R}^m$ we say that A *weakly dominates* B, denoted by $A \geq B$, iff every $b \in B$ is weakly dominated by an objective vector $a \in A$, formally:

$$A \geq B \iff \forall b \in B \; \exists a \in A \text{ s.t. } a \geq b$$

We say that A *dominates* B, denoted by $A > B$ iff every $b \in B$ is dominated by an objective vector $a \in A$, formally:

$$A > B \iff \forall b \in B \; \exists a \in A \text{ s.t. } a > b$$

And we say that A and B are *incomparable* iff $A \ngeq B$ and $B \ngeq A$.

Several *quality indicators* have been proposed to quantify the quality of approximate solution sets as a scalar value. These can be useful, for example, to distinguish between approximate solution sets that are incomparable in terms of dominance. One such indicator is the *hypervolume quality indicator* [24], which corresponds to the multi-dimensional area dominated by the objective vectors of a solution set with respect to a given reference point in the objective space. Formally, for a set $A \subset \mathbb{R}^m$ and a reference point $r \in \mathbb{R}^m$, the hypervolume is defined by:

$$H(A, r) = \lambda\left(\left\{q \in \mathbb{R}^m \mid \exists\, a \in A \colon r \geq q \geq a\right\}\right) \tag{2}$$

where λ is the Lebesgue measure. We also consider the *binary hypervolume quality indicator* [22], which corresponds to the multi-dimensional area dominated by a set but not by another, that is:

$$H(A, B, r) = H(A \cup B, r) - H(B, r) \tag{3}$$

We also consider the multiplicative ε-indicator [26], which, given two solution sets, corresponds to the minimum factor by which every objective vector of the former has to be multiplied such that it weakly dominates the latter. Formally, given two sets $A, B \subset \mathbb{R}^m$, the multiplicative ε-indicator is given by:

$$E(A, B) = \max_{a \in A} \min_{b \in B} \max_{i \in \{1, \ldots, m\}} \frac{b_i}{a_i} \tag{4}$$

An important characteristic of these quality indicators is that they are compatible with the weak dominance relation between solution sets. In particular, if for two sets $A, B \subset \mathbb{R}^m$ it holds that $A \geq B$, then it also holds that $H(A, r) \geq H(B, r)$ for a given reference point r, and that $H(A, R, r) \geq H(B, R, r)$ and $E(A, R) \leq E(B, R)$ for a given reference set $R \subset \mathbb{R}^m$. A relevant implication is that when adding solutions to a set the quality returned by these indicators improves monotonically, under the assumption that the reference point and/or reference set for the binary indicators are unchanged.

2.2 Anytime Behavior

To analyze the anytime performance of a run ω of an algorithm that keeps a solution archive we consider the *performance trace* $Q_\omega(t) \rightarrow \mathbb{R}^+$ that describes the trade-off between the quality of the archive and the execution time $t \in \mathbb{R}^+$. For this work, we consider the archive quality in terms of the unary hypervolume indicator defined in Equation 2, and execution time in terms of CPU time.

For analyzing the performance over multiple runs Ω we consider the *performance profile* [12, 14, 15] given by:

$$P_\Omega(t, q) = \frac{1}{|\Omega|} \sum_{\omega \in \Omega} I\{Q_\omega(t) \geq q\} \tag{5}$$

that denotes the probability of a run of the algorithm having an archive of quality greater or equal to $q \in \mathbb{R}$ at time $t \in \mathbb{R}^+$.

To measure the anytime quality of multiple independents runs Ω of an algorithm we aggregate those runs into a performance profile and compute the quality measure given by its bi-dimensional integral [15, 16], that is:

$$\int_0^\infty \int_0^{\bar{t}} P_\Omega(t, q)\, dt\, dq \tag{6}$$

where $\bar{t}$ denotes the timeout for the runs.

3 INDICATOR-BASED BRANCH AND BOUND

Branch and bound algorithms implicitly visit all feasible solutions. They recursively divide the solution space into subspaces, each of which encapsulates a subproblem of the original problem. For the purpose of this work we consider that each subproblem gives rise to a lower and an upper bound [10, 19]. The lower bound corresponds to a set of feasible and mutually non-dominated solutions that are weakly dominated by the efficient solutions to the subproblem. The upper bound corresponds to a set of mutually non-dominated objective vectors that weakly dominates the efficient set to the subproblem. Note that the branch and bound algorithm does not require the use of a lower bound since the solutions of a lower bound are eventually visited by the algorithm. However, finding the lower bound solutions earlier and adding them to the archive can improve the anytime performance of the branch and bound and lead to more pruning, which may impact the runtime of the algorithm. This can often outweigh the computational cost of calculating the lower bound and lead to a better runtime.

The subspace division in a branch and bound algorithm is commonly represented as a dynamically generated search tree where each *node* corresponds to a partial solution, e.g. a solution where some decision variables have been set but not others, and a *leaf* corresponds to a complete solution. Initially this tree contains only the *root* node, i.e. the original solution space. A node of the search tree is then processed at each iteration of the branch and bound algorithm. This involves three main tasks: selecting the node to process, computing the bounds, and branching from that node, i.e. further subdividing the solution space if possible.

Algorithm 1: Eager branch-and-bound

Input : Root node r
Output: Solution set S

```
1  S ← getLowerBound(r)
2  Q ← {r}
3  while Q ≠ ∅ do
4      node ← selectNode(Q)
5      for branch ∈ getBranches(node) do
6          if S ⊁ getUpperBound(branch) then
7              S ← S ∪ getLowerBound(branch)
8              Q ← Q ∪ {branch}
9          end
10     end
11 end
12 return S
```

3.1 MOCO Branch and Bound Framework

Algorithm 1 defines the general framework for branch and bound algorithms for MOCO that we will consider in this work. It describes an *eager* branch and bound design, meaning that the computation of lower and upper bounds of a node occurs as soon as that node is identified. By contrast, a *lazy* design only computes the bounds of a node once that node is needed, i.e. when it is selected for processing or when its upper bound is needed for node selection. In our preliminary experiments, the eager design had generally better runtime and anytime performance, and it also demanded less memory. As such, we choose to discard the lazy design for this work, and note that other approaches in the literature often use an eager design as well [17, 19, 20]. We remark that S in Algorithm 1 corresponds to the archive maintained by the branch and bound algorithm.

3.2 Node Selection

Two common strategies for selecting the next node to be explored are the depth-first selection (DFS) [17, 20] and the breadth-first selection (BFS) [20] strategies. Both can be trivially implemented within the framework of Algorithm 1 with a Last In First Out (LIFO) queue and a First In First Out (FIFO) queue respectively. We remark, that the BFS approach often suffers from memory issues since it needs to keep a large amount of nodes in memory. On the other hand, the DFS can easily get stuck deep in the tree, exploring solutions that do not improve the quality of the archive significantly. Nonetheless, both approaches can have good performance if the subspace division order leads to optimal, or close to optimal, solutions early on. Unfortunately, such orderings may not be available for every problem or it may not always be clear which order is the best since it can depend on the problem instance being solved [4]. A

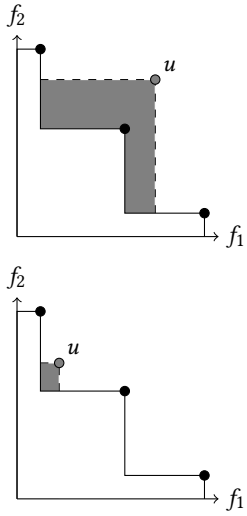

Figure 1: Illustration of the binary hypervolume indicator (in gray) for the upper bound $\{u\}$ of two distinct nodes.

third strategy is the best-first selection (BeFS), which corresponds to finding the most promising node in the queue. In single-objective optimization this could, for example, correspond to finding the node with the highest upper bound, since it suggests that searching in that direction will lead to a solution with a better objective value. A similar idea would be to select the node that may lead to a greater improvement in the quality of the archive. Note that, finding better solutions early, may not only lead to a better anytime performance, but also to an improvement in the overall runtime of the algorithm since other nodes can be pruned sooner.

In this work, we propose an indicator-based branch and bound (I-BB) approach, which considers the use of quality indicators for the BeFS strategy. In our approach, we consider that the "quality" of a node is given by the quality of its upper bound with respect to the archive. To measure this quality we can consider any binary quality indicator that measures the quality of a solution set with respect to another, such as the binary hypervolume indicator and the ε-indicator discussed in Section 2. Formally, given a queue Q of branch and bound nodes, the archive S, and assuming a maximizing binary quality indicator $I(A, B) \rightarrow \mathbb{R}$ with $A, B \subset \mathbb{R}^m$ that denotes the quality of a set of objective vectors A with respect to a solution set B, we define the `selectNode(Q)` function for Algorithm 1 by:

$$\texttt{selectNode}(Q) = \underset{node \in Q}{\text{argmax}}\ I\left(U(node), \{f(x) : x \in S\}\right) \quad (7)$$

where $U(node)$ denotes the upper bound of a node. As an example, consider Figure 1 that illustrates the upper bound of two distinct nodes and the calculation of the binary hypervolume indicator for each. We recall that all efficient solutions to the subproblem are weakly dominated by the upper bound. By selecting the node whose upper bound gives the best value of binary hypervolume indicator, in this case the topmost figure, we hope to find a solution that provides a more substantial contribution to the archive.

Unfortunately, the BeFS strategy may also suffer from memory issues like the BFS strategy. Another issue is that if the computation of the binary quality indicator is costly, as is the case for the binary hypervolume indicator with an increasing number of objectives, having a large amount of nodes in the queue can lead to a long selection time. Some implementation details, like caching previously computed values, can help mitigate such issues, but they may not be sufficient. As such, we also consider a *Best-Depth-First Selection (BeDFS)* strategy, which only evaluates the nodes that are in the deepest level of the search tree.

3.3 Other Aspects

There are two other key aspects for the implementation of a branch and bound approach not discussed here: how the solution space is divided, i.e. how branching occurs; and the computation of the bounds. Since defining these aspects heavily depends on the problem being solved, they are discussed in the following section for the considered multi-objective binary knapsack problem.

4 I-BB FOR MULTI-OBJECTIVE KNAPSACK

To validate our approach we conduct an empirical study on the multi-objective binary knapsack problem. In this section, we describe the problem and instances considered for the study, as well as the branch and bound implementation details for this problem.

4.1 Problem

For this study we consider the multi-objective binary knapsack problem, which can be defined as follows:

$$\underset{x \in \{0,1\}^n}{\text{argmax}} \left(\sum_{i=1}^{n} x_i v_i^1, \ldots, \sum_{i=1}^{n} x_i v_i^m \right) \quad (8)$$

$$\text{s.t.} \sum_{i=1}^{n} x_i c_i \leq C \quad (9)$$

where n is the number of items for the knapsack problem, $v_i^j \in \mathbb{Z}^+$ denotes the value associated with item $i \in \{1, \ldots, n\}$ for objective function $j \in \{1, \ldots, m\}$, $c_i \in \mathbb{Z}^+$ denotes the cost of item i, and $C \in \mathbb{Z}^+$ denotes the overall cost constraint.

We consider benchmark instances [2] with item values for the objective functions v_i^j and item cost values c_i randomly generated following a uniform distribution in the range $[1, 1000]$. Moreover, the cost constraint is set to half of the sum of all the costs, that is:

$$C = \frac{1}{2} \sum_{i=1}^{n} c_i \quad (10)$$

4.2 I-BB Implementation Details

In the following sections we describe the implementation details of the branch and bound algorithm with respect to branching, branching order, bounds computation and selection strategy.

4.2.1 Branching. We consider a *dichotomic* branching at each node, where the root node corresponds to the empty knapsack, i.e. no decision variable is set. Then, a branching operation corresponds to setting the next unset decision variable to either 0 or 1. Note that there is an exponential number of nodes with respect to the problem size. We also considered a *politomic* branching that sets the k next unset decision variables to 0, and the $k + 1$ unset decision variable to 1. However, our preliminary results showed

that it rarely achieved better runtime or anytime performance. As such, we choose to focus on the dichotomic branching.

4.2.2 Branching order. For the considered branching variants, we need to define the order in which the decision variables are set. One possibility is to consider a random order, which in our empirical analysis is given by the default (random) ordering of the items. Another possibility is to sort the decision variables beforehand according to a given ordering criteria. Importantly, our implementation of the DFS and BFS selection strategies processes first the children whose earlier variables are set to 1. As such, this ordering criterion should order the decision variables such that the first variables are more likely to be set to 1 in efficient solutions. In this work, we consider the *rank sum order* O^{sum} order [2], which is denoted by the increasing order of the sums of the ranks for items $i = \{1, \ldots, n\}$ given by the orders $O^j = v_i^j/c_i$ for each objective $j = \{1, \ldots, m\}$. Preliminary results comparing the O^{sum}, $O^{\max}$ [2], and $O^{\min}$ [2] orders, showed better anytime behavior for the former. For other possible orders, we refer to [4] where various branching orders and heuristics are analyzed.

4.2.3 Lower and upper bound. In the following we describe the procedure used to compute the lower and upper bounds of a node. Briefly, for a given partial solution we compute a number of feasible extensions that correspond to feasible solutions to the knapsack problem. These feasible extensions are first used to populate the lower bound of a node. Although the lower bound is not required for the branch and bound algorithm, having a diverse lower bound, whose solutions are then added to the archive, can allow the algorithm to prune nodes earlier, which often results in an improved anytime behavior and runtime. Moreover, these feasible extensions are then used to compute the upper bound of a node which is fundamental for branch and bound approaches.

Consider a node whose partial solution is given by x such that the first $k \leq n$ decision variables are set to either 0 or 1, and the remaining variables have not been set yet. Assume that this partial solution is feasible with respect to the cost constraint, i.e. $\sum_{i=1}^{k} x_i c_i \leq C$. Then, an extension for each objective j can be obtained by Dantzig's greedy algorithm [6]. In particular, assume that the items from $k+1$ to n are sorted in increasing order of the ratio v_i^j/c_i for an objective j. Then an extension for that objective function is given by adding all items from $k+1$ up to, but not including, b such that adding item b would break the cost constraint. The corresponding objective vector is given by

$$\ell^j = \left(\sum_{i=1}^{k} x_i v_i^1 + \sum_{i=k+1}^{b} v_i^1, \ldots, \sum_{i=1}^{k} x_i v_i^m + \sum_{i=k+1}^{b} v_i^m \right) \tag{11}$$

In addition, to refine the lower bound with a more balanced trade-off among the objectives, we consider the extension that is found with Dantzig's greedy algorithm [6] for the items from $k+1$ to n sorted by the ratio $(v_i^1 + \ldots + v_i^m)/c_i$. The objective vector of this extension is denoted by ℓ^{sum}. The topmost illustration in Figure 2 shows the objective vectors ℓ^1, ℓ^2, and ℓ^{sum} for two objectives. Since these extensions correspond to feasible solutions to the knapsack problem they are included in the lower bound of the node. Also, since the branching orders considered are fixed, we can precompute all the necessary orders for Dantzig's greedy algorithm.

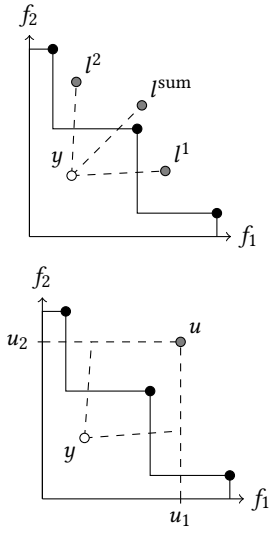

Figure 2: Illustration of the image of the lower bound (top) in the objective space $\{\ell^1, \ell^2, \ell^w\}$ for a node with partial objective vector y, and the corresponding upper bound (bottom) given by $\{u\}$.

For the upper bound of a node, we consider non-feasible extensions to the ℓ^j objective vectors. In particular, for each vector ℓ^j we add the fractional part of the item that violates the constraint, to the j-th objective value, formally:

$$u_j = \ell_j^j + C^j \frac{v_b^j}{c_b} \tag{12}$$

where C^j is the residual cost of ℓ^j. An upper bound to the node is then given by the objective vector:

$$u = (u_1, \ldots, u_m) \tag{13}$$

This upper bound is illustrated in Figure 2. A similar upper bound was considered in [17].

4.2.4 Selection strategies. For this study we consider the DFS, BFS, BeFS and BeDFS selection strategies described in Section 3. Moreover, for the latter two we consider two different binary quality indicators: the binary hypervolume indicator, and the ε-indicator described in Section 2.

Since our upper bound is given by a single objective vector u, the binary hypervolume computation can be formulated as a hypervolume contribution calculation:

$$H(A, u, r) = H(A \cup \{u\}, r) - H(A, r) \tag{14}$$

For two objectives the hypervolume contribution can be calculated in $O(\log \mu)$, where μ is the number of solutions in the archive, if the objective vectors of the archive solutions are kept sorted by one of the objective values and data pertaining to the hypervolume calculation with a dimension sweep is kept for each node. For three objectives, it can be calculated in $O(\mu)$ with the HV3D$^+$-U algorithm [13] if the objective vectors of the archive solutions are stored in the data structure of the HV3D$^+$ algorithm. For 4 or more objectives, we use the WFG algorithm [21], with the sorting and

slicing improvements discussed in that work, which computes the hypervolume contribution in $O(\mu^{m/2} \log \mu)$ for m objectives.

To compute the ε-indicator we maintain the maximum values for each objective among all solutions in the archive. Then, the ε-indicator for a set with a single objective vector can be found in $O(m)$ by dividing the maximum value for each objective by the corresponding value of the objective vector whose quality we are measuring. As such, calculating the ε-indicator is expected to be much faster than calculating the hypervolume contribution, which should be particularly noticeable for a high number of objectives.

Note that the value of the quality indicator for the nodes in the queue can be cached and kept in a priority queue. For a BeFS approach we need to keep a priority queue with all the nodes, whereas for BeDFS we would need a priority queue for each level of the search tree. However, for a dichotomic branching, the BeDFS strategy does not need to keep the values cached since after selecting the best node for a level there is only one node left.

Unfortunately, if the archive maintained by the branch and bound changes, then the cached values may need to be updated, and the priority queue recomputed, which can be detrimental to the performance of the algorithm if it happens frequently. For the BeFS strategy, our preliminary experiments suggest that changes to the maximum objective values were relatively rare. As such, for the ε-indicator BeFS we find it is best to keep the nodes in the priority queue at all times. For the binary hypervolume indicator we see that once the branch and bound reaches a relatively stable state where updates to the archive are rare, then it is worth keeping the items in a priority queue. However, when the archive is being regularly updated, recomputing the priority queue is detrimental to the performance of the algorithm. To balance both scenarios, we build and maintain a priority queue after the algorithm makes 10 consecutive calls to the selection methodology without updating the priority queue. However, if a solution is added to the archive, we discard the priority queue and rely on looping over all values of a vector to find the most promising node. The value 10 was chosen empirically during preliminary testing. Finally, to avoid unnecessary quality indicator computations when a priority queue is not used, we only recompute the quality indicator of the nodes whose previously cached value was better than the current best.

5 EMPIRICAL STUDY

Our empirical study focuses on comparing the performance of the various selection strategies described in the previous section with regards to two aspects of performance: the runtime of the approach, i.e. the time it takes to identify the efficient set, and its anytime quality, i.e. the quality of the archive at any given time.

To summarize, we consider the following alternatives for the implementation of the branch and bound approaches:

- **Branching**: *Dichotomic*;
- **Branching order**: *Random (Default)* or *Rank Sum*;
- **Selection strategy**: *DFS*, *BFS*, *ε-indicator BeFS*, *ε-indicator BeDFS*, *Hypervolume BeFS*, or *Hypervolume BeDFS*.

The algorithms were implemented in `C++20`, compiled with `GCC 10.2.0`, and the experiments were carried out in parallel, one per thread, on a machine with two `Intel(R) Xeon(R) Silver 4210R` processors. The algorithms had a timeout of 300 seconds and an 8Gb memory limit. The results shown below use instances of sizes $n \in \{10, 15, 20, 25, 30, 35, 40, 45, 50, 100, 200\}$, number of objectives $m \in \{2, 3, 5, 7\}$. For each combination of these parameters 10 instances were independently generated.

5.1 Runtime Analysis

Figure 3 shows the results of the average runtime for instances with a varying number of objectives, variables, and branching order. The runtime values are only shown for cases where the algorithms found the efficient set in all 10 generated instances for that particular combination of parameters. The results for two objectives (far left) suggest that the DFS strategy has the worst runtime performance for the default order, and that the BFS and the hypervolume BeFS strategies have the worst runtime performance for the rank sum order. As the number of objectives increases we can see that the runtime performance of the hypervolume BeFS and BeDFS strategies worsens substantially. This is to be expected due to the increasing complexity of the binary hypervolume calculation. We see that the ε-indicator BeFS and BeDFS approaches do not have this issue and that they often show the best, or second best, runtime. It is also worth noting that the runtime for the random order is generally similar to that of the rank sum order, and in fact a better runtime can be observed for some instances with 5 and 7 objectives.

5.2 Anytime Analysis

Figure 4 shows the anytime performance traces, with respect to the normalized hypervolume and CPU time, for instances of size $n = 100$ and 2, 3, 5, and 7 objectives. Note that no approach obtained the efficient set for the given time and memory limits. The hypervolume is normalized with respect to the minimum and maximum values computed for each instance during the execution of the branch and bound approaches, such that the minimum hypervolume considered for an instance is given by the hypervolume of the lower bound of the empty knapsack. For 2 objectives, we can see that the I-BB strategies, in particular the hypervolume BeFS strategy, show a very good anytime behavior. Moreover, we can see that for the default (random) order the I-BB strategies generally obtain the best anytime performance. By contrast, for the rank sum order, we see that the DFS strategy shows the overall best anytime performance. However, the results suggest that the I-BB BeFS strategies can surpass the DFS strategy when given enough time, as is the case for 2 and 3 objectives. The performance of the hypervolume based I-BB strategies shows a better anytime performance than the ε-indicator based I-BB strategies for 2 objectives, similar anytime performance for 3 objectives, and worse performance for 5 and 7 objectives. This is consistent with the increased complexity of the binary hypervolume indicator. Note that for 2 and 3 objectives the hypervolume based I-BB approaches for a default order have an anytime performance that is similar to the best anytime performance for a good branching order. For 5 and 7 objectives the results are not as good but still promising. The results were consistent for all tested instances with problem size $n = 100$.

Table 1 shows the ratio of instances for which a given selection strategy had the best anytime quality as given by Equation 6, e.g. a ratio of 0.6 means that for 60% of the tested instances the given

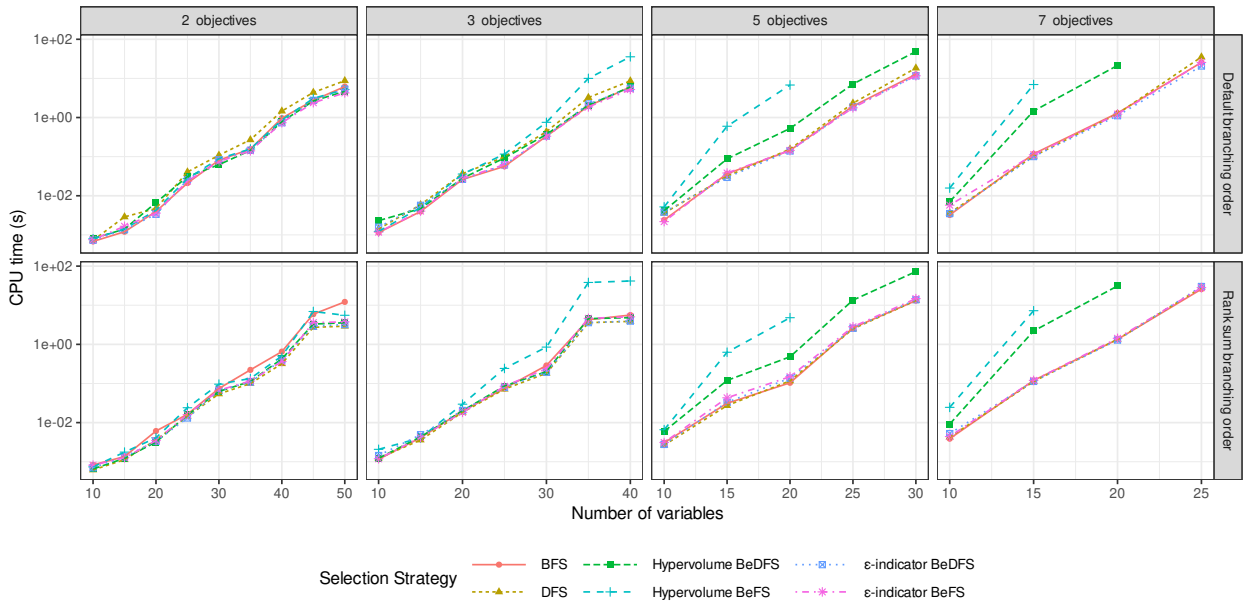

Figure 3: Runtime averages for varying number of objectives, variables, branching orders, and selection strategies. For each combination, the results are not shown if at least one instance did not finish within the 300 seconds timeout.

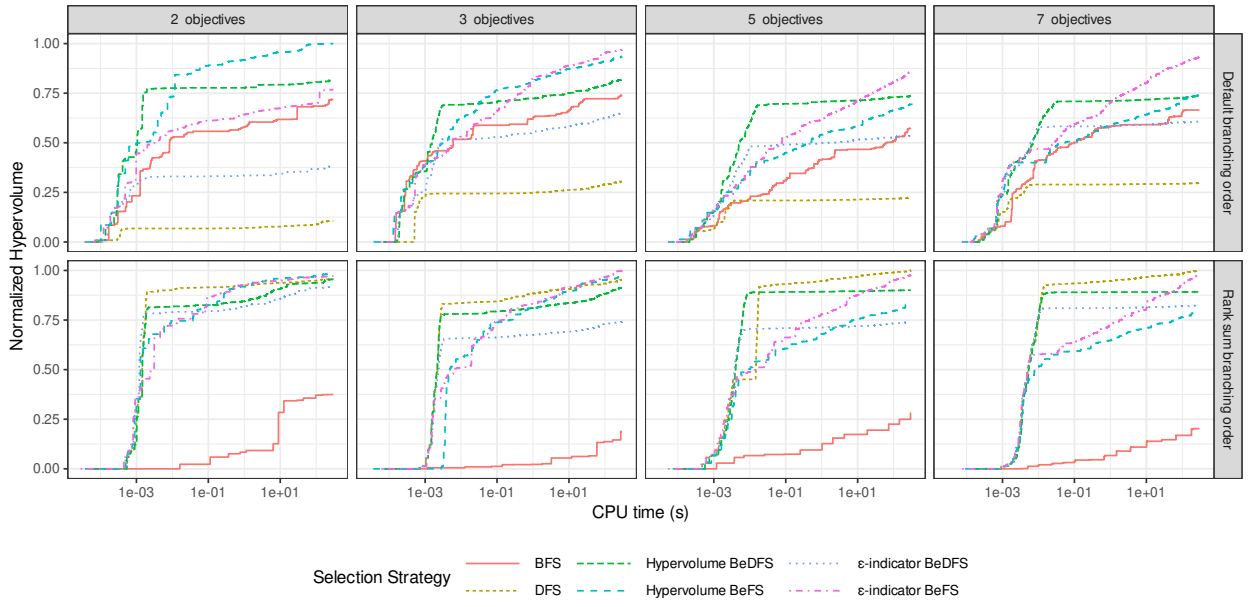

Figure 4: Anytime performance trace of the branch and bound approaches for instances with varying number of objectives, and problem size $n = 100$.

approach had the best anytime quality. Note that the I-BB strategies are always the best for the default (random) branching order. For 2 and 3 objectives we see that the hypervolume based I-BB approaches more often have the best anytime quality, whereas for 5 and 7 objectives the ε-indicator based I-BB approaches are better. Interestingly, the hypervolume based I-BB is capable of achieving a better anytime quality than its ε-indicator counterpart for 5 and 7 objectives when $n = 200$. This suggests that the ε-indicator based I-BB is not suited for increasing the quality of the archive in terms of hypervolume for large problem sizes.

Table 1: Ratio for the number of times a given selection strategy obtains the best anytime quality. Best values are in bold.

Branching Order	m	n	BFS	DFS	HV BeFS	HV BeDFS	ε-indicator BeFS	ε-indicator BeDFS
Default	2	50	0.0	0.0	**1.0**	0.0	0.0	0.0
		100	0.0	0.0	**1.0**	0.0	0.0	0.0
		200	0.0	0.0	**0.8**	0.2	0.0	0.0
	3	50	0.0	0.0	**0.9**	0.1	0.0	0.0
		100	0.0	0.0	0.4	0.0	**0.6**	0.0
		200	0.0	0.0	0.0	**1.0**	0.0	0.0
	5	50	0.0	0.0	0.0	0.0	**1.0**	0.0
		100	0.0	0.0	0.0	0.0	**1.0**	0.0
		200	0.0	0.0	0.0	**1.0**	0.0	0.0
	7	50	0.0	0.0	0.0	0.0	**1.0**	0.0
		100	0.0	0.0	0.0	0.0	**1.0**	0.0
		200	0.0	0.0	0.0	**1.0**	0.0	0.0
Rank Sum	2	50	0.0	0.2	**0.8**	0.0	0.0	0.0
		100	0.0	0.2	**0.8**	0.0	0.0	0.0
		200	0.0	**1.0**	0.0	0.0	0.0	0.0
	3	50	0.0	**0.8**	0.1	0.0	0.0	0.1
		100	0.0	**0.8**	0.0	0.0	0.2	0.0
		200	0.0	**1.0**	0.0	0.0	0.0	0.0
	5	50	0.0	0.1	0.0	0.0	**0.9**	0.0
		100	0.0	**0.9**	0.0	0.0	0.1	0.0
		200	0.0	**1.0**	0.0	0.0	0.0	0.0
	7	50	0.0	0.0	0.0	0.0	**1.0**	0.0
		100	0.0	**1.0**	0.0	0.0	0.0	0.0
		200	0.0	**1.0**	0.0	0.0	0.0	0.0

For the rank sum branching order, we see that the DFS strategy often shows the best anytime quality, as expected from the previous results. Still, we see that the I-BB approaches can sometimes have better anytime performance for smaller problem sizes. Further analysis of the instances with problem size $n = 200$, where our I-BB approach was never able to get the best anytime performance, suggests that if given enough time (and memory) the I-BB BeFS approaches would surpass the quality obtained by the DFS strategy.

6 CONCLUSIONS

In this article, we proposed an indicator-based branch and bound (I-BB) approach for multi-objective combinatorial optimization and analyzed it both in terms of runtime and anytime, i.e. heuristic, behavior. Our empirical results showed very promising results in terms of anytime performance, in particular when considering a random ordering of the decision variables. Moreover, the results indicated that the binary hypervolume I-BB had a good anytime performance for a small number of objectives, but that the ε-indicator I-BB showed better performance for a larger number of objectives. The results also highlighted some weaknesses of the I-BB that are worth investigating in the future. In particular, we note that the I-BB approach cannot always match the anytime behavior of a simple DFS strategy when considering a good branching order, particularly for larger problem sizes. However, a good branching order might not always be available, or may be difficult to design for new problem classes. Nonetheless, we see that the quality of the archive for the I-BB BeDFS strategies tends to quickly improve in the early stages of the search process but then stagnates, whereas for the BeFS strategies the quality is not as good initially but keeps improving throughout the search. As such, one possibility to improve the performance of our I-BB approach could be to consider a hybrid selection strategy that combines both the BeFS and BeDFS strategies. Another possibility would be to consider the use of heuristic approaches to approximately solve some of the subproblems, in particular those where the solutions found are only expected to marginally improve the quality of the solution archive. Moreover, it would be interesting to study different settings in terms of quality indicators and bound definitions. Finally, it would be relevant to study the performance of I-BB for other problem classes, particularly real-world problems, and to compare it against state-of-the-art heuristics, including multi-objective local search and evolutionary algorithms.

ACKNOWLEDGMENTS

This work was partially supported by the PICS project MOCO-SEARCH co-funded by the French National Center for Scientific Research (CNRS) and the Portuguese Foundation for Science and Technology (FCT). The first author further acknowledges the FCT for Ph.D. studentship SFRH/BD/132275/2017 co-funded by the European Social Fund and by the State Budget of the Portuguese Ministry of Education and Science. This work was partially funded by national funds through the FCT within the scope of the project CISUC - UID/CEC/00326/2020 and by the European Social Fund, through the Regional Operational Program Centro 2020.

REFERENCES

[1] Johannes Bader and Eckart Zitzler. 2010. HypE: An Algorithm for Fast Hypervolume-Based Many-Objective Optimization. *Evolutionary Computation* 19, 1 (July 2010), 45–76. https://doi.org/10.1162/EVCO_a_00009

[2] Cristina Bazgan, Hadrien Hugot, and Daniel Vanderpooten. 2009. Solving Efficiently the 0-1 Multi-Objective Knapsack Problem. *Computers & Operations Research* 36, 1 (Jan. 2009), 260–279. https://doi.org/10.1016/j.cor.2007.09.009

[3] Nicola Beume, Boris Naujoks, and Michael Emmerich. 2007. SMS-EMOA: Multiobjective Selection Based on Dominated Hypervolume. *European Journal of Operational Research* 181, 3 (Sept. 2007), 1653–1669. https://doi.org/10.1016/j.ejor.2006.08.008

[4] Audrey Cerqueus, Xavier Gandibleux, Anthony Przybylski, and Frédéric Saubion. 2017. On Branching Heuristics for the Bi-Objective 0/1 Unidimensional Knapsack Problem. *Journal of Heuristics* 23, 5 (Oct. 2017), 285–319. https://doi.org/10.1007/s10732-017-9346-9

[5] Van-Dat Cung, Mhand Hifi, and Bertrand Le Cun. 2000. Constrained Two-Dimensional Cutting Stock Problems a Best-First Branch-and-Bound Algorithm. *International Transactions in Operational Research* 7, 3 (Aug. 2000), 185–210. https://doi.org/10.1111/j.1475-3995.2000.tb00194.x

[6] George B. Dantzig. 1957. Discrete-Variable Extremum Problems. *Operations Research* 5, 2 (April 1957), 266–288. https://doi.org/10.1287/opre.5.2.266

[7] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. 2002. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (April 2002), 182–197. https://doi.org/10.1109/4235.996017

[8] Jérémie Dubois-Lacoste, Manuel López-Ibáñez, and Thomas Stützle. 2015. Anytime Pareto Local Search. *European Journal of Operational Research* 243, 2 (June 2015), 369–385. https://doi.org/10.1016/j.ejor.2014.10.062

[9] Matthias Ehrgott. 2005. *Multicriteria Optimization* (second ed.). Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-27659-9

[10] Matthias Ehrgott and Xavier Gandibleux. 2001. Bounds and Bound Sets for Biobjective Combinatorial Optimization Problems. In *Multiple Criteria Decision Making in the New Millennium*. Springer, Berlin, Heidelberg, 241–253. https://doi.org/10.1007/978-3-642-56680-6_22

[11] José Rui Figueira, Carlos M. Fonseca, Pascal Halffmann, Kathrin Klamroth, Luís Paquete, Stefan Ruzika, Britta Schulze, Michael Stiglmayr, and David Willems. 2017. Easy to Say They Are Hard, but Hard to See They Are Easy- Towards a Categorization of Tractable Multiobjective Combinatorial Optimization Problems. *Journal of Multi-Criteria Decision Analysis* 24, 1-2 (Jan. 2017), 82–98. https://doi.org/10.1002/mcda.1574

[12] Viviane Grunert da Fonseca, Carlos M. Fonseca, and Andreia O. Hall. 2001. Inferential Performance Assessment of Stochastic Optimisers and the Attainment Function. In *Evolutionary Multi-Criterion Optimization (EMO 2001)*. Springer, Berlin, Heidelberg, 213–225. https://doi.org/10.1007/3-540-44719-9_15

[13] Andreia P. Guerreiro and Carlos M. Fonseca. 2018. Computing and Updating Hypervolume Contributions in Up to Four Dimensions. *IEEE Transactions on Evolutionary Computation* 22, 3 (June 2018), 449–463. https://doi.org/10.1109/TEVC.2017.2729550

[14] Holger H. Hoos and Thomas Stützle. 2005. *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers, San Francisco, CA, USA.

[15] Alexandre D. Jesus, Arnaud Liefooghe, Bilel Derbel, and Luís Paquete. 2020. Algorithm Selection of Anytime Algorithms. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference (GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 850–858. https://doi.org/10.1145/3377930.3390185

[16] Manuel López-Ibáñez and Thomas Stützle. 2014. Automatically Improving the Anytime Behaviour of Optimisation Algorithms. *European Journal of Operational Research* 235, 3 (June 2014), 569–582. https://doi.org/10.1016/j.ejor.2013.10.043

[17] Ulungu-Ekunda Lukata and Jacques Teghem. 1997. Solving Multi-Objective Knapsack Problem by a Branch-and-Bound Procedure. In *Multicriteria Analysis*, João Clímaco (Ed.). Springer, Berlin, Heidelberg, 269–278. https://doi.org/10.1007/978-3-642-60667-0_26

[18] Luís Paquete, Tommaso Schiavinotto, and Thomas Stützle. 2007. On Local Optima in Multiobjective Combinatorial Optimization Problems. *Annals of Operations Research* 156, 1 (Aug. 2007), 83. https://doi.org/10.1007/s10479-007-0230-0

[19] Anthony Przybylski and Xavier Gandibleux. 2017. Multi-Objective Branch and Bound. *European Journal of Operational Research* 260, 3 (Aug. 2017), 856–872. https://doi.org/10.1016/j.ejor.2017.01.032

[20] Michaël Visée, Jacques Teghem, Marc Pirlot, and E.L. Ulungu. 1998. Two-Phases Method and Branch and Bound Procedures to Solve the Bi-Objective Knapsack Problem. *Journal of Global Optimization* 12, 2 (March 1998), 139–155. https://doi.org/10.1023/A:1008258310679

[21] Lyndon While, Lucas Bradstreet, and Luigi Barone. 2012. A Fast Way of Calculating Exact Hypervolumes. *IEEE Transactions on Evolutionary Computation* 16, 1 (Feb. 2012), 86–95. https://doi.org/10.1109/TEVC.2010.2077298

[22] Eckart Zitzler. 1999. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD Thesis. Swiss Federal Institute of Technology Zurich.

[23] Eckart Zitzler and Simon Künzli. 2004. Indicator-Based Selection in Multiobjective Search. In *Parallel Problem Solving from Nature — PPSN VIII*. Springer, Berlin, Heidelberg, 832–842.

[24] Eckart Zitzler and Lothar Thiele. 1998. Multiobjective Optimization Using Evolutionary Algorithms — A Comparative Case Study. In *Parallel Problem Solving from Nature — PPSN V*. Springer, Berlin, Heidelberg, 292–301. https://doi.org/10.1007/BFb0056872

[25] Eckart Zitzler, Lothar Thiele, and Johannes Bader. 2010. On Set-Based Multiobjective Optimization. *IEEE Transactions on Evolutionary Computation* 14, 1 (Feb. 2010), 58–79. https://doi.org/10.1109/TEVC.2009.2016569

[26] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. 2003. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation* 7, 2 (April 2003), 117–132. https://doi.org/10.1109/TEVC.2003.810758