Beam Search for integer multi-objective optimization

Thibaut Barthelemy Sophie N. Parragh Fabien Tricoire Richard F. Hartl

University of Vienna, Department of Business Administration, Austria
{thibaut.barthelemy,sophie.parragh,fabien.tricoire,richard.hartl}@univie.ac.at

Abstract

Beam search is a tree search procedure where, at each level of the tree, at most w nodes are kept. This results in a metaheuristic whose solving time is polynomial in w. Popular for single-objective problems, beam search has only received little attention in the context of multi-objective optimization. By introducing the concepts of oracle and filter, we define a paradigm to understand multi-objective beam search algorithms. Its theoretical analysis engenders practical guidelines for the design of these algorithms. The guidelines, suitable for any problem whose variables are integers, are applied to address a bi-objective 0-1 knapsack problem. The solver obtained outperforms the existing non-exact methods from the literature.

1 Introduction

Everyday life decisions are often a matter of multi-objective optimization. Indeed, life is full of tradeoffs, as reflected for instance by discussions in parliaments or boards of directors. As a result, almost all cost-oriented problems of the classical literature give also rise to quality, durability, social or ecological concerns. Such goals are often conflicting. For instance, lower costs may lead to lower quality and vice versa. A generic approach to multi-objective optimization consists in computing a set of several good compromise solutions that decision makers discuss in order to choose one.

Numerous methods have been developed to find the set of compromise solutions to multi-objective combinatorial problems. While exact algorithms, generating the optimal set, can only solve relatively small problem instances, heuristics are able to compute quickly good (but mostly suboptimal) sets without restriction on the problem size. Nowadays, most heuristics are problem-specific implementations of general guidelines called *metaheuristics* (Sörensen, 2013). Metaheuristics have been first designed for single-objective optimization and their success there inspired extensions to multi-objective optimization (Zhou et al., 2011; López-Ibánez and Stützle, 2012; Ghisu et al., 2010; Arroyo and Souza Pereira, 2011; Schaus and Hartert, 2013). With increasing computational power, combinations of exact algorithms and heuristics are more and more used in single-objective optimization. They are, however, barely investigated in the multi-objective domain (Ehrgott and Gandibleux, 2008). Beam search, among them, is typically built from a branch and bound tree, explored in a breadth-first manner and pruned by a decision-heuristic. The quality of the solutions found by a given implementation depends on one numerical parameter: the beam width w. It sets the maximum number of nodes kept at each level of the tree. Figure 1 illustrates a tree built with $w = 2$. The dotted lines depict discarded branches. Its size is polynomial in its depth, the combinatorial explosion is prevented.

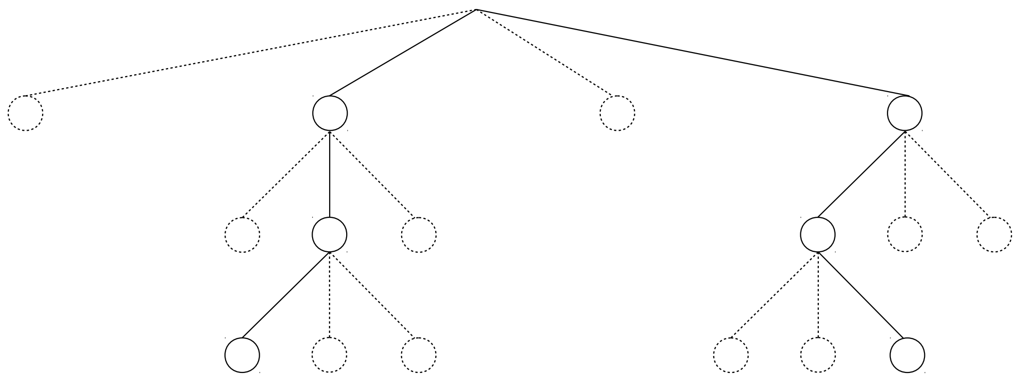

Figure 1: Tree built by a beam search with $w = 2$. Dotted lines are non-selected branches.

Historically, beam search has been introduced in the artificial intelligence community by Lowerre (1976) and the first translation to optimization problems came 12 years later (Ow and Morton, 1988). There, it has been mainly applied to single objective problems, often in the area of scheduling. According to the review of Sabuncuoglu et al. (2008), more than 30 contributions studied single-objective beam search since 1988. To the best of our knowledge, only two teams have addressed multi-objective beam search procedures. Honda et al. (2003) solve a bi-objective scheduling problem and Ponte et al. (2012) a bi-objective knapsack problem. Their methods are different from each other and our point of view brings a third approach.

The key feature of beam search algorithms is the rejection of nodes at each level of the search tree in order to keep at most w nodes to branch on. All authors naturally achieve this in two stages: an evaluation of the potential of the nodes which is then taken into account by a selection procedure. Both stages are more or less elaborated in their *implementation*, but the literature lacks a study on their *properties and interactions*. This article identifies desirable properties, studies how they interact with each other and how the other components of beam search influence them. On a practical point of view, it engenders guidelines for implementation. Our work is the first to give insight on what can be done in beam search algorithms and under which inter-dependent conditions, when tackling multi-objective combinatorial optimization problems.

To achieve the evaluation of the potential of a node in the search tree, authors use lower bounds and/or upper bounds. For instance, Ghirardi and Potts (2005) consider a linear combination of lower and upper bounds in the context of single-objective optimization, while the multi-objective approach of Honda et al. (2003) samples one point from the lower-bound set of the node and merges its objective values into a scalar indicator. In both cases, the nodes are ranked with respect to the resulting scalar values, and the selection stage consists in keeping the nodes associated with the best values. Ponte et al. (2012) follow a different approach. There, the objective values are not scalarized and the selection stage handles this data through an algorithm tailored to multi-objective optimization. Besides this fundamental difference between Honda et al. (2003) and Ponte et al. (2012), we notice a common point: the evaluation stage computes a low number of values. A small amount of data forecasts the quality of the numerous compromise solutions arising from a node. The time thus saved is exploited by performing backward steps during search. They step back in the tree respectively by backtracking and restarting from the root node. In contrast, our paradigm does not restrict the number of values at the evaluation stage (more values tend to improve the node selection correctness) and does not assume backward steps. Our node selection is tailored to multi-objective optimization. For both stages, we identify properties about their behavior which were targeted implicitly in the literature so far. We prove that they are together sufficient to find optimal solutions, but that some of them are unfortunately mutually exclusive if the algorithm must run in polynomial time (which is a goal of beam search), provided that P$\neq$NP.

These impossibility theorems are actually a wealth of knowledge. They imply guidelines to design the node evaluation and selection stages as well as other parts of beam search, with respect to the properties that the designer wants to meet or approach. The other parts in question are a selection of components that the literature considers usually and/or which must be carefully designed in light of the properties: the branching scheme (way to partition the search space), the variable selection (choosing the variable to restrict when branching on a node), and two anticipation stages which optimize the branching decisions. Since beam search has been introduced to the optimization area, authors have only studied a single branching scheme for the problem they tackle. For instance, in the context of scheduling, the employed branching scheme usually corresponds to assigning an unscheduled job to each machine in turn (see, e.g., Ghirardi and Potts, 2005; Sabuncuoglu et al., 2008). Other examples include Kim and Kim (1999) who tackle a transportation problem, Della Croce et al. (2004) who solve a location problem and Ponte et al. (2012) who address a bi-objective knapsack problem, evaluating only a single branching scheme each. As a result and to the best or our knowledge, the influence of enumeration schemes on the behavior of beam search algorithms has not been studied yet. We compare two schemes and show that, in the context of beam search, it can pay off to investigate non-standard schemes. Regarding the variable to restrict when branching, multi-objective optimization frameworks often set the enumeration order before entering the tree (Mavrotas and Diakoulaki, 2005; Sourd and Spanjaard, 2008; Vincent et al., 2013). Thus all branches enumerate the variables in the same sequence. Both existing multi-objective beam search algorithms follow this approach. However, Ponte et al. (2012) notice that the quality of their solution sets strongly depends on their pre-defined enumeration order. In this article, we propose a way to decide on the variable to restrict at each node during the execution of the algorithm. It leads to noticeably better solution sets than a predefined order. At last, we investigate the influence of what we call *anticipation rules*. Infeasibility anticipation rules perform a pre-improvement of future branching decisions while non-optimality anticipation rules (referred to as "recovering" in Della Croce et al. (2004) and related work), perform a post-improvement of previous branching restrictions.

The contributions of the paper are organized as follows. In Section 2, we provide a theoretical basis for multi-objective beam search by identifying properties of the algorithm that the literature considered implicitly so far. In Section 3, the properties are analyzed and we show that they face mutual incompatibilities when beam search must run in polynomial time. Discussions on how to design efficiently the components of beam search are given in light of this interrelationship. The guidelines are not only theoretical, they will be verified by experiments on two structurally different problems (a bi-objective knapsack problem and a bi-objective traveling salesman problem with profits). In Section 4, the beam search components that we use for the verification are detailed. In Section 5, the experiments are performed. Finally, since our knapsack problem is already addressed by two non-exact methods in the literature and their experimental data is available, we compare our implementation, which turns out to outperform them.

2 Theoretical grounds

In beam search, the search tree is explored level by level (breadth-first search). At each level l of the tree, we retain at most $w \geq 1$ nodes. Parameter w is called the *beam width*. Every node at a given level l is the child of a non-discarded node of the previous level $l-1$. $l=0$ denotes the level of the root node.

All statements of this article assume that the problem to solve is NP-hard (otherwise the development of a non-exact method does not appear to be justified) and all objective functions involve integer variables only (we restrict our study to discrete solution sets). When talking about branching schemes, we assume that they are *exhaustive*, in that no optimal solution is missing among the leaves that they would lead to if no pruning was done. We make use of standard terminology and operators of the multi-objective domain. Concepts such as optimality in the sense of Pareto, Pareto dominance, Pareto sets, supported and non-supported solutions are well known there and unfamiliar readers will find the definitions in Ehrgott (2005), Zitzler et al. (2010) or Jensen (2003) among others.

2.1 Notation

The set of Pareto-optimal solutions to the multi-objective sub-problem carried by a node σ^i in the search tree is denoted by $R(\sigma^i)$. The Pareto set of a given set G of points in the objective space is denoted by $P(G)$. We denote by $z(\sigma^i)$ the point whose coordinates are the objective function values of tree leaf (solution) σ^i. If σ^i is an infeasible solution, $z(\sigma^i) = \varnothing$. By extension, $z(\sigma)$ is the set of points arising from every leaf in node set σ. Given a branching scheme, the search tree which would be explored without pruning ($w \to +\infty$) is called *whole tree*. Given two functions $u(x)$ and $v(x)$, $u \circ v(x)$ denotes $u\big(v(x)\big)$. Given two points p_1 and p_2, $p_1 \preceq p_2$ means that p_1 weakly dominates p_2. All equations handling sets of points are written with set algebra but handle duplicates (when $p_1 = p_2$) such that $\{p_1\} \cup \{p_2\} = \{p_1, p_2\}$, $\{p_1, p_2\} \backslash \{p_1\} = \{p_2\}$ and $\{p_1, p_2\} \cap \{p_1\} = \{p_1\}$.

2.2 Definitions for node evaluation and selection

Beam search algorithms, whether they address single-objective or multi-objective optimization, handle a set of nodes at each level of the search tree and reject some of them. Naturally, all authors design a component (that we call *oracle*) to forecast the quality of the solutions that each node leads to, along with a component (that we call *filter*) to select the nodes with respect to the forecast. In this section, we introduce formal definitions for these two components, followed by two additional definitions needed for the rest of the study.

Definition 1. *An* oracle *g is an algorithm producing a set of points $g(\sigma^i)$ for a given node σ^i. By extension, we denote $g(\sigma) = \cup_{\sigma^i \in \sigma} g(\sigma^i)$.*

Definition 2. *Given a node set σ and two parameters w and g denoting respectively a beam width and an oracle, a* filter *F is a polynomial-time algorithm selecting a subset $F^g_w(\sigma) \subseteq \sigma$ such that $|F^g_w(\sigma)| = \min\{w, |\sigma|\}$ for any $w \geq 1$.*

Note that what we call a filter in our multi-objective paradigm is different from the concept of filter in the single-objective literature, used for instance in Ow and Morton (1988) and Sabuncuoglu et al. (2008).

Definition 3. *Given a node set σ and an oracle g, the* ideal beam width *w^* is the cardinality of the smallest subset $\sigma' \subseteq \sigma$ such that $P \circ g(\sigma') = P \circ g(\sigma)$.*

Whereas the beam width w is a constant parameter set by the user, the ideal beam width is defined for theoretical purposes and depends on the considered node set.

Definition 4. *A node σ^i is* feasible *for an oracle g if $g(\sigma^i) \neq \varnothing$.*

2.3 Properties for node evaluation and selection

Naturally, all authors of single-objective and multi-objective beam search algorithms expect their oracles to give a reliable measure of the potential of a node. In this section, we formalize the notion of *reliable measure* by means of two properties for the oracle (*constancy* and *leaf-compliance*). By construction, a multi-objective beam search needs a way to select nodes at each iteration. All authors tend to select nodes whose oracle points are non-dominated by the other nodes. In this section, we formalize this behavior by means of a property for the filter (*Pareto efficiency*). In addition, we introduce *independence of irrelevant alternatives*, a property for the filter that the literature neglected so far although a related notion is fundamental in Social Choice Theory[1] (Arrow et al., 2002).

Property 1. *Let σ_l be the set of nodes at level l of the whole tree. Given a branching scheme, let $b(\sigma^i) \subseteq \sigma_{l+1}$ be the set of nodes and leaves obtained by branching on node $\sigma^i \in \sigma_l$. An oracle g is* constant *if $P \circ g(\sigma^i) = P \circ g\big(b(\sigma^i)\big)$ for every node (except leaves) σ^i in the whole tree of any instance of the considered problem.*

Equivalently, a constant oracle is such that

$$\frac{\left|P \circ g(\sigma^i) \cap P \circ g\big(b(\sigma^i)\big)\right|^2}{\left|P \circ g(\sigma^i)\right| \cdot \left|P \circ g\big(b(\sigma^i)\big)\right|} = 1 \tag{1}$$

for every node (except leaves) σ^i. Informally, the *degree of constancy* of an oracle refers to the average value of the fraction over all possible whole trees, given a branching scheme.

Property 2. *Given a branching scheme, an oracle g is* leaf-compliant *if $z(\sigma^i) \preceq z(\sigma^j) \Leftrightarrow g(\sigma^i) \preceq g(\sigma^j)$ for every pair of leaves (σ^i, σ^j) in the whole tree of any instance of the considered problem.*

Equivalently, if we consider the usual case where $g(\sigma^i) = z(\sigma^i)$ for any leaf σ^i carrying a feasible solution, a leaf-compliant oracle is such that

$$|g(\sigma^i)| = |z(\sigma^i)| \tag{2}$$

for every leaf σ^i in the whole tree of any instance of the considered problem. It means that g does not compute more than one point per leaf and a leaf is feasible for g if and only if it is a feasible solution. Informally, the *degree of leaf-compliance* of an oracle refers to the proportion of leaves meeting Equation (2) over all possible whole trees, given a branching scheme, provided that the oracle meets $g(\sigma^i) = z(\sigma^i)$ for any leaf σ^i carrying a feasible solution.

Property 3. *An oracle is* perfect *if it is constant and leaf-compliant.*

Note that $z \circ R$ is a perfect oracle for any branching scheme, but no algorithm running in polynomial time is known to compute it.

Property 4. *Let g be an arbitrary oracle. A filter F is* Pareto-efficient *if $P \circ g\big(F^g_{w^*}(\sigma)\big) = P \circ g(\sigma)$ and $|F^g_w(\sigma) \cap F^g_{w^*}(\sigma)| = \min\{w, w^*\}$ for any instance of the considered problem, any beam width $w \geq 1$ and any subset σ of nodes in the whole tree.*

As a priority, a Pareto-efficient filter selects the nodes whose oracle points contribute to the Pareto set of the oracle points of the nodes of σ.

Property 5. *Given a branching scheme, let σ be the set of nodes of the whole tree at level l and σ^S be the set of leaves in the whole tree up to level l. Let $\sigma' \subseteq \sigma$ be the nodes submitted to the filter in beam search. Let w^* be the ideal beam width of $\sigma \cup \sigma^S$. Given an oracle g and a beam width w, a filter F is* independent of irrelevant alternatives *if $|F^g_w(\sigma') \cap F^g_{w^*}(\sigma \cup \sigma^S)| = \min\{w, |\sigma' \cap F^g_{w^*}(\sigma \cup \sigma^S)|\}$ for any instance of the considered problem and at any level l.*

If one removes some nodes from a set whose ideal beam width is w^*, an independent of irrelevant alternative filter selects as a priority the nodes left which are among the w^* nodes that it would have selected if the whole set was known. Note that this property depends on the branching scheme, the oracle and w.

Property 6. *A filter is* perfect *if it is Pareto-efficient and independent of irrelevant alternatives.*

Property 7. *A filter is d-*versatile *if its decisions*

- *rely only on the beam width and oracle points;*
- *do not exploit any specificity of certain d-objective optimization problems.*

3 Analysis and guidelines

Our theoretical analysis and implementation guidelines rely on the concepts introduced above. The reason is that oracle and filter perfection are sufficient for beam search to find Pareto-optimal solutions, as stated in Theorem 1.

Theorem 1. *Let $R(\sigma_0)$ be the set of all Pareto-optimal solutions to a given problem. If the oracle is perfect and the filter is perfect, then beam search finds at least $\min\{w, |R(\sigma_0)|\}$ solutions of $R(\sigma_0)$.* (Proof in Appendix B)

[1]area studying the selection of candidates according to voter evaluations, which is a particular case of our context (the contexts would be the same if the oracle computed only one point per node)

Note that this theorem does not assume that beam search runs in polynomial time. In practice, we aim for polynomiality for tractability reasons. Provided that P≠NP, obtaining optimal solutions to a NP-hard problem in polynomial time is impossible, that is why we consider the oracle and filter properties as guiding targets, worth considering to obtain likely good solutions.

The first two subsections identify the intrinsic limits due to polynomial-time solving. Furthermore, they provide food for thought to obtain oracles and filters "as perfect as possible" in order to tackle respectively node evaluation and selection. The next subsections study four branching components that the literature considers usually and/or which must be carefully designed in light of the properties (variable enumeration scheme, selection of variables to restrict for branching, and optimization of the restrictions by means of infeasibility and non-optimality anticipations). For each advice, concrete examples are given later in Section 4. The last subsection explains the influence of the beam width w on the quality of the solutions found.

3.1 Oracle

As explained above, incompatibilities between component properties have to be expected when aiming at a polynomial-time beam search. The following theorem identifies a first limit.

Theorem 2. *Provided that P≠NP, there does not exist a perfect oracle if and only if beam search is polynomial in the problem size.* (Proof in Appendix B)

Therefore, we can only have either constancy, leaf-compliance or none of these properties. The following discusses natural ways to implement an oracle and study their properties.

Linear relaxations. Such relaxations cannot give constant oracles because some fractional solutions are discarded by the branching. In detail, the solutions obtained at a node are not all among the solutions obtained at its child nodes. Then, term $\left|P \circ g(\sigma^i) \cap P \circ g\big(b(\sigma^i)\big)\right| / \left|P \circ g(\sigma^i)\right|$ in Equation (1) is within 0 and 1. Absence of non-supported solutions gives $0 \leq \left|P \circ g(\sigma^i) \cap P \circ g\big(b(\sigma^i)\big)\right| / \left|P \circ g\big(b(\sigma^i)\big)\right| \leq 1$ for the other term. Hence there is no guarantee on the degree of constancy. In practice, linear relaxations often provide accurate bounds in that the objective values of their solutions are close to optimal ones and the more accurate they are the more the integer solutions that they find are numerous. Translated to our paradigm, more integer solutions implies more solutions in common between a node and its subnodes, which means that the accuracy of a linear relaxation is correlated with its degree of constancy. Moreover, linear relaxations are leaf-compliant because every tree leaf is feasible for such an oracle if and only if it is feasible for the integer problem, and the objective values are the same. Since beam search procedures aim at good trade-offs between solution quality and solving time, the linear program should be solved by a tailored algorithm unless it is intrinsically easy for a generic simplex solver. Otherwise, the time needed to solve it might overweigh the leaf-compliance and the high degree of constancy, compared to the following options.

Combinatorial relaxations. Such relaxations consist in disregarding some constraints other than variable integrity. They are constant oracles if and only if the branching scheme misses no relaxed solution. However, computing their non-supported solutions is harder than computing the supported ones in general, so the former have to be skipped unless they are computed in polynomial time on average in practice. Then the oracle is not constant although its degree of constancy is positive: all solutions from a node are among the Pareto-optimal solutions to the union of its sub-node problems, hence $\left|P \circ g(\sigma^i) \cap P \circ g\big(b(\sigma^i)\big)\right| / \left|P \circ g(\sigma^i)\right| = 1$ in Equation (1); the converse is not true because the union of the sub-node problems has likely solutions which are non-supported solutions to the parent node (that it did not compute), so $0 < \left|P \circ g(\sigma^i) \cap P \circ g\big(b(\sigma^i)\big)\right| / \left|P \circ g\big(b(\sigma^i)\big)\right| \leq 1$ holds. Combinatorial relaxations have a positive degree of leaf-compliance because they produce at most one point at tree leaves, where the objective values are those of the corresponding original problem solution when it is feasible (see Equation 2). They allow for leaf-compliant oracles if and only if the branching scheme avoids all relaxed solutions which are infeasible for the original problem. However, when such a scheme is used, constancy is lost because the solutions carried by the disappeared infeasible leaves cannot be all excluded from the solutions that the oracle computes for parent nodes. So, in any case, combinatorial relaxations face a dilemma, which is a concrete example of the incompatibility stated by Theorem 2.

Oracles which are neither lower nor upper bounds can be made through combinatorial relaxations. For example, one may forget about the original problem and produce random variable values with biased randomness in order to tend to observe the problem constraints (without guarantee). Although no guarantee exists on the degrees of constancy and leaf-compliance, using a dedicated branching scheme (see Section 3.3) and improving the problems carried by the nodes (see Section 3.5) might increase the degrees enough. In Section 4.2.1, we suggest another uncommon approach: a combinatorial relaxation which is NP-hard so heuristically solved. It will be shown to be valuable.

Approximation schemes and heuristics. They are not constant oracles in general, because the constraints added by the branching change likely the sequence of decisions made by their algorithms (obviously, heuristics using randomness are not constant). Ideas and efforts might be focused on obtaining an approximation scheme or a heuristic whose solutions are changed by branching decisions as little as possible, thus optimizing the degree of constancy. Moreover, their accuracy is correlated with their degree of constancy for the same reason as linear relaxations. However, here again, accuracy is often related to time consumption, so practitioners are invited to balance them or regard the other options above. Approximation schemes and heuristics are leaf-compliant because they observe all constraints of the original problem.

From a general point of view, oracles may be computed with multi-objective solvers. It is likely intractable or complicated in practice. As usually done in the literature, we suggest to generate sets of points by enumerating various linear combinations of the objective functions and, for each so-obtained single-objective problem, compute a solution. The original objective values are deduced from the variable values, thus giving a point. Note that, when these solutions are optimal for the single-objective problem (typically, because a relaxation is solved exactly), linear combinations produce only supported solutions. For non-exact single-objective algorithms, supported and non-supported solutions are both likely to be missing. Thus, in both cases, constancy is not met. A popular method for enumerating the linear combinations is Aneja-Nair's algorithm (Aneja and Nair, 1979) whose modern description is given by Ponte et al. (2012) among others. Originally designed for two objectives, it has been extended to higher dimensions (Özpeynirci and Köksalan, 2010; Przybylski et al., 2010).

The oracles above are able to produce feasible solutions. We suggest to detect and add them to the set of solutions found by beam search.

3.2 Filter

The following theorem gives conditions for a filter to be perfect. It teaches us that this ability depends on the oracle and the beam width, thus identifying another limit of beam search algorithms.

Theorem 3. *Let us denote by w_l^* the ideal beam width of the node set known at level l of the search tree. If the oracle is constant or $\forall l \ w \geq w_l^*$, then there exists a perfect filter; otherwise, there does not exist a deterministic d-versatile filter which is perfect for any d-objective integer optimization problem.* (Proof in Appendix B)

In practice, we face two difficulties when the oracle is not constant. First, w is often fixed and smaller than w_l^* because w_l^* depends on the instance and may be exponentially large in the instance size. Then, designing a filter perfect for the considered problem is not trivial (probably impossible). We have to settle for either Pareto-efficiency, independence of irrelevant alternatives or none of these properties.

In any case, Pareto-efficiency relies on the following necessary condition.

Theorem 4. *Let g be an arbitrary oracle and σ be a set of nodes. If a deterministic filter is Pareto-efficient, then for every node $\sigma^i \in \sigma$ it compares (directly or through transitive relations) each point of $P \circ g(\sigma^i)$ to at least one point of every other node in σ.* (Proof in Appendix B)

Thus, some filters relying on well known indicators lack Pareto-efficiency. For instance, the filter described in Section 4.3.1, using the hyper-volume indicator, does not compare any pair of points. In the filter of Ponte et al. (2012) employing the ϵ indicator, relations between oracle points and the reference set are not transitive so dominance relations between oracle points are ignored. To meet Pareto-efficiency, we suggest to follow Property 4: identify the Pareto-set of the oracle points from all nodes, and select as a priority nodes having an oracle point in it.

When the oracle is not constant and $\forall l \ w \geq w_l^*$ does not hold, there is a sufficient condition to design a filter which is independent of irrelevant alternatives:

Theorem 5. *Given a set of nodes σ, sort the nodes with respect to a criterion such that removing any subset of nodes once sorted gives the same order as removing it before. Select the first $\min\{w, |\sigma|\}$ nodes. This filter is independent of irrelevant alternatives.* (Proof in Appendix B)

Concretely, the criterion may be a scalar value computed for each node independently of any data from the other nodes, such as the hypervolume of its oracle points.

When the oracle is constant but $\forall l \ w \geq w_l^*$ does not hold, this is how to build a perfect filter F for the considered d-objective integer optimization problem given a d-versatile Pareto-efficient filter f. During the search, keep in memory the discarded nodes and the leaves reached so far. At every level, create a set gathering them and the *candidate* nodes (the nodes to select from). Consider every candidate node having at least one oracle point in the Pareto-set of all points arising from the created set. If w is not larger than the considered subset, apply f to it in order to select w nodes. Otherwise, select the whole subset and complete by selecting with f enough nodes from the candidate set. The proof of Theorem 3 shows that this procedure is a perfect filter.

When $\forall l \ w \geq w_l^*$ due to an adaptive or large enough value, reaching perfection is trivial: any Pareto-efficient filter is independent of irrelevant alternatives. It is shown in the proof of Theorem 3.

3.3 Branching scheme

We advise to carefully choose the tree structure of beam search procedures and their interrelationship with the oracle. While most branch & bound procedures build a binary tree, such a scheme in our context is not guaranteed to give better results than problem-specific enumerations. For simplicity, we illustrate our arguments with a beam search for a d-objective traveling salesman problem (d-TSP). Given a graph where each edge has d distinct weights, the problem consists in finding every Pareto-optimal Hamiltonian cycle, whose the d lengths are minimized. We denote by n the number of vertices in the graph.

Let us consider two branching schemes. The first one is a classical binary enumeration, every edge is made alternatively mandatory and forbidden and all combinations over all edges are tried in the whole tree. The second scheme is tailored. Let us assign numbers from 1 to n to the vertices of the graph and, without loss of generality, consider that it is complete. Level 1 of the tree has $n - 1$ nodes: in each node, an edge is mandatory between vertex 1 and an adjacent vertex (such that every adjacent vertex is mandatory in one node). Similarly, at level 2, each node adds a mandatory edge between the vertex previously connected and one of the $n - 2$ remaining adjacent vertices, and so on. Thus, a path is drawn, growing level by level until no free vertex remains. Then, the path is closed to obtain a cycle.

In the following, we show that efficiency of the oracle to guess the quality of the nodes depends on the structure of the partial solutions. A partial solution is the set of constraints added by the branching along the path between the considered node and the root node of the tree. It is a set of mandatory and forbidden edges with the binary scheme, and a set of mandatory edges with the tailored scheme.

First, let us explain how the partial solution structure influences oracles based on combinatorial relaxations. For the d-TSP, the oracle may, e.g., rely on a 1-Tree relaxation (Held and Karp, 1970, 1971). Here, it is sufficient to know that a 1-Tree is based on a minimum spanning tree. With a binary scheme, this oracle mainly guides the beam search towards a set of low-weight spanning trees. Towards the end of the search, connecting the scattered mandatory edges of a partial solution tends to involve expensive edges due to the necessity of forming a cycle. On the opposite, every branch of the tailored scheme draws a path converging to a cycle, and expensive edges are seen and discarded progressively. Note that a tailored scheme prevents constancy of oracles based on combinatorial relaxations (in our example, certain spanning trees become infeasible each time the path grows) but it can bring leaf-compliance (in our example, all leaves are feasible for both the oracle and the original problem), whereas the binary scheme allows such oracles to be constant but prevents leaf-compliance. Therefore, the overall benefit has to be verified, especially when anticipations are implemented because they can modify the degrees of constancy and leaf-compliance (see Section 3.5).

If tailored schemes are likely successful, it is for the following reason. The points computed by an oracle depend, in practice, on how constrained the partial solutions are. The depth of the binary branching tree is $\mathrm{O}(n^2)$ and the depth of the tailored branching tree is $\mathrm{O}(n)$. Therefore, on average, the nodes at a given level l of the tailored tree carry subproblems that are more constrained than the nodes at level l of the binary tree. For this reason, the binary scheme is expected to produce less accurate oracle points and less discriminating differences between the nodes, thus leading the filter more often to wrong decisions than the tailored scheme.

3.4 Variable selection

Some branching schemes decide intrinsically which variables must be restricted in order to generate the children of a node. It is the case of the tailored scheme that we discussed in Section 3.3. However, often, the scheme gives us a choice of the variable to restrict. We suggest a straightforward way to decide it.

Usually, in single-objective branch and bound algorithms, each branch of the tree enumerates the variables in its own order: the decisions are made in each branch independently of the other branches. The reason is that the variables to restrict are chosen according to the structure of the relaxed solution of each node. In general, this creates sooner (formally, at a lower level of the tree) accurate bounds and/or feasible solutions. As a result, pruning is more intense so solvers are faster (Derpich and Vera, 2006). A deeper analysis has been suggested for single-objective branch and bound algorithms. It consists in computing a pool of solutions at each node which have equivalent objective values but various variable value combinations. The latter give the opportunity to compute statistics and to profit from richer information to select the restricted variable (Fischetti, 2014).

Like in Parragh and Tricoire (2015), we suggest to follow these footsteps in multi-objective beam search algorithms. A set of non-optimal and/or infeasible solutions at each node is computed. For example, the oracles described in Section 3.1 provide such data, or it can be achieved by a dedicated algorithm (the oracle is originally a component of the node pruning process and is not required to compute meaningful variable values). For binary decision variables, the average value of each over the solution set is computed and we choose the variable whose mean is the closest to 0.5. If the problem has non-binary integer variables, we may may prefer the highest relative standard deviation. The thus improved accuracy of the oracle points can increase the degree of constancy of the oracle (see Section 3.1 for the link between accuracy and constancy degree).

3.5 Infeasibility anticipation and non-optimality anticipation

In our framework, no backward step is performed while building the search tree. Instead, two components contribute to prevent bad branches from being selected and optimize the branches themselves. In the modern branch & bound literature, they are part of the concept of valid inequalities. Some valid inequalities may be complicated to implement, or even impossible (the oracle does not necessarily rely on linear programming). Instead, improvements on partial solutions are performed.

After a branching decision, certain values of some variables reachable by the oracle may be forbidden by the complete problem formulation. The *infeasibility anticipation* rules restrict such variables. Due to the lower number of free variables and/or their possible values, tree leaves are reached sooner. For this reason, infeasibility anticipation improves accuracy of the points computed by the oracle (see the arguments about tree depth in Section 3.3). However, these improvements are balanced by the fact that the fixes appear progressively along the branches so some solutions to a node problem are missing in its subnode problems. Section 3.1 explains the effects of accuracy and disappearing oracle points on the degree of constancy and that they are contradicting, so the overall benefit has to be verified. In any case, infeasibility anticipation improves the degree of leaf-compliance of the oracle as soon as it removes leaves which are feasible for the oracle but infeasible for the problem.

Neighborhood operators (as defined in the literature about heuristic solvers) can be turned into pruning rules in exact branch & bound algorithms without loosing optimality. In beam search, instead of pruning a node whose non-optimality is anticipated, its partial solution is optimized. This concept is known as *recovering* in the beam search literature (Della Croce et al., 2004) but we think that *non-optimality anticipation* is more consistent in our framework. Note that the improvement performed might generate partial solutions already carried by the node set. These duplicates must be discarded. Besides improving, non-optimality anticipation degrades the degree of constancy of the oracle because it changes the node problems (see Equation 1). Therefore, here again, the overall benefit has to be verified. The degree of leaf-compliance is not affected. Regarding implementation of the rule, all value combinations of the variables not restricted by the partial solution of a node should remain available once the rule is applied. Thus, no solution to the new node problem is worse than a solution to the former problem. Then, provided that the objective functions are linear, the improvement process of the rule can look for one point (computed from the partial solution) dominating the point of the original partial solution, rather than comparing the entire solution sets of the problems.

3.6 On the influence of the beam width

An analysis of Beam Search cannot miss a discussion on the beam width w, the parameter that the user tunes. Oracle constancy and leaf-compliance are defined over the whole tree, not over the sub-tree arising from w (that would not be sufficient for Theorem 1), so w has no influence on these properties. Filter perfection is achievable with large-enough w values (see Theorem 3). But, in practice, w is needed to be small (for time budget reasons) while the smallest large-enough value is likely exponential in the problem size. Therefore, in practical applications, w has no influence on filter Pareto efficiency and independence of irrelevant alternatives.

If both the oracle and the filter were perfect, Theorem 1 would directly show the influence of w on the quality of the solution-sets found: the guaranteed number of Pareto-optimal solutions in the set would increase with respect to w. In practice, we face imperfection. The resulting behavior can be interpreted with the following arguments. If the filter selected randomly, the probability to catch good solutions would increase with respect to w, so we expect that actual oracle and filter combinations show the same correlation. Moreover, good oracles and filters limit the number of solutions dominating each other, so the size of the set found tends to increase with respect to w.

4 Examples of beam search components

We now describe components implementing beam search for two structurally different problems (a bi-objective knapsack problem and a bi-objective traveling salesman problem with profits). All components discussed in the previous section are addressed. Most of them are problem-dependent (oracle, branching scheme, variable selection and anticipation rules), only our filters are general-purpose. This section has a goal in itself – illustrating our concepts and suggesting ways of proving component properties – but it also aims at supporting our guidelines by implementing the various possibilities discussed in Section 3 and evaluated in Section 5.

4.1 A bi-objective Knapsack Problem

Knapsack problems are popular to evaluate solving methods (see, e.g., Jaszkiewicz, 2004; Pisinger, 2005; Derpich and Vera, 2006; Özpeynirci and Köksalan, 2010; Gao et al., 2014). Several multi-objective knapsack problems (BKP) exist. Three versions are presented in Liefooghe et al. (2013). Another popular variant is addressed by the multi-objective beam search of Ponte et al. (2012), so we chose it. Given a set of n undividable items i, each having a weight w_i and two profits (p_i^1, p_i^2), the problem consists in selecting items such that both sums of profits are independently maximized while the total weight of the selected items does not exceed a given limit W.

4.1.1 Oracles for our BKP

Two algorithms will be compared to generate oracle points for our BKP.

Dantzig-based upper bound (DUB). This oracle relies on the classic and simple linear relaxation of Dantzig, described for instance in Martello and Toth (1990). This single-objective bound sorts the items by increasing ratio w_i/p_i, then selects the items in this order while W is not exceeded. The first item which cannot be selected is called the *critical item*. Then, a fraction of it is taken such that W is reached.

The structure of that single-objective bound allows us to enumerate in $\mathrm{O}(mn)$ all of the m extreme supported solutions to the linear relaxation of our BKP. To that aim, we use the tailored bi-objective simplex of Figueira et al. (2013).

Proposition 1. *This oracle is not constant but it is leaf-compliant.* (Proof in Appendix C)

By construction, DUB finds very unlikely integer solutions, hence an uncertain degree of constancy. However, the low complexity of DUB might be an advantage.

Greedy lower bound (GLB). Aneja-Nair's algorithm (see Section 3.1) is used to enumerate several solutions with the single-objective greedy heuristic described in Martello and Toth (1990). The heuristic sorts the items by increasing ratio w_i/p_i, then selects the items in that order as long as W is not exceeded. Unlike Dantzig's bound described in the previous paragraph, the critical item is not selected and the procedure keeps scanning the item list in order to select those which fit in the residual capacity. Each oracle point is obtained in $\mathrm{O}(n \log n)$.

Proposition 2. *This oracle is not constant but it is leaf-compliant.* (Proof in Appendix C)

GLB is a heuristic while DUB is a relaxation, thus we have an opportunity to experiment two completely different ways of measuring the potential of the nodes. The fact that GLB gives only feasible solutions might be an advantage over DUB. Anyway, when DUB is the oracle, we can use GLB in our experiments to produce feasible solutions at each node selected by the filter.

4.1.2 Branching scheme and variable selections for our BKP

The branching scheme is a classical binary enumeration. At each node, an item is selected. Then, two subproblems are created by making it respectively mandatory and forbidden. Note that some infeasible solutions are produced. Most of them are prevented by the rules given in Section 4.1.3. The following suggests two rules to select the item.

Statistical selection. It is the one advised for binary variables in Section 3.4.

Predefined ordering. It is similar to that used in Ponte et al. (2012). Before the search, all items i are sorted by increasing ratios $w_i/(p_i^1 + p_i^2)$. Then, at each level l of the tree, we branch on the item of rank $l + 1$.

4.1.3 Infeasibility and non-optimality anticipations for our BKP

It is infeasible to select items such that the capacity W is exceeded. Therefore, after each branching, we remove all items whose weight does not fit in the residual capacity. Hence infeasibilities are anticipated.

Here is how we anticipate non-optimality of some solutions. After selecting an item i, we look for an item j forbidden by previous branchings such that $w_j \leq w_i$ and $p_j^1 > p_i^1$ and $p_j^2 > p_i^2$ and $\max\{p_j^1/p_i^1, p_j^2/p_i^2\}$ is maximal. If such an item exists, i is replaced with j. Since the residual capacity resulting from this swap may be higher, items removed by infeasibility anticipations (as seen above) which fit again return to availability. It is clear that the conditions given at the end of Section 3.5 are met so no optimal solution is lost.

4.2 The bi-objective Traveling Salesman Problem with Profits

The bi-objective traveling salesman problem with profits (BTSPP) (Feillet et al., 2005; Jozefowiez et al., 2008; Bérubé et al., 2009; Filippi and Stevanato, 2012, 2013) allows us to evaluate the guidelines on a structurally different problem related to real world applications involving routing decisions. Given a graph of size n where each vertex $i \in \{1 \dots n\}$ has a profit p_i and each edge has a cost $c_{\{ij\}}$, it consists in finding all cycles such that the total profit of the selected vertices is maximized while the total cost of the traversed edges is minimized (Feillet et al., 2005). The cycles start and end at vertex 1.

4.2.1 Oracles for the BTSPP

As seen in Section 3.1, an oracle can be designed by combining linearly the objectives through various coefficients and, for each combination, addressing the so obtained single-objective problem in order to generate a point (for example, through a relaxation, an approximation scheme or a heuristic). For the TSP with Profits, we decide the coefficients with Aneja-Nair's algorithm (see Section 3.1). This section describes three different objective algorithms to address the single-objective problem, leading to three different oracles.

In a preliminary step, the single-objective problem considered by our algorithms is transformed. The weight of each edge $\{i, j\}$ is

$$w_{\{ij\}} = \alpha c_{\{ij\}} - (1-\alpha)\frac{p_i + p_j}{2} . \qquad (3)$$

One sees the edge costs combined to the profits through coefficient α. The latter are subtracted because we assume a minimizing objective while profits have to be maximized. This transformation is used in Righini and Salani (2006) and does not change the value of any feasible solution because each vertex i traversed by a cycle has two adjacent edges, thus counting twice $p_i/2$. This transformed problem is called *STSPP*.

Polynomial-size linear program (PLP). The first algorithm that we suggest consists in computing a linear relaxation of the STSPP. We had to choose between polynomial-size and exponential-size formulations of the STSPP. We noted that, when relaxing the classical single-objective traveling salesman problem, Roberti and Toth (2012) show experimentally that various polynomial-size formulations have solving times and accuracies close to each other, compared to the extremely short times and the excellent accuracy of Dantzig's exponential-size formulation. However, the latter would require to investigate efficient ways to incorporate cutting plane generation into multi-objective optimization algorithms, which is beyond the scope of this paper.

Therefore, the linear program that we consider is a polynomial-size formulation of the STSPP. It requires a directed graph, so every edge $\{i, j\}$ of the STSPP is split into two symmetric arcs (i, j) and (j, i). Given a set $\mathrm{V} = \{1 \ldots |\mathrm{V}|\}$ of vertices and a set $\mathrm{A} = \mathrm{V}^2 \backslash \{(i, i) \forall i \in \mathrm{V}\}$ of arcs, we describe the relaxed STSPP by

$$\left\{\begin{array}{lrcll}
\multicolumn{5}{l}{\min \sum_{a \in \mathrm{A}} w_a x_a} \\
\multicolumn{5}{l}{\text{s. t.}} \\
& \sum_{j \in \mathrm{V}\backslash\{i\}} x_{ij} & = & y_i \quad \forall i \in \mathrm{V} & \text{(A1)} \\
& \sum_{j \in \mathrm{V}\backslash\{i\}} x_{ji} & = & y_i \quad \forall i \in \mathrm{V} & \text{(A2)} \\
& x_{ij} + x_{ji} & \leq & y_i \quad \forall (i, j) \in \mathrm{A} & \text{(L)} \\
& u_j + (n-1)(1 - x_{ij}) & \geq & u_i + 1 \quad \forall i, j \in \mathrm{V}\backslash\{1\} & \text{(S)} \\
& x_{ij}, y_i & \in & [0\,;1] & \\
& u_i & \in & \mathbf{R}_+ &
\end{array}\right.$$

Regarding the variables, $x_a = 1$ if and only if arc a is part of the cycle and, similarly, $y_i = 1$ if and only if vertex i is part of the cycle. If vertex i is part of the cycle, then it is connected to one outgoing arc (constraint A1) and one incoming arc (constraint A2). Otherwise, no arc is connected to i. Constraints (L) are valid inequalities to improve the linear relaxation. Constraints (S) are Miller-Tucker-Zemlin's subtour elimination constraints (Miller et al., 1960).

Proposition 3. *This oracle is not constant but it is leaf-compliant.* (Proof in Appendix C)

Among oracles that we suggest for the BTSPP, PLP has the best accuracy with respect to the optimal solutions because the other oracles solve combinatorial relaxations of STSPP, so its degree of constancy is expected to be the highest. Note that it is the only leaf-compliant oracle when the branching scheme is binary (see Section 4.2.2).

Minimum Non-spanning 1-Tree (MN1T). Given a graph G, MN1Topt is the problem to compute T as follows.

1. Denote by G' the graph without the adjacent edges of vertex 1.
2. Find in G' the tree T' (not necessarily spanning) which includes all mandatory edges and whose total edge weight is minimal.

3. Make graph T from T' by adding two adjacent edges of vertex 1: select the mandatory edges as a priority, and complete if needed with the candidate(s) having the least weight.
4. If G has no mandatory edge and the total weight of T at Step 3 is positive, set $T = \varnothing$.

Proposition 4. *MN1Topt is a combinatorial relaxation of STSPP.* (Proof in Appendix C)

Proposition 5. *MN1Topt is NP-hard.* (Proof in Appendix C)

For tractability, MN1T replaces Step 2 with a $O(n^2 \log n)$ heuristic (the other steps are computable polynomially). The MN1T solutions are not optimal for MN1Topt. As a result, MN1T is neither a lower bound nor an upper bound. Here is the heuristic.

- Perform Kruskal's algorithm (Kruskal, 1956) but, at each iteration, do not connect two trees if the sum of the weight of the heaviest one plus the weight of the linking edge is greater than 0, unless they both contain a mandatory edge.
- Once the thus modified algorithm stops, keep the tree containing the mandatory edges. If no edge was mandatory, choose the tree with the lowest weight.

A (not necessarily spanning) tree is obtained and one attempts to minimize its total weight, thus approaching the requirements of Step 2.

Proposition 6. *This oracle is not constant. It is leaf-compliant if and only if all leaves of the tree are feasible for the BTSPP.* (Proof in Appendix C)

MN1T is based on a combinatorial relaxation. In Section 3.1, we saw that such relaxations tend to give oracles with a lower degree of constancy, but MN1T is faster to compute than PLP and observes more constraints of the problem than MN1F (see below). Leaf-compliance will be met with our path branching scheme and infeasibility anticipation (see Sections 4.2.2 and 4.2.3).

Minimum Non-spanning 1-Forest (MN1F). MN1Topt (see above) would lead to a NP-hard oracle, so a heuristic is used in MN1T. Here, we propose to relax a constraint of MN1Topt in order to obtain a problem solvable exactly in polynomial time. The relaxed problem is called Minimum Non-spanning 1-Forest.

In the definition of MN1Topt, at Step 2, a tree is computed. A tree is a connected graph without cycles. We remove the connection requirement and we just forbid cycles. As a result, Step 2 now consists in computing a forest whose total edge weight is minimal. It is obtained by simply discarding all edges having a positive weight in the instance and running Kruskal's algorithm on the remaining graph. Steps 1, 3, 4 are the same as MN1Topt. That takes $O(n^2 \log n)$.

Proposition 7. *This oracle is not constant. It is leaf-compliant if and only if all leaves of the tree are feasible for the BTSPP.* (Proof in Appendix C)

It will be interesting to compare the quality of the solutions found by beam search using MN1F and MN1T. Indeed, MN1F is more relaxed than MN1T but it is not heuristically computed; which of two works better is not obvious.

4.2.2 Branching schemes and variable selection for the BTSPP

Binary branching. It is a classical binary enumeration. At each node, an edge is considered according to the criteria suggested for binary variables in Section 3.4. Then, two sub-problems are created by making it respectively mandatory and forbidden. Some infeasible solutions are produced (so oracles MN1T and MN1F are not leaf-compliant). Most of them are prevented by the rules given in Section 4.2.3 (which improve the degrees of leaf-compliance).

Path branching. It is tailored to the problem, but does not observe completely the constraints, which allows us to study the benefits brought by the infeasibility anticipations rules. After the root, each node of the second level of the tree makes an edge mandatory between vertex 1 and one of its adjacent vertices. At the next level, for each of those vertices, an edge is made mandatory between it and every adjacent vertex (including those previously visited, unless the edge is already mandatory due to previous branchings), and so on. As a result, a path is built level by level and we consider that a leaf is reached as soon as a cycle is formed. All infeasible solutions produced by this scheme are prevented by the rules given in the following section (thus making oracles MN1T and MN1F leaf-compliant).

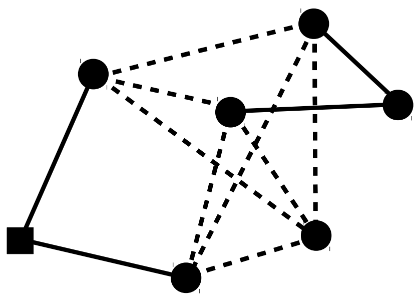
Figure 2: Infeasibility anticipation for the BTSPP. Selecting a non-depicted edge would cause infeasibility.

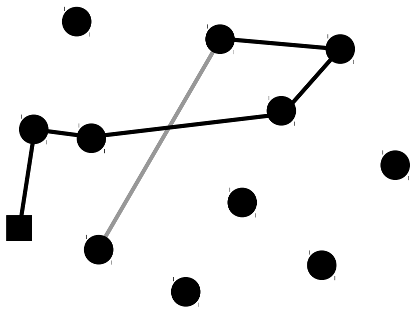
Figure 3: Non-optimality anticipation for the BTSPP. A 2-opt would improve the path length in this Euclidean example.

4.2.3 Infeasibility and non-optimality anticipations for the BTSPP

The number of infeasible solutions produced by MN1F and MN1T is decreased by the following four rules. Figure 2 depicts a graph where the square is vertex 1. The bold edges are mandatory due to the branching. The dotted edges remain available for future branching and for the oracle computations. The missing edges have been removed by the first two rules. The first rule removes all adjacent edges of a vertex if the latter has 2 mandatory edges. The second rule removes edges which would form a mandatory cycle not containing 1, as well as edges which would form a mandatory cycle missing a mandatory edge. A third rule, not depicted, makes mandatory the edges of vertex 1 as soon as two of them remain only. When the branching forbids an edge or when the rules above remove an edge, a vertex may be left with one adjacent edge only. In that case, a fourth rule removes this single edge, and it is applied recursively if need be.

Regarding non-optimality anticipation, Figure 3 depicts a graph where the costs of the edges are their Euclidean length for clarity. The gray edge is the last one added by our Path branching. The crossing edges can be uncrossed to decrease the total length of the path. This operation is known as *2-opt* (Johnson and McGeoch, 1997) and is valid also for non-Euclidean edge costs as soon as one finds a pair of edges such that "uncrossing" them decreases the total cost. Hence non-optimality of some partial solutions is anticipated. It is clear that the conditions given at the end of Section 3.5 are met so no optimal solution is lost.

4.3 Filters

We propose five filters having various behaviors and properties. They are d-versatile for any d. In this section, $d \geq 2$ denotes the dimension of the space where the oracle points lie (typically, the number of objectives of the problem) and n denotes the total number of points produced by the oracle for all nodes to be filtered at a given level.

4.3.1 Hypervolume-based with a fixed reference point (HVF)

In literature, many multi-objective heuristics are guided by a univariate indicator (Zitzler et al., 2010), and the interest in the hypervolume indicator for this purpose is growing (Auger et al., 2012; Bringmann and Friedrich, 2012). The hypervolume indicator has been introduced in Zitzler and Thiele (1998). Roughly, it computes the hypervolume enclosed between a set of points and an arbitrary reference point. Its success is due to its fundamental properties. In particular, if every point of a given set is dominated by a point from a second set, the hypervolume indicator is lower for the dominated set (Zitzler et al., 2003). The measure can be computed in $\mathrm{O}(n \log n)$ for $d \leq 3$ (Beume et al., 2009) and $\mathrm{O}(n^{d/2} \log n)$ for $d \geq 4$ (Beume, 2009). Note that it has been extended in order to take into account preferences for certain regions of the objective space (Zitzler et al., 2007), which would result in a region-oriented beam search.

The filter described here uses the regular hypervolume indicator. The reference point is determined before starting the beam search, such that it is sure or highly probable that no oracle point, for any node met, will have a coordinate exceeding that of the reference point (during the search, verifications are done and violations never happen in our implementations). At each level of the search tree, the nodes are ranked with respect to the hypervolume of their respective oracle points. At last, the w nodes having the highest hypervolumes are kept.

Proposition 8. *This filter is d-versatile for any $d \geq 2$. It is not Pareto-efficient but it is independent of irrelevant alternatives.* (Proof in Appendix C)

HVF has been chosen for its independence of irrelevant alternatives. We will compare it to HVA, described below, to evaluate the benefits that independence of irrelevant alternatives brings.

4.3.2 Hypervolume-based with a level-adaptive reference point (HVA)

This filter is a variant of the previous one. The only change is the reference point. At each level of the tree, the Nadir point of all oracle points of all nodes is identified and becomes the reference point for the level. The rest of the procedure is identical.

Proposition 9. *This filter is d-versatile for any $d \geq 2$. It is neither Pareto-efficient nor independent of irrelevant alternatives.* (Proof in Appendix C)

Although HVA meets no property at all, it allows us to evaluate benefits of being independent of irrelevant alternatives. Indeed, it has the same structure as HVF except the latter is independent of irrelevant alternatives.

4.3.3 Dominance-based 1 (D1)

Before describing the filter, we present the concept of non-dominated sorting (Deb et al., 2002). It consists in partitioning a set of points into subsets called *fronts*. Each front gathers only points which do not dominate each other. The fronts have ranks: every point of a front f_i is dominated by at least one point of front f_{i-1}. We denote by f the set of fronts. A well known algorithm to compute it is the $\mathrm{O}(dn^2)$ Fast Non Dominated Sort of NSGA-II (Deb et al., 2002). However, provided that the number of points to sort is distinctly higher than the number of objectives, a faster algorithm exists. It achieves a non-dominated sort in $\mathrm{O}(n \log^{d-1} n)$ and outperforms NSGA-II's procedure in practice (Buzdalov and Shalyto, 2014), that is why we use it.

The filter suggested here starts by merging all oracle points from all nodes of the current tree level. Then, they are partitioned by a non-dominated sort. Note that, in general, each front contains points from different nodes and the points of a given node are not all in the same front. Next, the nodes are given a grade depending on the proportion of points they have in the fronts. In detail, the proportion of points in front f_i which belong to node σ^j is

$$q^i_{\sigma^j} = \frac{|f_i \cap g(\sigma^j)|}{|f_i|} \tag{4}$$

and the grade of node σ^j is

$$D1(\sigma^j) = \sum_{i=1}^{|f|} M^{|f|-i} \lceil (M-1) \cdot q^i_{\sigma^j} \rceil, \tag{5}$$

where $(M-1)$ is a number used to transform the first decimal digits of $q^i_{\sigma^j}$ into an integer after rounding. Hence $D1(\sigma^j)$ is an integer value with radix M. We chose $M = 2^{11}$ to cut the grades into 11 buckets of 11 bits stored in 128 bit integers, meaning that only the first 11 fronts are taken into account in practice. It is enough because the first front only is required for Pareto-efficiency, the rest helps to resolve ties. Indeed, the last step of the filter consists in keeping the w nodes having the highest $D1$ values.

Proposition 10. *This filter is d-versatile for any $d \geq 2$. It is Pareto-efficient. It is independent of irrelevant alternatives if and only if w is not lower than the ideal beam width of every submitted node set.* (Proof in Appendix C)

D1 is our first example of a potentially perfect filter. Moreover, if the oracle is constant, the method given in Section 3.2 can be used to derive a perfect filter from D1.

4.3.4 Dominance-based 2 (D2)

This filter is a variant of the previous one. The only change is the meaning of $q^i_{\sigma^j}$. Here, $q^i_{\sigma^j}$ is the proportion of points of node σ^j which belong to front f_i. Formally,

$$q^i_{\sigma^j} \;=\; \frac{\left|f_i \cap g(\sigma^j)\right|}{\left|g(\sigma^j)\right|}\,. \tag{6}$$

The grade of node σ^j arises from Equation (5) but is denoted by $D2(\sigma^j)$. Then, the w nodes having the highest $D2$ values are kept.

Proposition 11. *This filter is d-versatile for any $d \geq 2$. It is Pareto-efficient. It is independent of irrelevant alternatives if and only if w is not lower than the ideal beam width of every submitted node set.* (Proof in Appendix C)

Like D1, D2 is an example of a potentially perfect filter. It can be combined with D1 to create another filter, DS, as described below.

4.3.5 Dominance-based summation (DS)

While D1 selects nodes whose oracle points hold a majority of the first front, D2 can select nodes whose points are a small part of the first front as soon as the first front is the main contribution of the node. DS is a combination of D1 and D2, aimed to balance their decisions.

First, the filter computes

$$DS(\sigma^j) \;=\; D1(\sigma^j) + D2(\sigma^j)\,. \tag{7}$$

Then, the w nodes having the highest DS values are kept.

Proposition 12. *This filter is d-versatile for any $d \geq 2$. It is Pareto-efficient. It is independent of irrelevant alternatives if and only if w is not lower than the ideal beam width of every submitted node set.* (Proof in Appendix C)

Like D1 and D2, DS is an example of a potentially perfect filter. It demonstrates that filters can be combined without loss of property.

5 Experiments

All algorithms are implemented in C++, compiled using gcc 4.4.7 with O3 optimizations and run on an Intel Xeon X5550 processor at 2.66 GHz with 24GB of RAM shared by 8 cores. Oracle PLP is implemented using Cplex 12.5.1. Our beam search is single-threaded but several experiments were run in parallel. All reported CPU times are in seconds. All instances and solution sets can be downloaded from `prolog.univie.ac.at/research/beamsearch/barthelemy-et-al.zip`.

5.1 Problem instances

The BKP instances and exact solution sets that we use are those of Bazgan et al. (2009a), in which weights and profits are uniformly distributed random values. Type A instances are fully random. For Type B, weights are random; profits are random with positive correlation: for each item i, p^1_i and p^2_i are close to each other. For type C, weights are random; profits are random with negative correlation: for each item i, a low p^1_i value implies a high p^2_i value and vice-versa. For Type D, the weight of each item i is random with positive correlation to $p^1_i + p^2_i$ (the weight is close to the sum); profits are random with negative correlation. The capacity W of every instance is half the sum of its item weights. The number of items depend on the type, as column "n" of Table 1 shows.

The BTSPP instances along with exact solution sets have been provided by Stevanato and Filippi. Due to unavailability, they are not the same as the ones presented in their articles (Filippi and Stevanato, 2012, 2013). The edge costs are rounded euclidean distances obtained from uniformly distributed random points in $\{1 \ldots 1000\}^2$. The vertex profits are constant for instances of Type 1, while they are uniformly drawn in $\{1 \ldots 10\}$ for instances of Type 2 and $\{1 \ldots 15\}$ for instances of Type 3. The graph sizes are quite small, as column "n" of Table 2 shows.

5.2 Experimental setting

Each experiment compares a subset of component combinations, meaning that the other components have a default setting. For the BKP, unless specified, the components used by default are: oracle DUB, filter D1, statistical variable selection, infeasibility anticipation. When oracle GLB is forecasting, all solutions that it produces during search are added to the set of solutions found. When oracle DUB is forecasting, we use heuristic GLB to produce feasible solutions at each node that the filter has selected, in order to add them to the set of solutions found. For the BTSPP, unless specified, the components used by default are: oracle MN1T, filter DS, path branching, infeasibility and non-optimality anticipations. All oracles for the BTSPP are able to produce a few feasible solutions, that we catch during the search and add to the set of solutions found. Of course, in all cases, once beam search terminates, only the non-dominated solutions that it found are returned.

As it is often the case with multi-objective optimization, reporting whole solution sets is impractical. In order to assess the quality of a solution set A in comparison to a reference set R, we compute the relative difference of their hypervolumes along with the multiplicative ϵ-indicator (Zitzler et al., 2003). In both cases, the reference set is the exact solution set. The nadir point used to compute hypervolumes is the nadir point of the union between the set produced by our beam search and the exact set. Lower hypervolume relative differences are better and a value of 0% means that the evaluated set is the exact set. For the BTSPP, profits are maximized while cycle costs are minimized, which is incompatible with the original definition of the ϵ-indicator. Therefore, we adapt it as follows:

$$\max_{z_r \in R} \min_{z_a \in A} \max \left\{ \frac{z_a^{cost}}{z_r^{cost}}, \frac{z_r^{prof}}{z_a^{prof}} \right\}.$$

It is the slightest possible adaptation of its initial meaning: here it gives the minimum amount by which our profits have to be multiplied and our costs divided in order for our solution set to dominate all solutions in the optimal set. Whatever the multiplicative ϵ-indicator is, lower values are better and a value of 1 means that the evaluated set is the exact set.

5.3 Combinations of oracles and filters

For each test problem, we suggested various oracles and filters with different properties. We now determine which oracle-filter pairs lead to the best solution sets on average. For that purpose, we test each oracle-filter pair with various beam widths and compare the results. The comparisons are presented in Figure 4 for the BKP and in Figure 5 for the BTSPP. On each figure, CPU time is reported on the x-axis, while the y-axis reports hypervolume indicator values. The scale of each axis is logarithmic. Each curve corresponds to a specific setting, i.e. to an oracle-filter pair. The filter used to get every point is identified by the point style; the oracle used to get every point is identified by the line style of the curve passing through the point. Along a curve, each point corresponds to a certain beam width and its coordinates are the average CPU time and hypervolume indicator value over all instances of the considered problem. From left to right, the beam widths corresponding to the different points are $1, 2, 5, 10, 20, 50, 100, 200, 500, 1000$. Wherever the oracle is PLP, only the first 4 widths are reported due to long running times. Since we are looking for good tradeoffs between CPU time and solution quality, a setting is better than another one if all of the points on its curve dominate the points of the other setting. The slope of the curves is also a criterion: the faster a curve decreases, the more the algorithm is sensitive to beam width variations.

In all cases, the wider the beam the better the hypervolume. It is explained in Section 3.6.

5.3.1 Oracles

It was not clear in Section 4.1.1 which oracle is the best for the BKP, because both DUB and GLB meet leaf-compliance, miss constancy and have opposed natures (DUB is a relaxation over-estimating the objective values; GLB is a heuristic under-estimating the objective values). Figure 4 shows that DUB is as reliable as GLB, and even slightly better. Hence both relaxations and heuristics can be good oracles. We recall that, in both cases, feasible solutions returned by beam search have been produced at each node by algorithm GLB. Hence incorporating separately in beam search the oracle and a generation of solutions is a valuable approach.

For the BTSPP, Figure 5 shows that the best oracle on average is the heuristically computed relaxation MN1T. We recall that it is fast to compute while the infeasibility anticipations make it leaf-compliant. MN1F, also fast to compute and leaf-compliant through the infeasibility anticipations, is the worst oracle. It is likely because it is a very weak relaxation of the problem, violating numerous constraints, so with a very low degree of constancy with our path branching (as explained in Sections 3.1 and 3.3 of the guidelines). As predicted in Section 3.1, oracle PLP is too slow. Its speed is the only reason why it is the worst oracle, because it leads to hypervolumes similar to those obtained with MN1T, which can be explained by its leaf-compliance and a good degree of constancy (predicted in Section 3.1). The similarity indicates that MN1T has a good degree of constancy too. It is not likely that a polynomial-size linear program other than PLP could beat MN1T:

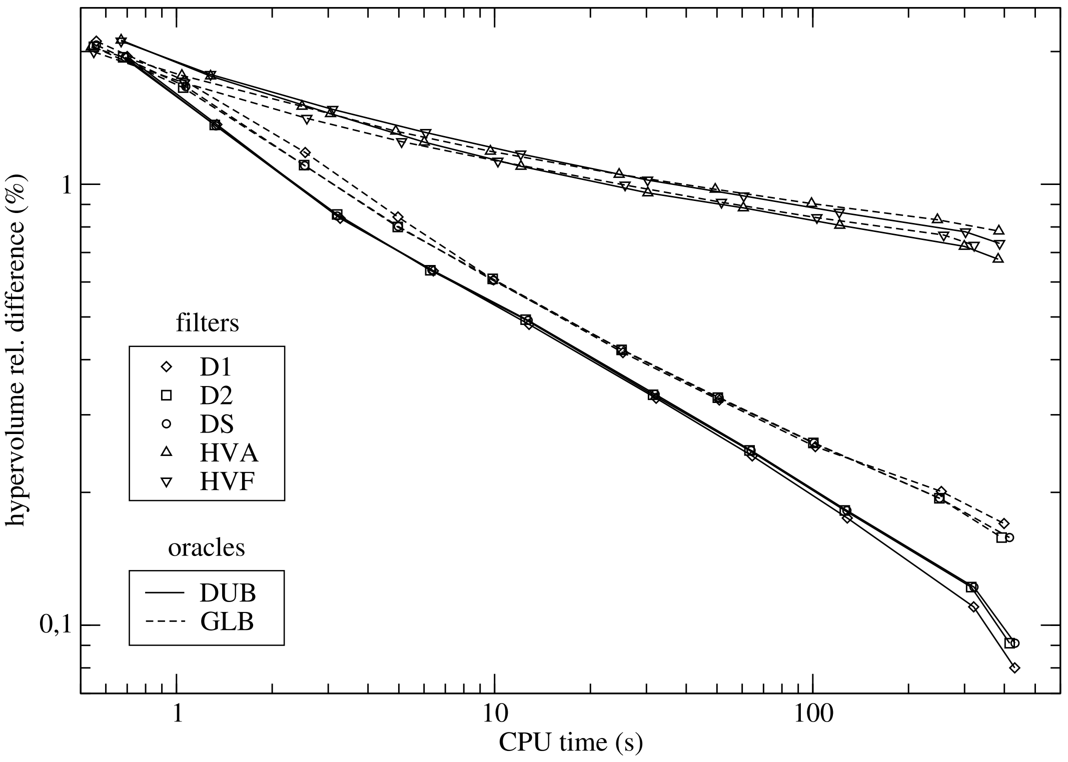

Figure 4: Comparing combinations of oracles and filters for the BKP

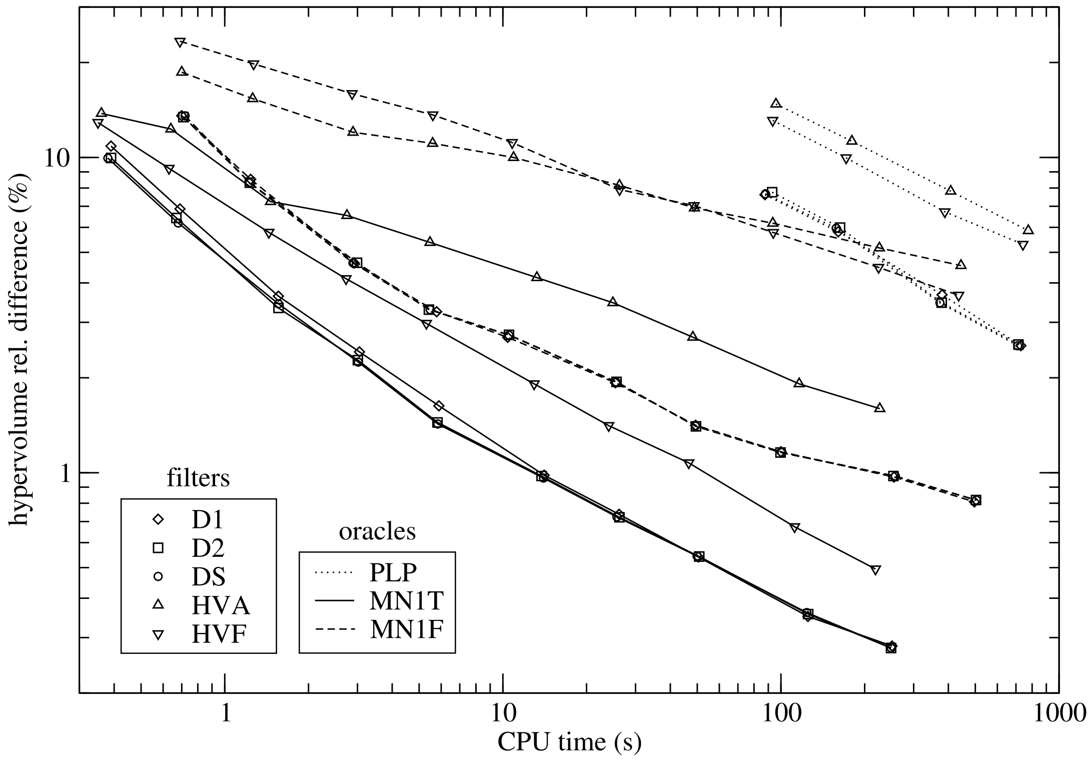

Figure 5: Comparing combinations of oracles and filters for the BTSPP

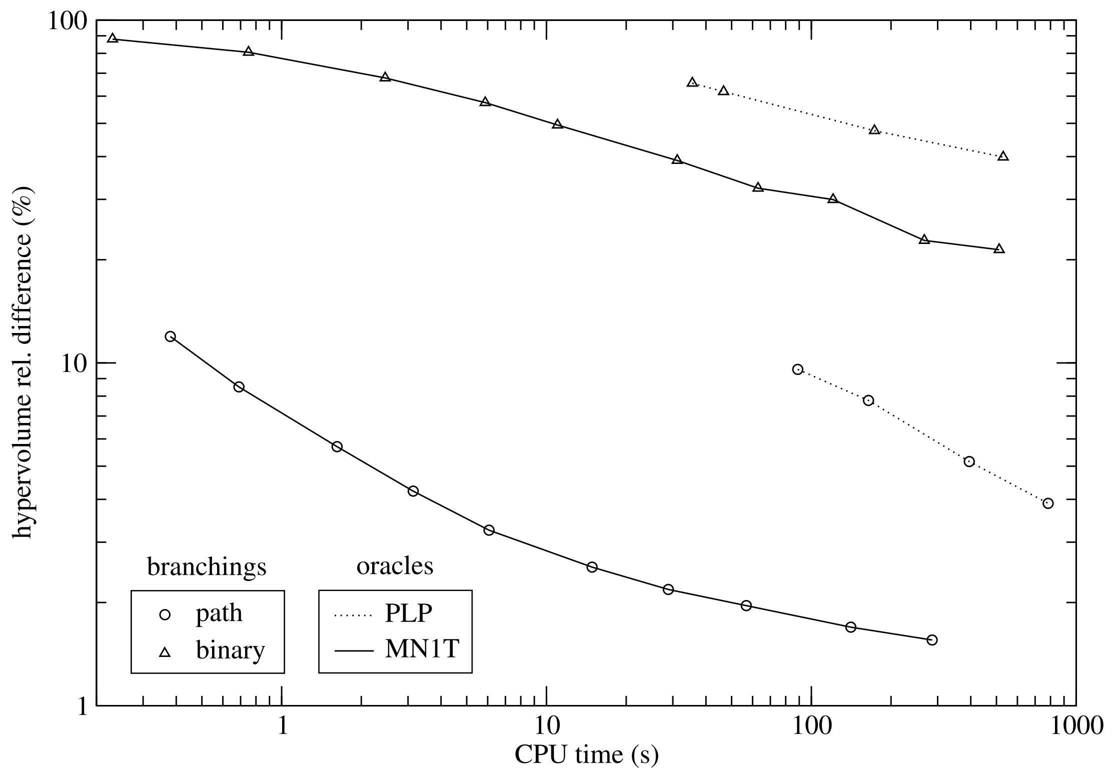

Figure 6: Comparison of two branching schemes for the BTSPP

Roberti and Toth (2012) show experimentally (on the classical single-objective traveling salesman problem) that the linear relaxation of Miller-Tucker-Zemlin's polynomial formulation (from which PLP is derived) ranges among the fastest to compute and its accuracy is of the same order as the accuracy of the other polynomial-size formulations tested (only Dantzig's exponential-size formulation has a clear advantage, but the investigation of how to best incorporate cutting plane generation into a multi-objective branch-and-bound based algorithm is beyond the scope of this paper).

5.3.2 Filters

Over Figures 4 and 5, we notice that filter HVF is better than HVA in 3 cases out of 5. HVF is worse than HVA only once. In the last case, they are incomparable and the curve slopes show that sensitivity to beam width is better with HVF. These observations confirm that independence of irrelevant alternatives is a desirable property, because the only difference between these filters is that HVF is independent of irrelevant alternatives while HVA is not so. However, the best filters are the Pareto-efficient ones: D1, D2 and DS.

5.4 Branching schemes and variable selection strategies

As seen in the guidelines of Sections 3.3 and 3.4, the quality of a beam search algorithm should depend on the branching scheme and the variable selection rule. We now evaluate the performance of the corresponding components designed in Section 4, for both the BKP and the BTSPP.

Figure 6 compares the results obtained for the BTSPP with path branching and binary branching. In order to evaluate their performance under the same conditions, non-optimality anticipations are deactivated (they have been implemented only for path branching). The curves show that path branching provides better results than binary branching. This is true for both MN1T and PLP oracles. As explained in Section 3.3, path branching leads to more constrained problems at a given level of the search tree, which has a positive impact on the behavior of oracles and filters, and, in the case of oracle MN1T, binary branching biases the search toward trees whereas path branching constructs tours and ensures leaf-compliance through infeasibility anticipation.

With the binary branching scheme of the BKP, we compare our two rules for selecting the variable to branch on at each node. Figure 7 shows the results obtained with predefined variable ordering and statistical variable selection. We notice that following a predefined order slows down the search and produces worse solutions than statistical selection, which confirms the expectations of Section 3.4.

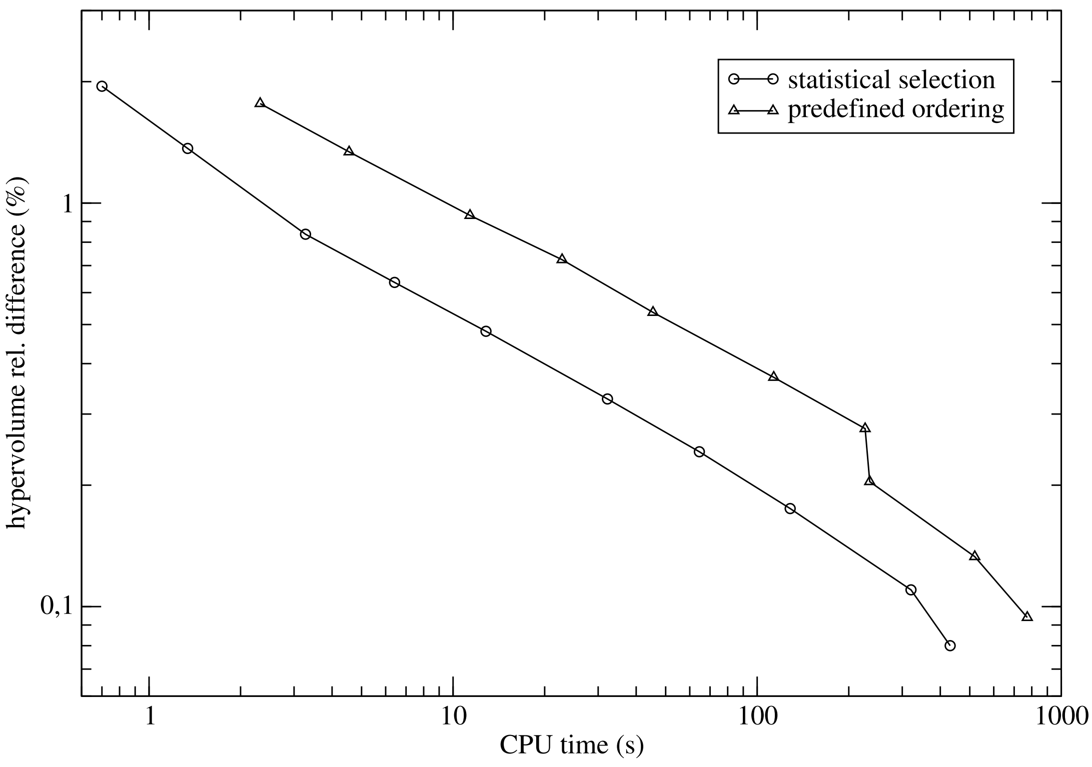

Figure 7: Comparison of two variable selection rules for the binary branching scheme of the BKP

5.5 Infeasibility and non-optimality anticipation

According to the guidelines of Section 3.5, improving partial solutions can help beam search to find good solutions. It is clearly shown by experiments on the BTSPP solver, but the behavior of the BKP solver confirms that exceptions exist. We explain this difference by the fact that the BTSPP has more constraints than the BKP. Constraints are what provides the information which is useful for anticipating non-optimality and infeasibility.

In detail, our infeasibility anticipation for the BKP does not lead to any improvement, as shown in Figure 8 (curves with and without infeasibility anticipation appear to be merged). Non-optimality anticipation slightly increases solving times and, for larger beam widths, it produces slightly worse solutions. In both cases, an explanation is given in the guidelines: the anticipation rules degrade oracle constancy and the degradation is not necessarily balanced by the improvements that they attempt on the partial solutions.

In contrast, Figure 9 reports that combining both types of anticipation for the BTSPP leads to noticeable improvements in solution quality as well as sensitivity to beam width. The highest improvements come from infeasibility anticipation, but non-optimality anticipation improves the hypervolume values significantly, too. This was expected in the guidelines: oracle MN1T relies on a combinatorial relaxation and the accuracy of such bounds is improved by avoiding infeasible variable combinations, which also make it leaf-compliant in our implementation. Non-optimality of partial solutions is corrected by a two-opt operator, which improves the leaves (final solutions) enough such that it balances the thus-lowered degree of constancy.

5.6 Benchmark results

Here, we use our two solvers with the best components that we determined above (for the BKP: oracle DUB, filter D1, statistical variable selection, infeasibility anticipation, solutions generated by GLB; for the BTSPP: oracle MN1T, filter DS, path branching, infeasibility and non-optimality anticipations). All tables provide CPU time and quality indicator values obtained for different beam widths. Each row is devoted to one instance set (instances having the same type and size). For the BKP, each value in a given row is an average over the 10 instances of the set while the BTSPP values assess 3 instances.

The aim of Tables 1 and 2 is to give an overview of our approach and to allow future comparison. Overall, our implementations work well even for short solving times. The BKP beam search produces very good solutions according to ϵ-indicators and hypervolumes. The BTSPP beam search produces good solutions on average according to the hypervolumes, while the ϵ-indicator values, although satisfying, indicate that a few solutions are farther from the Pareto-optimal solution set. Unfortunately, neither the instances, the solution sets nor the code used by the two non-exact methods existing for the BTSPP (Jozefowiez et al., 2008; Bérubé et al., 2009) are available.

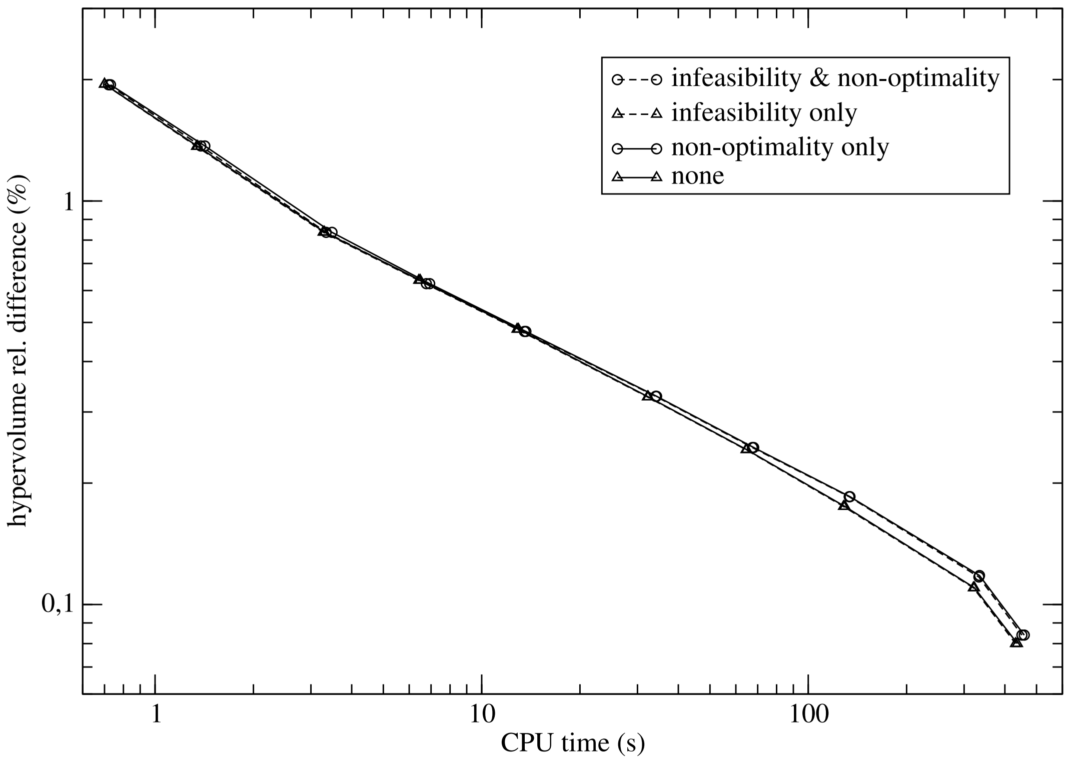

Figure 8: Effect of our infeasibility and non-optimality anticipations for the BKP

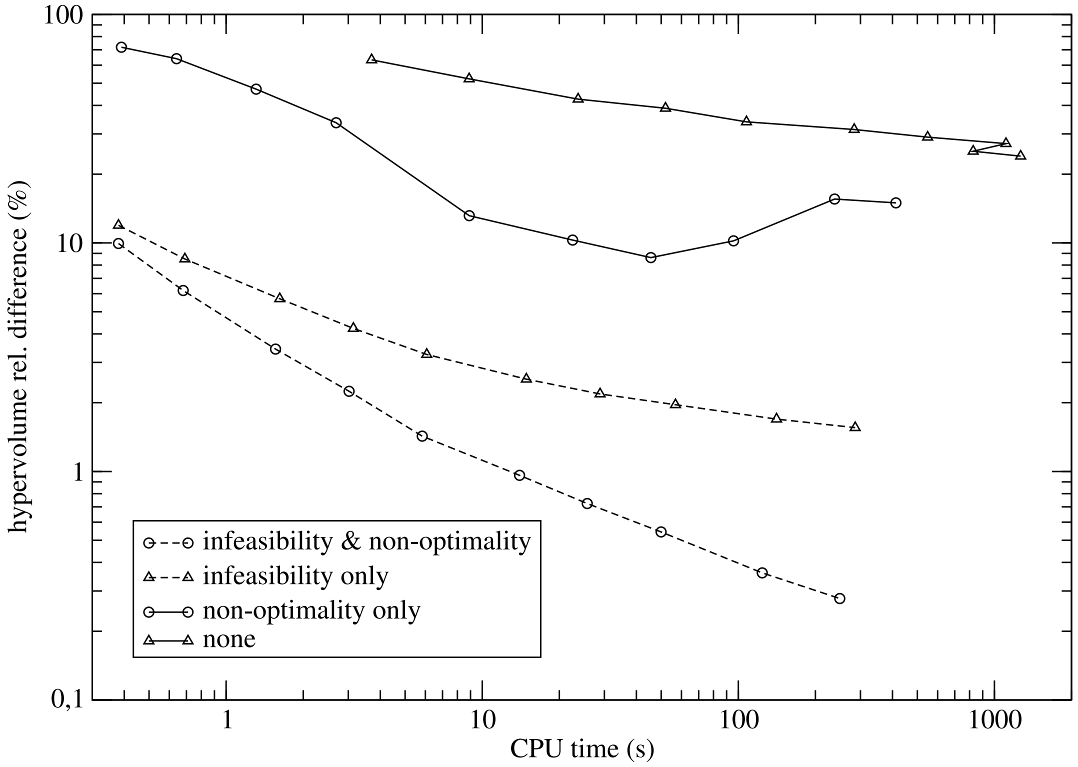

Figure 9: Effect of our infeasibility and non-optimality anticipations for the BTSPP

		$w = 1$			$w = 5$			$w = 10$			$w = 50$			$w = 100$		
type	n	time	ϵ	hv%	time	ϵ	hv%	time	ϵ	hv%	time	ϵ	hv%	time	ϵ	hv%
	100	0.1	1.0080	2.76	0.1	1.0043	1.01	0.1	1.0036	0.68	0.3	1.0024	0.23	0.6	1.0020	0.15
	200	0.1	1.0046	1.43	0.2	1.0025	0.56	0.4	1.0018	0.39	1.9	1.0011	0.17	3.6	1.0010	0.11
	300	0.2	1.0039	1.05	0.7	1.0016	0.36	1.3	1.0012	0.26	5.9	1.0007	0.12	11.6	1.0006	0.08
A	400	0.3	1.0029	0.82	1.4	1.0011	0.27	2.7	1.0009	0.20	12.7	1.0005	0.09	25.3	1.0004	0.06
	500	0.6	1.0027	0.67	2.6	1.0009	0.22	5.1	1.0006	0.16	24.2	1.0004	0.07	48.1	1.0003	0.05
	600	0.9	1.0021	0.58	4.3	1.0007	0.18	8.4	1.0005	0.13	41.4	1.0003	0.07	82.4	1.0003	0.05
	700	1.4	1.0021	0.49	6.9	1.0006	0.16	13.7	1.0005	0.12	67.0	1.0003	0.06	132.8	1.0002	0.04
	1000	0.4	1.0002	5.45	1.9	1.0001	2.39	3.6	1.0001	1.63	18.5	1.0001	0.78	38.2	1.0000	0.54
B	2000	1.9	1.0001	2.74	9.5	1.0001	1.06	18.9	1.0000	0.74	97.3	1.0000	0.37	196.6	1.0000	0.27
	3000	5.4	1.0001	1.76	26.3	1.0000	0.66	52.0	1.0000	0.48	267.2	1.0000	0.24	537.0	1.0000	0.18
	4000	11.3	1.0001	1.39	57.3	1.0000	0.52	114.3	1.0000	0.39	574.8	1.0000	0.21	1150.7	1.0000	0.17
	100	0.1	1.0152	1.92	0.1	1.0083	0.87	0.2	1.0065	0.63	0.5	1.0052	0.32	1.0	1.0041	0.24
	200	0.1	1.0092	1.32	0.4	1.0055	0.57	0.7	1.0046	0.46	3.3	1.0033	0.27	6.6	1.0028	0.22
C	300	0.3	1.0069	0.93	1.1	1.0036	0.40	2.2	1.0031	0.33	10.4	1.0025	0.20	20.8	1.0022	0.17
	400	0.5	1.0056	0.79	2.5	1.0028	0.33	4.8	1.0025	0.27	23.3	1.0019	0.17	46.5	1.0017	0.13
	500	1.0	1.0047	0.65	4.8	1.0023	0.26	9.3	1.0020	0.22	46.1	1.0015	0.13	91.7	1.0014	0.11
	100	0.1	1.0222	2.93	0.1	1.0138	1.58	0.2	1.0121	1.32	0.8	1.0094	0.81	1.6	1.0083	0.65
D	150	0.1	1.0177	2.34	0.3	1.0097	1.16	0.5	1.0086	0.97	2.4	1.0066	0.60	4.8	1.0060	0.47
	200	0.2	1.0131	1.75	0.6	1.0073	0.92	1.1	1.0068	0.77	5.2	1.0050	0.46	10.2	1.0045	0.37
	250	0.2	1.0112	1.44	1.0	1.0059	0.70	2.0	1.0052	0.58	9.6	1.0042	0.37	19.3	1.0036	0.29

Table 1: Average computation times in seconds, epsilon indicators (ϵ) and hypervolume percentage gaps (hv%) from our BKP solver.

		$w = 1$			$w = 5$			$w = 10$			$w = 50$			$w = 100$		
type	n	time	ϵ	hv%	time	ϵ	hv%	time	ϵ	hv%	time	ϵ	hv%	time	ϵ	hv%
	15	0.1	1.3142	16.47	0.1	1.0437	0.96	0.1	1.0397	0.63	0.2	1.0178	0.25	0.4	1.0178	0.25
1	25	0.1	1.1326	9.31	0.2	1.0919	2.07	0.4	1.0715	1.41	1.8	1.0425	0.50	3.3	1.0425	0.46
	35	0.2	1.2082	7.56	0.8	1.0751	2.88	1.7	1.0661	1.69	7.1	1.0488	0.40	13.4	1.0417	0.31
	50	0.9	1.2014	8.13	3.8	1.1213	4.25	7.2	1.0964	2.47	34.8	1.0740	1.58	65.1	1.0723	1.27
	15	0.0	1.2303	10.89	0.1	1.0871	1.59	0.1	1.0438	0.55	0.2	1.0115	0.17	0.5	1.0076	0.15
2	25	0.1	1.2722	10.57	0.3	1.0907	4.05	0.5	1.0884	3.55	1.8	1.0650	1.28	3.6	1.0537	0.85
	35	0.3	1.2569	8.29	1.1	1.1367	4.39	2.1	1.0942	3.70	9.4	1.0610	1.97	17.0	1.0485	1.55
	50	1.3	1.1677	9.16	5.6	1.1191	5.19	11.3	1.1021	4.39	51.8	1.0567	1.63	94.9	1.0517	1.28
	15	0.0	1.2265	5.75	0.1	1.0603	1.09	0.1	1.0393	0.45	0.2	1.0064	0.02	0.4	1.0047	0.01
3	25	0.1	1.2516	12.45	0.3	1.0706	3.10	0.5	1.0593	2.10	2.3	1.0455	0.93	4.0	1.0404	0.65
	35	0.3	1.2537	11.68	1.1	1.1404	6.52	2.1	1.0972	2.70	9.2	1.0378	0.80	17.6	1.0378	0.54
	50	1.2	1.2406	9.10	5.3	1.1038	5.09	10.2	1.0827	3.28	49.1	1.0453	2.00	88.2	1.0503	1.36

Table 2: Average computation times in seconds, epsilon indicators (ϵ) and hypervolume percentage gaps (hv%) from our BTSPP solver.

Tables 3 and 4 compare our beam search for the BKP to, respectively, the beam search of Ponte et al. (2012) and the approximation scheme of Bazgan et al. (2009b). Ponte et al. (2012) run their beam search on an AMD Phenom II X6 3.2 GHz. Various benchmarks[2] report that the performances of this processor are equivalent to our Intel Xeon X5550 2.66 GHz, so we compare directly our solving times to theirs in Table 3. Regarding Table 4, Bazgan et al. (2009b) run their approximation scheme on some Intel Xeon 3.4 GHz. Since their approximation scheme is derived from their exact algorithm (Bazgan et al., 2009a) that they test on the same processor and that one of the authors also runs on an AMD Phenom II X6 3.2 GHz (Figueira et al., 2013), we could calculate a speed ratio between these processors for each instance type and size, and we kept the most unfavorable for us ($\times 1.524$) in order to report fairly the times of Bazgan et al. (2009b) in Table 4.

In Tables 3 and 4, for each experiment done by Ponte et al. (2012) and Bazgan et al. (2009b), we run our beam search with a beam width producing an ϵ-indicator value not-worse than theirs: all our solution sets are at least as good as (in 50 cases out of 54, better than) their solution sets. In 43 cases (bold-faced in the tables) out of 54, our implementation is faster. In 5 other cases (also bold-faced), the running time is the same. In the 6 cases left, our durations are not much longer whereas our solutions are considerably better. To conclude, for all instance structures, our beam search is overall more efficient than the two existing non-exact approaches. The efficiency gap is considerable when the instances are large or the targeted quality is high.

6 Conclusions

We provided a paradigm to understand, analyze and design beam search algorithms for integer multi-objective optimization. The paradigm, suitable for linear as well as non-linear problems, decomposes beam search into

[2]cpuboss.com, cpubenchmark.net

		Ponte BS		our BS		Ponte BS		our BS		Ponte BS		our BS		Ponte BS		our BS	
type	n	time	ϵ	time	ϵ	time	ϵ	time	ϵ	time	ϵ	time	ϵ	time	ϵ	time	ϵ
	100	0.0	1.1000	0.1	1.0036	0.1	1.0100	**0.1**	**1.0036**								
A	300	0.1	1.1000	0.2	1.0039	1.1	1.0100	**0.2**	**1.0039**	11.0	1.0010	**2.4**	**1.0009**				
	500	0.3	1.1000	0.6	1.0027	2.9	1.0100	**0.6**	**1.0027**	28.0	1.0010	**2.6**	**1.0009**				
	700	0.7	1.1000	1.4	1.0021	9.1	1.0100	**1.4**	**1.0021**	80.3	1.0010	**6.9**	**1.0006**				
	1000	0.4	1.1000	**0.4**	**1.0002**	0.4	1.0100	**0.4**	**1.0002**	1.4	1.0010	**0.4**	**1.0002**	3.7	1.0001	**1.9**	**1.0001**
B	2000	2.5	1.1000	**1.9**	**1.0001**	2.7	1.0100	**1.9**	**1.0001**	9.1	1.0010	**1.9**	**1.0001**	41.0	1.0001	**1.9**	**1.0001**
	3000	6.8	1.1000	**5.4**	**1.0001**	7.4	1.0100	**5.4**	**1.0001**	25.0	1.0010	**5.4**	**1.0001**	159.6	1.0001	**5.4**	**1.0001**
	4000	14.2	1.1000	**11.3**	**1.0001**	15.6	1.0100	**11.3**	**1.0001**	55.4	1.0010	**11.3**	**1.0001**	411.6	1.0001	**11.3**	**1.0001**
	100	0.0	1.1000	0.1	1.0083	0.9	1.0100	**0.1**	**1.0083**								
	200	0.2	1.1000	**0.1**	**1.0092**	4.9	1.0100	**0.1**	**1.0092**								
C	300	0.3	1.1000	**0.3**	**1.0069**	12.7	1.0100	**0.3**	**1.0069**								
	400	0.6	1.1000	**0.5**	**1.0056**	26.6	1.0100	**0.5**	**1.0056**								
	500	1.1	1.1000	**1.0**	**1.0047**	75.0	1.0100	**1.0**	**1.0047**								
	100	0.2	1.1000	**0.1**	**1.0138**												
D	150	0.6	1.1000	**0.1**	**1.0138**	58.2	1.0100	**0.3**	**1.0097**								
	200	1.1	1.1000	**0.2**	**1.0131**	21.7	1.0100	**0.6**	**1.0073**								

Table 3: Comparison of our beam search for the BKP to the beam search of Ponte et al. (2012). In all cases, our solutions are at least as good as (mostly better than) those of Ponte et al.. When our solver is as fast or faster, the result is in bold.

		Bazgan appr.		our BS	
type	n	time*	ϵ	time	ϵ
	100	0.1	1.0159	**0.1**	**1.0036**
A	400	9.3	1.0076	**0.3**	**1.0029**
	700	49.2	1.0060	**1.4**	**1.0021**
	1000	0.2	1.0041	0.4	1.0002
B	2000	2.2	1.0028	**1.9**	**1.0001**
	4000	17.1	1.0023	**11.3**	**1.0001**
	100	0.3	1.0178	**0.1**	**1.0083**
C	300	15.4	1.0096	**0.3**	**1.0069**
	500	67.6	1.0064	**1.0**	**1.0047**
	100	3.6	1.0183	**0.1**	**1.0138**
D	200	55.2	1.0117	**0.3**	**1.0104**
	250	96.0	1.0098	**0.4**	**1.0087**

* scaled (see Section 5.6)

Table 4: Comparison of our beam search for the BKP to the approximation scheme of Bazgan et al. (2009b). In all cases, our solutions are better than those of Bazgan et al.. When our solver is as fast or faster, the result is in bold.

components whose properties and mutual influences are identified. In particular, we showed incompatibilities between properties of the key components (node evaluation and node selection, respectively implemented by the *oracle* and the *filter*). Besides understanding deeply why they cannot be perfect, it allowed us to derive guidelines. We studied popular ingredients such as linear relaxation, hypervolume indicator, binary branching scheme and predefined variable enumeration orders, but we showed that less common approaches are valuable.

The resulting beam search solver for the BKP outperforms both the beam search of Ponte et al. (2012) and the approximation scheme of Bazgan et al. (2009b). Our beam search solver for the BTSSP cannot be compared to the existing non-exact methods (Jozefowiez et al., 2008; Bérubé et al., 2009) due to data unavailability but it shows good results.

Further work should focus on finding ways to improve the properties of oracles and filters. The four sufficient properties that we introduced for beam search to find optimal solutions open prospects for decomposing them into more elementary conditions. That would bring richer guidance for designing beam search procedures, and perhaps the incompatibilities that we studied would turn out to lie between minor properties. At last, the beam width w is traditionally chosen by the user and constant over the search process. A way of investigation could consist in the automatic adjustments of w while progressing in the search tree.

Acknowledgments

This work has been funded by the project Austrian Science Fund (FWF): P23589-N13. The heavy computations achieved for our experiments would not have been possible without the Vienna Scientific Cluster. We thank Elisa Stevanato and Carlo Filippi for having kindly provided their BTSPP instances and exact solution sets.

Proofs

All theorems and propositions stated along the article are proven in Sections B and C. They rely on the lemmas of Section A. Through the appendix, the notation observes the conventions of Section 2.1. In addition, we denote the root node of the search tree by σ_0. The set of all leaves in the whole tree is denoted by σ_L^s.

A Prerequisites

Lemma 1. *Let g and z be two oracles such that equivalence of weak dominance relations $g\left(\sigma^i\right) \preceq g\left(\sigma^j\right) \Leftrightarrow z\left(\sigma^i\right) \preceq z\left(\sigma^j\right)$ holds for any pair of leaves (σ^i, σ^j) in a given leaf-set. Then, equivalence of equality relations $g\left(\sigma^i\right) = g\left(\sigma^j\right) \Leftrightarrow z\left(\sigma^i\right) = z\left(\sigma^j\right)$ holds and equivalence of strong dominance relations $g\left(\sigma^i\right) \prec g\left(\sigma^j\right) \Leftrightarrow z\left(\sigma^i\right) \prec z\left(\sigma^j\right)$ holds and equivalence of incomparability relations $g\left(\sigma^i\right) \,||\, g\left(\sigma^j\right) \Leftrightarrow z\left(\sigma^i\right) \,||\, z\left(\sigma^j\right)$ holds.*

Proof. For each relation, we prove an implication only, but exchanging g with z achieves the equivalence. Consider a pair (σ^i, σ^j) such that $g\left(\sigma^i\right) = g\left(\sigma^j\right)$. Then $g\left(\sigma^i\right) \preceq g\left(\sigma^j\right)$ and $g\left(\sigma^j\right) \preceq g\left(\sigma^i\right)$, which implies $z\left(\sigma^i\right) \preceq z\left(\sigma^j\right)$ and $z\left(\sigma^j\right) \preceq z\left(\sigma^i\right)$. So $z\left(\sigma^i\right) = z\left(\sigma^j\right)$ holds. Consider a pair (σ^i, σ^j) such that $g\left(\sigma^i\right) \prec g\left(\sigma^j\right)$. Then $g\left(\sigma^i\right) \preceq g\left(\sigma^j\right)$, which implies $z\left(\sigma^i\right) \preceq z\left(\sigma^j\right)$. Let us assume that $z\left(\sigma^i\right) = z\left(\sigma^j\right)$. Then $g\left(\sigma^i\right) = g\left(\sigma^j\right)$, thus contradicting $g\left(\sigma^i\right) \prec g\left(\sigma^j\right)$. So $z\left(\sigma^i\right) \prec z\left(\sigma^j\right)$ holds. Consider a pair (σ^i, σ^j) such that $g\left(\sigma^i\right) \,||\, g\left(\sigma^j\right)$. Then $g\left(\sigma^i\right) \preceq g\left(\sigma^j\right)$ is false, which implies that $z\left(\sigma^i\right) \preceq z\left(\sigma^j\right)$ is false. Let us assume that $z\left(\sigma^j\right) \prec z\left(\sigma^i\right)$. Then $g\left(\sigma^j\right) \prec g\left(\sigma^i\right)$, thus contradicting $g\left(\sigma^i\right) \,||\, g\left(\sigma^j\right)$. So $z\left(\sigma^j\right) \,||\, z\left(\sigma^i\right)$ holds. □

Lemma 2. *Let $R(\sigma_0) \subseteq \sigma_L^s$ be the set of Pareto-optimal solutions. Let $\sigma^g \subseteq \sigma_L^s$ be the set of solutions whose oracle points define $P \circ g(\sigma_L^s)$, where g is a leaf-compliant oracle. Then $\sigma^g = R(\sigma_0)$.*

Proof. Let h be an oracle. By definition, for any solution $a \in \sigma_L^s$, $h(a) \in P \circ h(\sigma_L^s)$ if and only if, for every $b \in \sigma_L^s$, either $h(a) = h(b)$, $h(a) \prec h(b)$ or $h(a) || h(b)$. Let z be as defined in Section 2.1. Due to leaf-compliance of g, the condition of Lemma 1 is fulfilled. So, given (a, b), one of the three dominance relations holds for $h = z$ if and only if it holds for $h = g$. Therefore, the solutions defining $P \circ z(\sigma_l^s)$ are the same as those defining $P \circ g(\sigma_l^s)$. □

Lemma 3. *Let σ_1 and σ_2 be two sets of nodes. Then $P \circ g(\sigma_1 \cup \sigma_2) = P\big(P \circ g(\sigma_1) \cup g(\sigma_2)\big)$, where g is an arbitrary oracle.*

Proof. $P\big(g(\sigma_1 \cup \sigma_2)\big)$ equals $P\big(g(\sigma_1) \cup g(\sigma_2)\big)$ by definition of an oracle subject to a node set. Properties of Pareto sets make it equal $P\big(P \circ g(\sigma_1) \cup g(\sigma_2)\big)$. □

Lemma 4. *Let σ_l be a set of nodes and σ_l^s a set of leaves, both part of the whole tree. Let us branch on the nodes of σ_l. The set of leaves arising from branching is σ. The set of nodes arising from branching (without leaves) is σ_{l+1}. The new set of leaves is $\sigma_{l+1}^s = \sigma_l^s \cup \sigma$. Then, $P \circ g(\sigma_l \cup \sigma_l^s) = P \circ g(\sigma_{l+1} \cup \sigma_{l+1}^s)$ for any (σ_l, σ_l^s) if and only if g is a constant oracle.*

Proof. By oracle constancy, $P \circ g(\sigma_l) = P \circ g(\sigma_{l+1} \cup \sigma)$. Then, $P \circ g(\sigma_l) \cup g(\sigma_l^s) = P \circ g(\sigma_{l+1} \cup \sigma) \cup g(\sigma_l^s)$ so $P\big(P \circ g(\sigma_l) \cup g(\sigma_l^s)\big) = P\big(P \circ g(\sigma_{l+1} \cup \sigma) \cup g(\sigma_l^s)\big)$. Through Lemma 3, we obtain $P \circ g(\sigma_l \cup \sigma_l^s) = P \circ g(\sigma_{l+1} \cup \sigma \cup \sigma_l^s)$. Lack of constancy means that there exists σ_l containing a node such that $P \circ g(\sigma_l) \neq P \circ g(\sigma_{l+1} \cup \sigma)$. □

Lemma 5. *If g is a constant oracle, then $P \circ g(\sigma_0) = P \circ g(\sigma_L^s)$.*

Proof. Let us consider an exhaustive branching scheme and its whole tree. At any level l, we denote the set of nodes without leaves by σ_l and the set of leaves met so far by σ_l^s. By iterating Lemma 4 from level 0 to the last level L, $P \circ g\big(\sigma_0 \cup \sigma_0^s\big) = P \circ g(\sigma_L \cup \sigma_L^s)$ holds. By construction, σ_0^s and σ_L are empty. □

Lemma 6. *Let g be an arbitrary oracle and F be a Pareto-efficient filter. Let σ be a node set whose ideal beam width is w^*. Define arbitrarily $\sigma'' \subseteq \sigma$. Let us denote $m = |\sigma'' \cap F_{w^*}^g(\sigma)|$ and $n = |P \circ g(\sigma'') \cap P \circ g(\sigma)|$. Then, $n \geq m$. Moreover, $n = 0$ if and only if $m = 0$.*

Proof. By definitions of Pareto-efficiency and ideal beam width, every node in $F_{w^*}^g(\sigma)$ has at least one oracle point in $P \circ g(\sigma)$. So every node in $\sigma'' \cap F_{w^*}^g(\sigma)$ has at least one oracle point in $P \circ g(\sigma)$, which is formally written $m \leq \big|\cup_{\sigma^i \in \sigma'' \cap F_{w^*}^g(\sigma)} \big(g(\sigma^i) \cap P \circ g(\sigma)\big)\big|$. The right-hand side is lower than or equal to $\big|\cup_{\sigma^i \in \sigma''} \big(g(\sigma^i) \cap P \circ g(\sigma)\big)\big|$, which is $|g(\sigma'') \cap P \circ g(\sigma)|$ by factoring. Due to properties of Pareto sets, no point in $g(\sigma'') \backslash P \circ g(\sigma'')$ is in $P \circ g(\sigma)$, so the latter cardinality is $|P \circ g(\sigma'') \cap P \circ g(\sigma)|$. Hence $m \leq n$, from which $n = 0 \Rightarrow m = 0$ arises. To prove the converse, note that equation $n > 0$ means $\exists \sigma^i \in \sigma''$ such that $P \circ g(\sigma^i) \cap P \circ g(\sigma) \neq \varnothing$. Since $F_{w^*}^g(\sigma)$ gathers every node having a point in $P \circ g(\sigma)$, $\sigma^i \in F_{w^*}^g(\sigma)$ holds so $m > 0$. □

B Proofs of the theorems

B.1 Proof of Theorem 1

We denote the set of Pareto-optimal solutions by $R(\sigma_0) \subseteq \sigma_L^s$. Let σ_l be the set of nodes of the whole tree at level l and σ_l^s be the set of leaves existing in the whole tree up to level l. Let $\sigma_l' \subseteq \sigma_l$ be the nodes submitted to the filter in beam search and $\sigma_l'^s$ be the set of leaves that it found so far. Let w^* be the ideal beam width of $\sigma = \sigma_l \cup \sigma_l^s$. The last level of the whole tree is L. By construction, $\sigma_L'^s$ is the set of solutions found by beam search once it has finished and $\sigma_L'^s \cap R(\sigma_0)$ is the set of Pareto-optimal solutions found.

First, note that applying Lemma 4 to $P \circ g(\sigma_l \cup \sigma_l^s)$ and iterating until the end of the whole tree gives $P \circ g(\sigma_L^s) = P \circ g(\sigma)$ because $\sigma_L = \varnothing$ by construction. In Property 5, two cases are possible. Let us denote $W = \min\{w, |R(\sigma_0)|\}$. In the first case, $|F_w^g(\sigma_l') \cap F_{w^*}^g(\sigma)| = w \geq W$, so $\big|\big(F_w^g(\sigma_l') \cup \sigma_l'^s\big) \cap F_{w^*}^g(\sigma)\big| \geq W$. Lemma 6 implies $\big|P \circ g\big(F_w^g(\sigma_l') \cup \sigma_l'^s\big) \cap P \circ g(\sigma_L^s)\big| \geq W$ independently of any assumption. The second case of Property 5 states $|\sigma_l' \cap F_{w^*}^g(\sigma)| - |F_w^g(\sigma_l') \cap F_{w^*}^g(\sigma)| = 0$. A few set algebra gives $|\sigma_l' \backslash F_w^g(\sigma_l') \cap F_{w^*}^g(\sigma)| = 0$. Lemma 6 implies $\big|P \circ g\big(\sigma_l' \backslash F_w^g(\sigma_l')\big) \cap P \circ g(\sigma_L^s)\big| = 0$. It means that the nodes rejected by the filter have no oracle point in $P \circ g(\sigma_L^s)$. Therefore, here again, $\forall l \in \{1 \dots L\}$ assuming $|P \circ g(\sigma_l' \cup \sigma_l'^s) \cap P \circ g(\sigma_L^s)| \geq W$ implies $\big|P \circ g\big(F_w^g(\sigma_l') \cup \sigma_l'^s\big) \cap P \circ g(\sigma_L^s)\big| \geq W$.

According to Lemma 4, $P \circ g\big(F_w^g(\sigma_{l-1}') \cup \sigma_{l-1}'^s\big) = P \circ g(\sigma_l' \cup \sigma_l'^s)$, so assuming $\big|P \circ g\big(F_w^g(\sigma_{l-1}') \cup \sigma_{l-1}'^s\big) \cap P \circ g(\sigma_L^s)\big| \geq W$ implies $\big|P \circ g\big(F_w^g(\sigma_l') \cup \sigma_l'^s\big) \cap P \circ g(\sigma_L^s)\big| \geq W$. According to Lemma 2, $|P \circ g(\sigma_L^s)| = |R(\sigma_0)|$. Through Lemma 5, that implies $|P \circ g(\sigma_0) \cap P \circ g(\sigma_L^s)| \geq |R(\sigma_0)|$. At the tree root, $l = 0$, $\sigma_0 = F_w^g(\sigma_l') \cup \sigma_l'^s$ because the filter rejects no node and $\sigma_0'^s = \varnothing$ by construction. Therefore, the assumption holds at level 1 and, by induction, the implication holds at the last tree level. There, it can be written $|P \circ g(\sigma_L'^s) \cap P \circ g(\sigma_L^s)| \geq W$ because $\sigma_L' = \varnothing$ by construction. That implies $|g(\sigma_L'^s) \cap P \circ g(\sigma_L^s)| \geq W$. Since the latter considers leaf sets and since the oracle gives exactly one point per leaf, there is a bijection between $\sigma_L'^s$ and $g(\sigma_L'^s)$ as well as between σ^g and $P \circ g(\sigma_L^s)$, where $\sigma^g \subseteq \sigma_L^s$ is the set of leaves whose points define $P \circ g(\sigma_L^s)$. Hence $|\sigma_L'^s \cap \sigma^g| \geq W$. Therefore, $|\sigma_L'^s \cap R(\sigma_0)| \geq W$ due to Lemma 2.

B.2 Proof of Theorem 2

If beam search is not polynomial in the problem size then there exists a perfect oracle: the exact solver. The converse could be straightforwardly proven if oracles had to compute meaningful decision variable values, but it is not part of their definition. Here is a way without such a hypothesis.

The oracle is perfect so a perfect filter can be used according to Theorem 3. Iterate the following and count iterations with i: perform beam search with $w = 2^i$. Repeat until no new solution is found. According to Theorem 1, this procedure stops when all Pareto-optimal solutions are found.

By contradiction, assume that beam search is polynomial in the problem size n and it calls the oracle at least once, so the oracle is polynomial in n, so the number of Pareto-optimal solutions to the considered problem is polynomial (or sub-polynomial) in n. The latter is due to $|P \circ g(\sigma_0)| = |R(\sigma_0)|$, obtained by combining Lemmas 2 and 5 and noting that $|\sigma^g| = |P \circ g(\sigma_L^S)|$. Therefore, the beam width reached by the procedure above when it stops is polynomial in n. Then, the overall complexity of the procedure is polynomial in n, so all Pareto-optimal solutions to the problem are found in polynomial time. Since the problem is NP-hard (see the introduction of Section 2), that implies P=NP.

B.3 Proof of Theorem 3

Let σ_l be the set of nodes of the whole tree at level l and σ_l^S be the set of leaves existing in the whole tree up to level l. Let $\sigma'_l \subseteq \sigma_l$ be the nodes to be filtered in beam search and $\sigma_l'^S$ be the set of leaves that it found so far. $\sigma_l'^D$ gathers the nodes discarded by the filter so far. We denote the ideal beam width of σ'_l by $w_l'^*$. Let us denote $\sigma = \sigma_l \cup \sigma_l^S$ and w^* be its ideal beam width. Let us denote $\sigma' = \sigma'_l \cup \sigma_l'^S \cup \sigma_{l-1}'^D$ and w'^* be its ideal beam width.

If w is never lower than the ideal beam width of σ'_l

Let F be a Pareto-efficient filter. By definition, $P \circ g(\sigma'_l) = P \circ g\big(F_w^g(\sigma'_l)\big)$. That implies $P \circ g(\sigma'_l) \backslash P \circ g\big(F_w^g(\sigma'_l)\big) = \varnothing$, meaning that every oracle point from the nodes rejected by the filter is dominated by points in $P \circ g(\sigma'_l)$. That implies $g\big(\sigma'_l \backslash F_w^g(\sigma'_l)\big) \cap P \circ g(\sigma'_l) = \varnothing$, so $\big|P \circ g\big(\sigma'_l \backslash F_w^g(\sigma'_l)\big) \cap P \circ g(\sigma)\big| = 0$. According to Lemma 6, $|\sigma'_l \backslash F_w^g(\sigma'_l) \cap F_{w^*}^g(\sigma)| = 0$ holds. By considering that $F_w^g(\sigma'_l) \subseteq \sigma'_l$, a few set algebra gives $|\sigma'_l \cap F_{w^*}^g(\sigma)| = |F_w^g(\sigma'_l) \cap F_{w^*}^g(\sigma)|$. By definition $|F_w^g(\sigma'_l)| = w$, so $|F_w^g(\sigma'_l) \cap F_{w^*}^g(\sigma)| \leq w$. Therefore, the equation of Property 5 is met.

If the oracle is constant

We build a perfect filter F for the considered d-objective integer optimization problem. Let f be a d-versatile Pareto-efficient filter. Let us denote $\sigma'' = \sigma'_l \cap f_{w'^*}^g(\sigma')$ and w' a beam width such that $\big|\sigma'' \cup \big(f_{w'}^g(\sigma'_l) \backslash \sigma''\big)\big| = \min\{w, |\sigma'_l|\}$. Define

$$F_w^g(\sigma'_l) = \begin{cases} f_w^g(\sigma'') & \text{if } w \leq |\sigma''| \\ \sigma'' \cup \big(f_{w'}^g(\sigma'_l) \backslash \sigma''\big) & \text{otherwise} \end{cases}$$

We omit the proof that F is Pareto-efficient, the following shows independence of irrelevant alternatives. Note that $\sigma'' = \sigma'' \cap f_{w'^*}^g(\sigma')$ by construction and $f_w^g(\sigma'') = f_w^g(\sigma'') \cap f_{w'^*}^g(\sigma')$ because $f_w^g(\sigma'') = f_w^g\big(\sigma'' \cap f_{w'^*}^g(\sigma')\big) = f_w^g\big(\sigma'' \cap f_{w'^*}^g(\sigma')\big) \cap f_{w'^*}^g(\sigma')$. If $w \leq |\sigma''|$ then $F_w^g(\sigma'_l) \cap f_{w'^*}^g(\sigma') = f_w^g(\sigma'') \cap f_{w'^*}^g(\sigma') = f_w^g(\sigma'')$ hence $w \leq |\sigma'_l \cap f_{w'^*}^g(\sigma')| \Rightarrow |F_w^g(\sigma'_l) \cap f_{w'^*}^g(\sigma')| = w$. If $w > |\sigma''|$ then $F_w^g(\sigma'_l) \cap f_{w'^*}^g(\sigma') = \sigma'_l \cap f_{w'^*}^g(\sigma') \cup \big(f_{w'}^g(\sigma'_l) \backslash \sigma''\big) \cap f_{w'^*}^g(\sigma')$ where $f_{w'}^g(\sigma'_l) \backslash \sigma'' \cap f_{w'^*}^g(\sigma') = \varnothing$ because $\sigma'_l \backslash \sigma'' \cap f_{w'^*}^g(\sigma') = \big(\sigma'_l \cap f_{w'^*}^g(\sigma')\big) \backslash \big(\sigma'' \cap f_{w'^*}^g(\sigma')\big) = \varnothing$. So $w > |\sigma'_l \cap f_{w'^*}^g(\sigma')| \Rightarrow |F_w^g(\sigma'_l) \cap f_{w'^*}^g(\sigma')| = |\sigma'_l \cap f_{w'^*}^g(\sigma')|$. The next paragraph proves that both implications match the equation of Property 5.

By construction, $\sigma_{l-1}'^D$ gathers all nodes such that branching on them without pruning would give $\sigma_l \backslash \sigma'_l \cup \sigma_l^S \backslash \sigma_l'^S$. Applying Lemma 4 iteratively on each node in $\sigma_{l-1}'^D$ until level l is reached leads to $P \circ g(\sigma_{l-1}'^D) = P \circ g(\sigma_l \backslash \sigma'_l \cup \sigma_l^S \backslash \sigma_l'^S)$ through Lemma 3. On the other hand, $P \circ g(\sigma'_l \cup \sigma_l'^S \cup \sigma_{l-1}'^D) = P\big(g(\sigma'_l \cup \sigma_l'^S) \cup P \circ g(\sigma_{l-1}'^D)\big)$ and $P \circ g(\sigma_l \cup \sigma_l^S) = P \circ g(\sigma'_l \cup \sigma_l'^S \cup \sigma_l \backslash \sigma'_l \cup \sigma_l^S \backslash \sigma_l'^S) = P\big(g(\sigma'_l \cup \sigma_l'^S) \cup P \circ g(\sigma_l \backslash \sigma'_l \cup \sigma_l^S \backslash \sigma_l'^S)\big)$ through Lemma 3. Hence $P \circ g(\sigma') = P \circ g(\sigma)$. By construction, $\sigma'_l \subseteq \sigma'$ and $\sigma'_l \subseteq \sigma$, so each node of σ'_l has an oracle point in $P \circ g(\sigma')$ if and only if it has an oracle point in $P \circ g(\sigma)$. Such nodes are selected by $f_{w'^*}^g(\sigma')$ as well as $F_{w^*}^g(\sigma)$ because $P \circ g(\sigma') = P \circ g\big(f_{w'^*}^g(\sigma')\big)$ and $P \circ g(\sigma) = P \circ g\big(F_{w^*}^g(\sigma)\big)$ by filter Pareto-efficiency. Formally, $\sigma'_l \cap f_{w'^*}^g(\sigma') = \sigma'_l \cap F_{w^*}^g(\sigma)$ due to the use of ideal beam widths. Then, $F_w^g(\sigma'_l) \subseteq \sigma'_l$ implies $F_w^g(\sigma'_l) \cap f_{w'^*}^g(\sigma') = F_w^g(\sigma'_l) \cap F_{w^*}^g(\sigma)$.

Otherwise and if the filter is deterministic and d-versatile

Let us consider instances of d-objective optimization problems such that

- a node is discarded at a certain level k (because w is lower than the ideal beam width of σ'_k),
- there exists a level $l > k$ such that the equation $P \circ g(\sigma_{l-1}'^D) = P \circ g(\sigma_l \backslash \sigma'_l \cup \sigma_l^S \backslash \sigma_l'^S)$ handled in the paragraph above is not met (according to Lemma 4 due to lack of constancy).

Not meeting this equation means that oracle points of discarded nodes are inconsistent for comparisons to the points of the nodes kept at further levels.

In this context, we give a counter-example for a deterministic and d-versatile filter F with $d = 2$ and $w = 1$. Figure 10 depicts oracle points at a certain tree level for two different instances. For each instance $i \in \{1, 2\}$, two nodes σ_i^A and σ_i^C are submitted to the filter at level l, hence $\sigma'_l = \{\sigma_i^A, \sigma_i^C\}$. One oracle point is computed

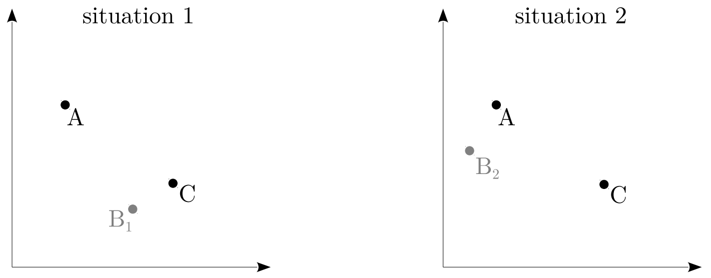

Figure 10: For each situation, we are minimizing. If B$_1$ and B$_2$ become unknown, it cannot be guessed which point among A and C is part of the original Pareto-set.

for each node: $g(\sigma_i^A) = A$ and $g(\sigma_i^C) = C$. Those points do not differ between the instances. Consider that $\sigma = \{\sigma_i^A, \sigma_i^B, \sigma_i^C\}$, so σ_i^B descends from a discarded node and its oracle point $g(\sigma_i^B) = B_i$ is unknown. Let us assume that $F_w^g(\sigma_l') = \{\sigma_1^A\}$ for instance 1, hence filter perfection. Then, for instance 2, $F_w^g(\sigma_l') = \{\sigma_2^A\}$ because the filter is d-versatile and deterministic and $g(\sigma_1^A) = g(\sigma_2^A)$ and $g(\sigma_1^C) = g(\sigma_2^C)$. Note that $F_{w^*}^g(\sigma) = \{\sigma_2^B, \sigma_2^C\}$ due to Pareto-efficiency. Hence $|F_w^g(\sigma_l') \cap F_{w^*}^g(\sigma)| \neq 1$ for the second instance. That violates the equation of Property 5. At last, note that counter-examples can be designed for any $d \geq 2$ and $w \geq 1$.

B.4 Proof of Theorem 4

By contraposition, let us assume that there is a point $p \in P$ that the filter does not compare to any point of another node, neither directly nor through transitive relations. If p is ignored or only compared to points belonging to the same node, it cannot be known whether an external point dominates p because p is incomparable or dominates the other points of the node. If p is not ignored and not compared to another point, then its coordinates are assessed such that its dominance relation to any other point is not retrieved. Again, it cannot be known whether an external point dominates p.

B.5 Proof of Theorem 5

In the following, we use the notation of Property 5. According to the theorem claim, the filter assigns a rank to every node σ^i. Let us denote by $r(\sigma^i)$ its rank when $\sigma \cup \sigma^S$ is submitted to the filter, and by $r'(\sigma^i)$ its rank when σ' is submitted. Set $F_w^g(\sigma') \cap F_{w^*}^g(\sigma \cup \sigma^S)$ can be written $\{\sigma^i \in \sigma', r'(\sigma^i) \leq w\} \cap \{\sigma^i \in \sigma \cup \sigma^S, r(\sigma^i) \leq w^*\}$, which equals $\{\sigma^i \in \sigma', r'(\sigma^i) \leq w \wedge r(\sigma^i) \leq w^*\}$. It equals either $\{\sigma^i \in \sigma', r'(\sigma^i) \leq w\}$ or $\{\sigma^i \in \sigma', r(\sigma^i) \leq w^*\}$ and the correct option is the smallest set. By construction, $|\{\sigma^i \in \sigma', r'(\sigma^i) \leq w\}| = w$ and $\{\sigma^i \in \sigma', r(\sigma^i) \leq w^*\} = \sigma' \cap F_{w^*}^g(\sigma \cup \sigma^S)$.

C Proofs of the propositions

C.1 Proof of Proposition 1

Section 3.1 shows that linear relaxations have a constancy degree lower than 1 in general while their leaf-compliance degree is 1.

C.2 Proof of Proposition 2

Section 3.1 shows that Aneja-Nair's algorithm cannot lead to a constant oracle and heuristics have a constancy degree lower than 1 in general, while their leaf-compliance degree is 1.

C.3 Proof of Proposition 3

Section 3.1 shows that linear relaxations have a constancy degree lower than 1 in general while their leaf-compliance degree is 1.

C.4 Proof of Proposition 4

Let us consider a STSPP instance. If the optimal cycle is empty, its objective value is 0. Step 4 ensures that the MN1Topt weight is not positive. If the optimal cycle C is not empty, removing from C the edges adjacent to vertex 1 gives a tree whose particular structure makes its total weight greater than or equal to the weight of the minimal tree computed at Step 2, and the weight of edges removed is also greater than or equal to the weight of the edges selected at Step 3. So MN1Topt is a lower bound. Moreover, every solution feasible for STSPP is feasible for MN1Topt by construction, and there exist feasible solutions to MN1Topt which are infeasible for STSPP. At last, all MN1Topt solutions are integer. Therefore, MN1Topt is a combinatorial relaxation.

C.5 Proof of Proposition 5

Consider an instance of the Minimum Steiner Tree Problem (which is NP-hard, see Karp, 1972) having m vertices $i \in \{2 \dots m+1\}$ to be connected and n non-mandatory vertices $j \in \{m+2 \dots m+n+1\}$. Add vertex 1 to the graph and two edges of weight 0 between vertex 1 and two arbitrary vertices. Add m vertices i' ($i \in \{2 \dots m+1\}$) to the graph along with mandatory edges $\{i, i'\}$ of weight 0. Compute MN1Topt and then remove all additional edges from the result. The set of edges left is the optimal Steiner tree of the original instance.

C.6 Proof of Proposition 6

Section 3.1 shows that Aneja-Nair's algorithm cannot lead to a constant oracle and heuristics have a constancy degree lower than 1 in general. It also shows that combinatorial relaxations are leaf-compliant if and only if the branching scheme avoids all relaxed solutions which are infeasible for the BTSPP.

C.7 Proof of Proposition 7

Section 3.1 shows that Aneja-Nair's algorithm cannot lead to a constant oracle in general. It also shows that combinatorial relaxations are leaf-compliant if and only if the branching scheme avoids all relaxed solutions which are infeasible for the BTSPP.

C.8 Proof of Proposition 8

The filter decisions consider only w and the oracle points and do not assume specific problem properties, hence d-versatility.

By contraposition of Theorem 4, the oracle points are not compared to each other hence lack of Pareto-efficiency. The reference point used to compute the hyper-volume of the oracle points is fixed, so the conditions of Theorem 5 are met, hence independence of irrelevant alternatives.

C.9 Proof of Proposition 9

The filter decisions consider only w and the oracle points and do not assume specific problem properties, hence d-versatility.

By contraposition of Theorem 4, the oracle points are not compared to each other hence lack of Pareto-efficiency.

To show the lack of independence of irrelevant alternatives, we give a counter-example. The following uses the notation of Property 5. Let us consider $\sigma' = \{\sigma^\circ, \sigma^+\}$. On Figure 11, $g(\sigma^\circ)$ is plotted with circles while $g(\sigma^+)$ is plotted with crosses, and their hypervolumes are respectively $\text{HV}_\circ$ and HV_+. On the left, $\sigma \cup \sigma^S$ is submitted to the filter. Each triangle is the oracle point of a leaf in σ^S. Note that $w^* = 2$. $\text{HV}_\circ$ is the white area delimited by dotted lines plus the hatched area, while HV_+ is the white area delimited by dotted lines plus the gray area. The hatched area being wider than the gray one, $\text{HV}_\circ > \text{HV}_+$. $g(\sigma^\triangle)$ dominates all points in $g(\sigma^+)$ so it has a greater hypervolume. Therefore, $F^g_{w^*}(\sigma \cup \sigma^S) = \{\sigma^\circ, \sigma^\triangle\}$. On the right, only σ' is submitted to the filter so the reference point (black square) changes. The hypervolumes change such that σ^+ is now ranked before σ°. Let us consider that $w = 1$. Then, $F^g_w(\sigma') = \{\sigma^+\}$ whereas $\sigma^+ \notin F^g_{w^*}(\sigma \cup \sigma^S)$, so $|F^g_w(\sigma') \cap F^g_{w^*}(\sigma \cup \sigma^S)| \neq 1$. That violates the equation of Property 5.

C.10 Proof of Proposition 10

The filter decisions consider only w and the oracle points and do not assume specific problem properties, hence d-versatility.

Regarding Pareto-efficiency, let us consider a node set σ'_l whose ideal beam width is w'^*_l. For every node $\sigma^j \in \sigma'_l$ having an oracle point in f_1, $q^1_{\sigma^j} > 0$ holds so $D1(\sigma^j) \geq M^{|f|-1}$. For every node $\sigma^k \in \sigma'_l$ having no oracle point in f_1, $q^1_{\sigma^k} = 0$ holds so $D1(\sigma^k) \leq (M-1)\sum_{i=2}^{|f|} M^{|f|-i}$ where the right-hand side equals $M^{|f|-1} - 1$ through a few algebra. Hence $D1(\sigma^j) > D1(\sigma^k)$. Therefore, the w'^*_l nodes contributing to $P \circ g(\sigma'_l) = f_1$ are ranked first, hence $P \circ g\big(F^g_{w'^*_l}(\sigma'_l)\big) = P \circ g(\sigma'_l)$. It is clear that the second equation of Property 4 is also met, we omit this part.

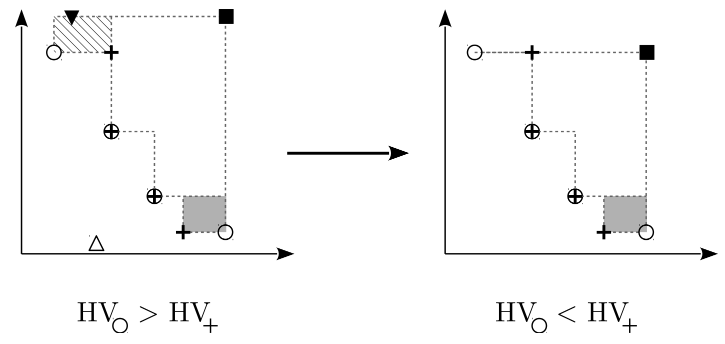

Figure 11: For each situation, we are minimizing. When the reference point (black square) moves, the hypervolume of the set of circle points becomes lower than the hypervolume of the set of cross points.

According to the first case handled in the proof of Theorem 3, Pareto-efficiency implies independence of irrelevant alternatives if $\forall l\ w \geq w_l'^*$. This Pareto-efficient filter is deterministic and d-versatile. Moreover, it does not known the oracle points of the discarded nodes. Then, according to the last case handled in the proof of Theorem 3, it is not independent of irrelevant alternatives in general if $\exists l\ w < w_l'^*$.

C.11 Proof of Proposition 11

The proof of Proposition 10 is written such that it is suitable for Proposition 11. Read "D2" instead of "D1".

C.12 Proof of Proposition 12

Using the notation of Sections C.10 and C.11, Pareto-efficiency of DS is due to $D1(\sigma^j) > D1(\sigma^k) \Leftrightarrow D2(\sigma^j) > D2(\sigma^k)$. Regarding independence of irrelevant alternatives, the counter-example given there is suitable for DS.

References

Yash P. Aneja and Kunhiraman P. K. Nair. Bicriteria transportation problem. *Management Science*, 25(1): 73–78, 1979.

Kenneth J. Arrow, Amartya K. Sen, and Kotaro Suzumura, editors. *Handbook of Social Choice and Welfare, Volume 1*. North Holland, 2002.

José Elias Claudio Arroyo and Ana Amélia Souza Pereira. A GRASP heuristic for the multi-objective permutation flowshop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 55 (5-8):741–753, 2011.

Anne Auger, Johannes Bader, Dimo Brockhoff, and Eckart Zitzler. Hypervolume-based multiobjective optimization: theoretical foundations and practical implications. *Theoretical Computer Science*, 425:75–103, 2012.

Cristina Bazgan, Hadrien Hugot, and Daniel Vanderpooten. Solving efficiently the 0-1 multi-objective knapsack problem. *Computers & Operations Research*, 36(1):260–279, 2009a.

Cristina Bazgan, Hadrien Hugot, and Daniel Vanderpooten. Implementing an efficient fptas for the 0-1 multi-objective knapsack problem. *European Journal of Operational Research*, 198(1):47–56, 2009b.

Jean-François Bérubé, Michel Gendreau, and Jean-Yves Potvin. An exact ϵ-constraint method for bi-objective combinatorial optimization problems: application to the traveling salesman problem with profits. *European Journal of Operational Research*, 194(1):39–50, 2009.

Nicola Beume. S-metric calculation by considering dominated hypervolume as klee's measure problem. *Evolutionary Computation*, 17(4):477–492, 2009.

Nicola Beume, Carlos M. Fonseca, Manuel López-Ibánez, Luís Paquete, and Jan Vahrenhold. On the complexity of computing the hypervolume indicator. *IEEE Transactions on Evolutionary Computation*, 13(5):1075–1082, 2009.

Karl Bringmann and Tobias Friedrich. Approximating the least hypervolume contributor: NP-hard in general, but fast in practice. *Theoretical Computer Science*, 425:104–116, 2012.

Maxim Buzdalov and Anatoly Shalyto. A provably asymptotically fast version of the generalized Jensen algorithm for non-dominated sorting. In *Parallel Problem Solving from Nature - PPSN XIII*, volume 8672 of *Lecture Notes in Computer Science*, pages 528–537. Springer International Publishing, 2014.

Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Thirunavukarasu Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary computation*, 6(2):182–197, 2002.

Federico Della Croce, Marco Ghirardi, and Roberto Tadei. Recovering beam search: enhancing the beam search approach for combinatorial optimization problems. *Journal of Heuristics*, 10(1):89–104, 2004.

Iván Derpich and Jorge R. Vera. Improving the efficiency of the branch and bound algorithm for integer programming based on flatness information. *European Journal of Operational Research*, 174(1):92–101, 2006.

Matthias Ehrgott. *Multicriteria Optimization.* Springer, 2005.

Matthias Ehrgott and Xavier Gandibleux. Hybrid metaheuristics for multi-objective combinatorial optimization. In *Hybrid Metaheuristics, An Emerging Approach to Optimization*, pages 221–259. Springer, 2008.

Dominique Feillet, Pierre Dejax, and Michel Gendreau. Traveling salesman problems with profits. *Transportation Science*, 39(2):188–205, 2005.

José Rui Figueira, Luís Paquete, Marco Simoes, and Daniel Vanderpooten. Algorithmic improvements on dynamic programming for the bi-objective 0,1 knapsack problem. *Computational Optimization and Applications*, 56(1):97–111, 2013.

Carlo Filippi and Elisa Stevanato. A two-phase method for bi-objective combinatorial optimization and its application to the tsp with profits. *Algorithmic Operations Research*, 7(2), 2012.

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.

Matteo Fischetti. Branchstorming (brainstorming about tree search). In *ISCO 2014: 3rd International Symposium on Combinatorial Optimization, Lisbon*, 2014.

Jiaquan Gao, Guixia He, Ronghua Liang, and Zhilin Feng. A quantum-inspired artificial immune system for the multiobjective 0-1 knapsack problem. *Applied Mathematics and Computation*, 230:120–137, 2014.

Marco Ghirardi and Chris N. Potts. Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. *European Journal of Operational Research*, 165(2):457–467, 2005. Project Management and Scheduling.

Tiziano Ghisu, Geoffrey T. Parks, Daniel M. Jaeggi, Jerome P. Jarrett, and P. John Clarkson. The benefits of adaptive parametrization in multi-objective tabu search optimization. *Engineering Optimization*, 42(10): 959–981, 2010.

Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees: part II. *Mathematical Programming*, 1:6–25, 1971.

Naoya Honda, Shintaro Mohri, and Hiroaki Ishii. Backtracking beam search applied to multi-objective scheduling problem. In *The Fifth Metaheuristics International Conference, Kyoto*, pages 1–6, 2003.

Andrzej Jaszkiewicz. On the computational efficiency of multiple objective metaheuristics. the knapsack problem case study. *European Journal of Operational Research*, 158(2):418–433, 2004.

Mikkel T. Jensen. Reducing the run-time complexity of multiobjective EAs: The NSGA-II and other algorithms. *IEEE Transactions on Evolutionary Computation*, 7(5):503–515, 2003.

David S. Johnson and Lyle A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, 1997.

Nicolas Jozefowiez, Fred Glover, and Manuel Laguna. Multi-objective meta-heuristics for the traveling salesman problem with profits. *Journal of Mathematical Modelling and Algorithms*, 7(2):177–195, 2008. ISSN 1570-1166.

Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972. ISBN 978-1-4684-2003-6.

Kap Hwan Kim and Ki Young Kim. Routing straddle carriers for the loading operation of containers using a beam search algorithm. *Computers & Industrial Engineering*, 36(1):109–136, 1999.

Joseph. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.

Arnaud Liefooghe, Luís Paquete, and José Rui Figueira. On local search for bi-objective knapsack problems. *Evolutionary Computation*, 21(1):179–196, 2013.

Manuel López-Ibánez and Thomas Stützle. An experimental analysis of design choices of multi-objective ant colony optimization algorithms. *Swarm Intelligence*, 6(3):207–232, 2012.

Bruce T. Lowerre. *The HARPY speech recognition system*. PhD thesis, Carnegie-Mellon University, Pittsburgh, 1976.

Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, New York, 1990. ISBN 0-471-92420-2.

George Mavrotas and Danae Diakoulaki. Multi-criteria branch and bound: a vector maximization algorithm for mixed 0-1 multiple objective linear programming. *Applied Mathematics and Computation*, 171(1):53–71, 2005.

Clair E. Miller, Albert W. Tucker, and Richard A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4):326–329, 1960.

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

Özgür Özpeynirci and Murat Köksalan. An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs. *Management Science*, 56(12):2302–2315, 2010.

Sophie N. Parragh and Fabien Tricoire. Branch-and-bound for bi-objective integer programming. *Optimization Online*, 2015.

David Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32(9):2271–2284, 2005.

Aníbal Ponte, Luís Paquete, and José R. Figueira. On beam search for multicriteria combinatorial optimization problems. In Nicolas Beldiceanu, Narendra Jussien, and Eric Pinson, editors, *Integration of AI and OR techniques in constraint programming for combinatorial optimization problems*, volume 7298 of *Lecture Notes in Computer Science*, pages 307–321. Springer Berlin Heidelberg, 2012.

Anthony Przybylski, Xavier Gandibleux, and Matthias Ehrgott. A recursive algorithm for finding all nondominated extreme points in the outcome set of a multiobjective integer programme. *INFORMS Journal on Computing*, 22(3):371–386, 2010.

Giovanni Righini and Matteo Salani. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 3(3):255–273, 2006. ISSN 1572-5286.

Roberto Roberti and Paolo Toth. Models and algorithms for the asymmetric traveling salesman problem: an experimental comparison. *EURO Journal on Transportation and Logistics*, 1(1):113–133, 2012.

Ihsan Sabuncuoglu, Yasin Gocgun, and Erdal Erel. Backtracking and exchange of information: Methods to enhance a beam search algorithm for assembly line scheduling. *European Journal of Operational Research*, 186(3):915–930, 2008.

Pierre Schaus and Renaud Hartert. Multi-objective large neighborhood search. In Christian Schulte, editor, *Principles and Practice of Constraint Programming*, volume 8124 of *Lecture Notes in Computer Science*, pages 611–627. Springer Berlin Heidelberg, 2013.

Kenneth Sörensen. Metaheuristics. In Saul I. Gass and Michael C. Fu, editors, *Encyclopedia of Operations Research and Management Science*, pages 307–321. Springer, 2013.

Francis Sourd and Olivier Spanjaard. A multiobjective branch-and-bound framework: application to the biobjective spanning tree problem. *INFORMS Journal on Computing*, 20(3):472–484, 2008.

Thomas Vincent, Florian Seipp, Stefan Ruzika, Anthony Przybylski, and Xavier Gandibleux. Multiple objective branch and bound for mixed 0-1 linear programming: corrections and improvements for the biobjective case. *Computers & Operations Research*, 40(1):498–509, 2013.

Aimin Zhou, Bo-Yang Qu, Hui Li, Shi-Zheng Zhao, Ponnuthurai Nagaratnam Suganthan, and Qingfu Zhang. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32–49, 2011.

Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithms - a comparative case study. *Lecture Notes in Computer Science*, 1498:292–301, 1998.

Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.

Eckart Zitzler, Dimo Brockhoff, and Lothar Thiele. The hypervolume indicator revisited: on the design of pareto-compliant indicators via weighted integration. *Lecture Notes in Computer Science*, 4403:862–876, 2007.

Eckart Zitzler, Lothar Thiele, and Johannes Bader. On set-based multiobjective optimization. *Evolutionary Computation, IEEE Transactions on*, 14(1):58–79, 2010.