Branch-and-bound for bi-objective integer programming

Sophie N. Parragh[1,2] and Fabien Tricoire[1]

[1]Department of Business Administration, University of Vienna
Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria
{sophie.parragh,fabien.tricoire}@univie.ac.at

[2]Institute for Transport and Logistics Management,
WU (Vienna University of Economics and Business)
Welthandelsplatz 1, 1020 Vienna, Austria

January 21, 2015

Abstract

In Pareto bi-objective integer optimization the optimal result corresponds to a set of non-dominated solutions. We propose a generic bi-objective branch-and-bound algorithm that uses a problem-independent branching rule exploiting available integer solutions, and cutting plane generation taking advantage of integer objective values. The developed algorithm is applied to the bi-objective team orienteering problem with time windows, considering two minimization objectives. Lower bound sets are computed by means of column generation, while initial upper bound sets are generated by means of multi-directional local search. Comparison to using the same ingredients in an ϵ-constraint scheme shows the effectiveness of the proposed branch-and-bound algorithm.

1 Introduction

Many practical problems involve several conflicting objectives and are more and more often considered as such. In particular, when it is not possible to aggregate the objectives, there is a requirement to produce the set of trade-off solutions, or at least a subset of it. This typically happens when the decision maker cannot elicit a preference function a priori, which can be the case when the different objectives are measured in non-comparable units, like for instance cost versus quality of service. This can also happen in other areas where the decision maker is interested in information on the actual trade-off relationship between two conflicting goals. Many of these problems can be modeled as bi-objective (mixed) integer linear programs. With theoretical progress as well as hardware improvements, exact solution approaches for single-objective (mixed) integer optimization have been flourishing over the last decade, be it problem-specific approaches or general-purpose frameworks. To some extent, this trend can also be observed in multi-objective optimization and especially in bi-objective (mixed) integer optimization. For a general introduction to multicriteria decision making, we refer to Ehrgott (2005).

Exact approaches in multi-objective (mixed) integer programming can be divided into two classes: those that work in the space of objective function values (referred to as criterion space search methods, e.g., by Boland et al. (2013a)) and those that work in the space of feasible solutions (generalizations of branch-and-bound algorithms).

Criterion space search methods solve a succession of single-objective problems in order to compute the set of Pareto optimal solutions. Therefore, they are able to exploit the power of single-objective mixed integer programming solvers. This appears to be one of their main advantages in comparison to generalizations of branch-and-bound algorithms (Boland et al. 2013b). However, many combinatorial single-objective problems cannot be solved efficiently by commercial solvers, e.g. the most efficient exact algorithms in the field of vehicle routing rely on column generation based techniques (see e.g. Baldacci et al. 2012). Thus, especially in this context but also in general, it is not clear whether a criterion space search method or a bi-objective branch-and-bound algorithm is more efficient. In this paper, we develop a general purpose bi-objective branch-and-bound framework and compare it to a criterion space search method.

One of the most popular criterion space search methods is the *ϵ-constraint method*, first introduced by Haimes et al. (1971). It consists in iteratively solving single-objective versions of a bi-objective problem. In every step, the first objective is optimized, but a constraint is updated in order to improve the quality of the solution with regards to the second objective. Thus the whole set of efficient solutions is enumerated. Laumanns et al. (2006) show that the ϵ-constraint method can be extended to more objectives, and provide a proof of concept for three objectives. The *ϵ-constraint method* is generic and simple to implement and it is among the best performing criterion space search algorithms when applied, e.g., to the bi-objective prize-collecting Steiner tree problem (Leitner et al. 2014).

A first theoretical characterization of efficient (or Pareto-optimal) solutions of integer problems is provided by Geoffrion (1968). The efficient frontier (or set of efficient points) is the image of all Pareto-optimal solutions of a multi-objective problem in objective space. The set of efficient solutions can be partitioned into *supported* and *non-supported* efficient solutions. Each supported efficient solution is optimal for at least one single-objective weighted-sum version of the multi-objective problem (with strictly positive weights), which does not hold for non-supported efficient solutions. Several criterion space search methods rely on this characterization of efficient solutions. Aneja and Nair (1979) describe an algorithm, sometimes referred to as the *weighted sum method*, to generate all extreme supported points in objective space. These points are the corner points of the boundary of the convex hull of the set of points corresponding to feasible solutions. They use this algorithm to solve the bicriteria transportation problem, which is formulated as a bi-objective linear program.

The *two-phase method* (Ulungu and Teghem 1995) also relies on this characterization: In the first phase, supported efficient solutions are generated using an algorithm similar to that of Aneja and Nair. In the second phase, each triangle defined by consecutive corner points on the convex hull boundary is searched for non-supported efficient solutions, for instance using a branch-and-bound algorithm. The points defining the triangle are used in the bounding in order to speed up the search. Tuyttens et al. (2000) and Przybylski et al. (2008) provide improved upper bounds for the second phase of the two-phase method.

The algorithm of Chalmet et al. (1986) is also similar to the algorithm of Aneja and Nair (1979). They iteratively solve a weighted sum scalarization considering bounds on both objectives that exclude previously generated solutions. Boland et al. (2013a) refer to this method as the *perpendicular method*.

The *weighted Tchebycheff method* is another criterion space search method. It considers a succession of reference points and minimizes the distance to these reference points using a weighted-sum objective function. Recent advances on the weighted Tchebycheff method as well as references are provided by Dächert et al. (2012).

Very recently, the *rectangle splitting method* for bi-objective 0-1 integer programs has been introduced by Boland et al. (2013a). Optimal solutions are computed for each objective and they define a rectangle. This rectangle is then split in half and in each half, the objective which has not been optimized yet in this half of the rectangle is then optimized. At this stage there are two non-overlapping rectangles (the objective space is partitioned). The same procedure is repeated recursively on these rectangles. In Boland et al. (2013b), using similar ideas, the *triangle splitting method* is developed. It is the first general-purpose criterion space search algorithm for bi-objective mixed integer programs.

Extensive notation and definitions for *bound sets* for bi-objective integer optimization are provided by Ehrgott and Gandibleux (2006): the concept of bound, so useful in single-objective (mixed) integer programming, is extended to the concept of bound set, since the optimum for a bi-objective optimization problem is a set of solutions and not a single solution. Interestingly, upper bound sets for minimization problems (resp. lower bound sets for maximization problems) are discrete sets but lower bound sets for minimization problems (resp. upper bound sets for maximization problems) are continuous sets. In the general case, the bottom-left part of the boundary of the convex hull of the image of the feasible set in objective space, which can be obtained using the algorithm of Aneja and Nair, provides a valid lower bound set for minimization problems.

Mavrotas and Diakoulaki (1998) provide generalizations of branch-and-bound to multiple objectives for mixed 0-1 integer programs. Their bounding procedure considers an ideal point for the upper bound (they work on a maximization problem), consisting of the best possible value for each objective at this node, and keeps branching until this ideal point is dominated by the lower bound set. Vincent et al. (2013) improve the algorithm by Mavrotas and Diakoulaki (1998), most notably by comparing bound sets instead of ideal points in the bounding procedure.

Masin and Bukchin (2008) propose a branch-and-bound method that uses a surrogate objective function returning a single numerical value. This value can be treated like a lower bound in the context of single objective minimization and conveys information on whether the node can be fathomed or not. They illustrate their method using a three-objective scheduling problem example but no computational study is provided.

Sourd and Spanjaard (2008) use separating hyperplanes between upper and lower bound sets in order to discard nodes in a general branch-and-bound framework for integer programs. The concept is in fact similar to the lower bound sets defined by Ehrgott and Gandibleux (2006).

Jozefowiez et al. (2012) introduce a branch-and-cut algorithm for integer programs in which discrete sets are used for lower bounds, so nodes can be pruned if every point in the lower bound set is dominated by a point in the upper bound set. Partial pruning is used in order to discard parts of the solutions represented by a given node of the search tree, thus speeding up the whole tree search.

Stidsen et al. (2014) provide a branch-and-bound algorithm to deal with a certain class of bi-objective mixed integer linear programs. Lower bounds correspond to solutions of a scalarized single-objective version of the original bi-objective problem. The fathoming rules of traditional single-objective branch-and-bound are modified in order to generate the whole Pareto set: nodes that provide integer solutions are not necessarily fathomed. Rather, no-good constraints are added to the current node so that previously generated solutions in the path from the root node are forbidden for that sub-tree. Since they only solve a single-objective problem, bound fathoming is also modified: a node is fathomed if it only yields solutions that are dominated by local nadir points. These local nadir points are derived from the current set of non-dominated solutions. They also propose two problem-independent improvements that rely on partitioning the objective space: slicing and Pareto branching. Slicing partitions the objective space into slices of equal size and slices dominated by available integer solutions can be disregarded. Pareto branching refers to a binary branching scheme that exploits available integer solution sets in order to disregard parts of the objective space. The method is applied to six different problems and compared to a criterion space search, namely a generic two-phase implementation. The proposed method performs better on five out of six data sets.

Belotti et al. (2013) are the first to introduce a general-purpose branch-and-bound algorithm for bi-objective mixed integer linear programming, where the continuous variables may appear in both objective functions. They build up on the previous work by Visée et al. (1998), Mavrotas and Diakoulaki (1998), Sourd and Spanjaard (2008) and Vincent et al. (2013) and like them, they use a binary branching scheme. Improved fathoming rules are introduced in order to discard more nodes during tree search.

Following these recent developments, we propose a generalization of the branch-and-bound algorithm to bi-objective optimization for integer programs. A few studies already exist on this topic but we still see room for improvement. First of all, few of them have proved to work in the general case: although they are general methods in theory, they are often only applied to specific cases, for instance to a problem where the single-objective scalarized problem is polynomial (Sourd and Spanjaard 2008), or to a problem where the values for one objective can be enumerated to a small set (Jozefowiez et al. 2012). To the best of our knowledge, the recent work by Belotti et al. (2013) is the only generalization of branch-and-bound to two objectives that is truly general for integer programs as well as mixed integer linear programs. The work by Vincent et al. (2013) is general for mixed 0-1 programs. The algorithm by Stidsen et al. (2014) is general for mixed 0-1 programs where only one of the two objectives uses continuous variables, but can probably easily be generalized to the case where continuous variables appear in both objective functions. Moreover, few approaches exploit the fact that the problem is bi-objective in their branching strategy.

Our algorithm, described in Section 2, works for any bi-objective integer program. We believe that, with the exception of the enhancements described in Section 2.5, it can easily be generalized to mixed integer programs, although we keep such a generalization as a research perspective. We introduce a new problem-independent branching rule for bi-objective optimization, which allows to discard the portions of objective space that are dominated by feasible solutions that have already been found, even in those cases where the bound set is not completely dominated. In addition, we propose several improvements exploiting the integrality of objective functions. As a proof of concept, we apply it to an orienteering problem, which is NP-hard even in the single-objective case. The lower bound sets are produced using column generation and the initial upper bound sets are obtained using multi-directional local search, as described in Section 3. Although column generation has been used in the context of bi-objective optimization, e.g. in the context of robust airline crew scheduling (Tam et al. 2011), vehicle routing (Sarpong et al. 2013), and bi-objective linear programming (Raith et al. 2012), this is, to the best of our knowledge, the first time column generation is used in a bi-objective branch-and-bound algorithm, resulting in a bi-objective branch-and-price algorithm. In order to validate our approach, we compare it to a criterion space search algorithm, namely the ϵ-constraint method, while using the same ingredients. Experimental results are presented in Section 4.

2 A branch-and-bound framework for bi-objective optimization

Branch-and-bound is a general purpose tree search method to solve (mixed) integer linear programs. As its name suggests, it has two main ingredients: branching and bounding. Branching refers to the way the search space is partitioned, while bounding refers to the process of using valid bounds to discard subsets of the search space without compromising optimality. In the following, we first describe the general framework of our bi-objective branch-and-bound (BIOBAB). We then describe its different ingredients in further detail.

2.1 General idea

For ease of exposition and without loss of generality, we always consider in the following that both objective functions, called f_1 and f_2, have to be minimized. As in other approaches, our BIOBAB is similar to a traditional branch-and-bound algorithm, except for the fact that we compare bound sets, in contrast with bounds as single numerical values. This is similar to most previously mentioned bi- or multi-objective approaches like Visée et al. (1998), Mavrotas and Diakoulaki (1998), Sourd and Spanjaard (2008), Vincent et al. (2013), Jozefowiez et al. (2012) and Belotti et al. (2013). This is however slightly different from the work by Stidsen et al. (2014), in which a single scalarized lower bound is compared to an upper bound set.

The general framework of our BIOBAB is outlined in Algorithm 1. Parameter UB is the upper bound (UB) set used and updated during the search. A node is a set of branching decisions (the root node of any search tree being the empty set), and the *bound* function calculates the lower bound (LB) set for that node. As a side effect, it also updates UB with any integer solution found during the bounding process. The LB set thus obtained is then filtered using the current UB set. The *branch* function takes a LB set as parameter and returns a set of branching decisions. If the LB set corresponds to a leaf (i.e. it is not possible to branch any more and the node corresponds to a unique integer feasible solution), or if it is completely dominated following filtering with the UB set, then an empty set is returned by *branch*. We assume that *push* adds a new element to a set and that *pop* retrieves an element from a set and deletes it from it. In the context of tree search, *push* adds a new node to the set Λ of nodes to process, and *pop* retrieves the next node to process from Λ. Depending on the data structure used for Λ, different strategies can be applied (e.g. depth-first when using a stack, breadth-first when using a queue).

Algorithm 1 *treeSearch*(UB)

```
1: $rootNode \leftarrow \emptyset$
2: $push(\Lambda, rootNode)$
3: while $\Lambda \neq \emptyset$ do
4:    $node \leftarrow pop(\Lambda)$
5:    $LB \leftarrow bound(node, UB)$
6:    $LB \leftarrow filterLB(LB, UB)$
7:    if $LB \neq \emptyset$ then
8:       $newBranches \leftarrow branch(LB)$
9:       for all $decision \in newBranches$ do
10:         $push(\Lambda, node \cup \{decision\})$
11:      end for
12:   end if
13: end while
```

Any non-dominated set of feasible solutions can be used as a valid UB set. We can for instance use a (meta)heuristic to provide such a set, or we can use the empty set. This is the bi-objective equivalent to upper bounds in single-objective branch-and-bound: if the current upper bound set dominates the lower bound set at a given node of the branch-and-bound tree, then this node can be fathomed. Every time a new feasible integer solution is found, which can happen during the bounding procedure, the UB set is updated with this solution: if there is no solution in the UB set that dominates this new solution then it must be added to the set; if this new solution dominates some solutions from the UB set, then we must remove them from the set.

In order to calculate LB sets, we use the linear relaxation of the original IP. As established by Ehrgott and Gandibleux (2006), the lower bound set of any relaxation of a given optimization problem is a valid lower bound set for that optimization problem. In the general case, the bottom-left boundary of the convex hull of the non-dominated set for any bi-objective minimization problem is a valid lower bound set. As mentioned earlier, the points on this convex hull boundary correspond to the supported efficient solutions. There are algorithms to compute the solutions corresponding to corner points of this convex hull, for instance the dichotomy algorithm by Aneja and Nair (1979). As established by the authors, this algorithm is linear in the number of corner points. Our *bound* function (see Algorithm 2) is based on this algorithm. It takes as input all branching decisions for a given node, as well as an upper bound set. However, our version is slightly different from the algorithm by Aneja and Nair: instead of keeping track of the corner points of the boundary of the convex hull, it keeps track of the segments between these corner points. This is useful for other components of our framework. Moreover, we use lexicographic weighted sums in order to compute the two extreme corner points e_1 and e_2, with ϵ being a small enough value for that purpose.

The function *solve* takes three parameters, the first being the objective function to minimize, the second being the set of branching decisions to consider, and the third being an upper bound set. It updates the UB set if necessary and returns the optimal solution to the linear relaxation

Algorithm 2 *bound*$(node, UB)$

```
 1: $C \leftarrow \emptyset$, $E \leftarrow \emptyset$
 2: $e_1 \leftarrow solve(f_1 + \epsilon f_2, node, UB)$
 3: if $e_1 = infeasible$ then
 4:     return $E$
 5: end if
 6: $e_2 \leftarrow solve(f_2 + \epsilon f_1, node, UB)$
 7: $push(C, (e_1, e_2))$
 8: while $C \neq \emptyset$ do
 9:     $(c_1, c_2) \leftarrow pop(C)$
10:     $\alpha \leftarrow (f_1(c_2) - f_1(c_1))/(f_2(c_1) - f_2(c_2))$
11:     $c \leftarrow solve(f_1 + \alpha f_2, node, UB)$
12:     if $f_1(c) + \alpha f_2(c) < f_1(c_1) + \alpha f_2(c_1)$ then
13:         $push(C, (c_1, c))$
14:         $push(C, (c, c_2))$
15:     else
16:         $E \leftarrow E \cup \{(c_1, c_2)\}$
17:     end if
18: end while
19: return $E$
```

of the original problem, considering all branching decisions. In the case where no feasible solution exists, it returns *infeasible*. In practice, the *solve* function can be a call to an existing LP solver or any black box solver. For instance in Section 3 we use column generation. The set C stores all corner point solution pairs between which additional corner point solutions may be found. The method terminates as soon as C is empty and returns E which is the LB set at the given node.

Since integrality constraints are relaxed, a LB set is continuous in the general case. This means that we have to compare a discrete set (upper bound) with a continuous set (lower bound) in order to determine whether a given node can be fathomed. We now explain how we represent LB sets in order to facilitate their filtering using UB sets (line 6 in Algorithm 1).

2.2 Lower bound segments and sets

We consider the set Ξ of all points in the objective space which are associated to feasible solutions to the bi-objective problem at hand. We now introduce the concept of *lower bound segment* that we use to represent LB sets.

Definition 1. *Two non-dominated points (x', y') and (x'', y''), such that $x' < x'' \wedge y' > y''$, define a lower bound segment iff $\{(x, y) \in \Xi | y < ax + b\} = \emptyset$, where $a = (y'' - y')/(x'' - x')$ is the slope of the line defined by the two points and $b = y' - ax'$ is the y-intercept of this same line.*

In other words, all points $(x, y) \in \Xi$ are on or above the line defined by (x', y') and (x'', y'').

The bottom-left part of the boundary of the convex hull of the Pareto set of any bi-objective minimization problem, which defines a valid LB set (cf. Ehrgott and Gandibleux 2006), can be split into lower bound segments (the proof is trivial: assume a given segment from the boundary of the convex hull is not a lower bound segment, then there exist feasible points below this line, which is impossible because this line is part of the convex hull). Since a valid LB set for the linear relaxation of a given problem is also a valid LB set for the original problem, it follows that the bottom-left part of the boundary of the convex hull (of the image of the feasible set in objective space) for the linear relaxation of the problem is a succession of LB segments for the original problem.

To any LB segment defined by points (x', y') and (x'', y'') we can associate the subset of objective space that it *covers*. This subset is the space dominated by any point on that segment, and

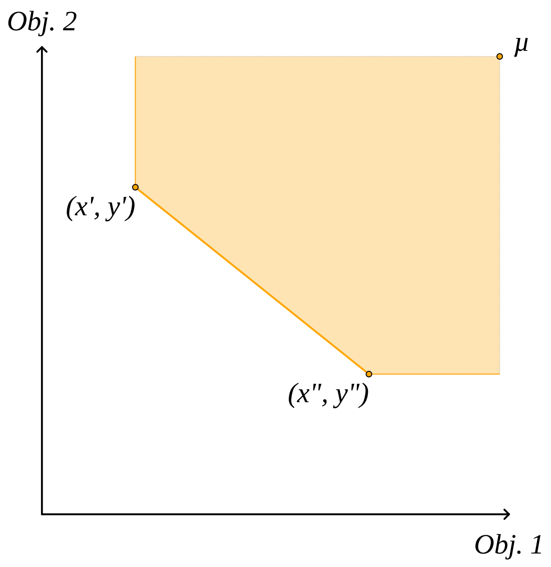

Figure 1: Space dominated by a given segment between points (x', y') and (x'', y''), and local nadir point μ in objective space.

represents the possibility to find feasible solutions (x, y) in the space above the segment such that $x \geq x'$ and $y \geq y''$. Figure 1 depicts this space (shaded).

To each segment s, we associate a top-right corner (or local nadir) point μ of which coordinates are valid single-objective upper bounds, one per objective, on the space dominated by this segment. In the most basic case the upper bounds can be arbitrarily large values. If the two extreme points of the Pareto front are known, we can deduce a tighter valid local nadir point from them. Similarly, we can associate to any LB set a valid local nadir point by considering, for each objective, the maximum value among the local nadir points of all segments in this LB set.

Here we note that we can partition the objective space, therefore the solution space, using values for any of the two objectives, or any linear combination of both with positive weights (as is done in Stidsen et al. 2014). In our BIOBAB, we partition the objective space using different intervals for the first objective, the union of these intervals covering the whole efficient objective space. A graphical interpretation is that the objective space can be split in vertical stripes.

This allows us to improve the initial local nadir point for segments of a connected sequence of segments: for any two segments connected by point (x', y'), we cut from the space covered by the left segment all points (x, y) such that $x \geq x'$, since these points are also covered by the right segment. This is illustrated in Figure 2: the local nadir point for the segment left of x' is now μ'. The local nadir point for the right-most segment is still μ.

By doing so, we partition the objective space covered by the LB set into different LB segments with associated local nadir points.

UB sets can also be used to provide better local nadir points. For instance, following our previous example from Figure 1, consider point (x, y), being the image of a feasible solution in objective space, such that $x \geq x''$ and $y < y''$, then x defines a valid bound on the first objective, since any point (u, v) such that $u \geq x$ and $v \geq y''$ is dominated by (x, y). This example is described graphically in Figure 3: for the depicted segment, using UB point (x, y), the initial local nadir point μ can be improved to μ'.

The fact that LB sets can be reduced using UB sets is the basis for a bi-objective branching rule that is presented in Section 2.4. We now describe the fathoming rule we derive from the concept of LB segments.

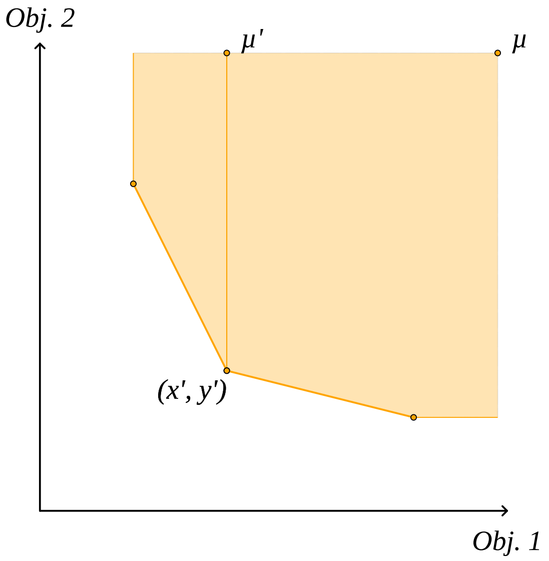

Figure 2: Space covered by the left segment can be reduced to points left of x', since all points right of x' are also covered by the right segment.

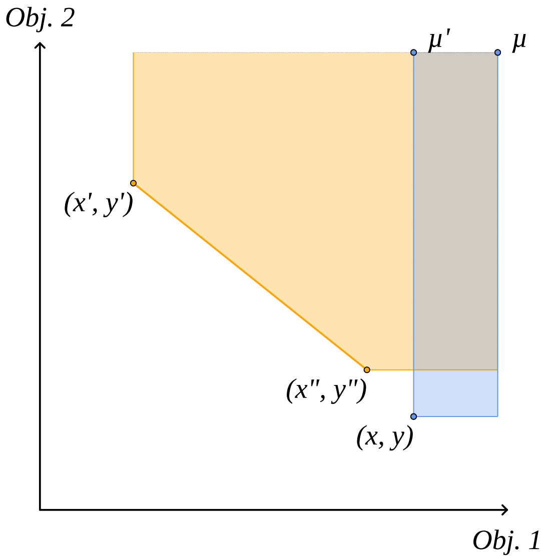

Figure 3: Space dominated by the segment between points (x', y') and (x'', y'') can be reduced by removing from it the space dominated by upper bound point (x, y).

2.3 Bound set comparisons and node fathoming

One major difficulty in previous approaches lies in the evaluation of the dominance of a given LB set by a given UB set. Multiple fathoming rules have been developed over the years (see Belotti et al. 2013, for the current state of the art). We now introduce the fathoming rule we use.

As seen above, it is possible to split a LB set into LB segments (with associated local nadir points). If we establish that each segment of a current LB set is dominated by the UB set, then we also establish that the whole LB set is dominated, therefore the current node of the branch-and-bound tree can be fathomed. In order to determine if a LB segment is dominated by an UB set, we simply subtract the space dominated by the UB set from the space covered by the LB segment: if the remaining space is empty then the LB segment is dominated, otherwise it may be tightened (as shown for example in Figure 3). The latter case is exploited by our branching strategy which is described in Section 2.4.

Algorithm 3 details how to subtract the space covered by an UB point u from a LB segment s. It shares some similarities with the dominance rules used by Vincent et al. (2013), but considers more cases. Notably, it updates the local nadir point even in cases where u does not dominate any point on s. We use object-oriented notation as follows. A point p has two attributes corresponding to its coordinates, $p.x$ and $p.y$. A segment s has five attributes: $s.p_1$ and $s.p_2$ define the two extremities of the segment, $s.c$ defines the local nadir point for this segment, $s.a$ is the slope and $s.b$ the y-intercept. Constructors are defined as $Point(x, y)$ and $Segment(p_1, p_2, c)$. Lines 2-7 deal with the case where u does not dominate any point on segment s. In that case, u may still be above $s.p_1$ or to the right of $s.p_2$, in which cases there is potential for improving $s.c$. If u is to the right of $s.p_2$ and below $s.p_2$ (Lines 5-7), we are then in the situation depicted in Figure 3 and $s.c.x$ can be reduced to $min(s.c.x, u.x)$. Similarly, if u is above $s.p_1$ and to the left of p_1 (Lines 3-4), $s.c.y$ can then be reduced to $min(s.c.y, u.y)$. Lines 8-15 deal with all the cases when at least one part of s is dominated by u. There are four different cases: (i) u does not dominate $s.p_1$ (Lines 9-11), (ii) u does not dominate $s.p_2$ (Lines 12-14), (iii) u dominates neither $s.p_1$ nor $s.p_2$, resulting in two new segments (both clauses in Lines 9-11 and 12-14 are matched), and (iv) u dominates both $s.p_1$ and $s.p_2$. In case (iv) no clause is matched in the algorithm, therefore an empty set is returned.

Algorithm 3 $filterSegment(s, u)$: filter LB segment s using UB point u

```
1: $S \leftarrow \emptyset$
2: if $u.y \geq s.a \cdot u.x + s.b \vee u.y \geq s.p_1.y \vee u.x \geq s.p_2.x$ then
3:     if $u.x \leq s.p_1.x$ then
4:         $S \leftarrow S \cup \{Segment(p_1, p_2, Point(c.x, min(c.y, u.y)))\}$
5:     else if $u.y \leq s.p_2.y$ then
6:         $S \leftarrow S \cup \{Segment(p_1, p_2, Point(min(c.x, u.x), c.y))\}$
7:     end if
8: else
9:     if $u.x > s.p_1.x$ then
10:        $S \leftarrow S \cup \{Segment(p_1, Point(u.x, s.a \cdot u.x + s.b), Point(u.x, c.y))\}$
11:    end if
12:    if $u.y > p_2.y$ then
13:        $S \leftarrow S \cup \{Segment(Point((u.y - s.b)/s.a, u.y), p_2, Point(c.x, u.y))\}$
14:    end if
15: end if
16: return $S$
```

Now in order to filter a whole LB set using an UB set, we filter each segment in the LB set with each point in the UB set. This procedure is described in Algorithm 4. If the remaining LB set is empty then the node can be fathomed, otherwise it must be kept.

Algorithm 4 *filterLB*(LB,UB): filter LB set LB using UB set UB

```
1: $S \leftarrow LB$
2: for all $u \in UB$ do
3:     $S' \leftarrow \emptyset$
4:     for all $s \in S$ do
5:         $S' \leftarrow S' \cup filterSegment(s, u)$
6:     end for
7:     $S \leftarrow S'$
8: end for
9: return $S$
```

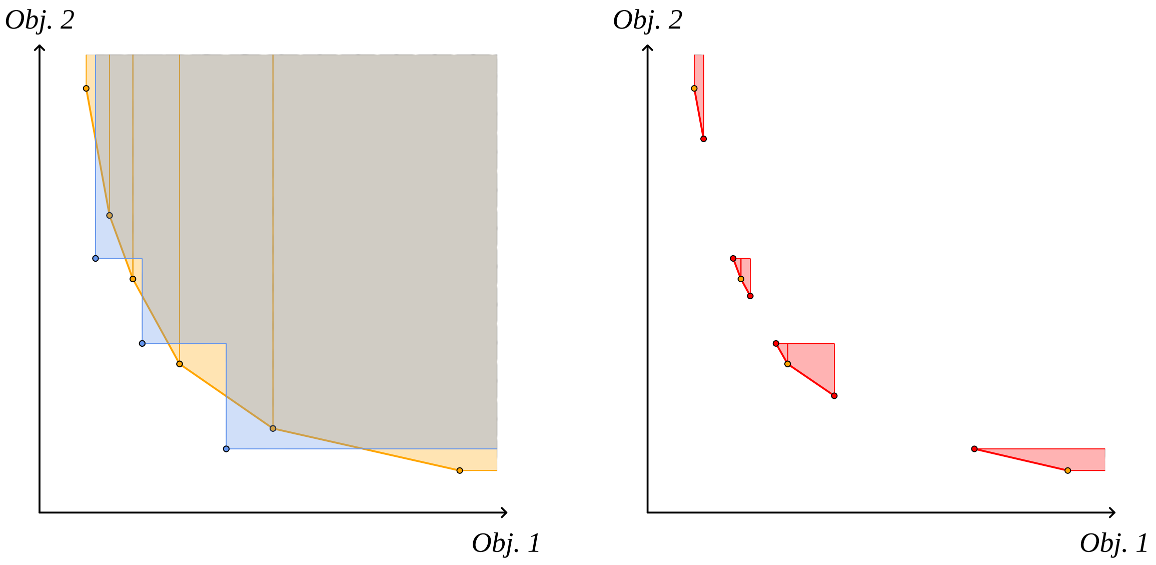

Figure 4: LB set is partially dominated by UB set, the node cannot be fathomed.

Because of branching decisions it can happen that the LP at a given node is infeasible, in that case the node is also fathomed.

2.4 A branching rule for bi-objective optimization

When dealing with bi-objective bound sets, a major annoyance is that the LB set at a given node of the branch-and-bound tree may be partially dominated but not completely, therefore the node cannot be fathomed. In that case we have to keep the whole node, although only the non-dominated part of the LB set is in fact relevant. This is illustrated in Figure 4.

In that scenario the traditional approach is to simply keep branching and bounding. However in the illustrated example it is clear that most of the objective space for this node is in fact already dominated by the UB set, therefore it is a waste of resources to calculate the new LB set for the whole objective space after branching, when we are in fact interested in a small portion of this objective space. We propose to branch on objective space in order to remedy that issue. More precisely, each continuous part of the filtered LB set gives rise to a new branch and the space between those parts is simply ignored. To generate the branch for a given continuous part, we consider its local nadir point μ and add two cutting planes so that both objectives are better than the value at this local nadir point: $f_1(x) \leq \mu.x$ and $f_2(x) \leq \mu.y$. Algorithm 5 gives the *branch* function of our branch-and-bound framework. Whenever the LB set after filtering is discontinuous, we perform objective space branching. Otherwise, we use problem specific branching rules. In the

Algorithm 5 *branch*(LB)

```
1: $N \leftarrow \emptyset$
2: if $LB$ is discontinuous then
3:     for all continuous subset $S \in LB$ do
4:         $\mu = Point(\max_{s \in S}(s.c.x), \max_{s \in S}(s.c.y))$
5:         $decision \leftarrow (f_1(x) \leq \mu.x \wedge f_2(x) \leq \mu.y)$
6:         $N \leftarrow N \cup \{decision\}$
7:     end for
8: else
9:     $N \leftarrow problemSpecificBranching(LB)$
10: end if
11: return $N$
```

example of Figure 4, function *branch* generates four different branches.

2.5 Improvements based on the integrality of objective functions

When solving integer programs, in many cases, objective values of feasible integer solutions only take integer values. It is in practice almost always possible to use integer numbers. The reasons include the fact that LP solvers have precision limitations, and that time-efficient floating-point numbers representations also have precision limitations. So rounding floating-point numbers is almost always inevitable, and if numbers are rounded to d decimals then they may as well be multiplied by 10^d and considered integers. We note that this property is exploited by other methods as well. For instance, the ϵ-constraint framework uses a known ϵ value which in our case would be 1. In some cases, it is even possible to use values higher than 1, as long as these values are valid: for instance if every coefficient for a given objective function is integer, then the greatest common divisor of these coefficients can be used as a valid value, just like it could be used in an ϵ-constraint framework. For the sake of simplicity and without loss of generality, we consider in the following that this valid value is 1. We now explore possibilities to speed up our bi-objective branch-and-bound by exploiting the fact that objective values of feasible solutions are always integer.

Any given LB segment covers a continuous part of the objective space, including continuous regions not containing any point with integer coordinates. These continuous regions can be disregarded during the search. This is illustrated in Figure 5, where dashed lines represent integer values for each objective. This general idea can be exploited in several ways in order to speed up the search.

First, any LB segment which does not cover any integer vector can be discarded. This can be tested in $\mathcal{O}(1)$: segment s covers integer vectors iff $(s.p1 = s.p2 \vee \lfloor s.c.y \rfloor \geq s.a\lfloor s.c.x \rfloor + s.b) \wedge \lfloor s.c.y \rfloor \geq \lceil s.p_2.y \rceil \wedge \lfloor s.c.x \rfloor \geq \lceil s.p_1.x \rceil$. This is tested right after filtering (Algorithm 1, line 6): only the segments in LB that cover an integer point are kept.

Second, we can use the fact that feasible integer solutions have integer objective values in order to produce tighter LB sets faster than the regular way. Let (x', y') and (x'', y'') be two extreme points from the convex hull boundary, defining a segment that needs to be processed in our function *bound* described in Algorithm 2. Processing this segment means determining whether it is part of the convex hull boundary or not. However it can happen that this segment already defines a valid LB segment, even if we do not know whether it is part of the convex hull boundary. In such a case, there is no need to investigate whether it is on the convex hull boundary. Such a situation can happen if there is no point with integer coordinates in the triangle defined by (x', y'), (x'', y'') and (x', y''). Point (x', y'') can in fact also be disregarded because it dominates both (x', y') and (x'', y''), although it is already established that they are both non-dominated; therefore (x', y'') cannot be associated to a feasible solution. Moreover, if the segment is not on the convex hull

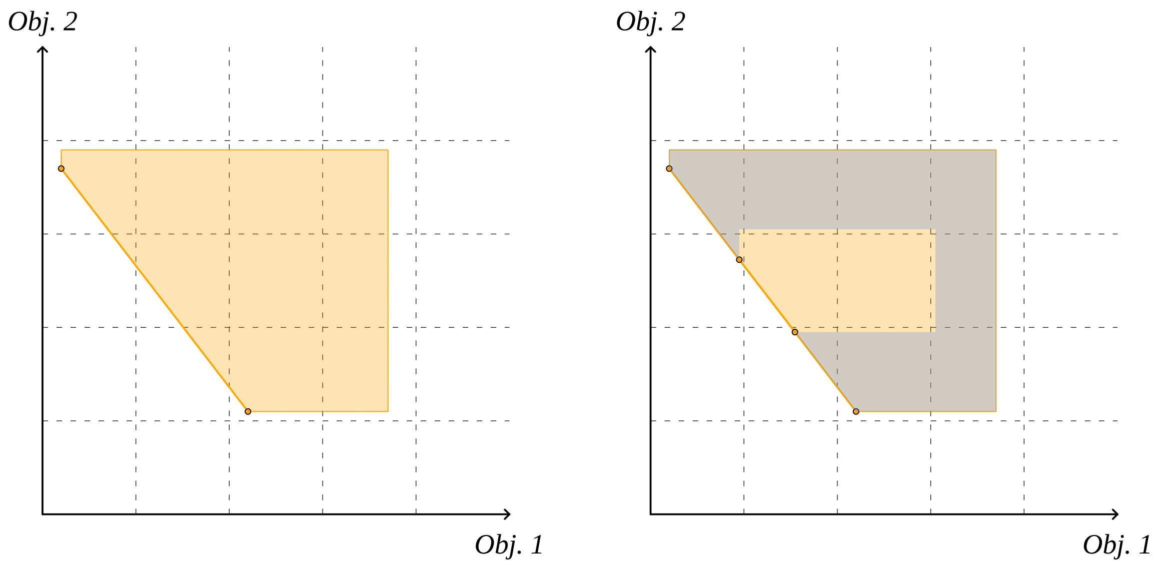

Figure 5: The space covered by a LB segment contains continuous regions which do not contain any point with integer coordinates; these parts are irrelevant to the search.

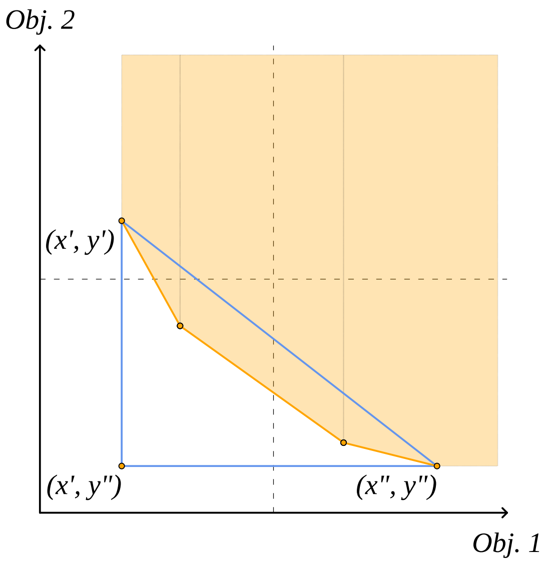

Figure 6: Segment between points (x', y') and (x'', y'') defines a valid LB segment even though it is not part of the convex hull boundary, because there is no vector of integer coordinates in the triangle defined by (x', y'), (x'', y'') and (x', y'').

boundary then it defines a tighter lower bound than the real convex hull boundary between (x', y') and (x'', y''). This situation is illustrated in Figure 6, where the line connecting (x', y') and (x'', y'') is not part of the convex hull boundary but still defines a valid lower bound segment.

As previously, testing for the existence of an integer vector in the given triangle can be achieved in $\mathcal{O}(1)$ and sometimes allows to skip some stages of the algorithm while providing a tighter LB set. In the notation of Algorithm 2, this means that after retrieving the next segment to be processed (c_1, c_2) from C (line 9, Algorithm 2), we perform the following check: Let s denote the segment defined by (c_1, c_2) (where c_1 corresponds (x', y') and c_2 to (x'', y'') in the Figure 6). If the point $(\lceil c_1.x\rceil, \lceil c_2.y\rceil)$ is above the line connecting c_1 and c_2, i.e. $\lceil c_2.y\rceil \geq s.b + s.a\lceil c_1.x\rceil$, then the triangle derived from (c_1, c_2) does not contain any integer point. In this case, (c_1, c_2) is appended to E in Algorithm 2 and lines 10–16 are skipped.

Third, when branching on objective space, it is also possible to exploit the integrality of objective values for feasible integer solutions. If we are in the situation of branching on objective space it is typically because part of the LB set has been dominated by the UB set. In that case, we already know that the given local nadir point of any LB segment is in fact also dominated by the UB set (this is actually how this local nadir point is calculated, see Algorithm 3). So this local nadir point can be tightened. Figure 7 illustrates this principle: After filtering the LB segment with (x', y') and (x'', y'') (Figures 7(a) and 7(b)) we obtain a reduced segment. The new local nadir point is actually dominated by both (x', y') and (x'', y'') (Figure 7(c)). Moreover, each point in the area which is in a darker shade (Figure 7(d)) is either (i) dominated by (x', y') and/or (x'', y'') or (ii) not integer. Therefore we can disregard the whole area. This can be achieved by subtracting any value ρ such that $0 \leq \rho < 1$ from both coordinates of the local nadir point μ after filtering, thus producing an improved nadir point μ'. In order to implement this idea in our algorithm, we simply subtract ρ from both coordinates of each upper bound point u used to filter each segment s in Algorithm 4.

3 The bi-objective team orienteering problem with time windows

In order to validate our approach, we apply it to the bi-objective team orienteering problem with time windows (BITOPTW). We first introduce the problem. Then we explain how to calculate lower bound sets using column generation and upper bound sets using heuristics. We also present other problem-specific mechanisms used in our BIOBAB framework such as branching rules.

3.1 Problem definition

In so-called orienteering problems, a term coined by Chao et al. (1996), the aim is to maximize the total score collected at the different locations (also referred to as control points) within a given time or cost budget. Having its roots in the orienteering game, in its very basic version, only a single route is planned. Considering a team of players, one route has to be planned for each of the players and each individual route may not exceed the budget constraint. To account for opening and closing times at the different locations time windows are added, recent application areas being found in mobile tourist guides. Considering the total travel cost as another objective, in addition to the total collected score or profit, gives rise to the bi-objective team orienteering problem with time windows (BITOPTW). In the context of tourism, cost can be seen as the effort required to visit points of interest, while profit is linked to the benefit taken from visiting said points, based on a user's personal preferences. For further variants and application areas we refer to Vansteenwegen et al. (2009).

More formally, the BITOPTW is defined on a directed graph $G = (V, A)$, where A is the set of arcs and V the set of vertices, representing the starting location (vertex 0), the ending location (vertex $n + 1$) and n control points. Each control point i is associated with a score S_i, a service time d_i and a time window $[e_i, l_i]$. Each route $k \in K$, with $|K| = m$, has to start at location 0

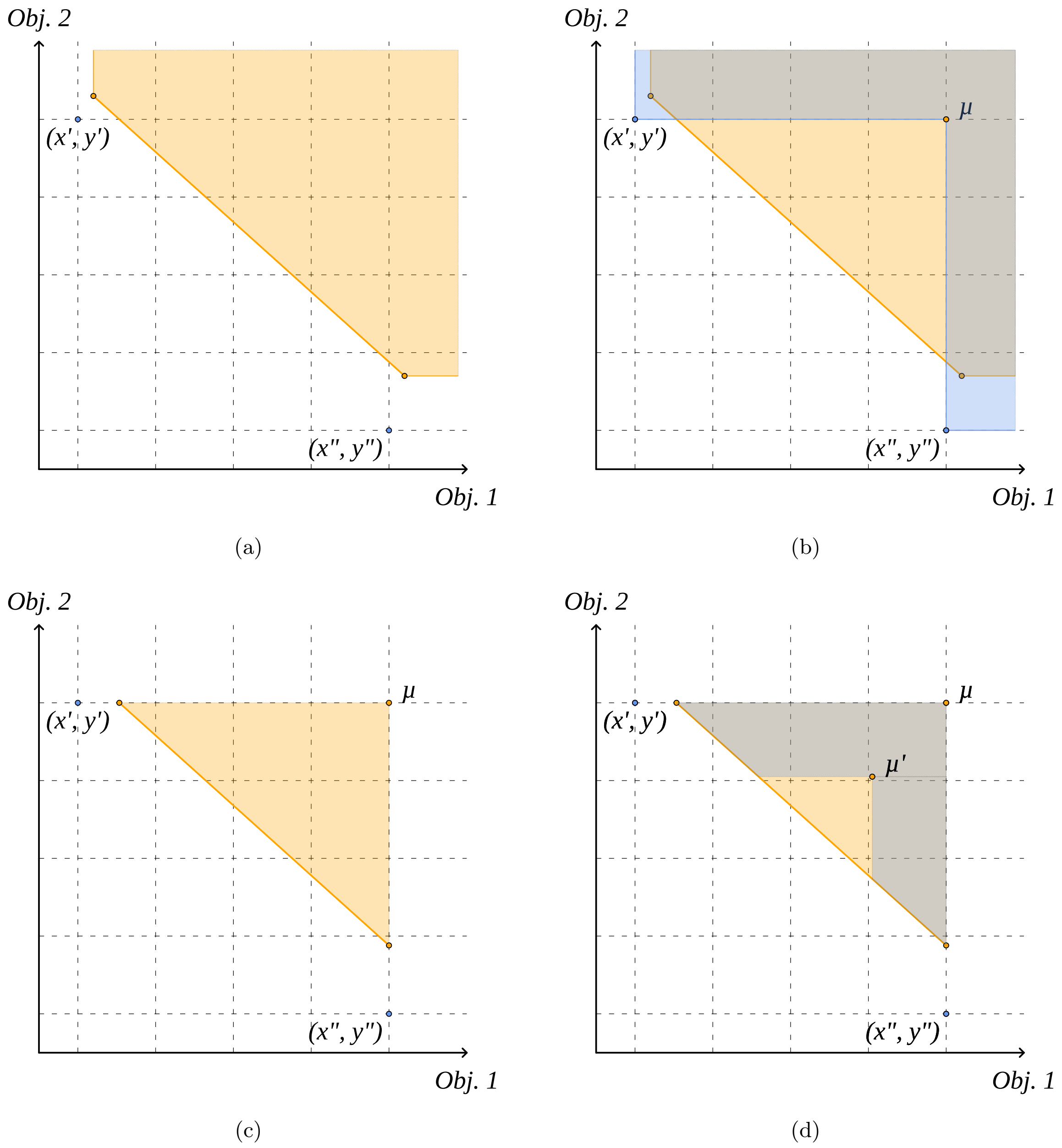

Figure 7: Part of the filtered LB is either still dominated by the UB or not containing any integer vector (darker shaded area in the fourth picture).

and end at location $n+1$ and each arc (i,j) is associated with travel cost c_{ij} and travel time t_{ij}. The aim is to maximize the total collected score and to simultaneously minimize the total travel cost. Using binary decision variables $z_i \in \{0,1\}$ equal to 1 if location i is visited and 0 otherwise, and $y_{ijk} \in \{0,1\}$ equal to 1 if arc (i,j) is traversed by route k and 0 otherwise, and continuous variables B_{ik}, denoting the beginning of service at i by route k, we formally define the BITOPTW as follows:

$$\min \sum_{k\in K}\sum_{(i,j)\in A} c_{ij}y_{ijk} \tag{1}$$

$$\max \sum_{i\in V\setminus\{0,n+1\}} S_i z_i \tag{2}$$

subject to:

$$\sum_{j\in V\setminus\{0\}} y_{0jk} = 1 \qquad \forall k\in K \tag{3}$$

$$\sum_{i\in V\setminus\{n+1\}} y_{i,n+1,k} = 1 \qquad \forall k\in K \tag{4}$$

$$\sum_{k\in K}\sum_{j\in V\setminus\{n+1\}} y_{jik} = z_i \qquad \forall i\in V\setminus\{0,n+1\} \tag{5}$$

$$\sum_{j\in V\setminus\{n+1\}} y_{jik} - \sum_{j\in V\setminus\{0\}} y_{ijk} = 0 \qquad \forall k\in K, i\in V\setminus\{0,n+1\} \tag{6}$$

$$(B_{ik}+d_i+t_{ij})y_{ijk} \le B_{jk} \qquad \forall k\in K,(i,j)\in A \tag{7}$$

$$e_i \le B_{ik} \le l_i \qquad \forall k\in K, i\in V \tag{8}$$

$$y_{ijk}\in\{0,1\} \qquad \forall k\in K,(i,j)\in A \tag{9}$$

$$z_i\in\{0,1\} \qquad \forall i\in V\setminus\{0,n+1\} \tag{10}$$

Objective function (1) minimizes the total routing costs while objective function (2) maximizes the total collected profit. Constraints (3) and (4) make sure that each route starts at the defined starting point and ends at the correct ending point. Constraints (5) link the binary decision variables and (6) ensure connectivity for visited nodes. Constraints (7) set the time variables and (8) make sure that time windows are respected. We note that constraints (7) are not linear but they can easily be linearized using big M terms.

We solve the BITOPTW with the proposed BIOBAB, where lower bound sets are generated by means of column generation. This is described next.

3.2 Lower bound sets: column generation

Since state-of-the-art exact methods for single-objective routing problems mostly rely on column generation based techniques (see, e.g. Baldacci et al. 2012), we also generate lower bound sets for the BITOPTW by means of column generation.

Let p_r denote the total score or profit achieved by route r, c_r the total travel cost of route r, and let a_{ir} indicate whether location i is visited by route r ($a_{ir}=1$) or not ($a_{ir}=0$). Using binary variables x_r equal to 1 if route r is selected from the set Ω of all feasible routes, the BITOPTW can also be formulated as a path-based model:

$$f_1 = \min \sum_{r\in\Omega} c_r x_r \tag{11}$$

$$f_2 = \max \sum_{r\in\Omega} p_r x_r \tag{12}$$

$$\sum_{r\in\Omega} a_{ir}x_r \leq 1 \qquad \forall i \in N \tag{13}$$

$$\sum_{r\in\Omega} x_r = m \tag{14}$$

$$x_r \in \{0,1\} \qquad \forall r \in \Omega. \tag{15}$$

Relaxing integrality requirements on the x_r variables, we replace constraints (15) with:

$$x_r \geq 0 \qquad \forall r \in \Omega \tag{16}$$

We also combine the two objective functions into a weighted sum (w_1 and w_2 giving the respective non-negative weights):

$$\min \sum_{r\in\Omega}(w_1c_r - w_2p_r)x_r \tag{17}$$

We thus obtain a single objective linear problem that can be solved by means of column generation, which allows us to compute LB sets using Algorithm 2. In column generation (see, e.g. Desrosiers and Lübbecke 2005), in each iteration a subset of promising columns is generated and appended to the restricted set of columns Ω' and the single objective linear program is re-solved on Ω'. Column generation continues as long as new promising columns exist. Otherwise, the optimal solution has been found (i.e. in that case, the optimal solution of the single objective linear program on Ω' is also the optimal solution of the single objective linear program on Ω). Promising columns are identified using dual information. Let π_i denote the dual variable associated with constraint (13) for a given i and α the dual variable associated with constraint (14), the pricing subproblem we have to solve corresponds to:

$$\min\ w_1c_r - w_2p_r - \sum_{i\in N} a_{ir}\pi_i - \alpha \tag{18}$$

subject to constraints (3)–(10), omitting subscript k. It is an elementary shortest path problem with resource constraints that can be solved by means of a labeling algorithm (cf. Feillet et al. 2004). In our labeling algorithm, a label carries the following information: the node the label is associated with, the time consumption until that node, the reduced cost so far, which nodes have been visited along the path leading to the node, and a pointer to the parent label. In order to compute the reduced cost of the path associated with a given label, we use a reduced cost matrix that is generated before the labeling algorithm is called. The reduced cost of arc (i,j) is $\bar{c}_{ij} = w_1c_{ij} - w_2S_i - \pi_i$, where if $i = 0$, π_i is replaced by α. Since the aim of this paper is not to investigate the most efficient pricing algorithm, we refrain from adopting all enhancements proposed in the literature (e.g., Righini and Salani 2009).

Note that during the execution of BIOBAB we keep all previously generated columns in the column pool. We only temporarily deactivate those columns that are incompatible with current branching decisions. When branching on objective space, it can happen that the current pool of columns does not allow to produce a feasible solution. This can be due to two reasons: either (i) because there exists no feasible solution to the current problem or (ii) because there exists a feasible solution but it cannot be reached with the currently available columns. In order to fix this issue and guarantee feasibility, we use a dummy column which allows to satisfy every constraint from the current problem, including the branching decisions described in Section 3.4. This column covers all mandatory control points, does not cover any forbidden control point, uses all the available vehicles, has a cost inferior to the maximum allowed cost and a profit superior to the minimum allowed profit. Using this column is penalized in the objective function, so that the column generation procedure converges to feasible solutions that do not use the dummy column. Assuming the variable associated to this dummy column is x_D, the objective function described in Equation (17) is modified as follows:

$$\min \sum_{r\in\Omega}(w_1 c_r - w_2 p_r)x_r + Mx_D, \tag{19}$$

where M is an arbitrarily large number. The dummy column is only activated when no feasible solution can be found, and it is systematically deactivated after column generation converges. At this point, if x_D has a strictly positive value then there does not exist a feasible solution that satisfies all the branching decisions.

3.3 Upper bound sets: multi-directional local search

Using starting upper bound sets has two advantages: uninteresting branches of the search tree may be pruned earlier and valid starting columns for our lower bound computations are available. For the BITOPTW we generate upper bound sets by means of multi-directional local search (MDLS). MDLS has been proposed in Tricoire (2012) where it is shown that MDLS is able to provide competitive results for the multi-objective multi-dimensional knapsack problem, the bi-objective set packing problem and the bi-objective orienteering problem. It obtains approximations of the Pareto frontier of either comparable or improved quality. Its main idea is to use several different local search procedures, each considering only one of the objectives. At each iteration, a solution is selected from the current set of non-dominated solutions. Then, local search is performed on this solution for each of the objectives in turn. Therefore, each call to a local search algorithm aims at a different "direction". Using the thus produced new solutions, the non-dominated set is updated and the next iteration is performed.

In this work we use large neighborhood search (LNS) (Shaw 1998, Ropke and Pisinger 2006) as single-objective local search. The LNS we apply works as follows. In every iteration, first, the number of nodes to be removed from the current solution is randomly chosen from $[1, 0.4\nu]$, where ν gives the number of currently visited nodes. Then, a destroy and a repair operator are chosen randomly and applied to the current solution. Destroy operators remove control points from the solution while repair operators insert control points into routes. In terms of destroy operators, we use a greedy and a random operator. In terms of repair operators, we use a greedy, a random and a "nil" operator. The greedy operators choose the points that help to improve the currently employed objective function the most or to deteriorate it the least. The random operators randomly select the points to be removed or inserted. The "nil" operator does not do anything. It is used to allow the application of a destroy operator without subsequent repairing, which leads to improved solutions when optimizing cost. In the repair phase, once a point has been selected for insertion, it is inserted at the cheapest position in terms of travel cost. The random destroy – random repair operator pair is not employed.

In addition, we add noise to the greedy operators by multiplying the selection value (either the difference in terms of profit or cost, depending on the objective being currently optimized) by a factor randomly chosen in $[0.4, 1]$.

3.4 Tree search

We now explain how the search tree is constructed, i.e. which branching rules are used, as well as how it is explored. In terms of tree exploration strategy, we use breadth-first search. We also implemented depth-first as well as best-first strategies, where the total area covered by a lower bound set is used as score for the best-first strategy. In a preliminary set of experiments neither of these performed significantly worse than breadth-first, but breadth-first still provided the best performance overall. In terms of branching rules, whenever the lower bound after filtering is discontinuous, we perform objective space branching (see Section 2.4). Since objective space branching involves setting bounds on both objectives, we include such constraints right from the start (initially setting the respective bounds to infinity) and later update them according to the branching decisions. In order to properly incorporate dual information from these two constraints into the subproblem, we modify the reduced cost matrix accordingly. In the case where the lower bound is not discontinuous, we either branch on control points or on arcs (Line 9 in Algorithm 5): First, we check if a control point is visited a fractional number of times. Since each lower bound represents a set of solutions, the number of times a control point is visited may take different values within the same LB set. In order to select a control point to branch on, we consider the extreme points of the boundary of the convex hull defining the current LB set, each of these being associated to a different solution. We then average the number of visits for each control point over this set of solutions. The control point with the average number of visits closest to 0.5 is then selected for branching. In the case where each control point is visited 0 or 1 time on average, we check for arcs that are traversed a fractional number of times on average, following the same procedure. Branching on control points can be achieved by modifying constraint (13) in the master problem. In order to force the visit of control point i, this constraint becomes $\sum\limits_{r \in \Omega} a_{ir} x_r \geq 1$. In order to forbid the visit of i, this constraint becomes $\sum\limits_{r \in \Omega} a_{ir} x_r \leq 0$. As is usual in branch-and-price for routing, branching on arcs involves adding constraints to the subproblem.

4 Experimental study

All algorithms are implemented in C++, compiled with g++ version 4.6.2 and run on an 2.67 GHz Intel Xeon CPU. We use ILOG Cplex 12.5 to solve linear programs in the column generation. We restrict Cplex to a single thread and set the parameter EpRHS to 10^{-7} in order to avoid numerical stability issues. For MDLS, we use the code available at http://prolog.univie.ac.at/research/MDLS and extend it, as explained above, to tackle the BITOPTW. All the code used in our experiments will be available online at http://prolog.univie.ac.at/research/BIOBAB.

To test our algorithms, we use instances from the data set of Righini and Salani (2009). In this data set, orienteering instances are derived from 29 instances for the vehicle routing problem with time windows (VRPTW) (Solomon 1987) and from 10 instances for the multi-depot periodic vehicle routing problem (MDPVRP) (Cordeau et al. 1997). From those derived from VRPTW instances, we use instance c101_100 (clustered locations), r101_100 (randomly spread locations) and rc101_100 (mix of clustered and random locations). From those derived from MDPVRP instances, we use instance pr01 (randomly spread locations). To obtain smaller instances, we consider the first $n = \{15, 20, 25, 30\}$ control points and $m = \{1, 2, 3, 4\}$ vehicles. Distances are given with 2-digit precision in the original instances, so we multiply them by 100 and only work with integer numbers.

4.1 Performance of the bi-objective branch-and-bound

We compare our BIOBAB algorithm to the ϵ-constraint algorithm, which is depicted in Algorithm 6, where P denotes the set of Pareto optimal solutions and δ is small enough to ensure that the objective is lexicographic. Function *solveMIP* performs a single-objective branch-and-price using the specified objective function and additional ϵ-constraint. Both algorithms use as starting

Algorithm 6 ϵ-constraint algorithm

```
$P \leftarrow \emptyset$
$\epsilon\text{-}constraint \leftarrow f_2 \leq \infty$
while MIP can be solved do
    $x \leftarrow solveMIP(f_1 + \delta f_2, \epsilon\text{-}constraint)$
    $P \leftarrow P \cup x$
    $\epsilon\text{-}constraint \leftarrow f_2 \leq f_2(x) - \epsilon$
end while
return P
```

	BIOBAB				ϵ-constraint			
n	#solved	$t_B < t_\epsilon$	$\frac{t_\epsilon}{t_B} > 1.5$	$\frac{t_\epsilon}{t_B} > 3$	#solved	$t_\epsilon < t_B$	$\frac{t_B}{t_\epsilon} > 1.5$	$\frac{t_B}{t_\epsilon} > 3$
15	16	3	0	0	16	13	1	0
20	16	1	1	1	16	15	4	0
25	14	8	2	1	14	6	0	0
30	13	7	2	0	13	6	0	0
35	11	6	2	1	11	5	0	0
Total	70	25	7	3	70	45	5	0

Table 1: CPU time comparison. "t_B": CPU time required by BIOBAB. "t_ϵ": CPU time required by ϵ-constraint method.

upper bound set the non-dominated union of the sets produced by 10 MDLS runs. Each MDLS run consists of 25,000 iterations. One iteration consists in two local search calls (one per objective), where one local search call consists of destroying and repairing the solution ten times. The longest MDLS run takes 2.13 seconds and in all cases all 10 MDLS runs are completed within 15 seconds. A total time limit of two hours is given to both BIOBAB and ϵ-constraint. The same code is used in both cases. There are only two differences, related to the way the methods work. First, the bounding procedure considers only one lexicographic objective for the ϵ-constraint framework, whereas it computes a bi-objective LB set for the BIOBAB (using a succession of weighted sums). Second, the tree search is called repeatedly in the ϵ-constraint framework (once for each point of the Pareto set), whereas the bi-objective branch-and-bound consists of a single run. Other than that, the code is exactly the same for both methods (although some parts of it are not actually called by the ϵ-constraint framework). This means for instance that the ϵ-constraint framework benefits from the existing UB set and can use it to fathom subtrees.

In Table 1 the CPU effort of the two methods is compared per instance group: n gives the number of control points of the instance group; #solved the number of instances solved within two hours (out of 16 per group); $t_B < t_\epsilon$ the number of times BIOBAB is faster than the ϵ-constraint algorithm, $\frac{t_\epsilon}{t_B} > 1.5$ the number of times BIOBAB is faster by a factor greater than 1.5; and $\frac{t_\epsilon}{t_B} > 3$ the number of times BIOBAB is faster by a factor greater than 3. In the remaining four columns the same information is provided taking the CPU time needed by the ϵ-constraint scheme as a basis and comparing it to the CPU time required by BIOBAB. We observe that both methods solve the same number (70 out of 80) instances to optimality. Comparing the CPU effort needed to solve these instances, the table shows that CPU times are usually of the same magnitude. The ϵ-constraint algorithm is faster for the smallest instances ($n = 15$ and $n = 20$), for five instances it is more than 1.5 times faster but less than 3 times. For all other instances, the ϵ-constraint algorithm is never more than 1.5 times faster while the BIOBAB is. Across all instances BIOBAB is more than 1.5 times faster in seven cases, and more than 3 times faster in three (instances rc101 with $m = 4$ and $n = 20$, $n = 25$, $n = 35$). In the case of instance rc101 with $m = 4$ and $n = 25$ the BIOBAB is even more than 10 times faster than the ϵ-constraint algorithm.

Per instance results underlying Table 1 can be found in the Appendix in Tables 4, 5, 6, 7 and 8, where the first four columns indicate the base instance (name), number of control points (n), number of vehicles (m) as well as the number of non-dominated solutions in the Pareto set (# sol.). Then for each method we indicate the number of columns generated in the process of producing the Pareto set (# col.), the number of nodes explored in the tree search algorithms (# nodes) and the total CPU time required to generate the Pareto set, in seconds (CPU (s)). If the CPU time limit of two hours is not sufficient to generate the whole Pareto set, "> limit" is indicated instead. The CPU time of the fastest method is bold-faced. These tables show that, usually, both methods generate a very similar number of columns. The difference in terms of the number of nodes explored is generally larger (in the case of the ϵ-constraint scheme, this entry gives the total number

	BIOBAB (no UB)				ϵ-constraint (no UB)			
n	#solved	$t_B < t_\epsilon$	$\frac{t_\epsilon}{t_B} > 1.5$	$\frac{t_\epsilon}{t_B} > 3$	#solved	$t_\epsilon < t_B$	$\frac{t_B}{t_\epsilon} > 1.5$	$\frac{t_B}{t_\epsilon} > 3$
15	16	13	9	0	16	3	0	0
20	16	14	11	5	16	2	0	0
25	14	14	7	2	13	0	0	0
30	13	12	9	2	11	1	0	0
35	10	9	6	2	7	1	0	0
Total	69	62	42	11	63	7	0	0

Table 2: CPU time comparison, no starting upper bound set. "t_B": CPU time required by BIOBAB. "t_ϵ": CPU time required by ϵ-constraint method.

of nodes explored across all single objective branch-and-bound calls). Fewer nodes are explored in the BIOBAB, and much fewer nodes in those cases where BIOBAB provides much shorter CPU times. There is a trade-off relationship between the time spent on the evaluation of each node and the number of nodes that can be explored in the same CPU time. Since the evaluation of each node is more costly in the BIOBAB than in the single objective branch-and-bound algorithm of the ϵ-constraint scheme, this result is expected and it indicates that the proposed ingredients help to keep the tree size small. They are evaluated in further detail later on.

Since in practice it is not always possible to produce good upper bound sets quickly, we reproduce the previous experiment but without giving a starting upper bound set to the BIOBAB and ϵ-constraint algorithms. Table 2 summarizes these results. For each instance class the same information as in Table 1 is given. Detailed tables are provided in the Appendix (Tables 9, 10, 11, 12 and 13).

The lack of starting upper bound set affects the performance of the ϵ-constraint algorithm more than that of the BIOBAB. Overall, the BIOBAB solves more instances than the ϵ-constraint and offers better performance. It never happens that the ϵ-constraint is much faster than the BIOBAB (it is never more than 1.5 times faster). However, it does happen that the BIOBAB is much faster. In 42 cases the BIOBAB is more than 1.5 times faster and in 11 cases it is more than 3 times faster. Sometimes the gap in performance is striking, like for example for instance rc101 with 25 control points and 3 vehicles (see Table 11 in the Appendix).

It is also interesting to compare the quality of the fronts produced by BIOBAB and by the ϵ-constraint framework when they are both suboptimal, i.e. when they both run out of time. As an example, we compare the upper bound set produced by both methods when they are called on what seems to be the hardest instance, c101 with 35 control points and 4 vehicles. Figure 8 shows a comparison of the final upper bound sets provided by the two algorithms when run without starting upper bound set. The ϵ-constraint framework only provides Pareto-optimal solutions. Although the bi-objective branch-and-bound does not find all Pareto optimal solutions (as evidenced in the top-left corner of the diagram), it does yield a better spread of solutions and many solutions it provides are actually part of the Pareto set.

We also conduct experiments to assess the impact of all the speedups which rely on the fact that feasible integer solutions have integer objective values, as well as the impact of objective space branching. In Table 3, we compare the results of BIOBAB without objective space branching (OSB) and the results of BIOBAB without using improvements exploiting integrality of objective values (INT) to the results obtained by means of BIOBAB using all ingredients. The different performance measures are given per instance class (indicated by the number of control points n). For each of the two versions, we first report the number of instances solved (out of 16), the average factor by which the number of columns ($\times col_B$), the number of nodes ($\times nodes_B$), and the CPU time ($\times t_B$) produced by BIOBAB using all ingredients has to be multiplied in order to obtain this measure for the respective version (per instance results are given in Tables 14, 15, 16,

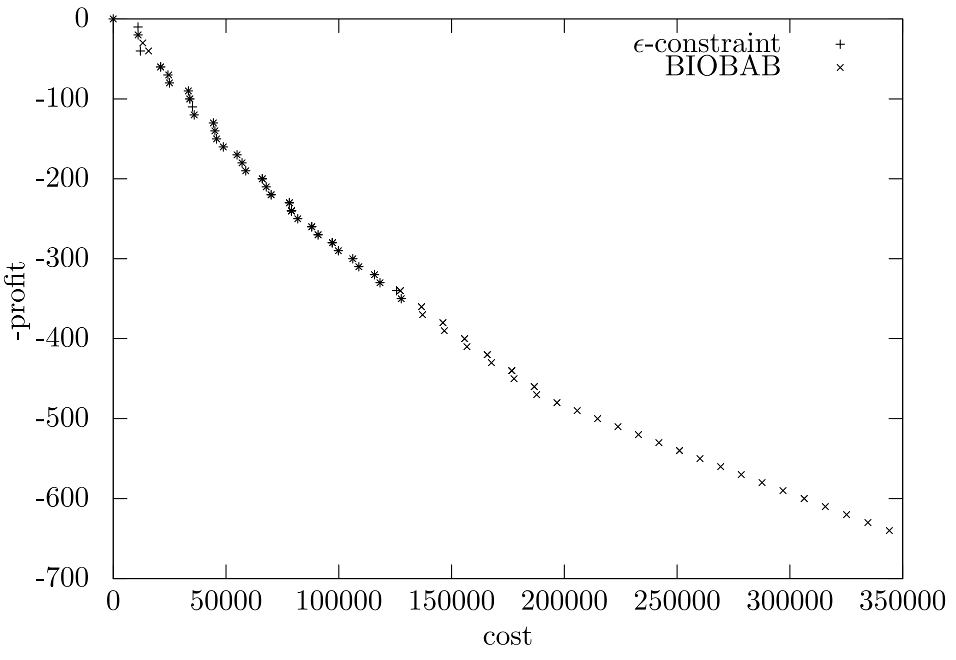

Figure 8: Compared upper bound sets found by bi-objective branch-and-bound and by ϵ-constraint framework when they are not able to provide Pareto sets and no initial upper bound set is provided. Instance c101_100, $n = 35$, $m = 4$.

17 and 18 in the Appendix). Results clearly show that both types of speedups do improve the performance of the algorithm. Without OBS, only 66 instead of 70 instances can be solved: the algorithm is on average 23% slower and produces 7% more columns but only 60% of the number of nodes are explored. The last result is expected: branching on objective space increases the number of nodes and therefore the size of the search tree. However, it decreases the time spent in evaluating each of these nodes and therefore it has a positive impact on the overall performance on the algorithm. Exploiting integrality of objective values leads to even more important speedups. Without this ingredient, BIOBAB produces more than 33% more columns and 40% more nodes, which translates into CPU times that are increased by a factor of more than 3. This increase in CPU times entails that 7 instances that can be solved by BIOBAB cannot be solved if speedups based on the integrality of objective values are deactivated.

4.2 Performance of the MDLS

We now evaluate the quality of the upper bound sets provided by the MDLS algorithm. As mentioned above, producing these bound sets never takes more than 15 seconds for 10 MDLS runs, so we assess here the quality of the non-dominated union of all 10 sets produced over these 10 runs. In order to evaluate their quality, we compare these sets to reference sets. Such reference sets are either exact Pareto sets when they are available, or the non-dominated union of the upper bound sets produced by BIOBAB and ϵ-constraint (when run with the result of MDLS as starting UB set). We then compare the hypervolume of the UB set produced by the MDLS with the hypervolume of the reference set. We report, for each test instance, the portion of the hypervolume of the reference set that is obtained when using MDLS, and the CPU time required for all 10 MDLS runs. We also report the hypervolume portion obtained by the BIOBAB and ϵ-constraint, when run without a starting UB set, as well as the CPU time they require. All results are reported in Tables 19, 20, 21, 22 and 23 (which can all be found in the appendix). Over all experiments, the worst hypervolume ratio achieved by MDLS is 95.76% and the average hypervolume ratio is 99.49%, so it is safe to say

	BIOBAB (no OSB)				BIOBAB (no INT)			
n	#solved	$\times col_B$	$\times nodes_B$	$\times t_B$	#solved	$\times col_B$	$\times nodes_B$	$\times t_B$
15	16	1.07	0.57	1.25	16	1.21	1.21	2.60
20	16	1.07	0.58	1.20	16	1.36	1.31	2.96
25	13	1.08	0.56	1.21	13	1.48	1.50	3.79
30	12	1.11	0.58	1.35	9	1.27	1.48	3.00
35	9	1.03	0.62	1.15	9	1.32	1.53	3.56
Avg	-	1.07	0.58	1.23		1.33	1.40	3.18
Total	66	-	-	-	63	-	-	-

Table 3: Comparison of BIOBAB without objective space branching (OSB) and of BIOBAB without improvements based on the integrality of objective values (INT) to BIOBAB. "$\times$": factor by which the respective performance measure for BIOBAB has to be multiplied in order to obtain this measure for BIOBAB (no OSB) or BIOBAB (no INT). "$col_.$": number of columns. "$nodes_.$": number of nodes. "$t_.$": CPU time. "B": BIOBAB

that MDLS always provides good solution sets in a short amount of time. For the harder instances, i.e. those that cannot be solved optimally by the exact methods within the allotted time, MDLS provides results very near to BIOBAB (even slightly better in two cases), and much better than the ϵ-constraint, also in a much shorter time. We conclude that MDLS is a suitable approach for providing good solution sets quickly.

5 Conclusion

We have introduced a new algorithm for bi-objective optimization. Although we only apply it to integer programs, we believe that it can be extended and applied to mixed integer programs as well. Such an extension will be the subject of future research. The algorithm is a generalization of the single-objective branch-and-bound algorithm to the bi-objective context. Although it is not the first attempt to generalize branch-and-bound to the bi-objective context, most previous attempts rely on specific problem properties. Moreover, very few of them exploit the fact that there are several objectives in order to speed up the search. Our algorithm is generic and uses a branching rule relying on the fact that there are two objectives, which can also be used with any bi-objective optimization problem. In order to demonstrate the versatility of our framework, we have applied it to a routing problem where the lower bound linear program is solved by means of column generation. This means that we also provide the first ever bi-objective branch-and-price algorithm. This algorithm is sped up by using heuristic upper bound sets, which we produce through a multi-directional local search algorithm. Experimental results show that our method is competitive with the ϵ-constraint method, which is a standard benchmark for multi-objective optimization. Moreover, when a starting upper bound set is not provided, our algorithm outperforms the ϵ-constraint method. We also note that our method explores the whole front of solutions in parallel rather than going from one side to the other (as the ϵ-constraint does), which means that it can be stopped early and still provide a good spread of solutions.

One research perspective lies in generalizing the concept of lower bound set to more than two objectives, thus allowing exact approaches relying on this notion for more than two objectives. However this will raise the issue of the cardinality of the efficient set, which typically increases exponentially with the number of objectives. Therefore another research perspective lies in deriving approximation methods for multi-objective optimization based on exact procedures, with the goal of sacrificing neither the quality of solutions nor the readability of the produced set of solutions.

Acknowledgments Financial support from the Austrian Science Fund (FWF): P23589-N13 is gratefully acknowledged.

A Additional experimental data

Instance				ϵ-constraint			BIOBAB		
name	n	m	# sol.	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)
c101	15	1	20	3911	1807	**10.00**	3933	1586	14.68
r101	15	1	11	162	286	**0.18**	158	250	0.20
rc101	15	1	13	849	666	**1.06**	826	564	1.58
pr01	15	1	28	3208	829	5.50	2918	580	**4.50**
c101	15	2	28	3818	6927	**44.87**	3902	6457	62.42
r101	15	2	27	166	932	**0.55**	169	771	0.62
rc101	15	2	27	882	3536	**5.60**	867	3228	7.66
pr01	15	2	38	1492	1331	4.98	1660	939	**4.46**
c101	15	3	28	3823	6881	**43.43**	3784	6457	58.26
r101	15	3	47	174	2136	**1.14**	178	1737	1.68
rc101	15	3	27	903	4414	**7.18**	867	3214	8.21
pr01	15	3	39	1282	1314	4.23	1382	948	**3.66**
c101	15	4	28	3838	6881	**46.32**	3821	6457	58.04
r101	15	4	53	159	2836	**1.48**	175	2070	2.53
rc101	15	4	27	878	3976	**6.85**	920	3215	7.40
pr01	15	4	39	1296	1314	**3.07**	1448	948	3.70

Table 4: Experimental results ($n = 15$).

Instance				ϵ-constraint			BIOBAB		
name	n	m	# sol.	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)
c101	20	1	25	10011	2953	**49.59**	9946	2457	62.59
r101	20	1	12	251	305	**0.24**	252	226	0.37
rc101	20	1	13	1343	882	**2.52**	1269	780	3.16
pr01	20	1	41	7748	1810	**26.91**	7280	1192	30.24
c101	20	2	41	9959	26443	**550.61**	9747	24092	689.96
r101	20	2	29	290	1104	**0.85**	295	843	1.38
rc101	20	2	29	1630	7458	**22.45**	1578	6189	24.60
pr01	20	2	73	5897	4168	**46.11**	6283	2351	54.79
c101	20	3	42	10007	31982	**647.46**	9595	30162	867.26
r101	20	3	46	293	2659	**2.22**	301	1915	3.57
rc101	20	3	36	1674	11475	**39.72**	1686	10204	45.41
pr01	20	3	69	4302	4028	**44.36**	5194	3001	48.53
c101	20	4	42	9648	31258	**787.05**	9592	30183	858.66
r101	20	4	58	298	3769	**3.40**	330	2871	5.21
rc101	20	4	38	1945	61385	260.30	1788	11738	**63.82**
pr01	20	4	69	4441	4010	**44.42**	4702	3036	44.98

Table 5: Experimental results ($n = 20$).

Instance				ϵ-constraint			BIOBAB		
name	n	m	# sol.	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)
c101	25	1	23	15812	3610	128.10	14263	3183	**119.59**
r101	25	1	12	378	359	0.48	384	252	**0.46**
rc101	25	1	15	1851	1165	**4.94**	1921	1022	6.75
pr01	25	1	73	20501	4494	260.56	19469	2912	**197.53**
c101	25	2	55	26678	57560	**4960.46**	25701	52255	5833.09
r101	25	2	34	431	1759	**2.52**	458	1140	2.66
rc101	25	2	34	2484	11314	**60.47**	2451	10394	63.39
pr01	25	2	121	19083	13160	698.25	18300	6602	**630.56**
c101	25	3	-	18593	138918	> limit	17694	109850	> limit
r101	25	3	56	473	3341	**4.52**	493	2264	5.11
rc101	25	3	52	2502	37503	**199.44**	2506	29134	232.25
pr01	25	3	135	12179	11954	411.52	14041	6564	**337.53**
c101	25	4	-	18850	125178	> limit	17547	110268	> limit
r101	25	4	67	469	5338	8.14	514	3675	**8.05**
rc101	25	4	56	3256	552520	3223.76	3014	36299	**312.09**
pr01	25	4	135	11903	12046	368.29	12002	6871	**241.87**

Table 6: Experimental results ($n = 25$).

Instance				ϵ-constraint			BIOBAB		
name	n	m	# sol.	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)
c101	30	1	24	25354	4210	306.24	22342	3749	**244.44**
r101	30	1	19	851	702	**1.60**	916	590	2.29
rc101	30	1	15	1918	1195	**5.38**	1950	1044	7.76
pr01	30	1	95	57558	7960	1838.20	55231	4424	**1424.19**
c101	30	2	-	42733	64073	> limit	32345	35689	> limit
r101	30	2	43	989	3190	**6.98**	1038	2477	10.22
rc101	30	2	34	2571	11728	**62.67**	2577	10835	75.67
pr01	30	2	157	47903	26332	4550.93	42703	12875	**3168.92**
c101	30	3	-	26716	104668	> limit	19462	63727	> limit
r101	30	3	65	995	5840	17.45	1022	4462	**15.08**
rc101	30	3	53	2699	35491	**234.35**	2699	32762	249.90
pr01	30	3	173	32890	28388	3140.76	32840	13579	**1916.33**
c101	30	4	-	26265	96482	> limit	18998	61861	> limit
r101	30	4	80	1038	8749	23.88	1050	5708	**23.08**
rc101	30	4	59	3295	67498	**512.71**	3118	55306	525.86
pr01	30	4	173	32921	30224	3083.09	30987	13540	**1698.85**

Table 7: Experimental results ($n = 30$).

Instance				ϵ-constraint			BIOBAB		
name	n	m	# sol.	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)
c101	35	1	26	31843	4870	500.55	28178	4265	**348.86**
r101	35	1	25	1336	976	**3.46**	1371	809	4.04
rc101	35	1	15	2335	1245	**7.99**	2322	1072	8.44
pr01	35	1	92	103477	9233	6116.43	93414	6142	**3926.60**
c101	35	2	-	45116	55476	> limit	28873	42694	> limit
r101	35	2	47	1478	4328	15.87	1454	2638	**13.17**
rc101	35	2	34	3235	12488	**98.31**	3185	11892	108.99
pr01	35	2	-	64285	27381	> limit	48821	8991	> limit
c101	35	3	-	30507	80876	> limit	22896	39442	> limit
r101	35	3	81	1571	9277	33.57	1492	5212	**27.79**
rc101	35	3	53	3543	44841	388.04	3414	39084	**377.46**
pr01	35	3	200	49253	32877	**4730.05**	56837	22583	7073.75
c101	35	4	-	29705	85297	> limit	18886	63637	> limit
r101	35	4	115	1577	15161	**55.73**	1605	8641	57.51
rc101	35	4	67	3859	499352	4797.14	3610	87785	**1136.95**
pr01	35	4	-	44772	36275	> limit	46970	23458	> limit

Table 8: Experimental results ($n = 35$).

Instance				ϵ-constraint (no UB)			BIOBAB (no UB)		
name	n	m	# sol.	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)
c101_100	15	1	20	4642	2309	22.54	3859	1599	**16.52**
r101_100	15	1	11	171	376	**0.20**	169	241	0.30
rc101_100	15	1	13	941	910	**1.53**	831	571	1.70
pr01	15	1	28	4933	1447	16.10	3232	599	**6.44**
c101_100	15	2	28	4445	12263	133.00	3757	6515	**66.21**
r101_100	15	2	27	176	1564	0.77	183	785	**0.70**
rc101_100	15	2	27	1018	10166	19.10	910	3331	**11.06**
pr01	15	2	38	1736	2623	9.36	1997	1021	**5.88**
c101_100	15	3	28	4361	12265	106.88	3782	6507	**78.10**
r101_100	15	3	47	172	4190	2.60	182	1966	**2.51**
rc101_100	15	3	27	1016	10166	23.14	907	3335	**11.14**
pr01	15	3	39	1622	2680	8.85	1488	1019	**4.38**
c101_100	15	4	28	4379	12263	107.57	3871	6486	**64.44**
r101_100	15	4	53	165	4966	**2.36**	175	2319	2.97
rc101_100	15	4	27	1010	10166	19.26	897	3335	**9.88**
pr01	15	4	39	1609	2690	7.29	1576	1018	**4.77**

Table 9: Experimental results, no starting upper bound set ($n = 15$).

Instance				ϵ-constraint (no UB)			BIOBAB (no UB)		
name	n	m	# sol.	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)
c101_100	20	1	25	12915	3623	110.73	9963	2511	**78.35**
r101_100	20	1	12	276	349	**0.44**	270	205	0.45
rc101_100	20	1	13	1554	1258	**3.68**	1425	817	4.07
pr01	20	1	41	13565	3350	104.73	8568	1278	**42.13**
c101_100	20	2	41	12993	75333	3133.47	10041	24257	**853.94**
r101_100	20	2	29	325	2034	1.83	294	873	**1.77**
rc101_100	20	2	29	1882	22938	90.96	1634	6499	**35.13**
pr01	20	2	73	7792	6870	115.67	6609	2829	**83.46**
c101_100	20	3	42	12605	97030	3888.29	9737	30448	**973.65**
r101_100	20	3	46	322	5565	5.56	311	2171	**3.12**
rc101_100	20	3	36	2001	102591	421.33	1768	10637	**60.34**
pr01	20	3	69	6368	7924	159.92	5748	4148	**95.88**
c101_100	20	4	42	12746	97030	3535.45	9943	30428	**899.54**
r101_100	20	4	58	314	8049	7.91	333	2821	**4.71**
rc101_100	20	4	38	2090	116887	396.18	1844	11983	**73.52**
pr01	20	4	69	5832	7886	144.97	5582	4129	**85.99**

Table 10: Experimental results, no starting upper bound set ($n = 20$).

Instance				ϵ-constraint (no UB)			BIOBAB (no UB)		
name	n	m	# sol.	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)
c101_100	25	1	24	20023	4240	228.77	15342	3225	**163.16**
r101_100	25	1	12	409	409	0.70	410	250	**0.49**
rc101_100	25	1	15	2089	1561	7.81	1875	1022	**7.04**
pr01	25	1	73	30068	8012	733.86	20838	3175	**262.86**
c101_100	25	2	56	31549	88801	> limit	25588	52470	**6137.06**
r101_100	25	2	34	488	2525	2.89	475	1284	**2.75**
rc101_100	25	2	34	2959	41796	232.09	2501	10559	**86.61**
pr01	25	2	121	23125	18702	1133.68	19820	7407	**815.92**
c101_100	25	3	-	20787	137629	> limit	17988	99187	> limit
r101_100	25	3	56	502	6023	8.72	513	2768	**6.09**
rc101_100	25	3	53	3037	704289	4223.95	2610	31065	**262.59**
pr01	25	3	135	14345	20124	847.68	14389	7787	**448.60**
c101_100	25	4	-	20838	146782	> limit	17487	91069	> limit
r101_100	25	4	67	493	10821	14.11	502	4119	**9.11**
rc101_100	25	4	56	3371	1057940	6432.59	3043	38794	**377.07**
pr01	25	4	135	14042	20010	756.67	13045	7773	**376.85**

Table 11: Experimental results, no starting upper bound set ($n = 25$).

Instance				ϵ-constraint (no UB)			BIOBAB (no UB)		
name	n	m	# sol.	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)
c101_100	30	1	24	33101	4916	567.93	22952	3794	**289.96**
r101_100	30	1	19	930	818	2.23	907	596	**1.92**
rc101_100	30	1	15	2081	1597	**7.03**	2002	1040	7.85
pr01	30	1	95	81343	14104	4410.73	58735	4795	**1809.00**
c101_100	30	2	-	33869	72490	> limit	32203	35120	> limit
r101_100	30	2	43	1097	4994	13.06	1017	2570	**10.51**
rc101_100	30	2	34	3043	42352	267.05	2677	11085	**101.15**
pr01	30	2	157	57995	33380	> limit	50926	13513	**4904.03**
c101_100	30	3	-	26311	97175	> limit	20206	56974	> limit
r101_100	30	3	65	1113	11958	32.81	1060	4645	**18.65**
rc101_100	30	3	53	3204	709383	4815.11	2777	32690	**344.53**
pr01	30	3	173	38020	39938	5729.96	37226	14504	**2541.46**
c101_100	30	4	-	26414	92725	> limit	20026	58049	> limit
r101_100	30	4	80	1100	17397	48.68	1106	6705	**30.75**
rc101_100	30	4	58	3108	1127408	> limit	3240	61031	**756.36**
pr01	30	4	173	36801	39670	4777.36	35220	14478	**2213.33**

Table 12: Experimental results, no starting upper bound set ($n = 30$).

Instance				ϵ-constraint (no UB)			BIOBAB (no UB)		
name	n	m	# sol.	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)
c101_100	35	1	28	41793	5792	785.37	29135	4256	**442.74**
r101_100	35	1	25	1415	1142	**3.86**	1334	799	5.12
rc101_100	35	1	15	2473	1619	13.17	2371	1068	**10.83**
pr01	35	1	92	110159	11394	> limit	95986	6265	**4441.00**
c101_100	35	2	-	44049	51866	> limit	29450	41524	> limit
r101_100	35	2	47	1521	5508	23.40	1455	2768	**13.55**
rc101_100	35	2	34	3789	43470	365.53	3360	12182	**134.63**
pr01	35	2	-	63019	26876	> limit	47645	5441	> limit
c101_100	35	3	-	32000	82915	> limit	25011	38469	> limit
r101_100	35	3	81	1653	14345	54.06	1574	5581	**38.53**
rc101_100	35	3	53	4196	733789	> limit	3557	38821	**466.93**
pr01	35	3	-	47207	39585	> limit	38131	18431	> limit
c101_100	35	4	-	30345	85255	> limit	21196	60456	> limit
r101_100	35	4	114	1672	23521	92.71	1671	8782	**63.61**
rc101_100	35	4	67	3921	941912	> limit	3749	91292	**1202.45**
pr01	35	4	-	46389	39132	> limit	34534	20671	> limit

Table 13: Experimental results, no starting upper bound set ($n = 35$).

Instance				BIOBAB			BIOBAB (no OSB)			BIOBAB (no INT)		
name	n	m	# sol.	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)
c101	15	1	20	3933	1586	**14.68**	4159	951	19.88	4554	2206	52.17
r101	15	1	11	158	250	**0.20**	162	117	0.22	173	193	0.26
rc101	15	1	13	826	564	**1.58**	849	339	1.78	907	578	2.32
pr01	15	1	28	2918	580	**4.50**	3256	343	4.66	4905	778	21.47
c101	15	2	28	3902	6457	**62.42**	4135	4165	73.99	4334	7801	163.03
r101	15	2	27	169	771	**0.62**	159	307	0.71	184	777	1.32
rc101	15	2	27	867	3228	**7.66**	979	1933	14.29	1001	3791	16.22
pr01	15	2	38	1660	939	**4.46**	1906	585	6.15	2474	1219	15.77
c101	15	3	28	3784	6457	**58.26**	4128	4165	69.11	4496	7816	148.00
r101	15	3	47	178	1737	**1.68**	178	783	1.70	193	2230	4.06
rc101	15	3	27	867	3214	**8.21**	933	1947	11.64	973	3761	15.39
pr01	15	3	39	1382	948	**3.66**	1616	591	4.06	2012	1249	12.38
c101	15	4	28	3821	6457	**58.04**	4319	4169	71.50	4350	7818	158.04
r101	15	4	53	175	2070	2.53	176	973	**2.40**	178	2905	5.38
rc101	15	4	27	920	3215	**7.40**	944	1947	12.05	970	3738	13.43
pr01	15	4	39	1448	948	**3.70**	1544	591	4.89	2006	1249	11.94

Table 14: Experimental results with and without improvements. "no OSB": no objective space branching. "no INT": no speedup based on the integrality of objective values for feasible solutions ($n = 15$).

Instance				BIOBAB			BIOBAB (no OSB)			BIOBAB (no INT)		
name	n	m	# sol.	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)
c101	20	1	25	9946	2457	**62.59**	8686	1337	65.20	12208	3183	287.55
r101	20	1	12	252	226	0.37	255	121	**0.32**	288	188	0.55
rc101	20	1	13	1269	780	**3.16**	1415	437	3.61	1409	856	6.49
pr01	20	1	41	7280	1192	**30.24**	9130	621	34.79	17820	2088	191.20
c101	20	2	41	9747	24092	**689.96**	10027	15429	817.37	12391	29270	1663.92
r101	20	2	29	295	843	1.38	294	459	**1.17**	328	974	2.54
rc101	20	2	29	1578	6189	**24.60**	1725	3891	39.81	1858	7721	53.63
pr01	20	2	73	6283	2351	**54.79**	7892	1287	79.70	10594	3417	239.32
c101	20	3	42	9595	30162	**867.26**	10986	20679	1192.52	12053	35483	2083.30
r101	20	3	46	301	1915	3.57	309	919	**2.58**	342	2601	7.02
rc101	20	3	36	1686	10204	**45.41**	1880	7133	85.68	2075	14085	126.59
pr01	20	3	69	5194	3001	48.53	5261	1641	**46.86**	8863	4693	172.77
c101	20	4	42	9592	30183	**858.66**	10891	20679	1322.52	12171	35476	2102.71
r101	20	4	58	330	2871	5.21	301	1415	**4.22**	324	3980	8.95
rc101	20	4	38	1788	11738	**63.82**	1897	7749	90.47	2121	16800	179.20
pr01	20	4	69	4702	3036	**44.98**	5263	1719	50.02	8613	4424	199.80

Table 15: Experimental results with and without improvements. "no OSB": no objective space branching. "no INT": no speedup based on the integrality of objective values for feasible solutions ($n = 20$).

Instance				BIOBAB			BIOBAB (no OSB)			BIOBAB (no INT)		
name	n	m	# sol.	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)
c101	25	1	24	14263	3183	**119.59**	16364	2221	222.81	23999	4583	767.76
r101	25	1	12	384	252	0.46	362	147	**0.38**	449	253	0.71
rc101	25	1	15	1921	1022	**6.75**	2066	705	7.52	2274	1165	15.14
pr01	25	1	73	19469	2912	**197.53**	24643	1285	258.29	48491	5147	1675.87
c101	25	2	56	25701	52255	**5833.09**	25810	31462	> limit	24543	33286	> limit
r101	25	2	34	458	1140	2.66	456	589	**1.90**	556	1455	5.24
rc101	25	2	34	2451	10394	**63.39**	2695	7045	118.07	2916	14208	166.33
pr01	25	2	121	18300	6602	**630.56**	22238	2649	694.68	35120	11379	2945.39
c101	25	3	-	17694	109850	> limit	18718	55013	> limit	19825	65492	> limit
r101	25	3	56	493	2264	5.11	472	1153	**4.46**	596	3391	12.29
rc101	25	3	53	2506	29134	**232.25**	2691	20171	313.94	2859	41579	462.75
pr01	25	3	135	14041	6564	337.53	14865	2969	**330.75**	23666	12180	1480.50
c101	25	4	-	17547	110268	> limit	18589	61507	> limit	19816	68731	> limit
r101	25	4	67	514	3675	**8.05**	491	1819	8.76	603	5676	22.51
rc101	25	4	56	3014	36299	**312.09**	3051	25451	437.30	3619	57579	1050.50
pr01	25	4	135	12002	6871	**241.87**	14457	3061	313.62	24175	12498	1532.16

Table 16: Experimental results with and without improvements. "no OSB": no objective space branching. "no INT": no speedup based on the integrality of objective values for feasible solutions ($n = 25$).

Instance				BIOBAB			BIOBAB (no OSB)			BIOBAB (no INT)		
name	n	m	# sol.	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)
c101	30	1	24	22342	3749	**244.44**	24421	2685	442.52	38968	5587	1672.51
r101	30	1	19	916	590	2.29	884	367	**2.02**	1110	861	5.16
rc101	30	1	15	1950	1044	**7.76**	2166	717	9.78	2273	1188	15.72
pr01	30	1	95	55231	4424	**1424.19**	66511	2241	1986.40	82379	6988	> limit
c101	30	2	-	32345	35689	> limit	32262	7708	> limit	25933	11816	> limit
r101	30	2	43	1038	2477	10.22	1012	1121	**8.15**	1258	3267	17.12
rc101	30	2	34	2577	10835	**75.67**	2885	7599	123.37	3052	15253	198.41
pr01	30	2	157	42703	12875	**3168.92**	60079	5245	6097.04	46697	12590	> limit
c101	30	3	-	19462	63727	> limit	23144	6840	> limit	23014	31385	> limit
r101	30	3	65	1022	4462	15.08	1008	2111	**14.08**	1276	6580	36.87
rc101	30	3	53	2699	32762	**249.90**	2986	22725	429.52	3196	51070	686.47
pr01	30	3	173	32840	13579	**1916.33**	40748	5723	2761.32	53615	20412	> limit
c101	30	4	-	18998	61861	> limit	21574	8262	> limit	21899	35801	> limit
r101	30	4	80	1050	5708	23.08	1097	3025	**20.30**	1297	8970	56.31
rc101	30	4	60	3118	55306	**525.86**	3319	41601	775.95	3897	103043	2079.66
pr01	30	4	173	30987	13540	**1698.85**	38689	5771	2537.24	52825	20526	> limit

Table 17: Experimental results with and without improvements. "no OSB": no objective space branching. "no INT": no speedup based on the integrality of objective values for feasible solutions ($n = 30$).

Instance				BIOBAB			BIOBAB (no OSB)			BIOBAB (no INT)		
name	n	m	# sol.	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)	# col.	# nodes	CPU (s)
c101	35	1	28	28178	4265	**348.86**	29268	2809	477.57	53428	6788	4400.88
r101	35	1	25	1371	809	4.04	1323	409	**3.39**	1739	1137	8.34
rc101	35	1	15	2322	1072	**8.44**	2437	743	10.46	2667	1226	16.27
pr01	35	1	92	93414	6142	**3926.60**	107216	2083	> limit	72624	5033	> limit
c101	35	2	-	28873	42694	> limit	35648	8990	> limit	31077	27462	> limit
r101	35	2	47	1454	2638	13.17	1411	1327	**10.77**	1903	3958	33.66
rc101	35	2	34	3185	11892	**108.99**	3520	8493	162.77	3872	16785	249.33
pr01	35	2	-	48821	8991	> limit	59265	2168	> limit	52370	5561	> limit
c101	35	3	-	22896	39442	> limit	30330	7339	> limit	28124	28735	> limit
r101	35	3	81	1492	5212	27.79	1478	2545	**25.35**	2000	9214	84.07
rc101	35	3	53	3414	39084	**377.46**	3771	29497	547.79	4016	61298	970.62
pr01	35	3	200	56837	22583	**7073.75**	51132	7750	> limit	39100	19100	> limit
c101	35	4	-	18886	63637	> limit	23901	8434	> limit	22117	48570	> limit
r101	35	4	115	1605	8641	57.51	1623	3963	**48.90**	2049	15141	136.16
rc101	35	4	67	3610	87785	**1136.95**	3877	72589	1571.44	4368	146439	2966.57
pr01	35	4	-	46970	23458	> limit	52881	9616	> limit	38325	21656	> limit

Table 18: Experimental results with and without improvements. "no OSB": no objective space branching. "no INT": no speedup based on the integrality of objective values for feasible solutions ($n = 35$).

Instance			BIOBAB (noUB)		ϵ-constraint (noUB)		MDLS	
name	n	m	H %	CPU (s)	H %	CPU (s)	H %	CPU (s)
c101	15	1	100	16.52	100	22.54	99.98	5.88
r101	15	1	100	0.3	100	0.2	99.41	2.89
rc101	15	1	100	1.7	100	1.53	99.90	3.37
pr01	15	1	100	6.44	100	16.1	99.51	3.71
c101	15	2	100	66.21	100	133	99.87	6.41
r101	15	2	100	0.7	100	0.77	99.92	4.13
rc101	15	2	100	11.06	100	19.1	99.97	6.08
pr01	15	2	100	5.88	100	9.36	99.60	4.69
c101	15	3	100	78.1	100	106.88	99.91	5.18
r101	15	3	100	2.51	100	2.6	99.77	4.47
rc101	15	3	100	11.14	100	23.14	99.20	5.68
pr01	15	3	100	4.38	100	8.85	99.90	5.08
c101	15	4	100	64.44	100	107.57	99.91	5.71
r101	15	4	100	2.97	100	2.36	99.40	4.9
rc101	15	4	100	9.88	100	19.26	99.56	5.25
pr01	15	4	100	4.77	100	7.29	99.90	5.35

Table 19: Comparison of MDLS hypervolume ratio (H %) and CPU time against exact methods (n=15)

Instance			BIOBAB (noUB)		ϵ-constraint (noUB)		MDLS	
name	n	m	H %	CPU (s)	H %	CPU (s)	H %	CPU (s)
c101	20	1	100	78.35	100	110.73	99.91	4.66
r101	20	1	100	0.45	100	0.44	99.75	3.34
rc101	20	1	100	4.07	100	3.68	100	4.07
pr01	20	1	100	42.13	100	104.73	98.57	5.8
c101	20	2	100	853.94	100	3133.47	99.88	6.91
r101	20	2	100	1.77	100	1.83	99.93	4.62
rc101	20	2	100	35.13	100	90.96	99.67	5.28
pr01	20	2	100	83.46	100	115.67	99.39	6.22
c101	20	3	100	973.65	100	3888.29	99.89	7.37
r101	20	3	100	3.12	100	5.56	99.85	6.14
rc101	20	3	100	60.34	100	421.33	99.85	6.34
pr01	20	3	100	95.88	100	159.92	99.66	6.96
c101	20	4	100	899.54	100	3535.45	99.88	7.32
r101	20	4	100	4.71	100	7.91	99.68	5.66
rc101	20	4	100	73.52	100	396.18	96.45	7.08
pr01	20	4	100	85.99	100	144.97	99.61	7.17

Table 20: Comparison of MDLS hypervolume ratio (H %) and CPU time against exact methods (n=20)

Instance			BIOBAB (noUB)		ϵ-constraint (noUB)		MDLS	
name	n	m	H %	CPU (s)	H %	CPU (s)	H %	CPU (s)
c101	25	1	100	163.16	100	228.77	100	5.34
r101	25	1	100	0.49	100	0.7	99.75	4.47
rc101	25	1	100	7.04	100	7.81	99.93	4.31
pr01	25	1	100	262.86	100	733.86	98.30	7.3
c101	25	2	100	6137.06	94.86	7200.08	100	9.06
r101	25	2	100	2.75	100	2.89	99.68	5.8
rc101	25	2	100	86.61	100	232.09	99.63	5.92
pr01	25	2	100	815.92	100	1133.68	98.67	7.8
c101	25	3	100	7200.09	92.43	7200.02	99.99	9.72
r101	25	3	100	6.09	100	8.72	99.83	5.54
rc101	25	3	100	262.59	100	4223.95	99.75	8.25
pr01	25	3	100	448.6	100	847.68	99.56	8.37
c101	25	4	100	7200.11	92.43	7200.03	99.88	9.16
r101	25	4	100	9.11	100	14.11	99.70	6.4
rc101	25	4	100	377.07	100	6432.59	98.26	8.3
pr01	25	4	100	376.85	100	756.67	99.39	8.84

Table 21: Comparison of MDLS hypervolume ratio (H %) and CPU time against exact methods (n=25)

Instance			BIOBAB (noUB)		ϵ-constraint (noUB)		MDLS	
name	n	m	H %	CPU (s)	H %	CPU (s)	H %	CPU (s)
c101	30	1	100	289.96	100	567.93	99.90	6.05
r101	30	1	100	1.92	100	2.23	99.95	5.98
rc101	30	1	100	7.85	100	7.03	100	5.7
pr01	30	1	100	1809	100	4410.73	98.57	6.97
c101	30	2	100	7200.3	93.35	7200.05	99.97	8.15
r101	30	2	100	10.51	100	13.06	99.86	6.53
rc101	30	2	100	101.15	100	267.05	99.76	6.54
pr01	30	2	100	4904.03	98.93	7200.16	97.89	9.63
c101	30	3	99.81	7200.04	81.14	7200.08	99.94	9.97
r101	30	3	100	18.65	100	32.81	99.94	7.08
rc101	30	3	100	344.53	100	4815.11	99.88	7.62
pr01	30	3	100	2541.46	100	5729.96	98.70	10.44
c101	30	4	99.81	7200.32	81.14	7200.06	99.94	13.18
r101	30	4	100	30.75	100	48.68	99.75	8.6
rc101	30	4	100	756.36	90.78	7200.01	99.74	9.03
pr01	30	4	100	2213.33	100	4777.36	98.52	12.01

Table 22: Comparison of MDLS hypervolume ratio (H %) and CPU time against exact methods (n=30)

Instance			BIOBAB (noUB)		ϵ-constraint (noUB)		MDLS	
name	n	m	H %	CPU (s)	H %	CPU (s)	H %	CPU (s)
c101	35	1	100	442.74	100	785.37	99.90	8.33
r101	35	1	100	5.12	100	3.86	100	5.71
rc101	35	1	100	10.83	100	13.17	100	5.82
pr01	35	1	100	4441	94.78	7200.39	97.96	10.08
c101	35	2	99.90	7200.82	91.35	7200.01	99.87	8.99
r101	35	2	100	13.55	100	23.4	99.02	8.53
rc101	35	2	100	134.63	100	365.53	99.76	7.32
pr01	35	2	99.50	7206.57	89.46	7200.23	95.76	10.44
c101	35	3	99.96	7200.04	80.46	7200.05	99.94	11.08
r101	35	3	100	38.53	100	54.06	98.75	7.82
rc101	35	3	100	466.93	98.29	7200.01	99.78	9.3
pr01	35	3	99.49	7200.06	92.10	7200.19	99.31	14.05
c101	35	4	99.94	7200.06	74.13	7200.06	99.94	12.19
r101	35	4	100	63.61	100	92.71	99.18	9.37
rc101	35	4	100	1202.45	85.90	7200.02	99.00	10.32
pr01	35	4	99.21	7200.5	93.32	7200.12	98.77	12.92

Table 23: Comparison of MDLS hypervolume ratio (H %) and CPU time against exact methods (n=35)

References

Aneja, Y. P., K. P. K. Nair. 1979. Bicriteria transportation problem. *Management Science* **25** 73–78.

Baldacci, Roberto, Aristide Mingozzi, Roberto Roberti. 2012. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research* **218**(1) 1 – 6. doi:http://dx.doi.org/10.1016/j.ejor.2011.07.037.

Belotti, P., B. Soylu, M. M. Wiecek. 2013. A branch-and-bound algorithm for biobjective mixed-integer programs. available online at http://www.optimization-online.org/DB_HTML/2013/01/3719.html.

Boland, N., H. Charkhgard, M. Savelsbergh. 2013a. Criterion space search algorithms for biobjective mixed 0-1 integer programming part i: 0-1 integer programs. available online at http://www.optimization-online.org/DB_HTML/2013/08/3987.html.

Boland, N., H. Charkhgard, M. Savelsbergh. 2013b. Criterion space search algorithms for biobjective mixed 0-1 integer programming part II: Mixed integer programs. available online at http://www.optimization-online.org/DB_FILE/2013/11/4107.pdf.

Chalmet, LG, L Lemonidis, DJ Elzinga. 1986. An algorithm for the bi-criterion integer programming problem. *European Journal of Operational Research* **25**(2) 292–300.

Chao, I, Bruce L Golden, Edward A Wasil, et al. 1996. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research* **88**(3) 475–489.

Cordeau, Jean-François, Michel Gendreau, Gilbert Laporte. 1997. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks* **30**(2) 105–119.

Dächert, K., J. Gorski, K. Klamroth. 2012. An augmented weighted tchebycheff method with adaptively chosen parameters for discrete bicriteria optimization problems. *Computers & Operations Research* **39** 2929–2943.

Desrosiers, Jacques, Marco E Lübbecke. 2005. A primer in column generation. Guy Desaulniers, Jacques Desrosiers, Marius M. Solomon, eds., *Column Generation*. Springer US, 1–32.

Ehrgott, Matthias. 2005. *Multicriteria optimization*, vol. 2. Springer.

Ehrgott, Matthias, Xavier Gandibleux. 2006. Bound sets for biobjective combinatorial optimization problems. *Computers & Operations Research* **34** 2674–2694.

Feillet, Dominique, Pierre Dejax, Michel Gendreau, Cyrille Gueguen. 2004. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks* **44**(3) 216–229.

Geoffrion, Arthur M. 1968. Proper efficiency and the theory of vector maximization. *Journal of Mathematical Analysis and Applications* **22** 618–630.

Haimes, Y., L. Lasdon, D Wismer. 1971. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on systems, man and cybernetics* **1** 296–297.

Jozefowiez, N., G. Laporte, F. Semet. 2012. A generic branch-and-cut algorithm for multiobjective optimization problems: application to the multilabel traveling salesman problem. *INFORMS Journal on Computing* **24** 554–564.

Laumanns, M., L. Thiele, E. Zitzler. 2006. An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European Journal of Operational Research* **169** 932–942.

Leitner, M., I. Ljubic, M. Sinnl. 2014. The bi-objective prize-collecting steiner tree problem. *INFORMS Journal on Computing* **to appear**.

Masin, Michael, Yossi Bukchin. 2008. Diversity maximization approach for multiobjective optimization. *Operations Research* **56**(2) 411–424. doi:10.1287/opre.1070.0413. URL `http://pubsonline.informs.org/doi/abs/10.1287/opre.1070.0413`.

Mavrotas, G., D. Diakoulaki. 1998. A branch and bound algorithm for mixed zero-one multiple objective linear programming. *European Journal of Operational Research* **107** 530–541.

Przybylski, Anthony, Xavier Gandibleux, Matthias Ehrgott. 2008. Two phase algorithms for the bi-objective assignment problem. *European Journal of Operational Research* **185**(2) 509–533.

Raith, Andrea, Siamak Moradi, Matthias Ehrgott, Michael Stiglmayr. 2012. Exploring bi-objective column generation. *Proceedings of the 46th Annual ORSNZ Conference*. 288–296.

Righini, Giovanni, Matteo Salani. 2009. Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. *Computers & Operations Research* **36**(4) 1191–1203.

Ropke, Stefan, David Pisinger. 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science* **40**(4) 455–472.

Sarpong, Boadu Mensah, Christian Artigues, Nicolas Jozefowiez. 2013. Column generation for bi-objective vehicle routing problems with a min-max objective. *13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2013, September 5, 2013, Sophia Antipolis, France*. 137–149.

Shaw, Paul. 1998. Using constraint programming and local search methods to solve vehicle routing problems. *Principles and Practice of Constraint Programming – CP98*. Springer, 417–431.

Solomon, Marius M. 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research* **35**(2) 254–265.

Sourd, F., O. Spanjaard. 2008. A multi-objective branch-and-bound framework: application to the biobjective spanning tree problem. *INFORMS Journal on Computing* **20** 472–484.

Stidsen, T., K. A. Andersen, B. Dammann. 2014. A branch and bound algorithm for a class of biobjective mixed integer programs. *Management Science* Available online.

Tam, Bassy, Matthias Ehrgott, David Ryan, Golbon Zakeri. 2011. A comparison of stochastic programming and bi-objective optimisation approaches to robust airline crew scheduling. *OR Spectrum* **33**(1) 49–75. doi:10.1007/s00291-009-0164-9. URL `http://dx.doi.org/10.1007/s00291-009-0164-9`.

Tricoire, Fabien. 2012. Multi-directional local search. *Computers & Operations Research* **39**(12) 3089–3101.

Tuyttens, D., J. Teghem, P. H. fortemps, K. Van Nieuwenhuyze. 2000. Performance of the mosa method for the bicriteria assignment problem. *Journal of Heuristics* **6** 295–310.

Ulungu, E.L., J. Teghem. 1995. The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences* **20**(2) 149–165.

Vansteenwegen, Pieter, Wouter Souffriau, Greet Vanden Berghe, Dirk Van Oudheusden. 2009. Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research* **36**(12) 3281–3290.

Vincent, T., F. Seipp, S. Ruzika, A. Przybylski, X. Gandibleux. 2013. Multiple objective branch and bound for mixed 0-1 linear programming: Corrections and improvements for the biobjective case. *Computers & Operations Research* **40** 498–509.

Visée, M., J. Teghem, M. Pirlot, E. L. Ulungu. 1998. Two-phases method and branch and bound procedures to solve the bi–objective knapsack problem. *J. of Global Optimization* **12**(2) 139–155.