RAIRO-Oper. Res. 49 (2015) 527–554
DOI: 10.1051/ro/2014054

RAIRO Operations Research
www.rairo-ro.org

USING COLUMN GENERATION TO COMPUTE LOWER BOUND SETS FOR BI-OBJECTIVE COMBINATORIAL OPTIMIZATION PROBLEMS

Boadu Mensah Sarpong[1,2], Christian Artigues[2,3] and Nicolas Jozefowiez[1,2]

Abstract. We discuss the use of column generation in a bi-objective setting. Just as in single objective combinatorial optimization, the role of column generation in the bi-objective setting is to compute dual bounds (*i.e.* lower bounds for minimization problems and upper bounds for maximization problems) which can be used to guide the search for efficient solutions or to evaluate the quality of approximate solutions. The general idea used in this paper is to first transform the bi-objective problem into single objective by a scalarization method and then solve the transformed problem several times by varying the necessary parameters. We show that irrespective of the scalarization method used, similar subproblems are solved when applying column generation. For this reason, we investigate possible ways of intelligently searching for columns for these subproblems in order to accelerate the column generation method.

Keywords. Multi-objective optimization, bound sets, combinatorial optimization, column generation.

Mathematics Subject Classification. 90C27, 90C29.

Received September 29, 2014. Accepted October 16, 2014.

[1] CNRS, LAAS, 7 avenue du colonel Roche, 31400 Toulouse, France. bmsarpon@laas.fr

[2] University de Toulouse, INSA, LAAS, 31400 Toulouse, France. artigues@laas.fr

[3] University de Toulouse, LAAS, 31400 Toulouse, France. njozefow@laas.fr

1. Introduction

Exact methods for solving difficult single objective optimization problems rely heavily on lower and upper bounds. The number of exact methods for multi-objective problems is very few and a possible explanation for this is that the notion of bounds is not very well developed for multi-objective problems in comparison to single objective problems. For this reason, it is necessary to develop models and strategies for computing good bounds for multi-objective problems and this paper aims at contributing to this goal. Without loss of generality, we suppose that all objective functions of the considered problems are to be minimized. We study the use of column generation in computing lower bound sets for bi-objective combinatorial optimization (BOCO) problems. This study is expected to serve a broad range of applications since an important class of combinatorial optimization problems can be formulated as integer programs with exponential number of variables that are efficiently solved by column generation methods. An advantage of this type of formulations is that they usually have good linear relaxations and thus strong lower bounds.

1.1. Multi-objective combinatorial optimization

A general multi-objective combinatorial optimization (MOCO) problem deals with the minimization of a vector of two or more functions $F(x) = (f_1(x), \ldots, f_r(x))$ over a finite domain of feasible solutions $\mathcal{X}$. The vector $x = (x_1, \ldots, x_n)$ is the decision variable or solution, $\mathcal{Y} = F(\mathcal{X})$ corresponds to the images of the feasible solutions in the objective space, and $y = (y_1, \ldots, y_r)$, where $y_i = f_i(x)$, is a point of the objective space. A solution x' *dominates* another solution x'', (denoted $x' \preceq x''$), if for any index $i \in \{1, \ldots, n\}$, $f_i(x') \leq f_i(x'')$ and there is at least one index $i \in \{1, \ldots, n\}$ for which $f_i(x') < f_i(x'')$. A feasible solution dominated by no other feasible solution is said to be *efficient* or *Pareto optimal* and its image in the objective space is said to be *nondominated*. The set of all efficient solutions is called the *efficient set* (denoted $\mathcal{X}_E$) and the set of all nondominated points is the *nondominated set* (denoted $\mathcal{Y}_N$). In general, more than one efficient solution may correspond to the same nondominated point and so solving a MOCO problem in most practical cases usually means finding at least one efficient solution for each nondominated point. The nondominated set defines what is known as the *Pareto frontier*. An efficient solution that maps onto a nondominated point lying on the convex part of the Pareto frontier is called a *supported efficient solution* and its image is called a *supported nondominated point*. An efficient solution that is not supported is called a *nonsupported efficient solution*.

1.1.1. Lower and upper bounds for a MOCO problem

Although the meaning of bounds in single objective optimization is well studied and understood, the situation is quite different in the multi-objective case. Ideal

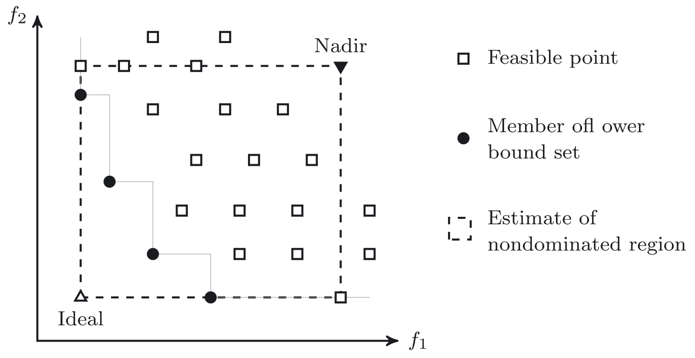

FIGURE 1. Lower and upper bounds for a BOCO problem.

and nadir points are well known lower and upper bounds, respectively, of the set $\mathcal{Y}_N$. The coordinates of the ideal point are obtained by optimizing each objective function independently of the others, whereas the coordinates of the nadir point correspond to the worst value of each objective function when we consider the set $\mathcal{X}_E$. From Figure 1, it can be seen that these points are usually poor bounds since they just estimate the whole region where a member of $\mathcal{Y}_N$ may lie. In this paper, we will be interested in bounds that can reduce the region where the members of $\mathcal{Y}_N$ are and thus narrow down the search for nondominated points.

Given that a MOCO problem is a discrete problem, its lower bound can be defined as a set of points $\boldsymbol{L}$ such that the image of every feasible solution is either a member of $\boldsymbol{L}$ or is dominated by at least one member of $\boldsymbol{L}$. The members of a lower bound set do not necessarily belong to $\mathcal{Y}$. That is, they may or may not correspond to feasible points of the problem. An upper bound may also be defined as a finite set of points in $\mathcal{Y}$ that do not dominate one another. The idea of using sets to define bounds for multi-objective problems was first introduced in [20]. The constuction and use of *bound sets* for bi-objective combinatorial optimization (BOCO) problems has recently been addressed by some authors [5, 8, 17]. The main idea used by these authors in defining a lower bound set is to transform the bi-objective problem into single objective by using a weighted sum method and solving the transformed problem for different weights in order to compute the complete set of supported nondominated points. A lower bound set for the considered bi-objective problem may be defined as the line connecting the set of supported nondominated points. This procedure is only possible if there is an efficient algorithm for solving the single objective problem associated to the bi-objective problem. If the single objective problem associated to the bi-objective problem is $\mathcal{NP}$-hard, then we have to resort to computing the set of supported nondominated points for a relaxation of the single objective problem. As pointed out in [8], any scalarization method may be used to convert a bi-objective problem into single objective when computing bound sets. Nevertheless, we could not find a published paper that uses a method not based on the weighted sum scalarization.

One reason for this may be that the weighted sum method converts a multi-objective problem into single objective by changing only the objective function. In this way, well known single objective methods can be used to solve the transformed problem. A well known disadvantage of the weighted sum method is that it cannot find nonsupported solutions no matter the choice of weights used. If there is a large number of nonsupported points, a lower bound set produced by using a weighted sum method can be very poor. For this reason, it seems worthy to study the use of another scalarization method like the ε-constraint method in computing bound sets. A disadvantage of an ε-constraint method is that it is not very easy to define values for the parameter ε in such a way that all members of the nondominated set are found. Nevertheless, this is possible for some practical problems (for Ex. [2,13]). An advantage of the ε-constraint method is that, unlike a weighted sum method, it is capable of finding both supported and nonsupported solutions.

An important class of combinatorial optimization problems (*e.g.* vehicle routing problems) can be formulated as (mixed-)integer linear programs with an exponential number of variables. These type of formulations (set covering, and set packing models) are efficiently solved by column generation based methods. In spite of the challenge posed by the large number of variables, an advantage of this type of models is that they usually have strong linear relaxations and so produce good lower bounds. In this paper, we discuss how column generation can be used to compute lower and upper bound sets for BOCO problems which are modeled by an exponential number of variables. In particular, we show that a set of similar subproblems are solved whether we use a weighted sum method or an ε-constraint method. For this reason, we investigate the possibility of treating some of these subproblems simultaneously in order to speed up the column generation algorithm.

1.2. Contributions and organization of work

A first contribution of this work is the application of column generation to BOCO problems. Although column generation and multi-objective optimization are well studied independently, the application of column generation to multi-objective problems seems to have been neglected. This current paper contributes in this respect. In particular, we propose different approaches to effectively search for columns when applying column generation to BOCO problems. Another contribution of this work is in the use of a scalarization method, different from the weighted sum, in computing lower bound sets for BOCO problems. To the best of our knowledge, this is the first attempt at designing such a method.

In Section 2, we discuss how column generation can be used in computing bound sets for BOCO problems. We show that whether we use a weighted sum method or an ε-constraint method, a similar type of subproblem is encountered. Due to this, we propose different strategies to effectively search for relevant columns. An application problem is presented in Section 3 and evaluation of the bound sets computed for this problem as well as a comparison of the different column search strategies are given in Section 4. In Section 5, we draw some conclusions and point out possible ways of extending this work.

2. Column generation for BOCO problems

In this section, we discuss how column generation can be used to compute lower and upper bound sets for BOCO problems by considering the following bi-objective set-covering based formulation, P:

$$\begin{aligned} \text{Minimize} \quad & (c^1)^T x && (2.1)\\ \text{Minimize} \quad & (c^2)^T x && (2.2)\\ \text{subject to :} \quad & Ax \geq b, && (2.3)\\ & x \geq 0 \text{ and integer.} && (2.4) \end{aligned}$$

In the above formulation, x is a vector of n decision variables and c^i is a vector of n integer coefficients in the ith objective function ($i = 1, 2$). The constraints are expressed by using an $m \times n$ matrix of coefficients, A, and a vector of m constants, b.

2.1. Computing lower and upper bound sets

The main idea used in computing bound sets for BOCO problems is to convert the problem into single objective by using a scalarization method and solving the resulting problem (or a relaxation of it) several times by varying the necessary parameters. In this paper, we consider the ε-constraint method and the weighted sum method.

The weighted sum method transforms Formulation (2.1)–(2.4) into single objective by using a vector of non-negative weights $\lambda = (\lambda_1, \lambda_2)$. The resulting problem $P(\lambda)$ is given by

$$\begin{aligned} \text{Minimize} \quad & \lambda_1 \cdot (c^1)^T x + \lambda_2 \cdot (c^2)^T x && (2.5)\\ \text{subject to :} \quad & Ax \geq b, && (2.6)\\ & x \geq 0 \text{ and integer.} && (2.7) \end{aligned}$$

On the other hand, the ε-constraint approach obtains a single objective problem by restricting the worst possible value of one objective (say the second one) to a constant $\varepsilon \in \mathbb{R}$. Thus, we obtain the single objective problem $P(\varepsilon)$:

$$\begin{aligned} \text{Minimize} \quad & (c^1)^T x && (2.8)\\ \text{subject to :} \quad & Ax \geq b, && (2.9)\\ & -(c^2)^T x \geq -\varepsilon, && (2.10)\\ & x \geq 0 \text{ and integer.} && (2.11) \end{aligned}$$

In a first instance, we suppose that Formulation (2.1)–(2.4) is manageable in the sense that we have a "small" number of variables in x (*i.e.* n is small). In such a case, there is no need for column generation in computing a lower bound set for problem P. This has been discussed extensively in [8] for the weighted sum method. Given that Formulation (2.5)–(2.7) is $\mathcal{NP}$-hard in the general case, we need to find the set of supported nondominated points for a relaxation of it. We consider the linear relaxation which is obtained by dropping the requirement for x to be integer. The set of supported nondominated points can be obtained by a method proposed in [1] which is used widely as the first phase of the two phases method [18]. A lower bound set can then be defined by the line joining the points of the computed set.

2.1.1. *Using an ε-constraint method*

The main idea in using an ε-constraint method to compute lower bound sets is similar to the case where we use a weighted sum method. That is, we need to solve linear relaxations of $P(\varepsilon)$ for different values of ε. In doing so, we need to ensure that every feasible point of the problem P is either generated or is dominated by at least one of the generated points. The general procedure is summarized in Algorithm 1 and we suppose that all feasible values for $(c^2)^T x$ lie in the range $[\min \varepsilon, \max \varepsilon]$. The output of Algorithm 1 is the set $\boldsymbol{L}$ and a sequence of values $\max \varepsilon = \varepsilon_1 > \varepsilon_2 > \ldots b > \min \varepsilon$ corresponding to the elements of the set. Note that each step size δ_i is chosen in such a way that there can be no feasible point $(f_1^* = (c^1)^T x^*, f_2^* = (c^2)^T x^*)$ of the original integer program such that $\varepsilon_{i+1} < f_2^* < f_2^i$ (see Fig. 2). This is a very difficult thing to do for a general bi-objective problem. For the considered problem, however, we have integer objective coefficients so we can use an idea similar to the one used in [13] and which was formalized in [2]. At iteration i, we define the step size as $\delta_i = 1 - (\lceil f_2^i \rceil - f_2^i)$. As we will later see, it is possible to define even better step sizes for specific problems. It can be clearly seen that the set, $\boldsymbol{L}$, returned by the algorithm is indeed a lower bound set for the considered problem.

Algorithm 1 Using an ε-constraint method to compute a lower bound set

```
1: Set $\boldsymbol{L} \leftarrow \emptyset$.
2: Set $i \leftarrow 1$, and $\varepsilon_i \leftarrow \max \varepsilon$.
3: while $\varepsilon_i \geq \min \varepsilon$ do
4:     Solve linear relaxation of $P(\varepsilon_i)$ and let $x^*$ be the optimal solution vector.
5:     Compute the corresponding objective values $f_1^i = (c^1)^T x^*$ and $f_2^i = (c^2)^T x^*$.
6:     Set $\boldsymbol{L} \leftarrow \boldsymbol{L} \cup (f_1^i, f_2^i)$.
7:     Choose $\delta_i > 0$ such that there can be no feasible point $(f_1^*, f_2^*)$ of the original
       integer program satisfying $f_2^i - \delta_i < f_2^* < f_2^i$.
8:     Set $i \leftarrow i + 1$ and $\varepsilon_i \leftarrow f_2^{i-1} - \delta_{i-1}$.
9: end while
```

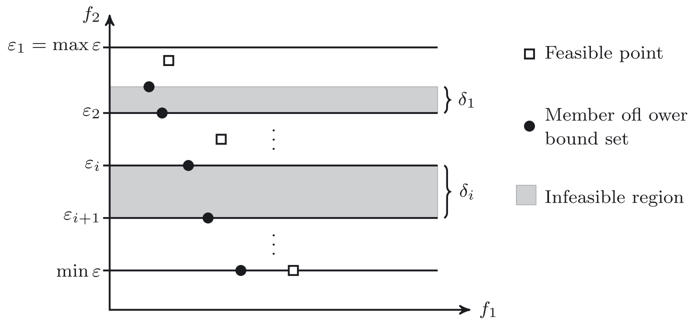

FIGURE 2. Constructing a lower bound set through an ε-constraint method.

Whether we use a weighted sum method or an ε-constraint method, the members of a lower bound set may correspond to points that are infeasible for problem P. The definition of an upper bound set, however, requires that its members are feasible points of P. For this reason, heuristics and metaheuristics are used in computing upper bound sets when necessary. An example is the greedy heuristic used in computing upper bound sets for instances of the bi-objective set covering problem in [8]. We stress that the heuristic or metaheuristic used in computing an upper bound set does not necessarily need to depend on a particular scalarization method. Any known heuristic or metaheuristic for problem P may be used.

2.2. Computing lower bound sets by column generation

Many practical problems can be formulated in the form of (2.1)–(2.4) with an exponential number of decision variables in order to obtain a tight linear programming relaxation. It is impractical and sometimes impossible to consider all the variables at once when dealing with such formulations. The idea of a column generation method for solving such a problem is to start with only a reasonable number of variables (and the corresponding columns of matrix A) for which the problem is primal feasible. The other variables together with their corresponding columns of A are introduced when necessary by solving an auxiliary problem. In column generation terminology, the original integer problem with an exponential number of variables (or columns) is called the integer programming master problem (IPM) and its linear relaxation is called the linear programming master problem (LPM). A restriction of IPM (respectively, LPM) to a reasonable number of columns is called a restricted IMP (RIPM), respectively, RLPM. The auxiliary problem solved to propose new variables to add to the RLPM or prove the convergence of the method is called the subproblem (S). Each iteration of column generation starts by solving the RLPM in order to obtain optimal primal and dual solutions. The subproblem (S) uses the optimal dual values to search for new columns that can possibly improve the current objective value of the RLPM. One or several of such columns, if found, are added to the RLPM and the process repeats. The method converges when the subproblem finds no new columns that can possibly improve the current objective value of the RLPM.

Next, we discuss how to use column generation in computing lower bounds for BOCO problems modelled with an exponential number of variables. We list the different models involved in a column generation method in the case of a weighted sum method and also those in the case of an ε-constraint method.

2.2.1. Using a weighted sum method

Linear Programming Master Problem (LPM(λ)):

$$\begin{aligned} \text{Minimize} \quad & \left(\lambda_1 c^1 + \lambda_2 c^2\right)^T x && (2.12)\\ \text{subject to :} \quad & Ax \geq b, && (2.13)\\ & x \geq 0. && (2.14) \end{aligned}$$

Dual of LPM (DLPM(λ)): let π be the vector of dual variables associated with Constraints (2.13). The dual formulation is

$$\begin{aligned} \text{Maximize} \quad & b^T \pi && (2.15)\\ \text{subject to :} \quad & A^T \pi \leq \lambda_1 c^1 + \lambda_2 c^2, && (2.16)\\ & \pi \geq 0. && (2.17) \end{aligned}$$

Subproblem (S(λ)): it is defined as finding variables that correspond to columns of matrix A and which satisfy an equality of the form

$$\lambda_1 c_j^1 + \lambda_2 c_j^2 - A_j^T \pi < 0, \tag{2.18}$$

where A_j is the jth column of matrix A.

2.2.2. Using an ε-constraint method

Linear Programming Master Problem (LPM(ε)):

$$\begin{aligned} \text{Minimize} \quad & (c^1)^T x && (2.19)\\ \text{subject to :} \quad & Ax \geq b, && (2.20)\\ & (-c^2)^T x \geq -\varepsilon, && (2.21)\\ & x \geq 0. && (2.22) \end{aligned}$$

Dual of LPM (DLPM(ε)): let π be the vector of dual variables associated with Constraints (2.20) and φ the dual variable associated with Constraint (2.21). The dual is formulation is

$$\begin{aligned} \text{Maximize} \quad & b^T \pi - \varepsilon \varphi && (2.23)\\ \text{subject to :} \quad & A^T \pi - \varphi c^2 \leq c^1, && (2.24)\\ & \pi,\ \varphi \geq 0. && (2.25) \end{aligned}$$

Subproblem (S(ε)): It is defined as finding variables that correspond to columns of matrix A and which satisfy an equality of the form

$$c_j^1 + \varphi c_j^2 - A_j^T \pi < 0, \tag{2.26}$$

where A_j is the jth column of matrix A.

2.2.3. General form of the subproblem

By comparing the inequalities (2.18) and (2.26), we can see that the subproblems encountered in both cases have a similar form. In addition, the general form of the subproblem obtained by using a particular scalarization method does not change when we modify the parameters. Only the values of the dual variables and coefficients change. A similar result is obtained if we consider problems with more than two objectives. This means that strategies we describe in solving the subproblems obtained by one of these scalarization methods can easily be adapted to the other scalarization methods. Moreover, for any given scalarization method, it is possible to treat more than one subproblem at the same time when searching for columns. We present different strategies to search for column in Section 2.3.

2.2.4. Computing an upper bound set

A direct and intuitive way of computing an upper bound set after computing a lower bound set by column generation is to solve the RLPM (with its current columns) as an integer program several times by following the idea of Algorithm 1. Before applying this algorithm, it is possible to use knowledge of the problem in order to generate more columns that were not necessary in computing a lower bound set but may be useful when solving the RLPM as an integer program. It is also possible to use metaheuristics and other problem specific heuristics if they are available.

2.3. Column search strategies

As we have already noted, the general form of subproblem obtained by using a particular scalarization method does not change when we modify the parameters. Also, irrespective of the scalarization method used, the subproblem has a similar form. We discuss some approaches to search for relevant columns when computing a lower bound set by column generation. We will demonstrate the strategies for the case where we use an ε-constraint method. They can, however, be easily adapted to the case where a weighted sum method is used. In the case where a particular idea applies uniquely to an ε-constraint method, this will be stated.

2.3.1. Point-by-point search (PPS)

A standard and very intuitive approach to apply column generation to a BOCO problem involves solving the RLPM completely for any given value of ε before continuing to the next value of ε. That is, for any given value of ε, RLPM(ε) is solved by column generation until the subproblem proposes no new columns that can improve the current objective value of the RLPM. We call this approach the point-by-point search (PPS) since the search is completely dedicated to finding a particular point of a lower bound set (corresponding to a particular value of ε) before moving on to another point. The algorithm for the PPS is similar to the one described in Algorithm 1 except that in Step 1 we need to solve an RLPM to optimality by column generation. The columns generated while solving the RLPM for a given value of ε are kept in the model throughout the whole algorithm. This is because they may still be relevant for other values of ε due to the similar structures of the subproblems. The PPS identifies the points of a lower bound set following a predictable order. The point corresponding to a smaller value of ε can be found only after all the points corresponding to greater values of ε have been found. Although the PPS is simple and easy to implement, it takes no advantage of the similar form of the subproblems solved for the different values of ε. The column generation method may also be slow to converge for a given value of ε but the PPS requires us to wait for it to converge before moving on to another value of ε. This can result in a huge number of "irrelevant" columns being added to the RLPM and a long computational time. Two approaches which aim at addressing some of these problems are discussed below.

2.3.2. k-step point-by-point search (k-PPS)

In this approach, we have the freedom to leave a given value of ε before the RLPM has converged for it and switch to another value of ε. If it is necessary, we will return to a previously skipped value of ε later on. The k-PPS approach is summarized in Algorithm 2. The first step of this approach is to define a condition, $\mathcal{O}$, under which a given value of ε for which the RLPM has not converged will be skipped either temporarily or permanently. The condition can be as simple as setting a fixed number, k, of column generation iterations. Instead of using a fixed number of iterations, we may also decide to skip a given value of ε when the objective value of the RLPM has not improved significantly after k iterations. This can be a way of addressing some column generation problems like *the plateau effect* and *the tailing-off effect* that were described in [19]. It is also possible to use any other condition we find appropriate for a given problem. In Step 2 of Algorithm 2, there is no need to solve RLPM(ε_i) if either of these two conditions are satisfied:

1. There is a converged point $(f_1^*, f_2^*) \in \boldsymbol{L}$ such that $f_2^* \le \varepsilon_i \le \varepsilon_*$, where ε_* denotes the value of ε for which (f_1^*, f_2^*) was computed.
2. There is a converged point $(f_1^*, f_2^*) \in \boldsymbol{L}$ and another point (f_1^j, f_2^j) such that $f_1^j - f_1^* \le 1$ and $f_2^j < \varepsilon_i < f_2^*$. Note that the point (f_1^j, f_2^j) may have converged or not.

If the first condition is satisfied, there is no need to solve RLPM(ε_i) since we are sure to end up with a solution that maps onto (f_1^*, f_2^*). In the case of the second condition, the new point that will be generated by solving RLPM(ε_i) to optimality will not make the current lower bound set any better since we have integral objective coefficients.

Algorithm 2 k-Step Point-by-Point Search (k-PPS)

```
1: Define a condition, $\mathcal{O}$, under which a given value of $\varepsilon$ will be skipped before RLPM($\varepsilon$)
   converges for this value.
2: Set $\boldsymbol{L} \leftarrow \emptyset$.
3: repeat
4:   Set $i \leftarrow 1$, and $\varepsilon_i \leftarrow \max \varepsilon$.
5:   while $\varepsilon_i \geq \min \varepsilon$ do
6:     if it is necessary to solve RLPM($\varepsilon_i$) then
7:       Solve RLPM($\varepsilon_i$) by column generation until the method converges or condition
         $\mathcal{O}$ is satisfied.
8:       Let $x^*$ be the current optimal solution vector.
9:       Compute the current optimal values $f_1^i = (c^1)_J^T x_J^*$ and $f_2^i = (c^2)_J^T x_J^*$, where $J$
         is the index set of the columns of $A$ in RLPM($\varepsilon_i$).
10:      if the method converged in Step 2 then
11:        Set $\boldsymbol{L} \leftarrow \boldsymbol{L} \cup (f_1^i, f_2^i)$.
12:      end if
13:      Choose an appropriate step size $\delta_i > 0$.
14:      Set $i \leftarrow i + 1$ and $\varepsilon_i \leftarrow f_2^{i-1} - \delta_{i-1}$.
15:    end if
16:  end while
17: until the RMP converges for all values of $\varepsilon$ in the inner loop.
```

Unlike in the case of the PPS, the points of a lower bound set are identified by the k-PPS in no predictable order. That is, the RLPM can converge for a smaller value of ε before it converges for a greater value. Figure 3 illustrates a possible order in which the points of a lower bound set are identified by the k-PPS. This illustration is based on a fictitious example. We suppose that feasible values for the ε-constraint objective function are integers between $\min \varepsilon = 1$ and $\max \varepsilon = 10$. Each complete iteration of the k-PPS approach will thus start by solving RLPM(ε) for $\varepsilon = 10$ as in Figure 3a. A description of the series of diagrams is given below. Note that we can obtain non-integral points since we are solving linear relaxations rather than integer programs. During the process, converged points are saved in the lower bound set.

Figure 3a. There are no converged points at the start of the algorithm. Condition $\mathcal{O}$ is satisfied before RLPM(10) converges and the current unconverged point is $(7.0, 3.0)$. Similarly, condition $\mathcal{O}$ is satisfied before RLPM(2) and RLPM(1) converge. The corresponding unconverged points are $(9.5, 2.0)$ and $(10.75, 1.0)$, respectively.

Figure 3b. This iteration of k-PPS also starts with no converged points from the previous iteration. Condition $\mathcal{O}$ is satisfied before RLPM(10), RLPM(7), and RLPM(6) converge. The corresponding points are $(1.25, 8.0)$, $(1.5, 7.0)$, and $(1.75, 6.0)$, respectively. RLPM(5) converges to the point $(1.75, 5.0)$ before condition $\mathcal{O}$ is satisfied. Four more unconverged points $(5.0, 4.0)$, $(5.75, 3.0)$, $(6.25, 2.0)$, and $(7.5, 1.0)$ corresponding to RLPM(4), RLPM(3), RLPM(2), and RLPM(1), respectively, are generated in this iteration.

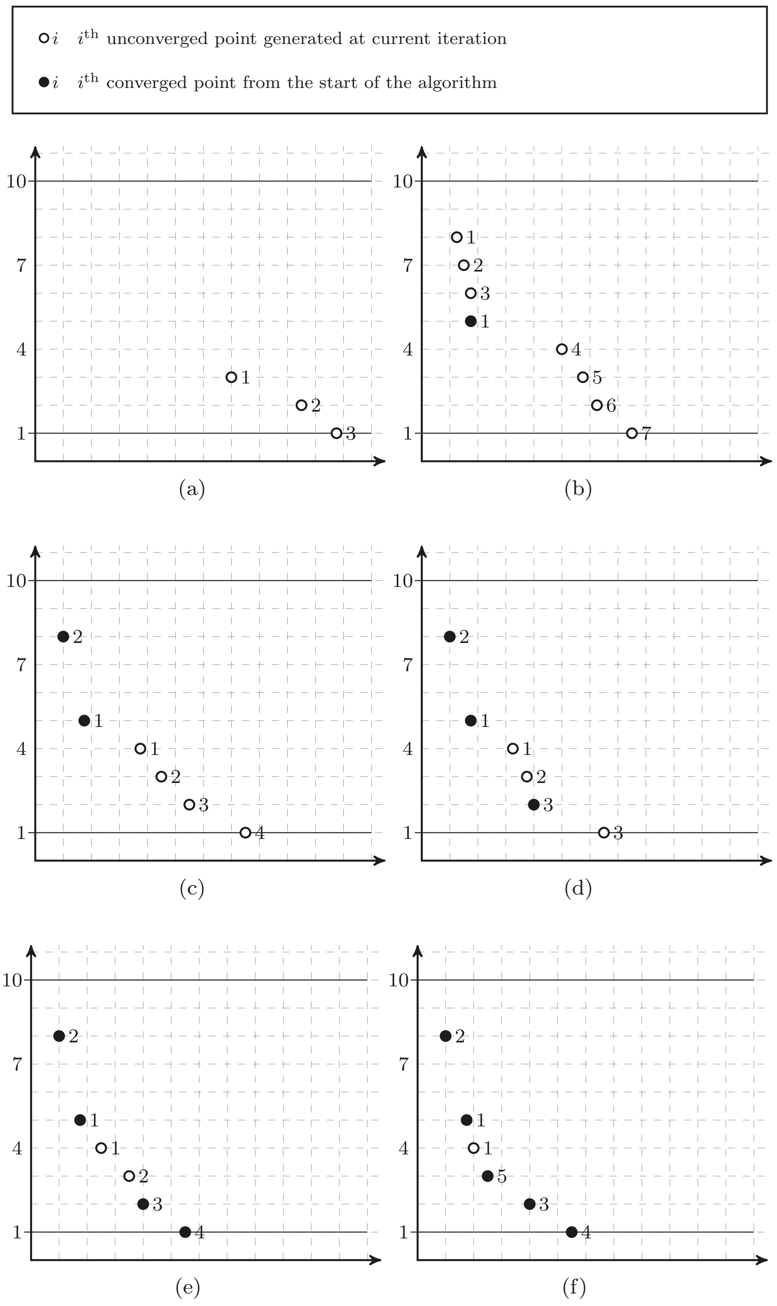

FIGURE 3. Possible order in which points of a lower bound set are identified by the k-PPS.

Figure 3c. RLPM(10) converges to the point $(1.0, 8.0)$ and becomes the second converged point since the start of the algorithm. RLPM(7) and RLPM(6) are permanently skipped for the rest of the algorithm since these values of ε satisfy the second condition for skipping a value of ε. Four unconverged points $(3.75, 4.0)$, $(4.5, 3.0)$, $(5.5, 2.0)$, and $(7.5, 1.0)$ corresponding to RLPM(4), RLPM(3), RLPM(2), and RLPM(1), respectively, are generated in this iteration.
Figure 3d. There are two previously converged points at the start of this iteration. RLPM(10), RLPM(9), and RLPM(8) are skipped permanently for rest of the algorithm based on the first and second conditions for skipping a value of ε. Condition $\mathcal{O}$ is satisfied before RLPM(4) and RLPM(3) converge and the corresponding points are $(3.25, 4.0)$ and $(3.75, 3.0)$, respectively. One converged point $(4.0, 2.0)$ corresponding to RLPM(2) is generated whereas an unconverged point RLPM(1) corresponding to $(6.5, 1.0)$ is also produced.
Figure 3e. There are three previously converged points at the start of this iteration. The values of ε which were skipped in previous iterations are also skipped here for the same reasons as before. Two unconverged points $(2.5, 4.0)$ and $(3.5, 3.0)$ corresponding to RLPM(4) and RLPM(3), respectively, are generated. RLPM(1) converges to the point $(5.5, 1.0)$.
Figure 3f. This iteration starts with RLPM(4) and which fails to converge before condition $\mathcal{O}$ is satisfied. The corresponding unconverged point is $(2.0, 4.0)$. RLPM(3) converges to the point $(2.5, 3.0)$. There is no need for an additional iteration to return to RLPM(4) thanks to the second condition for skipping a value of ε.

It is important to note that the set returned by the k-PPS represents a lower bound for the considered BOIP. In other words, if a given value of ε is skipped, then the algorithm will either return to the skipped value later on or prove that it is unnecessary to do so. Hence, no relevant points are missed. One main challenge of the k-PPS is that it is not easy to decide on a good condition $\mathcal{O}$ when we don't have enough information on how the RLPM behaves at the start of the algorithm. For this reason, it may be necessary to have an adaptive condition $\mathcal{O}$ that can be modified across the iterations when we have enough information.

2.3.3. Sequential search

The main idea behind the sequential search approach is to work on a whole frontier rather than deal with individual points. This is achieved by returning a set of columns that are relevant for several values of ε at each iteration of column generation. A description of the approach is given in Algorithm 3. Each iteration of the sequential search approach consists of two main steps. In step I (steps 3 to 3 of Algorithm 3), we solve the RLPM (without generating any columns) for several values of ε for which it has not converged. The search for relevant columns is done in step II (steps 3 to 3 of Algorithm 3). During an iteration, the sets Π and Ω are used to temporary store the vectors of dual values and the generated columns, respectively. Before solving the subproblem corresponding to a given point generated in step I, we first need to verify if any of the columns already found for the other points are relevant (has a negative reduced cost) for this new point. If this is the case the new point is skipped without solving the subproblem corresponding to it since the point is already represented by the set of generated columns. Thus, the subproblem corresponding to a new point is only solved if there are no previously found columns in the same iteration that are relevant for this point.

Algorithm 3 Sequential Search

```
1: Set $\boldsymbol{L} \leftarrow \emptyset$.
2: repeat
3:    Set $i \leftarrow 1$, $\varepsilon_i \leftarrow \max \varepsilon$, $\Pi \leftarrow \emptyset$, and $\Omega \leftarrow \emptyset$.
4:    while $\varepsilon_i \geq \min \varepsilon$ do
5:       if it is necessary to solve RLPM($\varepsilon_i$) then
6:          Solve RLPM($\varepsilon_i$) once without generating any columns.
7:          Let $x^*$ and $\pi^i$ be the optimal solution and dual vectors, respectively.
8:          Compute the current optimal values $f_1^i = (c^1)_J^T x_J^*$ and $f_2^i = (c^2)_J^T x_J^*$, where $J$
            is the index set of the columns of $A$ in RLPM($\varepsilon_i$).
9:          Set $\Pi \leftarrow \Pi \cup \pi^i$.
10:         Choose an appropriate step size $\delta_i > 0$.
11:         Set $i \leftarrow i + 1$ and $\varepsilon_i \leftarrow f_2^{i-1} - \delta_{i-1}$.
12:      end if
13:   end while
14:   for each element $\pi^i$ of $\Pi$ do
15:      if no column of $\Omega$ is of negative reduced cost for $\pi^i$ then
16:         Solve the subproblem corresponding to $\pi^i$.
17:         Let $\Lambda^i$ be the set of columns found and set $\Omega \leftarrow \Omega \cup \Lambda^i$.
18:         if $\Lambda^i = \emptyset$ then
19:            Set $\boldsymbol{L} \leftarrow \boldsymbol{L} \cup (f_1^i, f_2^i)$.
20:         end if
21:      end if
22:   end for
23:   Add all columns of $\Omega$ to the RMP.
24: until $\Omega = \emptyset$.
```

There is no particular order in which the points of a lower bound set are identified by a sequential search method. Note, however, that a point generated in step I of an iteration can be proven to have converged in step II only if the subproblem corresponding to it is solved directly by an exact algorithm and no relevant columns are found. A fictitious example to demonstrate a possible order in which the points of a lower bound set are identified by a sequential search approach is given in the series of diagrams in Figure 4. An explanation of each of the diagrams is given below.

Figure 4a. There are no converged points at the start of the algorithm. Step I produces three points $(7.0, 3.0)$, $(9.5, 2.0)$ and $(21.5, 1.0)$ corresponding to RLPM(10), RLPM(2), and RLPM(1), respectively. None of these points is proven to have converged in step II.

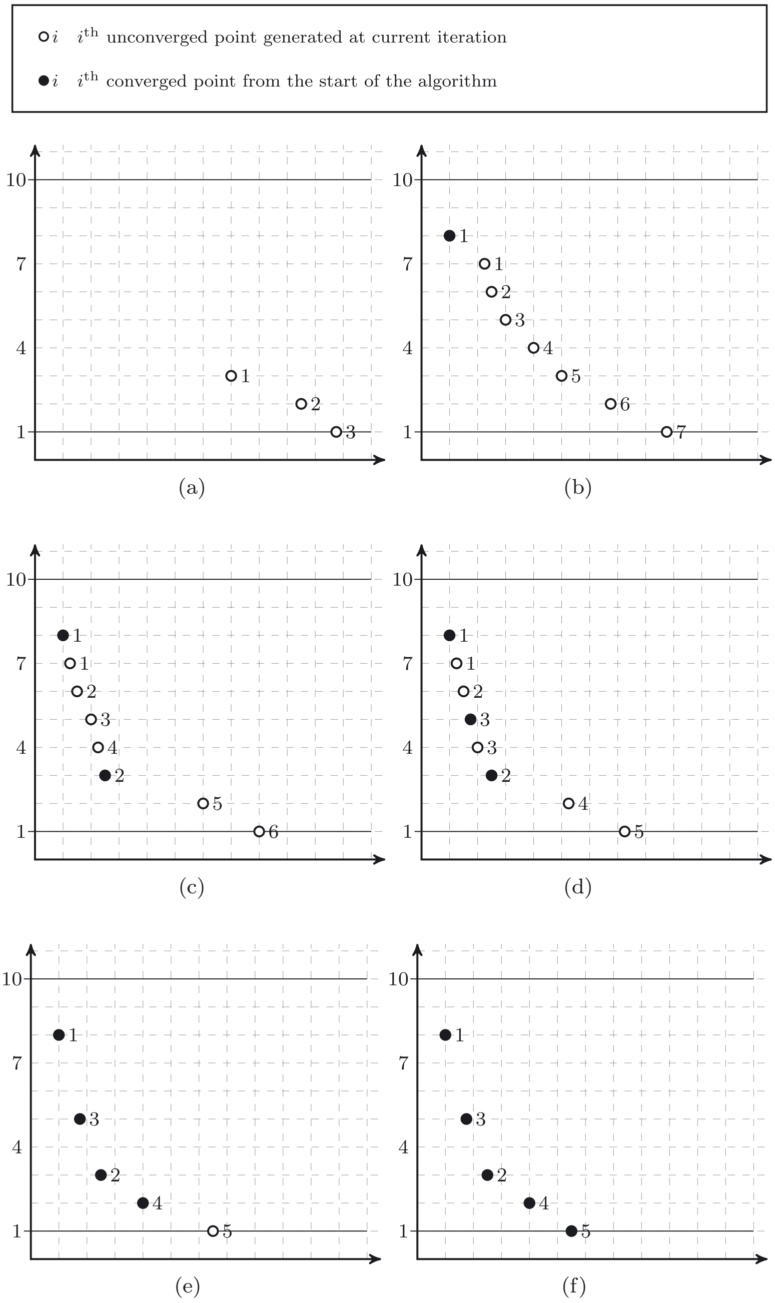

Figure 4. Possible order in which points of a lower bound set are identified by a sequential search approach.

Figure 4b. There are no converged points at the start of this iteration. In step I, seven points are generated. Out of these points, only one of them $(1.0, 8.0)$ corresponding to RLPM(10) is proven to have converged in step II. For the rest of the algorithm, RLPM(10), RLPM(9), and RLPM(8) will be skipped since the first condition for skipping a value of ε is satisfied for these values.
Figure 4c. This iteration starts from RLPM(7) and generates seven points in step I out of which only one is proven to have converged in step II. The converged point is $(2.5, 3.0)$ and corresponds to RLPM(3).
Figure 4d. The two points that converged in the preceding iterations are known at the beginning of this iteration. Step I starts from RLPM(7) and generates six points and one of these points, $(1.75, 5.0)$ corresponding to RLPM(5), is proven to have converged in step II. It is noted that the second condition for skipping a value of ε is satisfied for the first three unconverged points. For this reason, RLPM(7), RLPM(6), and RLPM(4) will be skipped for the rest of the algorithm.
Figure 4e. Step I produces two points $(4.0, 2.0)$ and $(6.5, 1.0)$ corresponding to RLPM(2) and RLPM(1), respectively. In step II, the point $(4.0, 2.0)$ is proven to have converged whereas the other point remains unconverged.
Figure 4f. RLPM(1) is solved by column generation until it converges to the point $(5.5, 1.0)$. Note that this is the same as repeating steps I and II the necessary number of times until RLPM(1) converges.

An advantage of the sequential search approach is that, it ensures some uniformity in the convergence of the members of $\boldsymbol{L}$. This can be a very useful technique in the design of heuristics and metaheuristics that are based on column generation since we are sure to find a set of points that is representative of the whole range of the Pareto frontier. Another advantage of the sequential search approach is that it encourages similar columns to appear in the solution of the points in a bound set and this is very necessary in some applications. An example is the case of creating treatment plans in the field of radiotherapy [16]. A bi-objective problem studied in this field is to treat defective cells and also reduce the side effects resulting from the use of radio waves. A column in the model used represents a weekly treatment plan which may require the expertise of several workers. A complete treatment program consists of several weekly treatment plans. Clearly, a treatment program that is made up of too many different weekly plans will be impractical to implement. Applying a sequential search approach to such a problem can help in obtaining more practical plans.

3. Application to the bi-objective multi-vehicle covering tour problem

In this section, we use the ideas presented in the preceding section to compute lower and upper bound sets for a bi-objective multi-vehicle extension of the covering tour problem (CTP) [10]. This extension is called the bi-objective multi-vehicle CTP (BOMCTP). We compute a lower bound set by using an ε-constraint scalarization since we have a condition to efficiently determine possible values of ε.

3.1. Problem description

The CTP consists in designing a route over a subset of locations with the aim of minimizing the length of the route. In addition, each location not visited by the route should lie within a fixed radius from a visited location. The fixed radius is called the *cover distance*. The CTP has a generic application in the design of bi-level transportation networks [4]. The aim in this kind of problems is to construct a primary route of minimum length in such a way that all points that are not on it can easily reach it. Specific applications arise in the problem of choosing where to locate post boxes among a set of candidate locations [14] and also in the delivery of medical services to villages in developing countries [4,12]. A bi-objective generalization [13] as well as a multi-vehicle extension [11] of the CTP have been proposed. The cover distance in the bi-objective version is not fixed in advance but rather induced by the constructed route. It is computed by assigning each non-visited location to the closest visited location and calculating the maximum of these distances. The objectives are to minimize the length of the route as well as the *induced cover distance*. In the multi-vehicle version, the combined length of a set of routes is minimized for a fixed cover distance. In addition, all routes must start from a common location (called the depot) and the number of locations that a single route can visit is limited by a predetermined constant p.

The BOMCTP discussed in this paper can be seen as a combination of the bi-objective and the multi-vehicle versions of the CTP and it is defined on a graph $G = (V \cup W, E)$. The nodes of V represent locations which may be visited by a route whereas the members of W are to be assigned to visited nodes of V. There is a subset of nodes $T \subseteq V$ which must be visited by at least one route. In particular, $v_0 \in T$ is the depot where all routes must start and also end. Set E consist of edges connecting all pairs of nodes in $V \cup W$ and a distance matrix $D = (d_{ij})$ satisfying the triangle inequality is defined on this set. The BOMCTP consists in designing a set of routes over a subset of V which should include all nodes of T and such that each route visits not more than p nodes of $V \backslash \{v_0\}$. The two objectives are to minimize the total length of the set of routes and the cover distance induced by the set.

3.2. A set covering formulation for the BOMCTP

Let Ω represent the set of all feasible routes. A feasible route is defined as a Hamiltonian cycle over a subset of V which includes the depot and visits not more than p nodes. The cost of a route $k \in \Omega$ is given by the sum of the cost of the edges it uses and we denote it by c_k. Let variable $\theta_k = 1$ if route k is selected in the solution and $\theta_k = 0$, otherwise. Constant $a_{ik} = 1$ if route k visits node $v_i \in V \backslash \{v_0\}$ and $a_{ik} = 0$ if this is not the case. Variable z_{ij} is used to indicate whether node $w_j \in W$ is assigned to $v_i \in V \backslash \{v_0\}$ in the solution ($z_{ij} = 1$) or not ($z_{ij} = 0$). Let $\Gamma_{\max}$ be the cover distance induced by a given set of routes. Note that $\Gamma_{\max} = \max\{d_{ij} z_{ij} : v_i \in V \backslash \{v_0\}$ and $w_j \in W\}$. The BOMCTP can be described with the following set covering model.

$$\text{Minimize} \quad \sum_{k\in\Omega} c_k\theta_k \tag{3.1}$$

$$\text{Minimize} \quad \Gamma_{\max} \tag{3.2}$$

$$\text{subject to :} \quad \Gamma_{\max} - d_{ij}z_{ij} \geq 0 \quad (v_i \in V\backslash\{v_0\}, w_j \in W), \tag{3.3}$$

$$\sum_{v_i\in V\backslash\{v_0\}} z_{ij} \geq 1 \quad (w_j \in W), \tag{3.4}$$

$$\sum_{k\in\Omega} a_{ik}\theta_k - z_{ij} \geq 0 \quad (v_i \in V\backslash\{v_0\}, w_j \in W), \tag{3.5}$$

$$\sum_{k\in\Omega} a_{ik}\theta_k \geq 1 \quad (v_i \in T\backslash\{v_0\}), \tag{3.6}$$

$$\Gamma_{\max} \geq 0, \tag{3.7}$$

$$z_{ij} \in \{0,1\} \quad (v_i \in V\backslash\{v_0\}, w_j \in W), \tag{3.8}$$

$$\theta_k \in \mathbb{N} \quad (k \in \Omega). \tag{3.9}$$

In this formulation, the objectives of minimizing the total length of the set of routes and the induced cover distance are represented in (3.1) and (3.2), respectively. Constraints (3.3) indicate that the induced cover distance should be large enough to respect the distances of all assignments of a node $w_j \in W$ to a node $v_i \in V\backslash\{v_0\}$. Constraints (3.4) and (3.5) specify that each node of W should be assigned to at least one node of $V\backslash\{v_0\}$ which is visited by a route selected in the solution. The requirement that each node of $T\backslash\{v_0\}$ is visited by at least one selected route is represented in Constraints (3.6). Constraints (3.7)–(3.9) define the domains of the decision variables used. We define θ_k to be a non-negative integer instead of binary in order to prevent constraints of the form $\theta_k \leq 1$ in the linear relaxation. The optimal solution of the problem is not affected by this change.

3.3. Restricted linear programming master problem

We restrict the value of the cover distance induced by the set of routes to be at most ε to obtain the constraint

$$-\Gamma_{\max} \geq -\varepsilon. \tag{3.10}$$

The LPM is defined by the linear relaxation of the formulation with objective function (3.1) and Constraints (3.3)–(3.10). The RLPM is obtained by replacing Ω in the LPM with a subset Ω_1 for which the problem is primal feasible.

Clearly, there is a finite number of values for $\Gamma_{\max}$ in the range $[\min\varepsilon, \max\varepsilon]$. The value for $\min\varepsilon$ can be obtained when all nodes of $V\backslash\{v_0\}$ are visited by a route so that each node of W can be assigned to the node of $V\backslash\{v_0\}$ which is closest to it. In a similar way, a value for $\max\varepsilon$ can be obtained by constructing a single route of the form $v_0 \to v_i \to v_0$ where v_i is the closest node of $V\backslash\{v_0\}$ to the depot, v_0. In such a situation, v_i will be required to cover all the nodes of W and so induce a maximum reasonable value for $\Gamma_{\max}$. From the definition of the induced cover distance ($\Gamma_{\max}$), the value of d_{ij} for every couple $(v_i, w_j) \in V\backslash\{v_0\} \times W$ is a candidate value for $\Gamma_{\max}$. Nevertheless, some of these candidate values will never be the cover distance. We use an idea introduced in [13] and which we state in the next proposition to determine the values that can be the cover distance for any given instance.

Proposition 1. *Given $v_i \in V\backslash\{v_0\}$ and $w_j \in W$, d_{ij} is a feasible value for $\Gamma_{\max}$ if and only if the following two conditions are satisfied.*

1. $\forall v_t \in T\backslash\{v_0\}$ *such that* $v_t \neq v_i$, $d_{ij} \leq d_{tj}$ *and*
2. $\forall w_l \in W$ *such that* $w_l \neq w_j$, $\exists v_h \in V\backslash\{v_0\}$ *such that* $d_{hl} \leq d_{ij} \leq d_{hj}$.

Proof. The proof is given in two parts. Part I shows that if d_{ij} is a feasible value for $\Gamma_{\max}$ then conditions (1) and (2) are satisfied. Part II also shows that if conditions (1) and (2) are satisfied then d_{ij} is a feasible value for $\Gamma_{\max}$.

Part I. The necessary conditions for d_{ij} to be the value of $\Gamma_{\max}$ are that v_i is visited by a selected route and w_j is assigned to v_i. Yet, each node of W is assigned to a visited node of $V\backslash\{v_0\}$ that is closest to it. Since all nodes $v_t \in T\backslash\{v_0\}$ are visited in any feasible solution, w_j will be assigned to v_i only if $d_{ij} \leq d_{tj}$. This proves condition (1). Next, suppose that $\Gamma_{\max} = d_{ij}$ but condition (2) is false. That is, there exists $w_l \in W$ such that $w_l \neq w_j$ and either $d_{ij} > d_{hj}$ or $d_{ij} < d_{hl}$ for all $v_h \in V\backslash\{v_0\}$. Then, either w_j will be assigned to v_h (since v_h is closer to it than v_i) or if w_j is assigned to v_i then d_{hl} is a better candidate for $\Gamma_{\max}$ (since it is greater than d_{ij} and $\Gamma_{\max}$ is determined by the maximum of the assigned distances). In both cases, d_{ij} cannot be the value for $\Gamma_{\max}$ and so condition (2) must be true if $\Gamma_{\max} = d_{ij}$.

Part II. If conditions (1) and (2) are satisfied then w_j will be assigned to v_i (if it is visited) rather than to a node $v_t \in T\backslash\{v_0\}$. If v_i is visited and w_j is assigned to v_i, then condition (2) implies that d_{ij} can possibly be the maximum value among all assigned distances and so a feasible value for $\Gamma_{\max}$. □

Using the two conditions in Proposition 1, all the feasible values of $\Gamma_{\max}$ can be computed for any given instance of the BOMCTP. Thus, instead of fixing ε to the next integer value less than the value of $\Gamma_{\max}$ obtained in a previous step, we rather fix it to the next feasible value of $\Gamma_{\max}$. By doing so, some unnecessary iterations can be avoided during the computation of the set of lower bounds.

3.4. Dual of LPM

Let γ_{ij} $(v_i \in V\backslash\{v_0\}, w_j \in W)$, β_j $(w_j \in W)$, α_{ij} $(v_i \in V\backslash\{v_0\}, w_j \in W)$, π_i $(v_i \in T\backslash\{v_0\})$ and λ be the non-negative dual variables associated with Constraints (3.3), (3.4), (3.5), (3.6) and (3.10), respectively. The dual formulation of the LPM is given by:

$$\text{maximize} \quad \sum_{w_j \in W} \beta_j + \sum_{v_i \in T\setminus\{v_0\}} \pi_i - \varepsilon\lambda \tag{3.11}$$

subject to:

$$\sum_{\substack{v_i \in V\setminus\{v_0\} \\ w_j \in W}} a_{ik}\alpha_{ij} + \sum_{v_i \in T\setminus\{v_0\}} a_{ik}\pi_i \le c_k \quad (k \in \Omega), \tag{3.12}$$

$$\beta_j - d_{ij}\gamma_{ij} - \alpha_{ij} \le 0 \quad (v_i \in V\setminus\{v_0\}, w_j \in W), \tag{3.13}$$

$$\sum_{\substack{v_i \in V\setminus\{v_0\} \\ w_j \in W}} \gamma_{ij} - \lambda \le 0, \tag{3.14}$$

$$\gamma_{ij} \ge 0 \quad (v_i \in V\setminus\{v_0\}, w_j \in W), \tag{3.15}$$

$$\alpha_{ij} \ge 0 \quad (v_i \in V\setminus\{v_0\}, w_j \in W), \tag{3.16}$$

$$\beta_j \ge 0 \quad (w_j \in W), \tag{3.17}$$

$$\pi_i \ge 0 \quad (v_i \in T\setminus\{v_0\}), \tag{3.18}$$

$$\lambda \ge 0. \tag{3.19}$$

3.5. Sub-problem

The sub-problem is to search for feasible routes $k \in \Omega\setminus\Omega_1$ (Ω_1 is the set of columns in the RLPM at any given time) such that

$$c_k - \sum_{\substack{v_i \in V\setminus\{v_0\} \\ w_j \in W}} a_{ik}\alpha_{ij} - \sum_{v_i \in T\setminus\{v_0\}} a_{ik}\pi_i < 0. \tag{3.20}$$

We define $\pi_i^* = \pi_i$ if $v_i \in T\setminus\{v_0\}$ and $\alpha_{ij}^* = \alpha_{ij}$ if $v_i \in V\setminus\{v_0\}, w_j \in W$. In all other cases these variables are set to 0. We also let $x_{ijk} = 1$ if route k visits v_j immediately after visiting v_i and $x_{ijk} = 0$ if this is not the case.

Note that $c_k = \sum_{(v_i,v_j) \in E} x_{ijk} d_{ij}$ and $\sum_{v_i \in V\setminus\{v_0\}} a_{ik} = \sum_{v_i \in V} a_{ik} = \sum_{(v_i,v_j) \in E} x_{ijk}$.

The starred versions of the dual variables (α_{ij}^* and π_i^*) also allow us to write the summations over $V\setminus\{v_0\}$ and $T\setminus\{v_0\}$ as a summation over V. We get

$$\sum_{\substack{v_i \in V\setminus\{v_0\} \\ w_j \in W}} a_{ik}\alpha_{ij} = \sum_{v_i \in V} \left(\sum_{w_h \in W} \alpha_{ih}^* \right) a_{ik} = \sum_{(v_i,v_j) \in E} \left(\sum_{w_h \in W} \alpha_{ih}^* \right) x_{ijk}$$

$$\text{and} \quad \sum_{v_i \in T\setminus\{v_0\}} a_{ik}\pi_i = \sum_{v_i \in V} a_{ik}\pi_i^* = \sum_{(v_i,v_j) \in E} \pi_i^* x_{ijk}.$$

Condition (3.20) can, thus, be simplified to

$$\sum_{(v_i,v_j)\in E}\left(d_{ij}-\pi_i^*-\sum_{w_h\in W}\alpha_{ih}^*\right)x_{ijk}<0. \tag{3.21}$$

Feasible routes satisfying condition (3.21) are searched for by solving an elementary shortest path problem with resource constraints (ESPPRC) given by

$$\text{Minimize}\quad \sum_{(v_i,v_j)\in E}\left(d_{ij}-\pi_i^*-\sum_{w_h\in W}\alpha_{ih}^*\right)x_{ijk}\quad \text{subject to}\quad k\in\Omega\backslash\Omega_1. \tag{3.22}$$

That is, we seek feasible routes with negative reduced costs where the reduced cost of $(v_i, v_j) \in V \times V$ is given by $d_{ij} - \pi_i^* - \sum_{w_h\in W} \alpha_{ih}^*$. The ESPPRC is very well studied since it appears as a subproblem in many vehicle routing problems solved by column generation. The problem is $\mathcal{NP}$-hard in the strong sense [7]. Several algorithms based on dynamic programming have been proposed to solve the ESPPRC. These algorithms rely mainly on the label setting algorithm for the shortest path problem with resource constraint (SPPRC) [6] where a route may visit a given node more than once. An exact algorithm for the ESPPRC is given in [9]. We solve the subproblem by the decremental state space relaxation (DSSR) algorithm [3, 15]. The only resource constraint considered is the limit on the total number of nodes a route can visit.

4. Computational experiments

We have performed numerical experiments to evaluate the quality of lower and upper bound sets for the BOMCTP and also compare the relative performance of the different column search approaches. In order to evaluate the quality of a lower bound set and a corresponding upper bound set, we used a distance based measure (μ_1), and an area based measure (μ_2) which were presented in [8]. Combining an area based measure with a distance based measure gives a good indication of the quality of the bounds. Roughly speaking, μ_1 represents the worst distance (with respect to the range of objective values) between a point of the upper bound set and a point of lower bound set closest to it. Also, μ_2 represents the fraction of the area that is dominated by the lower bound set but not by the upper bound set. This is, the area where additional points of $\mathcal{Y}_N$ can be found. If a lower bound set and a corresponding upper bound set are good, then we expect that both μ_1 and μ_2 will be small in value. The smaller both values are, the better the quality of the bounds. These two measures complement each other and as explained by the authors of [8], the measures can be seen to play a role analogous to the optimality gap in single objective optimization. The reader is referred to the relevant paper [8] for further explanation of the measures.

Before presenting the results obtained for the BOMCTP, we first give an overview of the quality of lower bound sets obtained by the weighted sum and the ε-constraint methods for the bi-objective set covering problem.

4.1. Quality of lower bound sets for the bi-objective set covering problem

We compare the quality of lower bound sets computed by the weighted sum and ε-constraint methods for the bi-objective set covering problem (BOSCP). Since the BOSCP serves as a base formulation for most problems solved by column generation, the aim of this section is to give a general idea of the quality of lower bounds computed by the two scalarization methods for this kind of problem and its extensions. The BOSCP is formulated as :

$$\text{Minimize} \quad \sum_{j=1}^{n} c_j^1 x_j \tag{4.1}$$

$$\text{Minimize} \quad \sum_{j=1}^{n} c_j^2 x_j \tag{4.2}$$

$$\text{subject to :} \quad \sum_{j=1}^{n} a_{ij} x_j \geq 1 \qquad i = 1, 2, \ldots, m, \tag{4.3}$$

$$x_j \in \{0,1\} \quad j = 1, 2, \ldots, n. \tag{4.4}$$

The integral objective coefficients are c_j^1 and c_j^2. The matrix coefficients are $a_{ij} \in \{0,1\}$ and we say that constraint i is *covered* by variable x_j if $a_{ij} = 1$. The BOSCP instances used for the tests can be found at http://xgandibleux.free.fr/MOCOlib/MOSCP.html. They are the same instances that were used in [8]. Since the goal is to evaluate the quality of lower bound sets, we used the exact nondominated sets computed by the algorithm proposed in [2] as upper bound sets. For this reason, the values of the two quality measures obtained for the weighted sum method are different from those obtained in [8] when greedy heuristics are used to compute upper bound sets. All computer codes were written in C/C++ and the LP/MIP optimizers of ILOG CPLEX 12.4 were used. The tests were run on an Intel(R) Core(TM)2 Duo CPU E7500 @ 2.93 GHz computer with a 2 GiB RAM and a summary of the results are presented in Table 1. In this table, $|\boldsymbol{U}^*|$ represents the number of points in the exact nondominated set, *time* is the computational time in cpu seconds, $|ext|$ is the number of extreme points computed when the weighted sum method is used, and $|\boldsymbol{L}|$ is the total number of points in a lower bound set when an ε-constraint method is used. Values of the two quality measures are expressed as percentages under the headings $\mu_1\%$ and $\mu_2\%$.

The results show that, the lower bound sets obtained by both methods for almost all the instances are of good quality. The lower bounds obtained in the case of the ε-constraint method are slightly better than those obtained for the weighted sum method. The situation is quite different when we compare the computational times and the computational complexities of the two methods. In the case of the weighted sum method, relatively few extreme points need to be calculated and this is done in negligible time for the instances tested. The number of points that needs to be calculated when an ε-constraint method is used can be very large (possibly

TABLE 1. Comparison of lower bound sets for the BOSCP.

		Weighted sum				ε-Constraint			
Instance	$\lvert\boldsymbol{U}^*\rvert$	Time	$\lvert ext\rvert$	μ_1%	μ_2%	Time	$\lvert\boldsymbol{L}\rvert$	μ_1%	μ_2%
2scp11A	39	0.00	13	2.9	2.5	0.02	192	2.7	2.1
2scp11C	10	0.00	21	13.7	34.1	0.01	117	13.9	32.7
2scp41A	105	0.01	23	1.5	1.3	0.14	1028	1.4	1.2
2scp41C	24	0.02	17	2.6	4.5	0.04	280	2.7	5.7
2scp42A	206	0.02	37	0.6	0.5	0.45	1663	0.5	0.4
2scp42C	87	0.04	51	3.0	3.4	0.44	2245	2.9	3.4
2scp43A	46	0.04	63	3.6	4.3	0.12	473	3.5	4.1
2scp43C	12	0.02	38	6.3	11.1	0.04	225	6.3	10.6
2scp61A	254	0.06	58	1.1	0.8	1.39	2829	1.1	0.8
2scp61C	28	0.02	17	1.0	3.7	1.02	732	0.9	4.4
2scp62A	98	0.28	236	3.5	3.6	1.01	1240	3.5	3.7
2scp62C	6	0.23	180	38.9	46.6	0.00	2	36.6	58.5
2scp81A	424	0.07	47	0.5	0.4	3.49	5580	0.4	0.3
2scp81C	14	0.12	77	0.3	0.5	0.60	147	0.4	0.7

exponential) and this may require a very long computational time. For example, the ε-constraint method computed 5580 points in a time of 3.49 cpu s for instance 2scp81A. This is very large when we compare it with the 47 points computed in 0.07 s when a weighted sum method was used. This stresses the need to reduce the number of points calculated when using an ε-constraint method by determining good step sizes.

4.2. Experiments for the BOMCTP

We now present results from experiments conducted to evaluate the quality of lower and upper bound sets computed for the BOMCTP. We also compare the relative perfomance of the different column search approaches.

4.2.1. Description of instances and experiments

The Mersenne Twister random number generator was used to generate instances similar to those described in the literature [10, 11, 13] but which are not publicly available. The generator can be obtained at http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html. The node sets were obtained by generating $|V|+|W|$ points in the $[0,100]\times[0,100]$ square with the depot restricted to lie in $[25,75]\times[25,75]$. Set T (respectively, V) is taken to be the first $|T|$ (respectively, $|V|$) points and set W is taken as the remaining points. The distance between two points is calculated as the Euclidean distance between them. Five instances for every combination of $|V|\in\{30,40,50\}$ and $|W|\in\{2|V|,3|V|\}$ were generated. Values of $|T|$ in $\{1,\lceil 0.25|V|\rceil,\lceil 0.50|V|\rceil\}$ and p in $\{5,8\}$ were tested. The exact instances used for our experiments can be found at http://homepages.laas.fr/bmsarpon/ctp_instances.zip. All computer codes were written in C/C++ and

TABLE 2. Quality of bound sets for the BOMCTP ($p = 5$).

				PPS			k-PPS			Sequential		
$\|T\|$	$\|V\|$	$\|W\|$	$\|\boldsymbol{L}^*\|$	$\|\boldsymbol{U}\|$	$\mu_1\%$	$\mu_2\%$	$\|\boldsymbol{U}\|$	$\mu_1\%$	$\mu_2\%$	$\|\boldsymbol{U}\|$	$\mu_1\%$	$\mu_2\%$
1	30	60	64	29	5.2	26.2	28	5.2	26.1	29	5.1	26.0
1	30	90	66	29	6.0	27.6	29	6.1	27.5	28	5.9	27.5
1	40	80	58	30	5.5	26.1	29	5.6	26.0	30	5.4	25.9
1	40	120	67	37	5.1	25.4	29	5.1	25.4	36	5.0	25.2
1	50	100	65	34	4.9	24.7	34	4.9	24.5	35	4.8	24.5
1	50	150	67	35	4.5	24.8	39	4.6	24.7	36	4.5	24.0
8	30	60	15	9	7.7	54.1	9	7.7	54.1	9	7.6	54.1
8	30	90	17	10	11.0	53.3	10	11.1	53.3	10	11.0	53.2
10	40	80	15	11	1.4	58.1	11	1.4	58.1	12	1.3	58.1
10	40	120	16	13	9.1	61.7	12	9.1	61.7	14	9.0	61.8
13	50	100	18	11	16.7	72.9	11	16.7	80.0	11	16.6	72.7
13	50	150	19	10	21.6	72.6	11	21.5	72.7	12	21.6	72.9
15	30	60	2	6	20.3	54.7	5	20.3	54.6	6	20.3	54.6
15	30	90	5	6	40.5	55.7	6	40.5	55.7	6	40.4	55.7
20	40	80	6	4	57.3	61.4	5	57.3	61.4	5	57.3	61.4
20	40	120	7	5	31.2	62.3	6	31.2	62.3	6	31.2	62.3
25	50	100	6	4	24.3	76.8	4	24.3	76.5	4	24.3	76.8
25	50	150	9	4	28.9	79.1	4	28.9	78.9	5	28.9	79.0

the RMP was solved with ILOG CPLEX 12.4. Tests were run on an Intel(R) Core(TM)2 Duo CPU E7500 @ 2.93GHz computer with a 2 GiB RAM. Summary of the results are given in Tables 2 to 5. All the values in these tables are averages over the five generated instances.

The condition we used in implementing the k-PPS, is to skip a current value of ε when the objective value of the RLPM has not improved (*i.e.* decreased) by more than 1 after $k = 3$ iterations of column generation. Given that column generation is an exact method for the RMP, the same lower bound set is obtained for a given instance irrespective of the column search approach used. The number of elements in the common lower bound set for each instance is given under the column with heading $|\boldsymbol{L}^*|$. The number of elements in an upper bound set is given under the columns with heading $|\boldsymbol{U}|$. We recall that for each approach, an upper bound set is computed after computing a lower bound set by following the idea of PPS. The only difference is that we solve integer programs rather than linear relaxations. For each instance, these values are different for the different column search approaches since different columns are generated by each approach when computing a lower bound set. The columns with headings *dssr* and *cols* give the number of times the sub-problem was solved with the DSSR algorithm and the total number of columns generated, respectively, when computing a lower bound set. All the other column headings have the same meanings as those for the BOSCP in Table 1.

TABLE 3. Quality of bound sets for the BOMCTP ($p = 8$).

				PPS			k-PPS			Sequential		
$\|T\|$	$\|V\|$	$\|W\|$	$\|\boldsymbol{L}^*\|$	$\|\boldsymbol{U}\|$	$\mu_1\%$	$\mu_2\%$	$\|\boldsymbol{U}\|$	$\mu_1\%$	$\mu_2\%$	$\|\boldsymbol{U}\|$	$\mu_1\%$	$\mu_2\%$
1	30	60	64	29	5.3	26.3	29	5.3	26.1	30	5.2	26.2
1	30	90	63	30	6.5	28.2	29	6.5	28.1	29	6.4	28.0
1	40	80	58	28	5.6	26.0	28	5.5	26.0	30	5.6	26.0
1	40	120	65	35	5.2	25.7	26	5.3	25.4	36	5.1	25.2
1	50	100	65	34	4.9	24.7	34	4.9	24.6	33	4.7	24.6
1	50	150	64	36	4.7	24.8	38	4.6	24.5	36	4.7	24.4
8	30	60	15	10	10.2	65.1	10	10.2	65.0	10	10.2	65.1
8	30	90	15	9	28.7	69.5	10	26.8	69.7	10	20.3	69.6
10	40	80	14	8	16.6	68.0	8	16.6	67.9	9	16.6	67.8
10	40	120	17	11	9.3	73.4	10	9.3	73.3	11	9.2	73.3
13	50	100	16	10	35.2	74.2	10	35.2	74.2	10	35.1	74.2
13	50	150	17	9	30.1	76.7	9	30.1	76.7	10	30.1	76.7
15	30	60	2	6	29.0	65.4	6	29.0	65.4	6	29.0	65.4
15	30	90	2	6	35.4	71.1	6	35.4	71.3	6	35.4	71.0
20	40	80	2	4	41.2	72.9	5	41.3	73.0	5	41.1	72.7
20	40	120	4	6	9.2	73.8	6	9.1	73.8	6	9.1	73.2
25	50	100	3	5	34.9	81.3	5	35.0	81.1	6	34.9	81.0
25	50	150	4	5	24.3	87.9	6	24.2	87.3	7	24.2	87.2

TABLE 4. Computational times for the BOMCTP ($p = 5$).

			PPS			k-PPS			Sequential		
$\|T\|$	$\|V\|$	$\|W\|$	Time	dssr	cols	Time	dssr	cols	Time	dssr	cols
1	30	60	7.78	160	358	6.72	126	388	7.00	98	372
1	30	90	12.81	93	332	10.82	95	364	9.13	73	351
1	40	80	14.99	72	416	11.37	70	479	7.83	59	461
1	40	120	15.98	129	435	12.51	112	521	7.76	90	478
1	50	100	25.96	152	570	13.33	147	574	8.81	135	583
1	50	150	24.66	137	591	15.29	116	573	11.51	106	619
8	30	60	6.32	165	808	8.74	147	823	7.89	123	747
8	30	90	15.28	143	837	13.06	139	891	14.14	104	865
10	40	80	18.60	157	969	19.37	142	1036	15.60	132	974
10	40	120	28.21	195	1048	28.16	176	986	23.92	155	1096
13	50	100	42.39	243	1160	39.49	198	1171	36.34	174	1235
13	50	150	51.69	286	1395	52.67	201	1343	42.11	194	1480
15	30	60	16.81	149	825	15.08	126	891	13.97	114	926
15	30	90	26.41	153	973	25.54	148	974	23.78	141	969
20	40	80	35.83	168	1232	36.83	160	1235	27.43	143	1198
20	40	120	53.45	192	1568	50.74	174	1549	42.81	158	1480
25	50	100	55.49	227	1913	49.03	195	1629	44.19	172	1670
25	50	150	68.22	285	2194	61.64	223	2160	53.30	206	2075

TABLE 5. Computational times for the BOMCTP ($p = 8$).

			PPS			k-PPS			Sequential		
$\|T\|$	$\|V\|$	$\|W\|$	Time	dssr	cols	Time	dssr	cols	Time	dssr	cols
1	30	60	11.82	245	374	15.83	183	392	6.85	119	449
1	30	90	14.86	144	357	14.23	141	383	8.33	132	417
1	40	80	53.97	124	436	22.24	109	451	17.54	93	491
1	40	120	47.20	154	438	29.32	127	492	18.30	120	524
1	50	100	96.60	189	618	22.55	142	575	15.09	132	633
1	50	150	86.50	199	566	31.13	132	573	19.58	111	578
8	30	60	52.13	231	738	53.92	211	718	48.12	176	619
8	30	90	58.00	195	953	62.17	176	949	51.30	164	946
10	40	80	81.38	206	1087	76.11	184	1093	69.26	153	1074
10	40	120	69.48	237	988	61.90	201	1027	54.67	159	1080
13	50	100	123.70	273	1439	115.14	258	1335	79.89	213	1326
13	50	150	110.41	294	1712	103.65	272	1726	87.10	219	1766
15	30	60	42.09	259	952	42.89	209	968	38.16	182	825
15	30	90	60.33	198	1071	58.67	177	1108	46.50	169	1151
20	40	80	153.70	281	1209	129.85	275	1160	109.25	227	1080
20	40	120	118.65	296	1841	103.09	278	1833	98.37	254	1766
25	50	100	209.85	317	2058	196.46	297	2051	182.32	261	2049
25	50	150	264.91	334	2314	251.92	312	2217	201.49	259	2147

4.2.2. Discussion of results

A first general comment is that, the number of points in a bound set decreases when the size of T increases. This can be seen in Tables 2 and 3 under the column headings $|\boldsymbol{L}^*|$ and $|\boldsymbol{U}|$. From these two tables, the quality of the bound sets obtained are quite good. There seem to be no clear preference for one column search approach with respect to another in terms of the quality of the bound sets obtained. Nevertheless, the quality of bound sets obtained by the sequential search approach are slightly better than those obtained by the PPS and k-PPS in most cases. This can be seen, for example, in row $|T| = 1$, $|V| = 40$, $|W| = 120$ of Table 3. The value of the pair $(\mu_1\%, \mu_2\%)$ for the PPS, the k-PPS, and the sequential search approaches are (5.2, 25.7), (5.3, 25.4), and (5.1, 25.2), respectivley.

In terms of computational times (Tabs. 4 and 5), the sequential search approach performs the best. The k-PPS also performs better than the PPS. For example, in the row for $|T| = 13$, $|V| = 50$, $|W| = 150$ of Table 5, the computational time for the PPS is 110.41, the one for the k-PPs is 103.65 and that of the sequential search is 87.10. From the same row, we can also see that the sequential search approach solves a fewer number of subproblems (219) with respect to the PPS (294) and the k-PPS (272). This is a very good sign in favour of the sequential search approach since solving the subproblem is usually the most costly operation in a column generation algorithm.

5. Conclusions

This paper discusses the use of column generation in computing bound sets for bi-objective combinatorial optimization problems. The main idea used is to convert the bi-objective problem into single objective by a scalarization method and solve the linear relaxation of the resulting single objective problem several times by varying the necessary parameters. Two very popular scalarization methods (weighted sum and ε-constraint) are studied for this purpose. We saw that the subproblems encountered for both scalarization methods have a similar structure. For this reason, we presented some strategies to effectively search for columns in order to possibly take advantage of the similar subproblems. The k-PPS and the sequential search approaches were proposed to address some challenges faced by a rather standard approach (the PPS). The ideas presented are applied to the bi-objective multi-vehicle covering tour problem and the results show that the proposed methodology and column search approaches are effective. In particular, the sequential search method is faster and also solves relatively few number of subproblems when computing a lower bound set. A possible extension of this current work is to test the presented strategies in the case of the weighted sum method and also develop other similar strategies. It will also be interesting to develop column generation based heuristics and metaheuristics that use the idea of the sequential search method.

References

[1] Y.P. Aneja and K.P.K. Nair, Bicriteria transportation problem. *Manage. Sci.* **25** (1979) 73–78.

[2] J.-F. Bérubé, M. Gendreau and J.-Y. Potvin, An exact ϵ-constraint method for bi-objective combinatorial optimization problems: Application to the Traveling Salesman Problem with Profits. *Eur. J. Oper. Res.* **194** (2009) 39–50.

[3] N. Boland, J. Dethridge and I. Dumitrescu, Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Oper. Res. Lett.* **34** (2006) 58–68.

[4] J.R. Current and D.A. Schilling, The median tour and maximal covering tour problems: Formulations and heuristics. *Eur. J. Oper. Res.* **73** (1994) 114–126.

[5] C. Delort and O. Spanjaard, Using bound sets in multiobjective optimization: Application to the biobjective binary knapsack problem, in *Experimental Algorithms*, Springer (2010) 253–265.

[6] M. Desrochers and F. Soumis, A generalized permanent labelling algorithm for the shortest path problem with time windows. *INFOR* **26** (1988) 191–212.

[7] M. Dror. Note on the complexity of shortest path models for column generation in VRPTW. *Oper. Res.* **42** (1994) 977–978.

[8] M. Ehrgott and X. Gandibleux, Bound sets for biobjective combinatorial optimization problems. *Comput. Oper. Res.* **34** (2007) 2674–2694.

[9] D. Feillet, P. Dejax, M. Gendreau and C. Gueguen, An exact algorithm for the Elementary Shortest Path Problem with Resource Constraints: Application to some vehicle routing problems. *Networks* **44** (2004) 216–229.

[10] M. Gendreau, G. Laporte and F. Semet, The Covering tour problem. *Oper. Res.* **45** (1997) 568–576.

[11] M. Hachicha, M.J. Hodgson, G. Laporte and F. Semet, Heuristics for the multi-vehicle covering tour problem. *Comput. Oper. Res.* **27** (2000) 29–42.

[12] M.J. Hodgson, G. Laporte and F. Semet, A Covering Tour Model for Planning Mobile Health Care Facilities in SuhumDistrict, Ghama. *J. Regional Sci.* **38** (1998) 621–638.

[13] N. Jozefowiez, F. Semet and E.-G. Talbi, The bi-objective covering tour problem. *Comput. Oper. Res.* **34** (2007) 1929–1942.

[14] M. Labbé and G. Laporte. Maximizing user convenience and postal service efficiency in post box location. *Belgian J. Oper. Res. Stat. Comput. Sci.* **26** (1986) 21–35.

[15] G. Righini and M. Salani, New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks* **51** (2008) 155–170.

[16] E. Salari and J. Unkelbach, A column-generation-based method for multi-criteria direct aperture optimization. *Phys. Med. Biol.* **58** (2013) 621–639.

[17] F. Sourd and O. Spanjaard, A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem. *INFORMS J. Comput.* **20** (2008) 472–484.

[18] E.L. Ulungu and J. Teghem. The two phases method: An efficient procedure to solve biobjective combinatorial optimization problems. *Found. Comput. Dec. Sci.* **20** (1995) 149–165.

[19] F. Vanderbeck, Implementing mixed integer column generation, in *Column Generation*, edited by G. Desaulniers, J. Desrosiers and M.M. Solomon. Springer (2005) 331–358.

[20] B. Villarreal and M.H. Karwan, Multicriteria integer programming: A (hybrid) dynamic programming recursive approach. *Math. Prog.* **21** (1981) 204–223.