Math. Prog. Comp. (2014) 6:199–231
DOI 10.1007/s12532-014-0065-z

FULL LENGTH PAPER

An exact cooperative method for the uncapacitated facility location problem

Marius Posta · Jacques A. Ferland · Philippe Michelon

Received: 25 March 2012 / Accepted: 13 January 2014 / Published online: 9 February 2014

Abstract In this paper, we present a cooperative primal-dual method to solve the uncapacitated facility location problem exactly. It consists of a primal process, which performs a variation of a known and effective tabu search procedure, and a dual process, which performs a lagrangian branch-and-bound search. Both processes cooperate by exchanging information which helps them find the optimal solution. Further contributions include new techniques for improving the evaluation of the branch-and-bound nodes: decision-variable bound tightening rules applied at each node, and a subgradient caching strategy to improve the bundle method applied at each node.

Mathematics Subject Classification 90B80 · 90C56 · 90C57

1 Introduction

The uncapacitated facility location problem is a well-known combinatorial optimization problem, also known as the warehouse location problem and as the simple plant location problem. The problem consists in choosing among n locations where plants can be built to service m customers. Building a plant at a location $i \in \{1, \ldots, n\}$ incurs

M. Posta (✉) · J. A. Ferland
Département d'Informatique et de Recherche Opérationelle, Université de Montréal,
C.P. 6128, Succursale Centre-Ville, Montreal, QC H3C 3J7, Canada
e-mail: postamar@iro.umontreal.ca

J. A. Ferland
e-mail: ferland@iro.umontreal.ca

P. Michelon
Laboratoire d'Informatique d'Avignon, Université d'Avignon et des Pays du Vaucluse,
84911 Avignon Cedex 9, France
e-mail: philippe.michelon@univ-avignon.fr

a fixed opening cost $\mathbf{c}_i$, and servicing a customer $j \in \{1, \ldots, m\}$ from this location incurs a service cost $\mathbf{s}_{ij}$. The objective is to minimize the total cost:

$$\min_{\mathbf{x}\in\{0,1\}^n} f(\mathbf{x}), \quad \text{with } f(\mathbf{x}) = \sum_{i=1}^{n} \mathbf{c}_i \mathbf{x}_i + \sum_{j=1}^{m} \min\left\{\mathbf{s}_{ij} : \mathbf{x}_i = 1, i \in \{1, \ldots, n\}\right\}.$$

Note that all solutions in $\{0, 1\}^n$ are feasible except for $\mathbf{0}$. The problem can be formulated as a 0–1 mixed integer programming problem. This requires the introduction of a vector of auxiliary variables $\mathbf{y} \in [0, 1]^{nm}$ in which each component $\mathbf{y}_{ij}$ represents the servicing of customer j by location i. The following model is the so-called 'strong' formulation, denoted by (P):

$$\min \quad \sum_{i=1}^{n} \mathbf{c}_i \mathbf{x}_i + \sum_{j=1}^{m} \sum_{i=1}^{n} \mathbf{s}_{ij} \mathbf{y}_{ij} \quad (P)$$

$$\text{s.to} \quad \sum_{i=1}^{n} \mathbf{y}_{ij} = 1, \qquad 1 \le j \le m, \tag{1}$$

$$0 \le \mathbf{y}_{ij} \le \mathbf{x}_i, \qquad 1 \le i \le n, 1 \le j \le m, \tag{2}$$

$$\mathbf{x} \in \{0, 1\}^n. \tag{3}$$

Note that this model has at least one optimal solution in which $\mathbf{y}$ is integer. Note also that vectors are represented by symbols in boldface font while scalars are in normal font, a convention which we shall adhere to throughout this paper.

Many different approaches to solving the UFLP have been studied over the years to the extent that a comprehensive overview of the literature falls beyond the scope of this paper. We refer the reader to [9,24,26] for relevant survey articles. More recent publications include approximation algorithms [7,12,29,30], lagrangian relaxation [2,27], metaheuristics [10,14,18,19,21,22,25,31,35,36], etc.

Most recent methods for solving the UFLP exactly are centered on the dual ascent methods proposed by Erlenkotter [11] and by Bilde and Krarup [5]. The `DualLoc` method of Erlenkotter, subsequently improved by Körkel [23] and recently by Letchford and Miller [28], is a branch-and-bound method in which a lower bound at each node is evaluated by a dual ascent heuristic. Being a linear program, the dual can also be solved by general-purpose solvers such as CPLEX or Gurobi, or by other specialized methods [8]. It seems however that dual ascent heuristics have been preferred, possibly because the exact resolution of such (often large) linear programs was not found to be cost-effective from a computational point of view. A study of the optimality gap of the dual ascent heuristic can be found in [32].

A different method which was proposed recently for solving the UFLP exactly is the semi-lagrangian approach of Beltran-Royo et al. [4], originally developed for solving the k-median problem [3]. Other methods include the primal-dual method proposed by Galvão and Raggi [13], which combines a primal procedure to find a good solution and a dual procedure to close the optimality gap. The heuristic proposed by Hansen et al. [19] can be adapted to solve the UFLP exactly. On a related note, Goldengorin et al. [16,17] propose some preprocessing rules to improve branch-and-bound search methods applied to the UFLP. These rules analyze the relationships

between the opening costs $\mathbf{c}$ and the service costs $\mathbf{s}$ to try to determine the optimal state of each location. However, many of the more difficult instances in the `UflLib` collection are purposely designed to have a contrived cost structure on which few deductions can be made. General purpose MIP solvers such as CPLEX or Gurobi struggle to solve these instances, in many cases spending a very long time just to evaluate the root node. Indeed, the constraints (2) make the linear relaxation of (P) difficult to solve using simplex algorithms because of degeneracy issues. There are also mn of them, and for many instances in `UflLib` we have $n = m = 1000$. It is possible to aggregate these constraints into $\sum_{j=1}^{m} \mathbf{y}_{ij} \leq m\mathbf{x}_i$ for all $i \in \{1, \ldots, n\}$, but the bound induced by the corresponding linear relaxation is too weak to be of any practical use, even when strengthened with cuts.

We propose a new method for solving (P) exactly, in which a metaheuristic and a branch-and-bound procedure cooperate to reduce the optimality gap. In this respect, this is somewhat similar to the work of Galvão and Raggi [13], however our method is *cooperative*: the metaheuristic and the branch-and-bound procedure exchange information such that they perform better than they would in isolation. Information is exchanged by message-passing between two processes. Each message has a sender, a receiver, a send date, a type and maybe some additional data. Each process sends and receives messages asynchronously at certain specific moments during their execution, and if at any of these moments there are several incoming messages of the same type, then all but the most recent are discarded. Similar approaches have been proposed before, a recent example is the work of Muter et al. [33] on solving the set-covering problem, but to our knowledge this paper introduces the first cooperative approach specifically for solving the UFLP exactly.

Section 2 describes the *primal process*, which improves an upper bound by searching for solutions through a metaheuristic. Section 3 describes the *dual process*, which improves a lower bound by enumerating a branch-and-bound search tree, using lower bounds derived from the lagrangian relaxation obtained by dualizing constraints (1). Contrary to most work in the literature which use dual-ascent heuristics (e.g., `DualLoc`), we use a bundle method to optimize the corresponding lagrangian dual. Additional contributions of this paper include new strategies for partitioning (P), as well as a subgradient caching technique to improve the performance of bundle method in the context of our branch-and-bound procedure.

Section 4 concludes this paper by presenting our computational results. Our method is very effective on almost all UFLP instances in `UflLib`, and solved many to optimality for the first time. These new optimal values can be found in the appendix, along with several implementation details which do not enrich this paper.

2 Primal process

The purpose of the primal process is to find good solutions for (P). Specifically, we use a variant of the tabu search procedure proposed by Michel and Van Hentenryck [31] which we shall now briefly describe. This local search method explores the 1OPT neighborhood of a given solution $\check{\mathbf{x}} \in \{0, 1\}^n$. This neighborhood is defined as the set of solutions obtained by flipping one component of $\check{\mathbf{x}}$: either opening a closed location, or closing an open location.

We let $\check{\mathbf{x}}^i$ denote the neighbor of $\check{\mathbf{x}}$ obtained by flipping its i-th component. For all components $i \in \{1, \ldots, n\}$, we let $\boldsymbol{\delta}_i$ denote the transition cost $f(\check{\mathbf{x}}^i) - f(\check{\mathbf{x}})$ associated with moving from $\check{\mathbf{x}}$ to $\check{\mathbf{x}}^i$. Michel and Van Hentenryck [31] show how to update the vector of transition costs $\boldsymbol{\delta}$ in $O(m \log n)$ time after each move in the 1OPT neighborhood (see also Sun [36]). They argue that this efficient evaluation of the neighborhood is the main reason why their local search scheme performs as well as it does. The basic idea is to maintain t he open locations in a heap for each customer $j \in \{1, \ldots, m\}$, ordered in increasing supply costs $\mathbf{s}_{ij}$: when opening a closed location add this location inside each heap, and when closing an open location remove this location from each heap.

Our tabu search procedure is a variant of that presented in [31]. Our main modification is that it allows some solution components to be fixed to their current values by having a tabu status with infinite tenure. Tabu status is stored in an array $\boldsymbol{\tau}$ of dimension n, where each component can take any value in $\mathbb{N} \cup \{\infty\}$ specifying the iteration at which the tabu status expires.

At each iteration, the search performs a 1OPT-move on a location $w \in \{1, \ldots, n\}$ selected as follows. The location w is the one decreasing the objective function the most. Furthermore the move should not be tabu or it should satisfy the aspiration criterion. If no such move exists, then we perform a diversification by selecting w at random among all non-tabu locations. If none of these exist, then we select w at random among all non-fixed locations. If none of these exist either, then there is nothing left to do and we end the search.

We modify the tabu status $\boldsymbol{\tau}_w$ according to the current tenure variable `tenure`, which indicates the number of subsequent iterations during which $\check{\mathbf{x}}_w$ cannot change. The value of `tenure`, initially set to 10, remains within the interval [2, 10]. During the search it is decreased by 1 following each improving move (if `tenure` > 2) and it is increased by 1 following each non-improving move (if `tenure` < 10).

Before performing the next iteration, the primal process performs its communication duties, dealing with incoming messages and sending outgoing messages to the dual process. Although this tabu search procedure works well enough for finding good solutions [31] on its own, communication with the dual process nonetheless allows us (among other things) to fix some solution components to their optimal values, thereby narrowing the search space. The following list specifies the kinds of messages our primal process sends and receives:

- Outgoing:
  - send the incumbent solution $\bar{\mathbf{x}}^{\texttt{primal}}$,
  - send a request for a *guiding solution*.
- Incoming:
  - receive a solution $\bar{\mathbf{x}}^{\texttt{dual}}$,
  - receive an *improving partial solution* $\bar{\mathbf{p}}$,
  - receive a guiding solution $\underline{\mathbf{x}}$.

Let $\bar{\mathbf{x}}^{\texttt{primal}}$ be the best solution known by the primal process. Whenever $\check{\mathbf{x}}$ is such that $f(\check{\mathbf{x}}) < f(\bar{\mathbf{x}}^{\texttt{primal}})$, the primal process sets $\bar{\mathbf{x}}^{\texttt{primal}}$ to $\check{\mathbf{x}}$ and sends it to the dual process as soon as possible. The dual process likewise maintains its own incumbent solution $\bar{\mathbf{x}}^{\texttt{dual}}$, which like $\bar{\mathbf{x}}^{\texttt{primal}}$ is a 0-1 n-vector. As we shall see further on, the

dual process may improve $\bar{\mathbf{x}}^{\texttt{dual}}$ through its own efforts. In this case, the dual process sends $\bar{\mathbf{x}}^{\texttt{dual}}$ to the primal process, which will update $\bar{\mathbf{x}}^{\texttt{primal}}$ if (as is likely) it is improved upon.

A guiding solution $\underline{\mathbf{x}} \in \{0, 1\}^n$ is used to perturb $\boldsymbol{\tau}$ such that a move involving any location $i \in \{1, \ldots, n\}$ such that $\check{\mathbf{x}}_i = \underline{\mathbf{x}}_i$ becomes tabu. As a consequence, $\check{\mathbf{x}}$ moves closer to $\underline{\mathbf{x}}$ during the next iterations. This is similar in spirit to path-relinking [15].

An improving partial solution $\bar{\mathbf{p}} \in \{0, 1, *\}^n$ defines a solution subspace which is known to contain all solutions cheaper than $\bar{\mathbf{x}}^{\texttt{dual}}$, and hence $\bar{\mathbf{x}}^{\texttt{primal}}$ upon synchronization. In other words, given any solution $\mathbf{x} \in \{0, 1\}^n$, if there exists $i \in \{1, \ldots, n\}$ for which $\bar{\mathbf{p}}_i \in \{0, 1\}$ and such that $\mathbf{x}_i \neq \bar{\mathbf{p}}_i$, then we know that $f(\mathbf{x}) \geq f(\bar{\mathbf{x}}^{\texttt{primal}})$. We shall see further down how the dual process generates $\bar{\mathbf{p}}$ at any given time by identifying common elements in all the remaining unfathomed leaves of the branch-and-bound search tree. In any case, having such a partial solution allows us to restrict the search space by fixing the corresponding components of $\check{\mathbf{x}}$, and consequently this speeds up the tabu search procedure.

Algorithm 1 summarizes our primal process. In the following section, we present our dual process.

3 Dual process

Our dual process computes and improves a lower bound for (P) by enumerating a lagrangian branch-and-bound search tree. In other words, the problem (P) is recursively partitioned into subproblems, and for each of these subproblems we compute a lower bound by optimizing a lagrangian dual. Recall that a lagrangian dual function maps a vector of lagrangian multipliers to the optimal value of the lagrangian relaxation parameterized by these multipliers.

This section is organized into subsections as follows. In Sect. 3.1 we specify the subproblems into which we recursively partition (P) and we explain how we compute a lower bound for them, given any vector of multipliers. In Sect. 3.2 we present our branch-and-bound procedure and the dual process. In Sect. 3.4 we explain how we search for a vector of multipliers inducing a good lower bound for a given node.

3.1 Subproblems and lagrangian relaxation

In general, a branch-and-bound method consists in expanding a search tree by separating open leaf nodes into subnodes. This is typically done by selecting a particular 0–1 variable which is unfixed in the open node, then fixing it to 0 in one subnode and to 1 in the other. In our method we apply this to the location variables $\mathbf{x}_i$, however we also separate on the number of open locations, i.e. by limiting the value of $\sum_{i=1}^{n} \mathbf{x}_i$ within an interval $[\underline{n}, \overline{n}]$, with $1 \leq \underline{n} \leq \overline{n} \leq n$. For this reason, we shall consider subproblems of (P) defined by the following parameters:

- a partial solution vector $\mathbf{p} \in \{0, 1, *\}^n$ specifying some locations which are forced to be open or closed,
- two integers $\underline{n}$ and $\overline{n}$ specifying the minimum and maximum number of open locations.

Algorithm 1 - Primal process:

0. Initialize the tabu search procedure with any solution $\check{\mathbf{x}} \in \{0,1\}^n$ of cost $\check{z} = f(\check{\mathbf{x}})$, and compute the corresponding transition cost vector $\boldsymbol{\delta}$. Initialize the incumbent solution $\bar{\mathbf{x}}^{\texttt{primal}} \leftarrow \check{\mathbf{x}}$, the iteration counter $\texttt{nmoves} \leftarrow 1$, the tabu status array $\boldsymbol{\tau} \leftarrow (0, 0, \ldots, 0)$, and the tabu tenure duration $\texttt{tenure} \leftarrow 10$. Initialize the counter $\texttt{elapsed} \leftarrow 0$.
1. Select w using the following index sets:

$$I^0 = \{i \in \{1, \ldots, n\} : \boldsymbol{\tau}_i < \infty\}$$
$$I^1 = \{i \in I^0 : \boldsymbol{\tau}_i < \texttt{nmoves}\}$$
$$I^2 = \{i \in I^1 : \boldsymbol{\delta}_i < 0\} \cup \{i \in I^0 : \boldsymbol{\delta}_i < f(\bar{\mathbf{x}}^{\texttt{primal}}) - f(\check{\mathbf{x}})\}$$

   (a) if I^2 is nonempty, then select w at random in $\arg\min\{\boldsymbol{\delta}_i : i \in I^2\}$,
   (b) else if I^1 is nonempty, then select w at random in I^1,
   (c) else if I^0 is nonempty, then select w at random in I^0,
   (d) else end the search because all components of $\check{\mathbf{x}}$ are fixed.
2. If $\boldsymbol{\delta}_w < 0$, then decrement $\texttt{tenure}$ by 1 if $\texttt{tenure} > 2$, else increment $\texttt{tenure}$ by 1 if $\texttt{tenure} < 10$. Update $\boldsymbol{\tau}_w \leftarrow \texttt{nmoves} + \texttt{tenure}$.
3. Perform the move:
   (a) flip the w-th component of $\check{\mathbf{x}}$,
   (b) update $\boldsymbol{\delta}$ accordingly in $O(m \log n)$ time,
   (c) increment $\texttt{nmoves}$ and $\texttt{elapsed}$ by 1,
   (d) if $f(\check{\mathbf{x}}) < f(\bar{\mathbf{x}}^{\texttt{primal}})$, then update $\bar{\mathbf{x}}^{\texttt{primal}} \leftarrow \check{\mathbf{x}}$, reset $\texttt{elapsed} \leftarrow 0$, and send the new $\bar{\mathbf{x}}^{\texttt{primal}}$ to the dual process.
4. Do asynchronously and in no particular order:
   – If $\texttt{elapsed}$ has reached a predefined threshold, then reset $\texttt{elapsed} \leftarrow 0$ and send a request to the dual process for a guiding solution.
   – Upon reception of a solution $\bar{\mathbf{x}}^{\texttt{dual}}$ of cost $f(\bar{\mathbf{x}}^{\texttt{dual}}) < f(\bar{\mathbf{x}}^{\texttt{primal}})$:
     (a) set $\bar{\mathbf{x}}^{\texttt{primal}} \leftarrow \bar{\mathbf{x}}^{\texttt{dual}}$, set $\check{\mathbf{x}} \leftarrow \bar{\mathbf{x}}^{\texttt{dual}}$ and rebuild $\boldsymbol{\delta}$,
     (b) for all $i \in \{1, \ldots, n\}$ for which $\boldsymbol{\tau}_i < \infty$, set $\boldsymbol{\tau}_i \leftarrow 0$.
   – Upon reception of an improving partial solution $\bar{\mathbf{p}} \in \{0, 1, *\}^n$, for all $i \in \{1, \ldots, n\}$ for which $\bar{\mathbf{p}}_i = 0$ or 1:
     (a) set $\boldsymbol{\tau}_i \leftarrow \infty$,
     (b) if $\check{\mathbf{x}}_i \neq \bar{\mathbf{p}}_i$, then flip the i-th component of $\check{\mathbf{x}}$, update $\boldsymbol{\delta}$, and increment $\texttt{nmoves}$ by 1.
   – Upon reception of a guiding solution $\underline{\mathbf{x}}$, for all $i \in \{1, \ldots, n\}$ for which $\boldsymbol{\tau}_i < \infty$ and $\underline{\mathbf{x}}_i = \check{\mathbf{x}}_i$, set $\boldsymbol{\tau}_i \leftarrow \texttt{nmoves} + \texttt{tenure}$.
5. Return to step 1.

We let $SP(\mathbf{p}, \underline{n}, \overline{n})$ denote the subproblem formulated as follows:

$$\min \quad \sum_{i=1}^{n} \mathbf{c}_i \mathbf{x}_i + \sum_{j=1}^{m} \sum_{i=1}^{n} \mathbf{s}_{ij} \mathbf{y}_{ij} \quad (SP(\mathbf{p}, \underline{n}, \overline{n}))$$

$$\text{s.to} \quad \sum_{i=1}^{n} \mathbf{y}_{ij} = 1, \qquad 1 \le j \le m, \tag{1}$$

$$0 \le \mathbf{y}_{ij} \le \mathbf{x}_i, \qquad 1 \le i \le n, 1 \le j \le m, \tag{2}$$

$$\mathbf{x} \in \{0,1\}^n, \tag{3}$$

$$\mathbf{x}_i = \mathbf{p}_i, \qquad 1 \le i \le n, \ \ \mathbf{p}_i \in \{0,1\}, \tag{4}$$

$$\underline{n} \le \sum_{i=1}^{n} \mathbf{x}_i \le \overline{n}. \tag{5}$$

We compute lower bounds for such subproblems using the lagrangian relaxation obtained by dualizing the m constraints (1). We let $LR(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu})$ denote the lagrangian relaxation of $SP(\mathbf{p}, \underline{n}, \overline{n})$ using the vector of multipliers $\boldsymbol{\mu} \in \mathbb{R}^m$, and let $\phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu})$ denote the lagrangian dual at $\boldsymbol{\mu}$:

$$\phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}) = \min \sum_{i=1}^{n} \mathbf{c}_i \mathbf{x}_i + \sum_{j=1}^{m} \sum_{i=1}^{n} \mathbf{s}_{ij} \mathbf{y}_{ij} + \sum_{j=1}^{m} \boldsymbol{\mu}_j \left(1 - \sum_{i=1}^{n} \mathbf{y}_{ij}\right)$$
$$\text{s.to } (2-5);$$

To eliminate the decision variables $\mathbf{y}$ from the formulation, we introduce a vector of reduced costs $\bar{\mathbf{c}}^{\boldsymbol{\mu}}$ defined as follows for all $i \in \{1, \ldots, n\}$:

$$\bar{\mathbf{c}}_i^{\boldsymbol{\mu}} = \mathbf{c}_i + \sum_{j=1}^{m} \min\left\{0, \mathbf{s}_{ij} - \boldsymbol{\mu}_j\right\}.$$

This yields the following formulation for $LR(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu})$:

$$\phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}) = \sum_{j=1}^{m} \boldsymbol{\mu}_j + \min \sum_{i=1}^{n} \bar{\mathbf{c}}_i^{\boldsymbol{\mu}} \mathbf{x}_i$$

$$\text{s.to } \underline{n} \leq \sum_{i=1}^{n} \mathbf{x}_i \leq \overline{n}, \tag{5}$$

$$\mathbf{x}_i = \mathbf{p}_i, \qquad 1 \leq i \leq n, \ \mathbf{p}_i \in \{0, 1\}, \tag{4}$$

$$\mathbf{x} \in \{0, 1\}^n. \tag{3}$$

If $LR(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu})$ has no feasible solution, then for the sake of consistency we let $\phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}) = \infty$.

Since $\phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu})$ is a lower bound on the optimal value of $SP(\mathbf{p}, \underline{n}, \overline{n})$ for all $\boldsymbol{\mu} \in \mathbb{R}^m$, we are interested in searching for

$$\bar{\boldsymbol{\mu}} \approx \arg \max_{\boldsymbol{\mu} \in \mathbb{R}^m} \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}).$$

There exist several methods for this purpose, we shall discuss this matter further in Sect. 3.4. We now show how to efficiently evaluate ϕ.

Evaluating a lagrangian dual function $\phi(\mathbf{p}, \underline{n}, \overline{n}, \cdot)$ for a particular vector of multipliers $\boldsymbol{\mu} \in \mathbb{R}^m$ consists in searching for an optimal solution $\underline{\mathbf{x}}$ of the corresponding lagrangian relaxation $LR(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu})$. We do this in three stages:

1. we compute the reduced costs $\bar{\mathbf{c}}^{\boldsymbol{\mu}}$;
2. we try to generate an optimal partition $\Pi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}) = (F^1, L^1, L^*, L^0, F^0)$ of the set of locations, i.e. $F^1 \cup L^1 \cup L^* \cup L^0 \cup F^0 = \{1, \ldots, n\}$;
3. if successful, we generate $\underline{\mathbf{x}}$ using $\Pi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu})$ and $\bar{\mathbf{c}}^{\boldsymbol{\mu}}$.

We generate the location sets F^1, L^1, L^*, L^0 and F^0 as follows. The set F^1 contains the locations fixed open, i.e. F^1 is the set of all locations $i \in \{1, \ldots, n\}$ for which $\mathbf{p}_i = 1$. Likewise, F^0 contains the locations fixed closed. Now consider constraint (5), and notice that it cannot be satisfied if $|F^1| > \overline{n}$ or if $n - |F^0| < \underline{n}$. In this case $LR(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu})$ is infeasible, we fail to generate the partition $\Pi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu})$, and $\phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}) = \infty$. Otherwise, note that if $|F^1| < \underline{n}$, then in order to satisfy constraint (5) at least $\underline{n} - |F^1|$ additional unfixed locations must be open. Let L^1 be this set, the optimal decision is for L^1 to contain the unfixed locations which are the least expensive in terms of $\bar{\mathbf{c}}^{\boldsymbol{\mu}}$. Likewise, if $n - |F^0| < \overline{n}$, then at least $\overline{n} - n + |F^0|$ unfixed locations must be closed. Let L^0 be this set, the optimal decision is for L^0 to contain the unfixed locations which are the most expensive in terms of $\bar{\mathbf{c}}^{\boldsymbol{\mu}}$. Let L^* be the set of locations in $\{1, \ldots, n\}$ not in F^1, L^1, L^0 or F^0. The optimal decision for each location $i \in L^*$ is to be open or closed if $\bar{\mathbf{c}}_i^{\boldsymbol{\mu}}$ is negative or positive, respectively.

The optimal value of a lagrangian relaxation $LR(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu})$ which is feasible, i.e. if and only if $\underline{n} \leq n - |F^0|$ and $|F^1| \leq \overline{n}$, is

$$\phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}) = \sum_{j=1}^{m} \boldsymbol{\mu}_j + \sum_{i \in F^1 \cup L^1} \bar{\mathbf{c}}_i^{\boldsymbol{\mu}} + \sum_{i \in L^*} \min\left\{0, \bar{\mathbf{c}}_i^{\boldsymbol{\mu}}\right\}.$$

We can use the reduced costs and the optimal partition to identify an optimal solution $\underline{\mathbf{x}}$:

$$\forall i \in \{1, \ldots, n\}, \quad \underline{\mathbf{x}}_i = \begin{cases} 1 & \text{if } i \in F^1 \cup L^1, \\ 0 & \text{if } i \in L^0 \cup F^0, \\ 1 & \text{if } i \in L^* \text{ and } \bar{\mathbf{c}}_i^{\boldsymbol{\mu}} < 0, \\ 0 & \text{if } i \in L^* \text{ and } \bar{\mathbf{c}}_i^{\boldsymbol{\mu}} > 0, \\ 0 \text{ or } 1 & \text{otherwise.} \end{cases}$$

Proposition 1 *The lagrangian relaxation $LR(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu})$ can be solved in $O(mn)$ time.*

Proof To begin with, the computation of the reduced costs $\bar{\mathbf{c}}^{\boldsymbol{\mu}}$ requires $O(mn)$ time, and the generation of the sets F^0 and F^1 requires $O(n)$ time. Next, the generation of L^1 requires us to select a certain number of the cheapest locations in $\{1, \ldots, n\} \backslash (F^0 \cup F^1)$. This can be done using an appropriate selection algorithm such as `quickselect` which requires $O(n)$ time. We generate L^0 in a similar fashion, and generate L^* in $O(n)$ time also. Finally, we generate $\underline{\mathbf{x}}$ simply by enumerating the elements in these sets and hence the whole resolution is done in $O(mn)$ time. □

We now use the concepts introduced in this subsection to present our dual process.

3.2 Branch-and-bound procedure and dual process

Throughout the search, the dual process maintains a incumbent solution $\bar{\mathbf{x}}^{\texttt{dual}}$ as well as a global upper bound $\bar{z}^{\texttt{dual}} = f(\bar{\mathbf{x}}^{\texttt{dual}})$. Initially, $\bar{\mathbf{x}}^{\texttt{dual}}$ is undefined and $\bar{z}^{\texttt{dual}}$

is set to ∞. The dual process then performs a preprocessing phase before solving the problem (P) by branch-and-bound. The preprocessing phase consists in computing, for all ranks $k \in \{1, \ldots, n\}$, a lower bound $\boldsymbol{\ell}_k$ on the cost of any solution of (P) with k open locations:

$$\forall \mathbf{x} \in \{0,1\}^n,\ \sum_{i=1}^{n} \mathbf{x}_i = k \implies \boldsymbol{\ell}_k \leq f(\mathbf{x}).$$

These lower bounds are subsequently used in the following manner during the resolution of (P). Note that

$$\forall \mathbf{x} \in \{0,1\}^n,\ f(\mathbf{x}) \leq \bar{z}^{\texttt{dual}} \implies \left(\sum_{i=1}^{n} \mathbf{x}_i\right) \in \left\{k \in \{1, \ldots, n\} : \boldsymbol{\ell}_k \leq \bar{z}^{\texttt{dual}}\right\}.$$

During the resolution of (P) we therefore discard all solutions whose number of open locations is not in the interval

$$\left[\min\left\{k \in \{1, \ldots, n\} : \boldsymbol{\ell}_k \leq \bar{z}^{\texttt{dual}}\right\}, \max\left\{k \in \{1, \ldots, n\} : \boldsymbol{\ell}_k \leq \bar{z}^{\texttt{dual}}\right\}\right].$$

In practice, given good enough bounds $\boldsymbol{\ell}$ and $\bar{z}^{\texttt{dual}}$, this interval is small enough to justify the extra effort required for computing $\boldsymbol{\ell}$.

Before giving an overview of the dual process, we begin this subsection by specifying our `Branch-and-bound` procedure. We shall see that it is applied both to compute $\boldsymbol{\ell}$ during the preprocessing phase as well as to solve (P) afterwards.

3.2.1 Branch-and-bound

The `Branch-and-bound` procedure described here performs a lagrangian branch-and-bound search on (P), updating $\bar{\mathbf{x}}^{\texttt{dual}}$ and $\bar{z}^{\texttt{dual}}$ whenever it finds a cheaper solution. At all times, our `Branch-and-bound` procedure maintains an open node queue $\mathcal{Q}$ storing all the open leaf nodes of the search tree. Any node of the search tree is represented by a 4-tuple $(\mathbf{p}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}})$: the subproblem corresponding to this node is $SP(\mathbf{p}, \underline{n}, \overline{n})$ and a lower bound for this node is $\phi(\mathbf{p}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}})$. Let $*$ be the n-dimensional vector of $*$-components. We compute

$$\bar{\boldsymbol{\mu}}^{root} \approx \arg\max_{\boldsymbol{\mu} \in \mathbb{R}^m} \phi(*, 1, n, \boldsymbol{\mu})$$

using a bundle method as explained in Sect. 3.4, and initialize the open node queue $\mathcal{Q}$ with $(*, 1, n, \bar{\boldsymbol{\mu}}^{root})$. This 4-tuple corresponds to the root node of the search tree. The `Branch-and-bound` procedure then iteratively expands the search tree until all leaf nodes have been closed, i.e. until $\mathcal{Q} = \emptyset$.

At the beginning of each iteration of `Branch-and-bound`, we select a node $(\mathbf{p}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}}) \in \mathcal{Q}$ according to a predefined strategy `NodeSelect`. For example, our implementation applies one of the following two node selection strategies:

- `LFS`, which selects the node with the lowest lower bound in $\mathcal{Q}$ (lowest-first search, also known as best-first search),
- `DFS`, which selects the node most recently added into $\mathcal{Q}$ (depth-first search).

Note that since `LFS` always selects the node with the smallest lower bound, i.e. $(\mathbf{p}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}}) \in \mathcal{Q}$ which minimizes ϕ, this lower bound is then also a global lower bound for (P).

We then improve this lower bound by updating $\bar{\boldsymbol{\mu}}$ with a bundle method such that

$$\bar{\boldsymbol{\mu}} \approx \arg\max_{\boldsymbol{\mu} \in \mathbb{R}^m} \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}).$$

In the case where the lower bound does not exceed the upper bound $\bar{z}^{\texttt{dual}}$, we try to separate this node into subnodes by applying a given branching procedure `Branch`. We define a branching procedure `Branch` as a procedure which, when applied to a 4-tuple $(\mathbf{p}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}})$ returns either nothing or two 3-tuples $(\mathbf{p}', \underline{n}', \overline{n}')$ and $(\mathbf{p}'', \underline{n}'', \overline{n}'')$ which satisfy the following property: for all solutions $\mathbf{x}$ feasible for $SP(\mathbf{p}, \underline{n}, \overline{n})$ and cheaper than $\bar{z}^{\texttt{dual}}$, $\mathbf{x}$ must be feasible either for $SP(\mathbf{p}', \underline{n}', \overline{n}')$ or for $SP(\mathbf{p}'', \underline{n}'', \overline{n}'')$. If $\phi(\mathbf{p}', \underline{n}', \overline{n}', \bar{\boldsymbol{\mu}}) < \bar{z}^{\texttt{dual}}$, then we insert the subnode $(\mathbf{p}', \underline{n}', \overline{n}', \bar{\boldsymbol{\mu}})$ into the open node queue $\mathcal{Q}$, and likewise for $(\mathbf{p}'', \underline{n}'', \overline{n}'', \bar{\boldsymbol{\mu}})$.

The final step of our `Branch-and-bound` iteration consists in communicating with the primal process and possibly replacing $\bar{\mathbf{x}}^{\texttt{dual}}$ to decrease $\bar{z}^{\texttt{dual}} = f(\bar{\mathbf{x}}^{\texttt{dual}})$. Let us list all types of messages which our dual process may send and receive (mirroring the list in Sect. 2) before discussing how to handle them:

- incoming:
  - receive a solution $\bar{\mathbf{x}}^{\texttt{primal}}$,
  - receive a request for a guiding solution.
- outgoing:
  - send the solution $\bar{\mathbf{x}}^{\texttt{dual}}$,
  - send the improving partial solution $\bar{\mathbf{p}}$,
  - send the guiding solution $\underline{\mathbf{x}}$.

The first step is to try replacing $\bar{\mathbf{x}}^{\texttt{dual}}$ by a cheaper solution. Consider $\underline{\mathbf{x}}$, the optimal solution of $LR(\mathbf{p}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}})$: if $f(\underline{\mathbf{x}}) < \bar{z}^{\texttt{dual}}$, then we replace $\bar{\mathbf{x}}^{\texttt{dual}}$ with $\underline{\mathbf{x}}$ and $\bar{z}^{\texttt{dual}}$ with $f(\underline{\mathbf{x}})$. Likewise, if we have received a solution $\bar{\mathbf{x}}^{\texttt{primal}}$ from the primal process and if $f(\bar{\mathbf{x}}^{\texttt{primal}}) < \bar{z}^{\texttt{dual}}$, then we replace $\bar{\mathbf{x}}^{\texttt{dual}}$ with $\bar{\mathbf{x}}^{\texttt{primal}}$ and $\bar{z}^{\texttt{dual}}$ with $f(\bar{\mathbf{x}}^{\texttt{primal}})$. Note that an improved upper bound $\bar{z}^{\texttt{dual}}$ may allow us to fathom some open nodes in $\mathcal{Q}$.

Next, we occasionally update the improving partial solution $\bar{\mathbf{p}}$ and send it to the primal process if it has changed. The improving partial solution (a notion already introduced in Sect. 2) is a partial solution $\bar{\mathbf{p}} \in \{0, 1, *\}^n$ which satisfies the following for all $i \in \{1, \ldots, n\}$:

$$\bar{\mathbf{p}}_i = 1 \implies \forall (\check{\mathbf{p}}, \check{\underline{n}}, \check{\overline{n}}, \check{\boldsymbol{\mu}}) \in \mathcal{Q},\ \check{\mathbf{p}}_i = 1,$$
$$\bar{\mathbf{p}}_i = 0 \implies \forall (\check{\mathbf{p}}, \check{\underline{n}}, \check{\overline{n}}, \check{\boldsymbol{\mu}}) \in \mathcal{Q},\ \check{\mathbf{p}}_i = 0.$$

Since the effort required to compute an improving partial solution $\bar{\mathbf{p}}$ with a minimum number of $*$-components scales with the size of $\mathcal{Q}$, the frequency of the updates of $\bar{\mathbf{p}}$ must not be too high (this is set by a predefined parameter).

Finally, if a request for a guiding solution was received from the primal process, then we send $\underline{\mathbf{x}}$ as a guiding solution. If the open node queue $\mathcal{Q}$ is now empty, then we end the search, otherwise we perform the next iteration of the `Branch-and-bound` procedure.

We specify the whole `Branch-and-bound` procedure in Algorithm 2. The procedure takes two callback procedures as arguments: `NodeSelect` which selects a node in the open node queue, and a branching procedure `Branch`.

Algorithm 2 - `Branch-and-bound(NodeSelect, Branch)`:

0. **Initialization -** Search for

$$\bar{\boldsymbol{\mu}}^{root} \approx \arg \max_{\boldsymbol{\mu} \in \mathbb{R}^m} \phi(*, 1, n, \boldsymbol{\mu}),$$

then initialize the open node queue $\mathcal{Q}$ with the 4-tuple $(*, 1, n, \bar{\boldsymbol{\mu}}^{root})$.

1. **Selection -** Apply `NodeSelect`$(\mathcal{Q})$ to select a 4-tuple $(\mathbf{p}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}})$ in $\mathcal{Q}$. Remove it from $\mathcal{Q}$.
2. **Evaluation -** Update $\bar{\boldsymbol{\mu}}$ such that:

$$\bar{\boldsymbol{\mu}} \approx \arg \max_{\boldsymbol{\mu} \in \mathbb{R}^m} \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}).$$

3. **Separation -**
   (a) If $\phi(\mathbf{p}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}}) \geq \bar{z}^{\texttt{dual}}$, then go straight to step 4.
   (b) Apply `Branch`$(\mathbf{p}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}})$ to try to generate two 3-tuples $(\mathbf{p}', \underline{n}', \overline{n}')$ and $(\mathbf{p}'', \underline{n}'', \overline{n}'')$.
   (c) If successful and if $\phi(\mathbf{p}', \underline{n}', \overline{n}', \bar{\boldsymbol{\mu}}) < \bar{z}^{\texttt{dual}}$, then insert $(\mathbf{p}', \underline{n}', \overline{n}')$ into $\mathcal{Q}$. Do the same for $(\mathbf{p}'', \underline{n}'', \overline{n}'')$.
4. **Upper bound update and communication -**
   (a) Let $\underline{\mathbf{x}}$ be the optimal solution of $LR(\mathbf{p}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}})$. If we have received a new solution $\bar{\mathbf{x}}^{\texttt{primal}}$ from the primal process, then let $\mathbf{x} = \arg\min\{f(\underline{\mathbf{x}}), f(\bar{\mathbf{x}}^{\texttt{primal}})\}$, else let $\mathbf{x} = \underline{\mathbf{x}}$. If $f(\mathbf{x}) < \bar{z}^{\texttt{dual}}$, then:
      i. Set $\bar{\mathbf{x}}^{\texttt{dual}} \leftarrow \mathbf{x}$ and $\bar{z}^{\texttt{dual}} \leftarrow f(\mathbf{x})$.
      ii. If $\bar{\mathbf{x}}^{\texttt{dual}} = \underline{\mathbf{x}}$ then send the new best solution $\bar{\mathbf{x}}^{\texttt{dual}}$ to the primal process.
      iii. For all $(\check{\mathbf{p}}, \underline{\check{n}}, \overline{\check{n}}, \check{\boldsymbol{\mu}}) \in \mathcal{Q}$: if $\phi(\check{\mathbf{p}}, \underline{\check{n}}, \overline{\check{n}}, \check{\boldsymbol{\mu}}) \geq \bar{z}^{\texttt{dual}}$ then remove $(\check{\mathbf{p}}, \underline{\check{n}}, \overline{\check{n}}, \check{\boldsymbol{\mu}})$ from $\mathcal{Q}$.
   (b) Occasionally, do the following:
      i. Initialize $\bar{\mathbf{p}} \leftarrow *$.
      ii. For all $i \in \{1, \ldots, n\}$: if for all $(\check{\mathbf{p}}, \underline{\check{n}}, \overline{\check{n}}, \check{\boldsymbol{\mu}}) \in \mathcal{Q}$ we have $\check{\mathbf{p}}_i = 0$, then set $\bar{\mathbf{p}}_i \leftarrow 0$,
      iii. For all $i \in \{1, \ldots, n\}$: if for all $(\check{\mathbf{p}}, \underline{\check{n}}, \overline{\check{n}}, \check{\boldsymbol{\mu}}) \in \mathcal{Q}$ we have $\check{\mathbf{p}}_i = 1$, then set $\bar{\mathbf{p}}_i \leftarrow 1$.
      iv. Send the improving partial solution $\bar{\mathbf{p}}$ to the primal process.
   (c) Upon reception of a request for a guiding solution, send $\underline{\mathbf{x}}$ to the primal process as a guiding solution.
   (d) If $\mathcal{Q}$ is nonempty, then go back to step 1.

3.2.2 Dual process overview

At the beginning of this subsection we mentioned that the dual process begins by performing a preprocessing phase to generate a vector $\boldsymbol{\ell} \in \mathbb{R}^n$ which satisfies

$$\forall \mathbf{x} \in \{0,1\}^n,\ \sum_{i=1}^{n} \mathbf{x}_i = k \implies \boldsymbol{\ell}_k \leq f(\mathbf{x}).$$

For this purpose, we initialize $\boldsymbol{\ell}_k \leftarrow \infty$ for all $k \in \{1,\ldots,n\}$, and apply `Branch-and-bound` using an appropriate branching procedure which will dictate how the nodes are subdivided in step 3b of Algorithm 2. This particular branching procedure is denoted by `BrP` and is specified recursively in Algorithm 3. Its input is a 4-tuple $(\mathbf{p}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}})$ specifying a node to be split, and its final output is either nothing or two 3-tuples specifying the subnodes. By defining this branching procedure recursively, we may effectively tighten the constraint (5) in each of the two resulting subproblems, which we hope improves the search overall. Since this procedure branches on the number of open locations $\sum_{i=1}^{n} \mathbf{x}_i$, by applying it we separate (P) into subproblems $SP(*, \underline{n}, \overline{n})$ with $1 \leq \underline{n} \leq \overline{n} \leq n$.

Algorithm 3 - `BrP`$(\mathbf{p}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}})$:

case $\underline{n} = \overline{n}$
  set $\boldsymbol{\ell}_{\underline{n}} \leftarrow \phi(\mathbf{p}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}})$ and return $\emptyset$;
case $\phi(\mathbf{p}, \underline{n}, \underline{n}, \bar{\boldsymbol{\mu}}) \geq \bar{z}^{\mathtt{dual}}$
  return `BrP`$(\mathbf{p}, \underline{n}+1, \overline{n}, \bar{\boldsymbol{\mu}})$;
case $\phi(\mathbf{p}, \overline{n}, \overline{n}, \bar{\boldsymbol{\mu}}) \geq \bar{z}^{\mathtt{dual}}$
  return `BrP`$(\mathbf{p}, \underline{n}, \overline{n}-1, \bar{\boldsymbol{\mu}})$;
case otherwise
  return $(\mathbf{p}, \underline{n}, \lfloor \frac{\underline{n}+\overline{n}}{2} \rfloor)$ and $(\mathbf{p}, \lfloor \frac{\underline{n}+\overline{n}}{2} \rfloor + 1, \overline{n})$.

Note that this branching procedure requires several evaluations of ϕ. In general this is done in $O(mn)$ time, however we shall see in Sect. 3.3 that it is possible to compute $\Pi(\mathbf{p}, \underline{n}+1, \overline{n}, \bar{\boldsymbol{\mu}})$ and $\phi(\mathbf{p}, \underline{n}+1, \overline{n}, \bar{\boldsymbol{\mu}})$ in $O(n)$ time by reusing $\bar{\mathbf{c}}^{\bar{\boldsymbol{\mu}}}$, $\Pi(\mathbf{p}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}})$ and $\phi(\mathbf{p}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}})$. The same can also be done for $(\mathbf{p}, \underline{n}, \overline{n}-1, \bar{\boldsymbol{\mu}})$. When `Branch-and-bound` terminates, we set $\boldsymbol{\ell}_k \leftarrow \bar{z}^{\mathtt{dual}}$ for all $k \in \{1,\ldots,n\}$ for which $\boldsymbol{\ell}_k = \infty$. In this manner, for all $k \in \{1,\ldots,n\}$, $\boldsymbol{\ell}_k$ is a lower bound on the solutions of (P) with k open locations, i.e. on the solutions of $SP(*, k, k)$.

Having completed preprocessing, we solve (P) by applying `Branch-and-bound` using another recursive branching procedure `BrS`, which we specify in Algorithm 4. In this case, recursion allows us to tighten constraint (4) as well as (5) in the resulting subproblems. For all $\mathbf{p} \in \{0,1,*\}^n$ and for all $k \in \{1,\ldots,n\}$, we define $\mathbf{p}^{+k}$ and $\mathbf{p}^{-k}$ as follows. For all $i \in \{1,\ldots,n\}$,

$$\mathbf{p}_i^{+k} = \begin{cases} 1 & \text{if } k = i, \\ \mathbf{p}_i & \text{otherwise.} \end{cases} \qquad \mathbf{p}_i^{-k} = \begin{cases} 0 & \text{if } k = i, \\ \mathbf{p}_i & \text{otherwise.} \end{cases}$$

In a sense, for all $i \in \{1,\ldots,n\}$ such that $p_i = *$, the value $|\bar{\mathbf{c}}_i^{\bar{\boldsymbol{\mu}}}|$ is a measure of the impact of the decision to branch on the unfixed location i, and `BrS` chooses to branch on the unfixed location with the smallest impact. Like `BrS`, this procedure is recursive and seemingly requires many $O(mn)$ evaluations of ϕ which in fact can be performed in $O(n)$. We explain how in the following subsection.

Algorithm 4 - $\texttt{BrS}(\mathbf{p}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}})$:

case $\mathbf{p} \in \{0,1\}^n$
  return $\emptyset$,
case $\boldsymbol{\ell}_{\underline{n}} \geq \bar{z}^{\texttt{dual}}$ or $\phi(\mathbf{p}, \underline{n}, \underline{n}, \bar{\boldsymbol{\mu}}) \geq \bar{z}^{\texttt{dual}}$
  return $\texttt{BrS}(\mathbf{p}, \underline{n}+1, \overline{n}, \bar{\boldsymbol{\mu}})$;
case $\boldsymbol{\ell}_{\overline{n}} \geq \bar{z}^{\texttt{dual}}$ or $\phi(\mathbf{p}, \overline{n}, \overline{n}, \bar{\boldsymbol{\mu}}) \geq \bar{z}^{\texttt{dual}}$
  return $\texttt{BrS}(\mathbf{p}, \underline{n}, \overline{n}-1, \bar{\boldsymbol{\mu}})$;
case $\exists k \in \{1,\ldots,n\},\ p_k = *,\ \phi(\mathbf{p}^{+k}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}}) \geq \bar{z}^{\texttt{dual}}$
  return $\texttt{BrS}(\mathbf{p}^{-k}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}})$;
case $\exists k \in \{1,\ldots,n\},\ p_k = *,\ \phi(\mathbf{p}^{-k}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}}) \geq \bar{z}^{\texttt{dual}}$
  return $\texttt{BrS}(\mathbf{p}^{+k}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}})$;
case otherwise
  select $k = \arg\min\left\{|\bar{\mathbf{c}}_i^{\bar{\boldsymbol{\mu}}}| : p_i = *, i \in \{1,\ldots,n\}\right\}$,
  return $(\mathbf{p}^{+k}, \underline{n}, \overline{n})$ and $(\mathbf{p}^{-k}, \underline{n}, \overline{n})$.

3.3 Reoptimization

Given any vector $\boldsymbol{\mu} \in \mathbb{R}^m$, suppose that we have evaluated $\phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu})$ and that this value is not ∞. We thus also have the reduced costs $\bar{\mathbf{c}}^{\boldsymbol{\mu}}$ as well as an optimal partition $\Pi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}) = (F^1, L^1, L^*, L^0, F^0)$. The following proposition and its corollary specify how to reuse these to compute $\phi(\mathbf{p}, \underline{n}+k, \overline{n}, \boldsymbol{\mu})$ and $\phi(\mathbf{p}, \underline{n}, \overline{n}-k, \boldsymbol{\mu})$ in $O(n)$ time, for all $k > 0$. Without loss of generality, suppose that the values in $\bar{\mathbf{c}}^{\boldsymbol{\mu}}$ are all different.

Proposition 2 *For all $k > 0$, if $\underline{n} + k > \min\{\overline{n}, n - |F^0|\}$ then*

$$\phi(\mathbf{p}, \underline{n}+k, \overline{n}, \boldsymbol{\mu}) = \infty,$$

else if $\underline{n} + k \leq |F^1|$ then

$$\Pi(\mathbf{p}, \underline{n}+k, \overline{n}, \boldsymbol{\mu}) = (F^1, \emptyset, L^*, L^0, F^0),$$
$$\phi(\mathbf{p}, \underline{n}+k, \overline{n}, \boldsymbol{\mu}) = \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}),$$

else, let $\tilde{L}$ be the set of the k least expensive locations in L^* in terms of $\bar{\mathbf{c}}^{\boldsymbol{\mu}}$, we have

$$\Pi(\mathbf{p}, \underline{n}+k, \overline{n}, \boldsymbol{\mu}) = (F^1, L^1 \cup \tilde{L}, L^* \backslash \tilde{L}, L^0, F^0),$$
$$\phi(\mathbf{p}, \underline{n}+k, \overline{n}, \boldsymbol{\mu}) = \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}) + \sum_{i \in \tilde{L}} \max\left\{0, \bar{\mathbf{c}}_i^{\boldsymbol{\mu}}\right\}.$$

Proof If $\underline{n}+k > \overline{n}$, then by definition of $SP(\mathbf{p}, \underline{n}+k, \overline{n})$, this subproblem is infeasible, hence $\phi(\mathbf{p}, \underline{n}+k, \overline{n}, \boldsymbol{\mu}) = \infty$. Likewise, if $\underline{n} + k > n - |F^0|$, then there exists no $\mathbf{x} \in \{0,1\}^n$ which can satisfy both $\underline{n} + k \leq \sum_{i=1}^n \mathbf{x}_i$ and $\mathbf{x}_i = 0$ for all $i \in \{1,\ldots,n\}$ for which $\mathbf{p}_i = 0$, i.e. for all $i \in F^0$.

Suppose henceforth that $\underline{n} + k \leq \min\{\overline{n}, n - |F^0|\}$, and let $\Pi(\mathbf{p}, \underline{n}+k, \overline{n}, \boldsymbol{\mu}) = (F'^1, L'^1, L'^*, L'^0, F'^0)$. We know that $F'^1 = F^1$ and $F'^0 = F^0$ because both these sets are induced solely by $\mathbf{p}$.

If $\underline{n} + k \leq |F^1|$, then all $\mathbf{x} \in \{0,1\}^n$ which satisfy $\mathbf{x}_i = 1$ for all $i \in F^1$ also satisfy $\underline{n} + k \leq \sum_{i=1}^{n} \mathbf{x}_i$, consequently L^1 and L'^1 are empty by definition of these sets. Likewise, $L'^0 = L^0$ and $L'^* = L^*$, hence $\Pi(\mathbf{p}, \underline{n} + k, \overline{n}, \boldsymbol{\mu}) = \Pi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu})$ and $\phi(\mathbf{p}, \underline{n} + k, \overline{n}, \boldsymbol{\mu}) = \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu})$.

Suppose now that $\underline{n} + k > |F^1|$. The set L'^1 consists of the $\underline{n} + k - |F^1|$ unfixed locations which are least expensive in terms of $\bar{\mathbf{c}}^{\boldsymbol{\mu}}$, and we know that this is a subset of $L^1 \cup L^*$ because

$$\min\{\overline{n}, n - |F^0|\} = n - |L^0 \cup F^0| = |F^1| + |L^1| + |L^*|,$$

hence $L'^1 = L^1 \cup \tilde{L}$ and $L'^* = L^* \backslash \tilde{L}$. Consequently,

$$\begin{aligned}
\phi(\mathbf{p}, \underline{n} + k, \overline{n}, \boldsymbol{\mu}) &= \sum_{i \in F'^1 \cup L'^1} \bar{\mathbf{c}}_i^{\boldsymbol{\mu}} + \sum_{i \in L'^*} \min\left\{0, \bar{\mathbf{c}}_i^{\boldsymbol{\mu}}\right\}, \\
&= \sum_{i \in F^1 \cup L^1} \bar{\mathbf{c}}_i^{\boldsymbol{\mu}} + \sum_{i \in \tilde{L}} \bar{\mathbf{c}}_i^{\boldsymbol{\mu}} + \sum_{i \in L^*} \min\left\{0, \bar{\mathbf{c}}_i^{\boldsymbol{\mu}}\right\} - \sum_{i \in \tilde{L}} \min\left\{0, \bar{\mathbf{c}}_i^{\boldsymbol{\mu}}\right\}, \\
&= \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}) + \sum_{i \in \tilde{L}} \max\left\{0, \bar{\mathbf{c}}_i^{\boldsymbol{\mu}}\right\}.
\end{aligned}$$

□

Corollary 1 *For all $k > 0$, if $\overline{n} - k < \max\{\underline{n}, |F^1|\}$ then*

$$\phi(\mathbf{p}, \underline{n}, \overline{n} - k, \boldsymbol{\mu}) = \infty,$$

else if $n - (\overline{n} - k) \leq |F^0|$ then

$$\begin{aligned}
\Pi(\mathbf{p}, \underline{n}, \overline{n} - k, \boldsymbol{\mu}) &= (F^1, L^1, L^*, \emptyset, F^0), \\
\phi(\mathbf{p}, \underline{n}, \overline{n} - k, \boldsymbol{\mu}) &= \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}),
\end{aligned}$$

else, let $\tilde{L}$ be the set of the k most expensive locations in L^* in terms of $\bar{\mathbf{c}}^{\boldsymbol{\mu}}$, we have

$$\begin{aligned}
\Pi(\mathbf{p}, \underline{n}, \overline{n} - k, \boldsymbol{\mu}) &= (F^1, L^1, L^* \setminus \tilde{L}, L^0 \cup \tilde{L}, F^0), \\
\phi(\mathbf{p}, \underline{n}, \overline{n} - k, \boldsymbol{\mu}) &= \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}) - \sum_{i \in \tilde{L}} \min\left\{0, \bar{\mathbf{c}}_i^{\boldsymbol{\mu}}\right\}.
\end{aligned}$$

The next propositions show how to reoptimize $LR(\mathbf{p}^{+k}, \underline{n}, \overline{n}, \boldsymbol{\mu})$ for all $k \in \{1, \ldots, n\}$ for which $\mathbf{p}_k = *$, i.e. for all $k \in (L^1 \cup L^* \cup L^0)$. For this purpose we need to identify the following locations:

$$\begin{aligned}
i_>^1 &= \arg\max\left\{\bar{\mathbf{c}}_i^{\boldsymbol{\mu}} : i \in L^1\right\}, & i_>^* &= \arg\max\left\{\bar{\mathbf{c}}_i^{\boldsymbol{\mu}} : i \in L^*\right\}, \\
i_<^* &= \arg\min\left\{\bar{\mathbf{c}}_i^{\boldsymbol{\mu}} : i \in L^*\right\}, & i_<^0 &= \arg\min\left\{\bar{\mathbf{c}}_i^{\boldsymbol{\mu}} : i \in L^0\right\}.
\end{aligned}$$

Each proposition also has a corollary stating a symmetric result for $\mathbf{p}^{-k}$.

Proposition 3 *For all $k \in L^1$,*

$$\begin{aligned}\Pi(\mathbf{p}^{+k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) &= (F^1 \cup \{k\}, L^1 \backslash \{k\}, L^*, L^0, F^0),\\ \phi(\mathbf{p}^{+k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) &= \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}).\end{aligned}$$

Proof Let $\Pi(\mathbf{p}^{+k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) = (F'^1, L'^1, L'^*, L'^0, F'^0)$. Since $\mathbf{p}^{+k}$ differs from $\mathbf{p}$ only in its k-th component, for which we have $p_k = *$ and $\mathbf{p}^{+k} = 1$, it follows that $F'^1 = F^1 \cup \{k\}$ and that $F'^0 = F^0$. Recall that by definition, if $|F'^1| \geq \underline{n}$ then $L'^1 = \emptyset$ else L'^1 contains the $\underline{n} - |F'^1|$ cheapest unfixed locations in terms of $\bar{\mathbf{c}}^{\boldsymbol{\mu}}$. Consequently, we have $L'^1 = L^1 \backslash \{k\}$, and by using a similar reasoning we also have $L'^0 = L^0$, hence $L'^* = L^*$. Note that $F'^1 \cup L'^1 = F^1 \cup L^1$, hence $\phi(\mathbf{p}^{+k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) = \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu})$. □

Corollary 2 *For all $k \in L^0$,*

$$\begin{aligned}\Pi(\mathbf{p}^{-k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) &= (F^1, L^1, L^*, L^0 \backslash \{k\}, F^0 \cup \{k\}),\\ \phi(\mathbf{p}^{-k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) &= \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}).\end{aligned}$$

Proposition 4 *For all $k \in L^*$, if L^1 is empty then*

$$\begin{aligned}\Pi(\mathbf{p}^{+k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) &= (F^1 \cup \{k\}, \emptyset, L^* \backslash \{k\}, L^0, F^0),\\ \phi(\mathbf{p}^{+k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) &= \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}) + \max\left\{0, \bar{\mathbf{c}}^{\boldsymbol{\mu}}_k\right\},\end{aligned}$$

else

$$\begin{aligned}\Pi(\mathbf{p}^{+k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) &= (F^1 \cup \{k\}, L^1 \backslash \{i^1_>\}, (L^* \backslash \{k\}) \cup \{i^1_>\}, L^0, F^0),\\ \phi(\mathbf{p}^{+k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) &= \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}) + \max\left\{0, \bar{\mathbf{c}}^{\boldsymbol{\mu}}_k\right\} - \max\left\{0, \bar{\mathbf{c}}^{\boldsymbol{\mu}}_{i^1_>}\right\}.\end{aligned}$$

Proof Let $\Pi(\mathbf{p}^{+k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) = (F'^1, L'^1, L'^*, L'^0, F'^0)$. As in the proof of Proposition 3, we have $F'^1 = F^1 \cup \{k\}$, $L'^0 = L^0$, and $F'^0 = F^0$. If L^1 is empty, then $|F^1| \geq \underline{n}$ and therefore L'^1 is empty also, hence:

$$\begin{aligned}\phi(\mathbf{p}^{+k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) &= \sum_{i \in F'^1 \cup L'^1} \bar{\mathbf{c}}^{\boldsymbol{\mu}}_i + \sum_{i \in L'^*} \min\left\{0, \bar{\mathbf{c}}^{\boldsymbol{\mu}}_i\right\},\\ &= \sum_{i \in F^1 \cup L^1} \bar{\mathbf{c}}^{\boldsymbol{\mu}}_i + \sum_{i \in \tilde{L}} \bar{\mathbf{c}}^{\boldsymbol{\mu}}_i + \sum_{i \in L^*} \min\left\{0, \bar{\mathbf{c}}^{\boldsymbol{\mu}}_i\right\} - \sum_{i \in \tilde{L}} \min\left\{0, \bar{\mathbf{c}}^{\boldsymbol{\mu}}_i\right\},\\ &= \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}) + \sum_{i \in \tilde{L}} \max\left\{0, \bar{\mathbf{c}}^{\boldsymbol{\mu}}_i\right\}.\end{aligned}$$

If L^1 is not empty, then L'^1 must contains one location less than L^1, which therefore must be $i^1_>$. Consequently, $i^1_> \in L'^*$ and

$$\begin{aligned}\phi(\mathbf{p}^{+k},\underline{n},\overline{n},\boldsymbol{\mu}) &= \sum_{i\in F^1\cup\{k\}\cup L^1\setminus\{i^1_>\}}\bar{\mathbf{c}}^{\boldsymbol{\mu}}_i + \sum_{i\in(L^*\setminus\{k\})\cup\{i^1_>\}}\min\left\{0,\bar{\mathbf{c}}^{\boldsymbol{\mu}}_i\right\},\\ &= \sum_{i\in F^1\cup L^1}\bar{\mathbf{c}}^{\boldsymbol{\mu}}_i + \bar{\mathbf{c}}^{\boldsymbol{\mu}}_k - \bar{\mathbf{c}}^{\boldsymbol{\mu}}_{i^1_>}\\ &\qquad + \sum_{i\in L^*}\min\left\{0,\bar{\mathbf{c}}^{\boldsymbol{\mu}}_i\right\} - \min\left\{0,\bar{\mathbf{c}}^{\boldsymbol{\mu}}_k\right\} + \min\left\{0,\bar{\mathbf{c}}^{\boldsymbol{\mu}}_{i^1_>}\right\},\\ &= \phi(\mathbf{p},\underline{n},\overline{n},\boldsymbol{\mu}) + \max\left\{0,\bar{\mathbf{c}}^{\boldsymbol{\mu}}_k\right\} - \max\left\{0,\bar{\mathbf{c}}^{\boldsymbol{\mu}}_{i^1_>}\right\}.\end{aligned}$$

□

Corollary 3 *For all $k\in L^*$, if L^0 is empty then*

$$\begin{aligned}\Pi(\mathbf{p}^{-k},\underline{n},\overline{n},\boldsymbol{\mu}) &= (F^1, L^1, L^*\setminus\{k\},\emptyset, F^0\cup\{k\}),\\ \phi(\mathbf{p}^{-k},\underline{n},\overline{n},\boldsymbol{\mu}) &= \phi(\mathbf{p},\underline{n},\overline{n},\boldsymbol{\mu}) - \min\left\{0,\bar{\mathbf{c}}^{\boldsymbol{\mu}}_k\right\},\end{aligned}$$

else

$$\begin{aligned}\Pi(\mathbf{p}^{-k},\underline{n},\overline{n},\boldsymbol{\mu}) &= (F^1, L^1, (L^*\setminus\{k\})\cup\{i^0_<\}, L^0\setminus\{i^0_<\}, F^0\cup\{k\}),\\ \phi(\mathbf{p}^{-k},\underline{n},\overline{n},\boldsymbol{\mu}) &= \phi(\mathbf{p},\underline{n},\overline{n},\boldsymbol{\mu}) - \min\left\{0,\bar{\mathbf{c}}^{\boldsymbol{\mu}}_k\right\} + \min\left\{0,\bar{\mathbf{c}}^{\boldsymbol{\mu}}_{i^0_<}\right\}.\end{aligned}$$

Proposition 5 *For all $k\in L^0$, if L^1 and L^* are both empty then*

$$\phi(\mathbf{p}^{+k},\underline{n},\overline{n},\boldsymbol{\mu}) = \infty,$$

else if L^1 is nonempty and L^* is empty then

$$\begin{aligned}\Pi(\mathbf{p}^{+k},\underline{n},\overline{n},\boldsymbol{\mu}) &= (F^1\cup\{k\}, L^1\setminus\{i^1_>\},\emptyset,(L^0\setminus\{k\})\cup\{i^1_>\}, F^0),\\ \phi(\mathbf{p}^{+k},\underline{n},\overline{n},\boldsymbol{\mu}) &= \phi(\mathbf{p},\underline{n},\overline{n},\boldsymbol{\mu}) + \bar{\mathbf{c}}^{\boldsymbol{\mu}}_k - \bar{\mathbf{c}}^{\boldsymbol{\mu}}_{i^1_>},\end{aligned}$$

else if L^1 is empty and L^* is nonempty then

$$\begin{aligned}\Pi(\mathbf{p}^{+k},\underline{n},\overline{n},\boldsymbol{\mu}) &= (F^1\cup\{k\},\emptyset, L^*\setminus\{i^*_>\},(L^0\setminus\{k\})\cup\{i^*_>\}, F^0),\\ \phi(\mathbf{p}^{+k},\underline{n},\overline{n},\boldsymbol{\mu}) &= \phi(\mathbf{p},\underline{n},\overline{n},\boldsymbol{\mu}) + \bar{\mathbf{c}}^{\boldsymbol{\mu}}_k - \min\left\{0,\bar{\mathbf{c}}^{\boldsymbol{\mu}}_{i^*_>}\right\},\end{aligned}$$

else

$$\begin{aligned}\Pi(\mathbf{p}^{+k},\underline{n},\overline{n},\boldsymbol{\mu}) &= (F^1\cup\{k\}, L^1\setminus\{i^1_>\},(L^*\setminus\{i^*_>\})\cup\{i^1_>\},(L^0\setminus\{k\})\cup\{i^*_>\}, F^0),\\ \phi(\mathbf{p}^{+k},\underline{n},\overline{n},\boldsymbol{\mu}) &= \phi(\mathbf{p},\underline{n},\overline{n},\boldsymbol{\mu}) + \bar{\mathbf{c}}^{\boldsymbol{\mu}}_k - \max\left\{0,\bar{\mathbf{c}}^{\boldsymbol{\mu}}_{i^1_>}\right\} - \min\left\{0,\bar{\mathbf{c}}^{\boldsymbol{\mu}}_{i^*_>}\right\}.\end{aligned}$$

Proof If $L^1 = L^* = \emptyset$, then $|\{i \in \{1, \ldots, n\} : \mathbf{p}_i = 1\}| = |F^1| = \overline{n}$ and thus $SP(\mathbf{p}^{+k}, \underline{n}, \overline{n})$ is infeasible and $\phi(\mathbf{p}^{+k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) = \infty$. Suppose henceforth that $L^1 \cup L^* \neq \emptyset$, and let $\Pi(\mathbf{p}^{+k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) = (F'^1, L'^1, L'^*, L'^0, F'^0)$. Again, as in the proof of Proposition 3, we have $F'^1 = F^1 \cup \{k\}$ and $F'^0 = F^0$. Also, $k \in L^0$ implies that L^0 is not empty and hence $|F^0| + |L^0| = n - \overline{n}$.

In the case where $L^1 \neq \emptyset$ and $L^* = \emptyset$, we have $|F^1| + |L^1| = \underline{n} = \overline{n}$. Consequently L'^1 must contain one location less than L^1, this location therefore being $i^1_>$, and L'^0 must remain the same size as L^0, hence $L'^0 = (L^0 \backslash \{k\}) \cup \{i^1_>\}$.

In the case where $L^1 = \emptyset$ and $L^* = \emptyset$, we have $|F^1| \geq \underline{n}$ and thus L'^1 is also empty. Consequently L'^0 must remain the same size as L^0, hence $L'^0 = (L^0 \backslash \{k\}) \cup \{i^*_>\}$.

In the final case where both L^1 and L^* are nonempty, we have $|F^1| + |L^1| = \underline{n}$. Similarly as in both previous cases, L'^1 must contain one location less than L^1 and hence $L^1 = L^1 \backslash \{i^*_>\}$, and L'^0 must remain the same size as L^0. Note that $\bar{\mathbf{c}}^{\boldsymbol{\mu}}_{i^*_>} = \max_{i \in L^*} \bar{\mathbf{c}}^{\boldsymbol{\mu}}_i < \bar{\mathbf{c}}^{\boldsymbol{\mu}}_{i^1_>}$, hence $L'^0 = (L^0 \backslash \{k\}) \cup \{i^*_>\}$. We deduce the values of $\phi(\mathbf{p}^{+k}, \underline{n}, \overline{n}, \boldsymbol{\mu})$ like in the proof of Proposition 4. □

Corollary 4 *For all $k \in L^1$, if L^* and L^0 are both empty then*

$$\phi(\mathbf{p}^{-k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) = \infty,$$

else if L^* is empty and L^0 is nonempty then

$$\begin{aligned}
\Pi(\mathbf{p}^{-k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) &= (F^1, (L^1 \backslash \{k\}) \cup \{i^0_<\}, \emptyset, L^0 \backslash \{i^0_<\}, F^0 \cup \{k\}), \\
\phi(\mathbf{p}^{-k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) &= \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}) - \bar{\mathbf{c}}^{\boldsymbol{\mu}}_i + \bar{\mathbf{c}}^{\boldsymbol{\mu}}_{i^0_<},
\end{aligned}$$

else if L^* is nonempty and L^0 is empty then

$$\begin{aligned}
\Pi(\mathbf{p}^{-k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) &= (F^1, (L^1 \backslash \{k\}) \cup \{i^*_<\}, L^* \backslash \{i^*_<\}, \emptyset, F^0 \cup \{k\}), \\
\phi(\mathbf{p}^{-k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) &= \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}) - \bar{\mathbf{c}}^{\boldsymbol{\mu}}_i + \max\left\{0, \bar{\mathbf{c}}^{\boldsymbol{\mu}}_{i^*_<}\right\},
\end{aligned}$$

else

$$\begin{aligned}
\Pi(\mathbf{p}^{-k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) &= (F^1, (L^1 \backslash \{k\}) \cup \{i^*_<\}, (L^* \backslash \{i^*_<\}) \cup \{i^0_<\}, L^0 \backslash \{i^0_<\}, F^0 \cup \{k\}), \\
\phi(\mathbf{p}^{-k}, \underline{n}, \overline{n}, \boldsymbol{\mu}) &= \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}) - \bar{\mathbf{c}}^{\boldsymbol{\mu}}_i + \min\left\{0, \bar{\mathbf{c}}^{\boldsymbol{\mu}}_{i^0_<}\right\} + \max\left\{0, \bar{\mathbf{c}}^{\boldsymbol{\mu}}_{i^*_<}\right\}.
\end{aligned}$$

3.4 Bundle method

In this subsection we describe how we adapt a bundle method to optimize the lagrangian dual of any subproblem $SP(\mathbf{p}, \underline{n}, \overline{n})$ with $\mathbf{p} \in \{0, 1, *\}^n$, $1 \leq \underline{n} \leq \overline{n} \leq n$:

$$\bar{\boldsymbol{\mu}} \approx \arg \max_{\boldsymbol{\mu} \in \mathbb{R}^m} \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}),$$

Recall that such subproblems correspond to nodes evaluated by the `Branch-and-bound` procedure, and that the optimization of the lagrangian duals takes place in steps 0 and 2 of Algorithm 2.

There exist many different methods to compute $\bar{\boldsymbol{\mu}}$: the simplex method, dual-ascent heuristics [11], the volume algorithm [1], simple subgradient search, Nesterov's algorithm [34], etc. However, our preliminary experiments have indicated that bundle methods works well for this problem. Indeed, on the one hand, optimizing $\phi(\mathbf{p}, \underline{n}, \overline{n}, \cdot)$ exactly with the simplex method is too difficult in practice for all but the easiest and smallest instances of the UFLP, and in any case we are not interested in exact optimization per se. On the other hand, subgradient search methods might not capture enough information about $\phi(\mathbf{p}, \underline{n}, \overline{n}, \cdot)$ during the optimization process to be as effective as the bundle method, as suggested by some preliminary experiments with a bundle method using very small bundle size limits. Bundle methods therefore seem like a good compromise between two extremes. After giving a brief summary of the bundle method, we shall explain how we adapted it to solve our problem. A detailed description of bundle and trust-region methods in general lies beyond the scope of this paper, see, e.g., [20] for a detailed treatment on the subject.

3.4.1 Overview

The bundle method is a trust-region method which optimizes a lagrangian dual function iteratively until a termination criterion is met. At each iteration t, a *trial point* $\boldsymbol{\mu}^{(t)} \in \mathbb{R}^m$ is determined. In the case of the first iteration (i.e. $t = 0$), the trial point depends on where we are in the dual process:

- during preprocessing, in step 0 of Algorithm 2: this is the very first application of the bundle method, and the initial trial point is the result of a dual-ascent heuristic such as `DualLoc`;
- during the resolution of (P), in step 0 of Algorithm 2: the initial trial point is $\bar{\boldsymbol{\mu}}^{\texttt{root}}$, the best vector of multipliers for the root node (computed during preprocessing);
- otherwise, i.e. in step 2 of Algorithm 2: the initial trial point is $\bar{\boldsymbol{\mu}}$, the best vector of multipliers for the parent node.

At any subsequent iteration $t > 0$, $\boldsymbol{\mu}^{(t)}$ is determined by optimizing a *model* approximating $\phi(\mathbf{p}, \underline{n}, \overline{n}, \cdot)$.

Next, $\phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}^{(t)})$ is evaluated, and this value and a subgradient are used to update the model of the dual. A new iteration is then performed, unless either a convergence criterion, or a predefined iteration limit, or a bound limit is reached, in which case we update the best multipliers:

$$\bar{\boldsymbol{\mu}} \leftarrow \arg\max_t \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}^{(t)}).$$

The convergence criterion is satisfied when the bundle method determines that its model of the dual $\phi(\mathbf{p}, \underline{n}, \overline{n}, \cdot)$ provides a good enough approximation of the dual in the neighborhood of an optimal vector of multipliers. The bound limit is reached when the current trial point $\boldsymbol{\mu}^{(t)}$ satisfies $\phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}^{(t)}) \geq \bar{z}$, where $\bar{z}$ is an upper bound on the optimal value of (P).

The model of the lagrangian dual is specified using a bundle $\mathcal{B}$, defined by a set of vector-scalar pairs $(\boldsymbol{\sigma}, \epsilon)$ such that the vector $\boldsymbol{\sigma} \in \mathbb{R}^m$ and the scalar $\epsilon \in \mathbb{R}$ define a first-order outer approximation of $\phi(\mathbf{p}, \underline{n}, \overline{n}, \cdot)$, i.e.

$$\forall \boldsymbol{\mu} \in \mathbb{R}^m, \quad \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}) \leq \sum_{j=1}^{m} \boldsymbol{\sigma}_j \boldsymbol{\mu}_j + \epsilon.$$

At each iteration t, the bundle $\mathcal{B}$ [and hence, the model of $\phi(\mathbf{p}, \underline{n}, \overline{n}, \cdot)$] is updated using a pair $(\boldsymbol{\sigma}^{(t)}, \epsilon^{(t)})$ generated as follows. First, we determine an optimal solution $\underline{\mathbf{x}}^{(t)}$ of the lagrangian relaxation $LR(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}^{(t)})$. Assuming that it exists, let

$$z^{(t)} = \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}^{(t)}) = \sum_{j=1}^{m} \boldsymbol{\mu}_j^{(t)} + \sum_{i=1}^{n} \bar{\mathbf{c}}_i^{\boldsymbol{\mu}^{(t)}} \underline{\mathbf{x}}_i^{(t)}.$$

To determine a subgradient $\boldsymbol{\sigma}^{(t)} \in \mathbb{R}^m$, we specify a vector $\underline{\mathbf{y}}^{(t)}$ associated with the optimal solution $\underline{\mathbf{x}}^{(t)}$ of $LR(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}^{(t)})$:

$$\forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\}, \quad \underline{\mathbf{y}}_{ij}^{(t)} = \begin{cases} 0 & \text{if } \underline{\mathbf{x}}_i^{(t)} = 0 \text{ or } \mathbf{s}_{ij} - \boldsymbol{\mu}_j^{(t)} > 0, \\ 1 & \text{if } \underline{\mathbf{x}}_i^{(t)} = 1 \text{ and } \mathbf{s}_{ij} - \boldsymbol{\mu}_j^{(t)} < 0, \\ 0 \text{ or } 1 & \text{otherwise.} \end{cases}$$

For all suitable $\underline{\mathbf{y}}^{(t)}$, the vector in $\mathbb{R}^m$ with components $\left(1 - \sum_{i=1}^{n} \underline{\mathbf{y}}_{ij}^{(t)}\right)_j$ for all $j \in \{1, \ldots, m\}$ is a subgradient of $\phi(\mathbf{p}, \underline{n}, \overline{n}, \cdot)$. We let

$$\boldsymbol{\sigma}_j^{(t)} = 1 - \left| \left\{ i \in \{1, \ldots, n\} : \underline{\mathbf{x}}_i^{(t)} = 1, \mathbf{s}_{ij} < \boldsymbol{\mu}_j^{(t)} \right\} \right|, \quad \forall j \in \{1, \ldots, m\}.$$

and

$$\epsilon^{(t)} = z^{(t)} - \sum_{j=1}^{m} \boldsymbol{\sigma}_j^{(t)} \boldsymbol{\mu}_j^{(t)}.$$

The function $\phi(\mathbf{p}, \underline{n}, \overline{n}, \cdot)$ is concave, thus $\phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}) \leq \sum_{j=1}^{m} \boldsymbol{\sigma}_j^{(t)} \boldsymbol{\mu}_j + \epsilon^{(t)}$ for all $\boldsymbol{\mu} \in \mathbb{R}^m$. We therefore update the bundle $\mathcal{B}$ using the pair $(\boldsymbol{\sigma}^{(t)}, \epsilon^{(t)})$ specifying a first-order outer approximation of $\phi(\mathbf{p}, \underline{n}, \overline{n}, \cdot)$ at $\boldsymbol{\mu}^{(t)}$.

3.4.2 Subgradient cache

Recall that the subproblems of the form $SP(\mathbf{p}, \underline{n}, \overline{n})$ are associated with the nodes in a branch-and-bound search tree, therefore if a bundle method is to be performed at each node, then it stands to reason that some lagrangian dual functions may be very similar to one another. In particular, one $(\boldsymbol{\sigma}, \epsilon)$ pair generated during the optimization

of one dual function may also specify a valid first-order outer approximation function of another dual function. For this reason, our method maintains a cache $\mathcal{C}$ in which $(\boldsymbol{\sigma}, \epsilon)$ pairs are stored as soon as they are generated. Thus, before optimizing any dual function $\phi(\mathbf{p}, \underline{n}, \overline{n}, \cdot)$, our method retrieves some suitable pairs from $\mathcal{C}$ to populate the current bundle (which is otherwise empty), thus improving the model and hopefully also the quality of the trial points. Note that although there exist other ways of re-using the subgradient information generated during the application of the bundle method, ours is simple, effective, and requires only a limited amount of memory. We now introduce the result which tells us if and how a pair is suitable.

Proposition 6 *Suppose that* $\underline{\mathbf{x}}$ *is an optimal solution of any lagrangian relaxation* $LR(\mathbf{p}', \underline{n}', \overline{n}', \boldsymbol{\mu})$*, and let* $(\boldsymbol{\sigma}, \epsilon)$ *be the corresponding pair which defines a first-order outer approximation of* $\phi(\mathbf{p}', \underline{n}', \overline{n}', \cdot)$*. This pair defines a first-order outer approximation of any other function* $\phi(\mathbf{p}'', \underline{n}'', \overline{n}'', \cdot)$ *if* $\underline{\mathbf{x}}$ *is also an optimal solution of* $LR(\mathbf{p}'', \underline{n}'', \overline{n}'', \boldsymbol{\mu})$*.*

Proof Since $\boldsymbol{\sigma}$ is generated using only $\bar{\mathbf{c}}^{\boldsymbol{\mu}}$ and $\underline{\mathbf{x}}$, and since $\underline{\mathbf{x}}$ is optimal for $LR(\mathbf{p}'', \underline{n}'', \overline{n}'', \boldsymbol{\mu})$ as well as for $LR(\mathbf{p}', \underline{n}', \overline{n}', \boldsymbol{\mu})$, then:

- $\phi\left(\mathbf{p}'', \underline{n}'', \overline{n}'', \boldsymbol{\mu}\right) = \phi\left(\mathbf{p}', \underline{n}', \overline{n}', \boldsymbol{\mu}\right) = \sum_{j=1}^{m} \boldsymbol{\mu}_j + \sum_{i=1}^{n} \bar{\mathbf{c}}_i^{\boldsymbol{\mu}} \underline{\mathbf{x}}_i$,
- $\boldsymbol{\sigma}$ is a subgradient of $\phi(\mathbf{p}'', \underline{n}'', \overline{n}'', \cdot)$ as well as of $\phi(\mathbf{p}', \underline{n}', \overline{n}', \cdot)$.

This last function is concave, therefore for all $\tilde{\boldsymbol{\mu}} \in \mathbb{R}^m$:

$$\phi\left(\mathbf{p}'', \underline{n}'', \overline{n}'', \tilde{\boldsymbol{\mu}}\right) \leq \sum_{j=1}^{m} \boldsymbol{\sigma}_j \tilde{\boldsymbol{\mu}}_j + \left(\phi\left(\mathbf{p}'', \underline{n}'', \overline{n}'', \boldsymbol{\mu}\right) - \sum_{j=1}^{m} \boldsymbol{\sigma}_j \boldsymbol{\mu}_j\right),$$

$$\phi\left(\mathbf{p}'', \underline{n}'', \overline{n}'', \tilde{\boldsymbol{\mu}}\right) \leq \sum_{j=1}^{m} \boldsymbol{\sigma}_j \tilde{\boldsymbol{\mu}}_j + \epsilon.$$

□

Remark 1 Suppose that $\mathbf{p}''$, $\underline{n}''$ and $\overline{n}''$ are such that $SP(\mathbf{p}'', \underline{n}'', \overline{n}'')$ is a subproblem of $SP(\mathbf{p}', \underline{n}', \overline{n}')$. In this special case, $\underline{\mathbf{x}}$ only needs to be feasible for $SP(\mathbf{p}'', \underline{n}'', \overline{n}'')$, because it is then also optimal for $LR(\mathbf{p}'', \underline{n}'', \overline{n}'', \boldsymbol{\mu})$.

We therefore specify the cache $\mathcal{C}$ as containing key-value pairs of the form $\left((\underline{\mathbf{x}}, \mathbf{p}', \underline{n}', \overline{n}'),\ (\boldsymbol{\sigma}, \epsilon)\right)$. Prior to optimizing a dual function $\phi(\mathbf{p}'', \underline{n}'', \overline{n}'', \cdot)$, we update the bundle using $(\boldsymbol{\sigma}, \epsilon)$ pairs obtained by traversing $\mathcal{C}$ for keys which match the following conditions:

- $\underline{n}' \leq \underline{n}'' \leq \sum_{i=1}^{n} \underline{\mathbf{x}}_i \leq \overline{n}'' \leq \overline{n}'$, and
- for all $i \in \{1, \ldots, n\}$: either $\underline{\mathbf{x}}_i = \mathbf{p}_i''$ or $\mathbf{p}_i'' = *$, and either $\mathbf{p}_i'' = \mathbf{p}_i'$ or $\mathbf{p}_i' = *$.

When properly implemented, these tests can be performed quickly enough for the cache to be of practical use.

Consider now the problem of populating such a cache $\mathcal{C}$. The straightforward way to do so is as follows: at each iteration t of the bundle method applied to any dual function

$\phi(\mathbf{p}, \underline{n}, \overline{n}, \cdot)$, simply add the key-value pair $\big((\underline{\mathbf{x}}^{(t)}, \mathbf{p}, \underline{n}, \overline{n}),\ (\boldsymbol{\sigma}^{(t)}, \epsilon^{(t)})\big)$. However, we can instead try to improve this key in terms of increasing the likelihood of subsequently reusing $(\boldsymbol{\sigma}^{(t)}, \epsilon^{(t)})$. For this purpose we generate a vector $\tilde{\mathbf{p}}$ in the following manner, using the partition $\Pi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}^{(t)}) = (F^1, L^1, L^*, L^0, F^0)$ as well as the reduced costs $\bar{\mathbf{c}}^{\boldsymbol{\mu}^{(t)}}$ computed during the evaluation of $\phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}^{(t)})$. Let

$$\tilde{F}^1 = \left\{ i \in F^1 : \forall k \in (L^* \cup L^0),\ \bar{\mathbf{c}}_i^{\boldsymbol{\mu}^{(t)}} \leq \bar{\mathbf{c}}_k^{\boldsymbol{\mu}^{(t)}} \right\},$$
$$\tilde{F}^0 = \left\{ i \in F^1 : \forall k \in (L^1 \cup L^*),\ \bar{\mathbf{c}}_i^{\boldsymbol{\mu}^{(t)}} \geq \bar{\mathbf{c}}_k^{\boldsymbol{\mu}^{(t)}} \right\},$$

we initialize $\tilde{\mathbf{p}} \leftarrow \mathbf{p}$ and set $\tilde{\mathbf{p}}_i \leftarrow *$ for all $i \in \tilde{F}^1 \cup \tilde{F}^0$.

Proposition 7 *This partial solution* $\tilde{\mathbf{p}}$ *is such that* $\underline{\mathbf{x}}^{(t)}$ *is also an optimal solution of* $LR(\tilde{\mathbf{p}}, \underline{n}, \overline{n}, \boldsymbol{\mu}^{(t)})$.

Proof The locations in $\tilde{F}^1$ form a subset of F^1 and are not more expensive (in terms of $\bar{\mathbf{c}}^{\boldsymbol{\mu}^{(t)}}$) than any location in L^* or L^0, therefore there exists an optimal partition $\Pi(\tilde{\mathbf{p}}, \underline{n}, \overline{n}, \boldsymbol{\mu}^{(t)}) = (F'^1, L'^1, L'^*, L'^0, F'^0)$ such that $\tilde{F}^1 \subset L'^1$. Likewise, there also exists L'^0 such that $\tilde{F}^0 \subset L'^0$. Consequently, $F'^1 \cup L'^1 = F^1 \cup L^1$, $F'^0 \cup L'^0 = F^0 \cup L^0$ and $L'^* = L^*$. □

Corollary 5 $\phi(\tilde{\mathbf{p}}, \underline{n}, \overline{n}, \boldsymbol{\mu}^{(t)}) = \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}^{(t)})$.

This result ensures that $(\boldsymbol{\sigma}^{(t)}, \epsilon^{(t)})$ is also a first-order outer approximation of $\phi(\tilde{\mathbf{p}}, \underline{n}, \overline{n}, \cdot)$ at $\boldsymbol{\mu}^{(t)}$. By associating the key $(\underline{\mathbf{x}}^{(t)}, \tilde{\mathbf{p}}, \underline{n}, \overline{n})$ instead of $(\underline{\mathbf{x}}^{(t)}, \mathbf{p}, \underline{n}, \overline{n})$ with the pair $(\boldsymbol{\sigma}^{(t)}, \epsilon^{(t)})$ in $\mathcal{C}$, we increase its potential for reuse.

A proper implementation of such a cache is crucial to ensure good performance. Details are provided in the Appendix.

4 Computational results

We begin this section by outlining how our implementation executes the primal and dual processes concurrently. Following this, we present our test parameters and methodology, and then we present our results. The complete source code, in C, can be freely obtained from the authors.

4.1 Concurrency

Recall that the primal and dual processes both consist of a loop: in the primal process one iteration corresponds to one 1OPT move, and in the dual process one iteration corresponds to one branch-and-bound node. Although it could be, our implementation is not concurrent. The main reason behind this choice is that concurrency induces significant variance in execution times, as we found out during preliminary tests.[1]

[1] Naturally, in a practical setting, this argument may be completely irrelevant. Furthermore, our method may benefit from having more than just one primal process. We can easily imagine that better results may be achieved by running several different primal heuristics concurrently in addition to our tabu search procedure.

Instead of being concurrent, our implementation performs iterations of the primal and dual processes sequentially and alternately. The dual and primal iterations are not performed in equal proportions: at the beginning of the search 100 primal iterations are performed for every dual iteration, and this ratio progressively inverts itself until 100 dual iterations are performed for every primal iteration. The inversion speed is set by a predefined parameter, and in our implementation the default setting is such that a few thousand primal iterations should have been performed before the steady state of 100 dual per 1 primal iterations is reached. This policy seems to work well for reducing overall CPU time for all instances. Indeed the primal process is much more likely to find an improving solution at the beginning of the search than later on, and vice-versa for the dual process. Additionally, the dual process also benefits from having a good upper bound early in the search.

4.2 Parameters

The machines which we used for our tests are identical workstations equipped with Intel i7-2600 CPUs clocked at 3.4 GHz. We have tested our implementation on all instances in `UflLib`, with a maximum execution time limit set at 2 h of CPU time. To date, the `UflLib` collection consists of the following instance sets: `Bilde-Krarup`; `Chessboard`; `Euclidian`; `Finite projective planes` (`k = 11`, `k = 17`); `Galvão-Raggi`; `Körkel-Ghosh` (symmetric and asymmetric, $n = m \in \{250, 500, 750\}$); `Barahona-Chudak`; `Gap (a, b, c)`; `M*`; `Beasley` (a.k.a. `ORLIB`); `Perfect codes`; `Uniform`. Most of these instances are such that $n = m = 100$, but in the largest case (some instances in `Barahona-Chudak`) we have $n = m = 3000$. Part of the motivation behind applying our method on so many instances stems from how few results on the exact resolution of the UFLP have been published in the last decade.

We performed some preliminary experiments to determine good maximal bundle size settings, and we noticed that execution times seem to be little affected by this parameter. Without going into the details, it appears that for comparatively small ($n = m = 100$) and easy `UflLib` instances, a maximal bundle size between 20 and 50 is best. However since we hope to solve the larger and more difficult instances, we set this parameter to 100 for all of our tests. We use the following bundle method iteration limits: `iter_limit_initial` = 2000 for the very first application of the bundle method during the dual process; `iter_limit_other` = 120 for all the subsequent applications. Other search parameters are set as follows: the improving partial solution $\bar{p}$ is updated after every 250 consecutive iterations of the dual process, or whenever the best-known solution is updated; likewise, the primal process sends requests for guiding solutions after 250 consecutive primal process iterations with no improvement of the best-known solution. Again, these settings are not very sensitive.

4.3 Tests

Our first series of tests aims to illustrate the effectiveness of our branching rules and of maintaining a cache $\mathcal{C}$. We performed these tests on the 30 instances in `Gap-a`, which

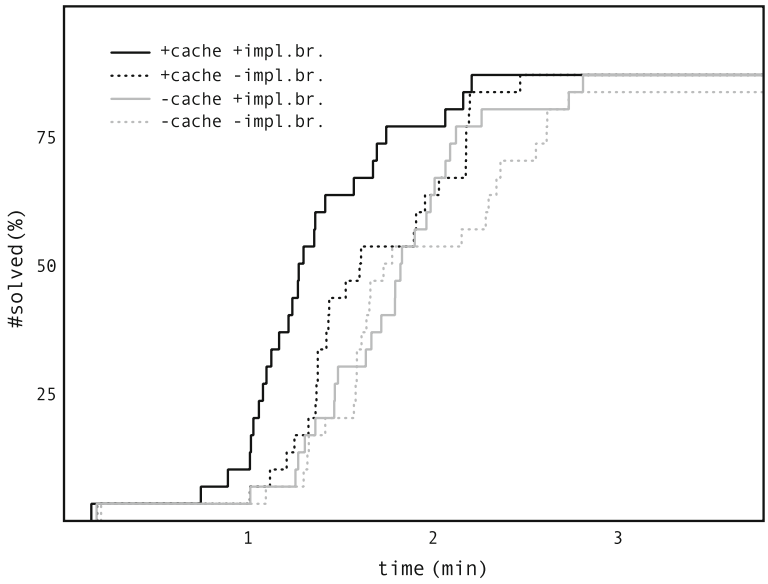

Fig. 1 Preprocessing and branching settings

are the easier instances in `Gap`. The instances in this set are such that $n = m = 100$, and they are characterized by their large duality gap, inducing suitably large search trees for our tests to be meaningful. Because we are not measuring anything related to the primal process, in our tests we disable the primal process and initialize the upper bound $\bar{z}^{\texttt{dual}}$ with the optimal value of the corresponding instance. As a consequence, when solving any one instance, the search trees obtained at the end of the dual process should in theory be identical to one another whichever node selection strategy is used (`LFS` or `DFS`). In practice, many components of our implementation are not numerically stable and hence results differ a little. In fact the mean difference in the number of nodes of the search trees obtained by solving with `DFS` and with `LFS` is 5.4 %, with a standard deviation of 3.3 %.

Figure 1 illustrates the profile curves corresponding to solving with `DFS` and the following settings:

‘`-cache`’ without cache,
‘`+cache`’ with cache, limited to 1,024 elements,
‘`-branch`’ without preprocessing, and only considering the first and last cases in the branching procedures `BrP` and `BrS` (i.e. cases ‘$\underline{n} = \overline{n}$’, ‘$\mathbf{p} \in \{0, 1\}^n$’ and ‘otherwise’),
‘`+branch`’ with preprocessing, with `BrP` and `BrS` as specified in Algorithms 3 and 4.

Observe that ‘`+branch`’ is always worthwhile when used in conjunction with ‘`+cache`’, despite the extra computational effort involved. In fact, performance without the cache is still improved when solving the more difficult instances. For this reason, we use ‘`+branch`’ in all of our subsequent tests.

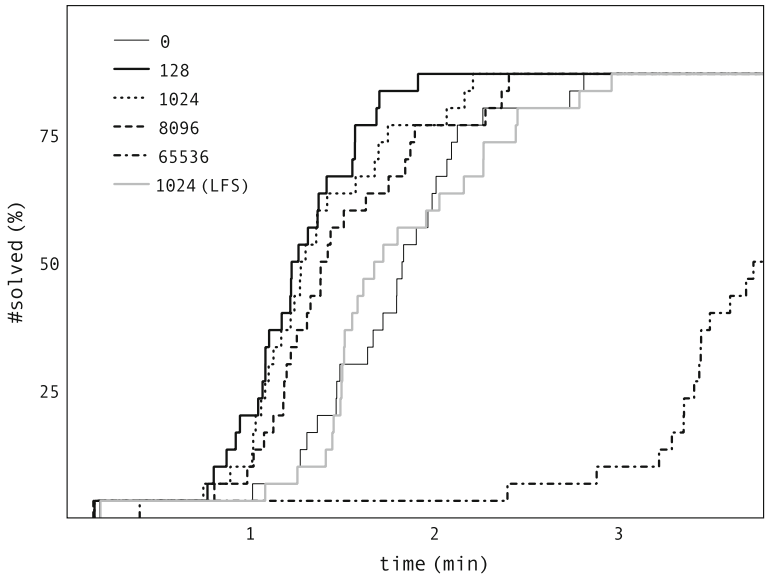

Fig. 2 Cache settings

Figure 2 illustrates the profile curves corresponding to solving with DFS (unless otherwise specified) with the associated maximal cache size. For example, the curve '1024' corresponds to the results using DFS and $|\mathcal{C}| < 1024$. Notice that the cache does not improve performance when used in conjunction with LFS, unlike with DFS. This is not surprising because in large branch-and-bound trees, the nodes consecutively selected by LFS often bear little relation to one another and too few elements of the cache are reused for it to be worthwhile. Conversely, the nodes consecutively selected by DFS are often closely related. Otherwise, we can see that the use of a cache in conjunction with DFS improves performance provided that the cache remains small enough.

4.4 Solving UflLib

For purposes of comparison, we solve all instances with the Gurobi solver (version 5.0.1) in single-threaded mode, and summarize the execution time results in Table 1. The columns 'min', 'median', 'max' list the execution time of the easiest, median and hardest instance in each set, respectively. The column 'mean' lists the mean execution time for the instances in each set which were solved before reaching the time limit, and the column 'std.dev.' lists the corresponding standard deviation. Timeouts are represented by a dash (recall that the time limit is 2 CPU-hours), all other times are in minutes and seconds, rounded up.

Finally, we apply our method on all instances in UflLib with the following two settings:

Table 1 Solving with Gurobi

Instances	Solved	Min	Median	Max	Mean	Std.dev.
Barahona-Chudak	15/18	5”	8’55”	–	17’20”	28’31”
Beasley	17/17	1”	1”	11”	1”	3”
Bilde-Krarup	220/220	1”	2”	15”	4”	4”
Chessboard	30/30	1”	1”	1”	1”	1”
Euclidian	30/30	1”	1”	1”	1”	1”
FPP k=11	30/30	3”	14”	23”	12”	8”
FPP k=17	30/30	56”	2’28”	7’21”	3’18”	2’18”
Gap-a	30/30	58”	3’18”	23’4”	5’15”	5’37”
Gap-b	30/30	1’25”	8’36”	33’46”	10’29”	8’00”
Gap-c	2/30	19’56”	–	–	60’14”	56’59”
Galvao-Raggi	50/50	1”	1”	1”	1”	1”
KG-250	5/30	30’36”	–	–	55’45”	18’02”
KG-500	0/30	–	–	–	–	–
KG-750	0/30	–	–	–	–	–
M*	16/22	3”	8’32”	–	14’51”	28’43”
Perfect Codes	30/30	1”	1”	1”	1”	1”
Uniform	30/30	3”	14”	1’39”	16”	17”

1. `DFS` and a cache with less than 128 elements,
2. `LFS` and no cache.

Tables 2 and 3 summarize the respective execution time results for each instance set. Figure 3 illustrates the corresponding profile curves for the instance sets whose hardest instance required at least one minute: `Barahona-Chudak`, `Finite projective planes`, `Gap`, `Körkel-Ghosh` and `M*`.

First, note that the only instance set for which our method does not work well is `Barahona-Chudak`. These are large instances, for which

$$n = m \in \{500, 1000, 1500, 2000, 2500, 3000\}.$$

We noticed that the tabu search procedure does not always work very well, and that our branch-and-bound approach often fails to partition (P) adequately. Conversely, the Gurobi solver solved 15 out of 18 instances within 2 h, and even more dramatically, the semi-lagrangian approach of Beltran-Royo et al. [4] was able to solve 16 out of these 18 instances to optimality in a matter of seconds. Likewise, the primal-dual approach of Hansen et al. [19] also works very well on instances of this type.

Nonetheless, our method works well for all other instances in `UflLib` and can solve many instances which until now were beyond the capabilites of other general or specialized solvers. We can see that several instance sets are better solved by one setting than by the other. This is especially the case for the `Finite projective planes` instances, for which ‘`DFS` with $|\mathcal{C}| < 128$’ performs very badly, in stark contrast to ‘`LFS`’. Looking more closely we noticed that these poor results were

Table 2 Solving with DFS and $|\mathcal{C}| < 128$

Instances	Solved	Min	Median	Max	Mean	Std.dev.
Barahona-Chudak	6/18	1”	–	–	15’07”	25’42”
Beasley	17/17	1”	1”	1”	1”	1”
Bilde-Krarup	220/220	1”	1”	3”	1”	1”
Chessboard	30/30	1”	1”	12”	1”	3”
Euclidian	30/30	1”	1”	1”	1”	1”
FPP k=11	30/30	1”	7”	3’59”	35”	1’05”
FPP k=17	28/30	2”	10’22”	–	22’02”	30’11”
Gap-a	30/30	31”	5’05”	20’09”	7’27”	5’39”
Gap-b	30/30	3’13”	11’49”	35’02”	12’43”	7’29”
Gap-c	30/30	70’13”	83’60”	117’15”	86’38”	9’25”
Galvao-Raggi	50/50	1”	1”	1”	1”	1”
KG-250	27/30	3”	12’28”	–	24’39”	35’45”
KG-500	7/30	44’31”	–	–	70’08”	22’10”
KG-750	0/30	–	–	–	–	–
M*	22/22	1”	1”	40’50”	1’57”	8’30”
Perfect Codes	32/32	1”	1”	1”	1”	1”
Uniform	30/30	1”	3”	17”	3”	4”

Table 3 Solving with LFS and no cache

Instances	Solved	Min	Median	Max	Mean	Std.dev.
Barahona-Chudak	8/18	1”	–	–	3’60”	4’37”
Beasley	17/17	1”	1”	1”	1”	1”
Bilde-Krarup	220/220	1”	1”	3”	1”	1”
Chessboard	30/30	1”	1”	1”	1”	1”
Euclidian	30/30	1”	1”	1”	1”	1”
FPP k=11	30/30	1”	1”	1”	1”	1”
FPP k=17	30/30	1”	2”	2”	2”	1”
Gap-a	30/30	15”	1’57”	25’46”	4’01”	6’35”
Gap-b	30/30	39”	7’10”	30’22”	10’24”	9’20”
Gap-c	24/30	7’24”	115’08”	–	107’39”	21’26”
Galvao-Raggi	50/50	1”	1”	1”	1”	1”
KG-250	27/30	6”	14’36”	–	25’52”	34’40”
KG-500	3/30	71’07”	–	–	74’31”	3’25”
KG-750	0/30	–	–	–	–	–
M*	22/22	1”	2”	109’29”	5’15”	22’46”
Perfect Codes	32/32	1”	1”	1”	1”	1”
Uniform	30/30	1”	4”	23”	5”	5”

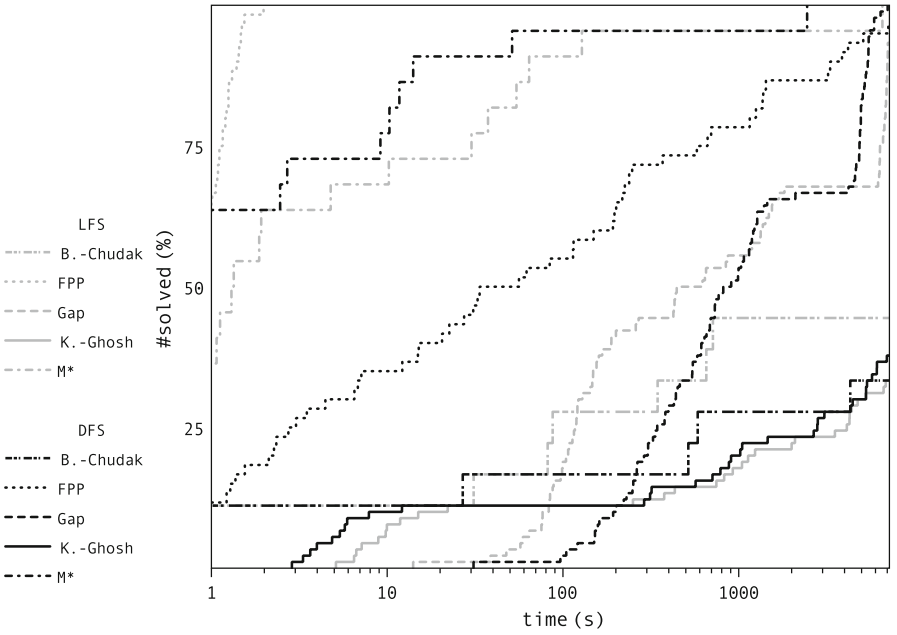

Fig. 3 Profile curves for `UflLib`

correlated to a very poor incumbent solution. Therefore it follows that the tabu search procedure in the primal process remains stuck in a local optimality trap very early in the search. It is therefore up to the dual process to send information to the primal process to enable it to escape from its trap. Evidently this does occur with `LFS` node selection, but not with `DFS`. We performed another series of tests on the `Finite projective planes` instances using `LFS` node selection but with the primal process disabled. We found that the dual process was able to find the optimal solution on its own in a matter of seconds, but not as quickly as in the original '`LFS`' tests. This illustrates the pertinence of the cooperative aspect of our approach.

Note that in the case of the `Gap` instance set, the easier instances in the `a` and `b` subsets are better solved with '`LFS`' while the opposite is true of the harder instances in the `c` subset. On the other hand, the '`DFS` with $|\mathcal{C}| < 128$' setting performs better for `M*` and `Körkel-Ghosh`. The instances in these sets have not yet all been solved to optimality, unlike all other sets in `UflLib`. Until now, only about half of those in `M*` had known optima, yet our method solves all instances but the largest one to optimality in less that a minute on a modern computer, and the largest one in less than an hour. Our method solves all instances in `M*` and a good proportion of those in `Körkel-Ghosh` to optimality. Also, to the best of our knowledge, only 2 of the 90 instances in `Körkel-Ghosh` had known optima [4], yet our method was able to find 28 more.

Table 4 illustrates our results for the instances of the `Körkel-Ghosh` set which we were able to solve to optimality. To the best of our knowledge, most of these were unknown until now. Table 5 illustrates our results for the `M*` set, which we were able to solve to optimality in its entirety in less than an hour. The column '#nodes' lists

Table 4 Optimal values for the Körkel-Ghosh instance set

Instance	Opt	Time	#Nodes	#Lag.eval.	#Moves
ga250a-2	257502	5'20"	69,950	4,352,212	10,829
ga250a-4	257987	116'04"	1,555,510	96,300,841	96,301
ga250b-1	276296	24'21"	452,946	22,557,355	23,118
ga250b-2	275141	4'50"	87,894	4,467,390	10,092
ga250b-3	276093	13'07"	249,354	12,146,379	16,655
ga250b-4	276332	17'27"	324,970	15,686,289	18,916
ga250b-5	276404	15'05"	286,054	13,942,841	17,883
ga250c-1	334135	6"	2,510	80,553	966
ga250c-2	330728	3"	1,220	39,043	639
ga250c-3	333662	5"	2,106	67,897	878
ga250c-4	332423	4"	1,426	45,501	697
ga250c-5	333538	6"	2,602	82,561	976
ga500c-1	621360	44'31"	175,544	5,673,939	9,331
ga500c-2	621464	88'23"	328,682	10,324,601	12,499
ga500c-3	621428	101'56"	383,096	11,840,896	13,299
ga500c-4	621754	88'21"	312,472	9,600,336	11,919
gs250a-1	257964	51'12"	617,345	38,626,708	38,627
gs250a-2	257573	17'01"	199,171	12,187,024	18,473
gs250a-3	257626	95'32"	1,245,908	77,751,785	77,752
gs250a-4	257961	72'06"	964,562	59,278,305	59,279
gs250a-5	257896	101'45"	1,332,652	82,664,426	82,665
gs250b-1	276761	89'30"	1,659,142	82,929,645	82,930
gs250b-2	275675	11'49"	224,214	10,927,163	15,774
gs250b-3	275710	14'40"	276,632	13,602,650	17,729
gs250b-4	276114	5'14"	97,728	4,799,499	10,325
gs250b-5	275916	9'29"	173,536	8,815,952	14,381
gs250c-1	332935	4"	1,750	54,895	772
gs250c-2	334630	8"	3,476	110,912	1,152
gs250c-3	333000	6"	2,320	73,653	915
gs250c-4	333158	4"	1,568	49,979	739
gs250c-5	334635	13"	5,362	172,336	1,474
gs500c-1	620041	46'58"	185,670	5,850,888	9,368
gs500c-2	620434	46'40"	187,538	5,865,590	9,348
gs500c-4	620437	74'10"	300,514	9,343,075	11,822

the number of nodes in the search tree for each instance, and the column '#lag.eval.' lists the number of times a lagrangian relaxation was solved as per Sect. 3.1. The column '#moves' lists the number of 1OPT moves performed by the primal process. As before, all times are rounded up.

Table 5 Optimal values for the M* instance set

Instance	Opt	Time	#Nodes	#Lag.eval.	#Moves
MO1	1305.95	1”	68	2,880	139
MO2	1432.36	1”	58	3,088	149
MO3	1516.77	1”	92	3,774	165
MO4	1442.24	1”	26	1,296	85
MO5	1408.77	1”	62	2,690	134
MP1	2686.48	1”	142	6,231	227
MP2	2904.86	1”	168	6,605	232
MP3	2623.71	1”	52	2,264	120
MP4	2938.75	1”	232	9,688	296
MP5	2932.33	1”	362	15,348	393
MQ1	4091.01	1”	108	4,436	182
MQ2	4028.33	1”	180	7,338	249
MQ3	4275.43	1”	106	4,256	177
MQ4	4235.15	1”	138	5,608	211
MQ5	4080.74	3”	608	25,728	537
MR1	2608.15	10”	890	38,963	692
MR2	2654.73	3”	280	9,187	275
MR3	2788.25	12”	1,582	48,423	714
MR4	2756.04	15”	1,744	61,719	855
MR5	2505.05	11”	1,142	44,873	732
MS1	5283.76	52”	1,090	37,352	635
MT1	10069.80	40’50”	20,094	663,007	3,080

5 Concluding remarks

In this paper, we presented a cooperative method to solve (P) exactly, in which a primal process and a dual process exchange information to improve the search. The primal process performs a variation of the tabu search procedure proposed by Michel et al. [31], and the dual process performs a lagrangian branch-and-bound search. We selected this particular metaheuristic for its simplicity and its good overall performance, however any other metaheuristic can be used instead.

Our main contributions lie in the dual process. Partitionning (P) into subproblems $SP(\mathbf{p}, \underline{n}, \overline{n})$ allows us to apply sophisticated branching strategies. Our branching rules rely on results for quickly reoptimizing lagrangian relaxations with modified parameters $\mathbf{p}$, $\underline{n}$ or $\overline{n}$, and hence require relatively little computational effort. Note that the branch-and-bound method presented in this paper could be supplemented with the preprocessing and branching rules presented by Goldengorin et al. [16,17].

Furthermore, we introduced a subgradient caching technique which helps improve performance of the bundle method, which we apply to compute a lower bound at each node. The subgradients are stored in the cache as soon as they are generated during the

application of the bundle method, and may be retrieved to initialize the bundle at the beginning of a subsequent application. We introduced results to improve the potential for reuse of a subgradient within the specific context of the UFLP, but these possibly could be extended to lagrangian branch-and-bound methods for other problems. Maintaining a subgradient cache may not be the ideal way of reusing information obtained during the optimization of the lagrangian dual, but this technique has certain practical advantages: it requires constant time and space, it is conceptually simple (and hence, easy to implement), its behavior is easy to control and it works well enough in conjunction with a depth-first node selection strategy.

Note also that by computing the lower bound at each node using a bundle method allows us a certain level of control on the computational effort expended at each node, which more or less directly translates into bound quality. This is in contrast to the dual-ascent heuristic of `DualLoc`, and also to integer programming solvers such as CPLEX or Gurobi which apply the dual simplex algorithm to completion.

Finally, we presented extensive computational results. Our method performs well for a wide variety of problem instances, several of which having been solved to optimality for the first time.

Acknowledgments We wish to thank Paul-Virak Khuong who kindly helped us with our code, improving its performance significantly. We are also grateful to Antonio Frangioni for his bundle method implementation and for taking the time to answer our questions. Finally, we wish to thank the referees and editors which helped improve this paper.

Appendix A: Cache implementation details

An efficient implementation of $\mathcal{C}$ is crucial to its performance. In our implementation, $\mathcal{C}$ is a FIFO queue whose maximal size is a predefined parameter. Consequently, an insertion of a new element into a full $\mathcal{C}$ is preceded by the removal of its oldest element. When inserting a new element into $\mathcal{C}$, our implementation does not examine the contents of $\mathcal{C}$ to detect if it is already present, but instead it tests its presence using a counting Bloom filter. A Bloom filter is a data structure originally proposed by Bloom [6] which allows testing efficiently whether a set contains an element (with occasional false positives, but no false negatives). A counting Bloom filter is a variant defined using any vector $\boldsymbol{\beta}$ of positive integers and a set H of different hash functions $h \in H$ which each map any element possibly in $\mathcal{C}$ to an index of $\boldsymbol{\beta}$:

- if an element e is added into $\mathcal{C}$, then increment $\boldsymbol{\beta}_{h(e)}$ by 1 for all $h \in H$,
- therefore an element e is certainly not in $\mathcal{C}$ if there exists $h \in H$ for which $\boldsymbol{\beta}_{h(e)} = 0$,
- if an element e is removed from $\mathcal{C}$, then decrement $\boldsymbol{\beta}_{h(e)}$ by 1 for all $h \in H$.

Bloom filters are efficient in time and in space even for very large sets. By hard-coding the maximum size of $\mathcal{C}$ to $2^{16} - 1$ elements, we implemented $\boldsymbol{\beta}$ as an array of 16-bit unsigned integers. In our implementation, the dimension of this array is in the low hundred thousands, and $|H| = 7$. As a consequence, the probability of false positives when testing whether $e \in \mathcal{C}$ is less than 1%.

When trying to insert a new element e into $\mathcal{C}$, the following steps are performed:

1. If $\boldsymbol{\beta}_{h(e)} > 0$ for all $h \in H$, then e is probably already in $\mathcal{C}$: go to step 4.
2. If $\mathcal{C}$ is full, then remove the oldest element e' in $\mathcal{C}$ and decrement $\boldsymbol{\beta}_{h(e')}$ by 1 for all $h \in H$.
3. Insert e into $\mathcal{C}$ and increment $\boldsymbol{\beta}_{h(e)}$ by 1 for all $h \in H$.
4. Done.

The counting Bloom filter thus ensures the unicity of the elements in $\mathcal{C}$.

Appendix B: Bundle method iteration limits

Recall that the elements stored in $\mathcal{C}$ are key-value pairs specifying first-order outer approximations of lagrangian dual functions, and the purpose of $\mathcal{C}$ is to provide valid approximations to populate the bundle. Recall also that in our method we apply a bundle method during the evaluation of each node $(\mathbf{p}, \underline{n}, \overline{n}, \bar{\boldsymbol{\mu}})$ selected in $\mathcal{Q}$ by the `Branch-and-bound` procedure, and that the initial trial point $\boldsymbol{\mu}^{(0)}$ is set to $\bar{\boldsymbol{\mu}}$, i.e. the best vector of multipliers found for the parent node. Once our implementation has identified a subset S of valid $(\boldsymbol{\sigma}, \epsilon)$ pairs present in $\mathcal{C}$, it does not immediately update the bundle with all pairs in S. We have found it more convenient to update the bundle using at most b pairs in S, with b being the maximum bundle size minus 1. This seemingly arbitrary choice was partly due to certain limitations of `[B]TT` (the bundle method implementation which we use), however it seems to be effective. In the case where the valid subset size exceeds this number, we update the bundle with the b pairs in S selected as follows: let $\boldsymbol{\mu}^{(0)}$ be the initial trial point of the bundle method, we select $(\boldsymbol{\sigma}, \epsilon) \in S$ minimizing the value $\sum_{j=1}^{m} \boldsymbol{\mu}^{(0)} \boldsymbol{\sigma}_j + \epsilon$. In this manner, we select the most accurate first-order outer approximations, the value $\sum_{j=1}^{m} \boldsymbol{\mu}^{(0)} \boldsymbol{\sigma}_j + \epsilon - \phi(\mathbf{p}, \underline{n}, \overline{n}, \boldsymbol{\mu}^{(0)})$ being the error at $\boldsymbol{\mu}^{(0)}$ of the approximation defined by $(\boldsymbol{\sigma}, \epsilon)$.

Recall that in our implementation, a hard limit on the number of iterations to be performed by an application of the bundle method is set. This is either `iter_limit_initial` for the very first application (at the beginning of which the cache is empty), or `iter_limit_other` for the others. In all applications other than the first, we use `iter_limit_other` - $\min\{b, |S|\}$ as the actual iteration limit. In this manner, and provided that `iter_limit_other` > b, the intensity of the search for the optimal multiplier is inversely proportional to the size of the initial bundle, and hence indirectly to the quality of the model of the lagrangian dual.

Appendix C: Fast floating point arithmetic

A large part of the computational effort is expended in the following computations within the bundle method:

- computing reduced costs $\bar{\mathbf{c}}^{\boldsymbol{\mu}}$,
- generating a subgradient $\boldsymbol{\sigma}^{(t)}$,
- computing $\sum_{j=1}^{m} \boldsymbol{\mu}_j^{(0)} \boldsymbol{\sigma}_j + \epsilon$.

While the theoretical complexity of these operations cannot be reduced, in practice we can speed these up by performing as many of them as possible in the SSE registers which are now standard on x86 processors. Consider the computation of the reduced costs given $\boldsymbol{\mu} \in \mathbb{R}^m$:

$$\forall i \in \{1, \ldots, n\}, \quad \bar{\mathbf{c}}_i^{\boldsymbol{\mu}} \leftarrow \mathbf{c}_i + \sum_{j=1}^{m} \min\left\{0, \mathbf{s}_{ij} - \boldsymbol{\mu}_j\right\}.$$

The sum of the minima can be performed in the SSE registers several minima at a time. Similarly, the computation of $\sum_{j=1}^{m} \boldsymbol{\mu}_j^{(0)} \boldsymbol{\sigma}_j + \epsilon$ for all suitable $(\boldsymbol{\sigma}, \epsilon)$ pairs identified at the beginning of a bundle method consists mainly of a dot product. This operation can be performed by the BLAS function `ddot`, and recent implementations of BLAS will perform several products simultaneously.

The generation of the subgradient $\boldsymbol{\sigma}^{(t)}$ was specified earlier as follows:

$$\forall j \in \{1, \ldots, m\}, \quad \boldsymbol{\sigma}_j^{(t)} \leftarrow 1 - \left|\left\{i \in \{1, \ldots, n\} : \underline{\mathbf{x}}_i^{(t)} = 1, \mathbf{s}_{ij} < \boldsymbol{\mu}_j^{(t)}\right\}\right|,$$

but can be rewritten using the following vectors $\mathbf{v}^i \in \{0, 1\}^m$, $i \in \{1, \ldots, n\}$:

$$\boldsymbol{\sigma}^{(t)} \leftarrow (1, 1, \ldots, 1) - \sum_{\substack{i=1 \\ \underline{\mathbf{x}}_i^{(t)}=1}}^{n} \mathbf{v}^i,$$

$$\text{with } \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\}, \ \mathbf{v}_j^i = \begin{cases} 1 & \text{if } \mathbf{s}_{ij} < \boldsymbol{\mu}_j^{(t)}, \\ 0 & \text{otherwise.} \end{cases}$$

Using SSE registers, we can compute several components of a vector $\mathbf{v}^i$ simultaneously.

References

1. Barahona, F., Anbil, R.: The volume algorithm: producing primal solutions with a subgradient method. Math. Program. **87**(3), 385–399 (2000)
2. Beasley, J. E.: Lagrangean heuristics for location problems. Eur. J. Oper. Res. **65**(3), 383–399 (1993)
3. Beltran, C., Tadonki, C., Vial, J. P.: Solving the p-median problem with a semi-lagrangian relaxation. Comput. Optim. Appl. **35**(2), 239–260 (2006)
4. Beltran-Royo, C., Vial, J.P., Alonso-Ayuso, A.: Semi-lagrangian relaxation applied to the uncapacitated facility location problem. Comput. Optim. Appl. **51**(1), 387–409 (2012)
5. Bilde, O., Krarup, J.: Sharp lower bounds and efficient algorithms for the simple plant location problem. Ann. Discret. Math. **1**, 79–97 (1977)
6. Bloom, B. H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
7. Byrka, J., Aardal, K.: An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. SIAM J. Comput. **39**(6), 2212–2231 (2010)
8. Conn, A. R., Cornuejols, G.: A projection method for the uncapacitated facility location problem. Math. Program. **46**(1), 273–298 (1990)

9. Cornuejols, G., Nemhauser, G.L., Wolsey, L.A.: The uncapacitated facility location problem. In: Mirchandani, P.B., Francis, R.L. (eds.) Discrete Location Theory, pp. 1–54. Wiley, New York (1983)
10. Cura, T.: A parallel local search approach to solving the uncapacitated warehouse location problem. Comput. Ind. Eng. **59**(4), 1000–1009 (2010)
11. Erlenkotter, D.: A dual-based procedure for uncapacitated facility location. Oper. Res. **26**(6), 992–1009 (1978)
12. Flaxman, A. D., Frieze, A. M., Vera, J. C.: On the average case performance of some greedy approximation algorithms for the uncapacitated facility location problem. Comb. Probab. Comput. **16**(05), 713–732 (2007)
13. Galvão, R. D., Raggi, L. A.: A method for solving to optimality uncapacitated location problems. Ann. Oper. Res. **18**(1), 225–244 (1989)
14. Ghosh, D.: Neighborhood search heuristics for the uncapacitated facility location problem. Eur. J. Oper. Res. **150**(1), 150–162 (2003)
15. Glover, F.: A template for scatter search and path relinking. In: Artificial evolution, pp. 1–51. Springer, Berlin (1998)
16. Goldengorin, B., Ghosh, D., Sierksma, G.: Branch and peg algorithms for the simple plant location problem. In: Computers and operations research. Comput. Oper. Res. **30**(1), 967–981 (2003)
17. Goldengorin, B., Tijssen, G. A., Ghosh, D., Sierksma, G.: Solving the simple plant location problem using a data correcting approach. J. Global Optim. **25**(4), 377–406 (2003)
18. Guner, A. R., Sevkli, M.: A discrete particle swarm optimization algorithm for uncapacitated facility location problem. J. Artif. Evol. Appl. **2008**, 1–9 (2008)
19. Hansen, P., Brimberg, J., Urošević, D., Mladenović, N.: Primal-dual variable neighborhood search for the simple plant-location problem. INFORMS J. Comput. **19**(4), 552 (2007)
20. Hiriart-Urruty, J.B., Lemaréchal, C.: Convex analysis and minimization algorithms: Fundamentals, vol. 305. Springer, Berlin (1996)
21. Homberger, J., Gehring, H.: A two-level parallel genetic algorithm for the uncapacitated warehouse location problem. In: Hawaii international conference on system sciences, proceedings of the 41st Annual, p. 67. IEEE (2008)
22. Julstrom, B.: A permutation coding with heuristics for the uncapacitated facility location problem. Recent advances in evolutionary computation for combinatorial optimization. pp. 295–307 (2008)
23. Körkel, M.: On the exact solution of large-scale simple plant location problems. Eur. J. Oper. Res. **39**(2), 157–173 (1989)
24. Krarup, J., Pruzan, P. M.: The simple plant location problem: survey and synthesis. Eur. J. Oper. Res. **12**(1), 36–81 (1983)
25. Kratica, J., Tošic, D., Filipović, V., Ljubić, I.: Solving the simple plant location problem by genetic algorithm. Oper. Res. **35**(1), 127–142 (2001)
26. Labbe, M., Louveaux, F.: Location problems. In: Dell'Amico, M., Maffioli, F., Martello, S. (eds.) Annotated bibliographies in combinatorial optimization, pp. 261–281. Wiley, New York (1997)
27. Letchford, A. N., Miller, S. J.: An aggressive reduction scheme for the simple plant location problem. Tech. rep., Department of Management Science, Lancaster University (2011)
28. Letchford, A. N., Miller, S. J.: Fast bounding procedures for large instances of the simple plant location problem. Comput. Oper. Res. **39**(5), 985–990 (2012)
29. Li, S.: A 1.488 approximation algorithm for the uncapacitated facility location problem. Automata, languages and programming, pp. 77–88 (2011)
30. Mahdian, M., Ye, Y., Zhang, J.: Approximation algorithms for metric facility location problems. SIAM J. Comput. **36**(2), 411–432 (2006)
31. Michel, L., Van Hentenryck, P.: A simple tabu search for warehouse location. Eur. J. Oper. Res. **157**(3), 576–591 (2004)
32. Mladenović, N., Brimberg, J., Hansen, P.: A note on duality gap in the simple plant location problem. Eur. J. Oper. Res. **174**(1), 11–22 (2006)
33. Muter, İ., Birbil, S. I., Sahin, G.: Combination of metaheuristic and exact algorithms for solving set covering-type optimization problems. INFORMS J. Comput. **22**(4), 603–619 (2010)
34. Nesterov, Y.: Smooth minimization of non-smooth functions. Math. Program. **103**(1), 127–152 (2005)
35. Resende, M. G.C., Werneck, R. F.: A hybrid multistart heuristic for the uncapacitated facility location problem. Eur. J. Oper. Res. **174**(1), 54–68 (2006)
36. Sun, M.: Solving the uncapacitated facility location problem using tabu search. Comput. Oper. Res. **33**(9), 2563–2589 (2006)