A Branch-and-Price Algorithm for the Capacitated *p*-Median Problem

Alberto Ceselli, Giovanni Righini
Dipartimento di Technologie dell'Informazione, Università degli Studi di Milano, via Bramante 65, 26013 Crema, Italy

The capacitated *p*-median problem is the variation of the well-known *p*-median problem in which a demand is associated to each user, a capacity is associated to each candidate median, and the total demand of the users associated to the same median must not exceed its capacity. We present a branch-and-price algorithm, that exploits column generation, heuristics and branch-and-bound to compute optimal solutions. We compare our branch-and-price algorithm with other methods proposed so far, and we present computational results both on test instances taken from the literature and on random instances with different values of the ratio between the number of medians and the number of users. © 2005 Wiley Periodicals, Inc. NETWORKS, Vol. 45(3), 125–142 2005

Keywords: integer programming; *p*-median; column generation; branch-and-price

1. INTRODUCTION

The *p*-median problem (PMP) is one of the most widely studied problems in location theory. It consists of partitioning the vertices of a given graph into p subsets and to choose a *median* vertex in each subset, minimizing the sum of the distances between each vertex of the graph and the median of its subset. The PMP arises in many different contexts such as network design, telecommunications, distributed database design, transportation, and distribution logistics. Kariv and Hakimi [14] proved that the PMP is $\mathcal{NP}$-hard. Optimization algorithms based on Lagrangean relaxation have been proposed in [2, 4, 5, 21] and approaches based on dual formulations are illustrated in [9, 11]. A survey on the PMP can be found in [22].

If medians represent facilities providing a certain service and each vertex of the graph represents a user who requires that service, it is natural to introduce capacity constraints, so that the sum of the demands of the users assigned to each facility is forced not to exceed the capacity of that facility. This yields the capacitated *p*-median problem (CPMP). Algorithms devised for the uncapacitated PMP cannot be adapted to the CPMP in a straightforward way: even finding a feasible solution is $\mathcal{NP}$-complete when capacities are considered.

Very recently, two heuristic algorithms for the CPMP have been presented: Lorena and Senne [16] followed a column generation approach, finding good solutions on real instances with up to 402 vertices, while Diaz and Fernández [7] attacked an instance with 737 vertices through hybrid scatter search and path relinking.

In this article we deal with exact optimization algorithms for the CPMP. The optimization of the CPMP can be pursued through the adaptation of an algorithm developed by Pirkul [25] for the capacitated concentrator location problem (CCLP): the CCLP is similar to the CPMP in that it involves the allocation of a set of indivisible users' demands to capacitated facilities, but the number of facilities to be used is not fixed; instead, a cost is incurred for each facility used. The algorithm is based on Lagrangean relaxation and branch-and-bound, and it can solve the CCLP to optimality on graphs with up to 100 vertices in a few minutes. Another approach was suggested by Ross and Soland [27], and consists of the reformulation of CPMP instances as generalized assignment problem (GAP) instances; although based on an elegant transformation, this technique is not competitive because the GAP instances it produces are very large. More recently, Baldacci et al. [1] developed an algorithm that either proves optimality or provides a bound on the approximation obtained; the authors report about experiments for problems with up to 200 vertices and 20 medians, when their algorithm is initialized with quasioptimal feasible solutions. In this article we present a branch-and-price algorithm for the exact solution of the CPMP, and we compare it with a general purpose integer linear programming solver (CPLEX) and with the algorithm of Pirkul (adapted to the CPMP). We also compare our results with those obtained by the algorithm of Baldacci et al. [1] when it is properly initialized.

The article is organized as follows. In Section 2 we define the mathematical model of the CPMP, and we present

Received October 2002; accepted December 2004
Correspondence to: G. Righini; e-mail: righini@dti.unimi.it
DOI 10.1002/net.20059
Published online in Wiley InterScience (www.interscience.wiley.com).

a disaggregated formulation; then we compare the lower bounds obtained from the corresponding linear relaxations; in Section 3 we describe our branch-and-price algorithm and we illustrate some implementation details; in Section 4 we describe the algorithms we used as benchmarks and we present experimental results.

2. FORMULATIONS

Consider a graph $G = (\mathcal{N}, E)$ and a subset $\mathcal{M} \subseteq \mathcal{N}, \mathcal{M} = \{1 \dots M\}$ of its vertices that are candidates to be medians. For each vertex $i \in \mathcal{N}$ an integer weight w_i represents its demand. For each candidate median $j \in \mathcal{M}$ an integer coefficient Q_j represents its capacity. Integer coefficients d_{ij} (usually referred to as *distances*) describe the cost of allocating each vertex $i \in \mathcal{N}$ to each median $j \in \mathcal{M}$. We make the assumption that $d_{ij} \geq 0 \ \forall i \in \mathcal{N}, j \in \mathcal{M}$. The vertex set $\mathcal{N}$ must be partitioned into p subsets (*clusters*), where p is given. The capacitated p-median problem (CPMP) is formulated as follows:

$$CPMP) \min \upsilon = \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{M}} d_{ij} x_{ij}$$

s.t.

$$\sum_{j \in \mathcal{M}} x_{ij} = 1 \qquad \forall i \in \mathcal{N} \tag{1}$$

$$\sum_{i \in \mathcal{N}} w_i x_{ij} \leq Q_j y_j \qquad \forall j \in \mathcal{M} \tag{2}$$

$$\sum_{j \in \mathcal{M}} y_j = p \tag{3}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in \mathcal{N}, \quad \forall j \in \mathcal{M}$$

$$y_j \in \{0, 1\} \qquad \forall j \in \mathcal{M}$$

Binary variables x are assignment variables: $x_{ij} = 1$ if and only if vertex i is assigned to a median located in vertex j. Binary variables y correspond to location decisions: $y_j = 1$ if and only if vertex j is selected to be a median. The objective is to minimize the sum of allocation costs. Set partitioning constraints (1) impose that each vertex is assigned to a median. Capacity constraints (2) impose that the sum of the vertex weights in each cluster do not exceed the capacity of the median and forbid the assignment of vertices to unselected medians. To satisfy constraint (3) exactly p medians must be selected.

The linear relaxation of this formulation can be strengthened in two ways: first, including the inequalities $x_{ij} \leq y_j$, $\forall i \in \mathcal{N}$, $j \in \mathcal{M}$, arising from constraints (2); second, by dynamic generation of valid inequalities. The illustration of the test instances we have used is presented in Subsection 2.4. Our computational experience using CPLEX 6.5 as a general purpose solver is reported in Section 4.

2.1. Dantzig-Wolfe Decomposition

Hereafter we derive an alternative formulation of the CPMP amenable to a column generation approach.

Let $(\mathbf{1}, p)$ be the vector of the right-hand side terms of constraints (1) and (3), $d^j = (d_{1j} \dots d_{Nj})$ be the vector of the distances between each vertex and the candidate median $j \in \mathcal{M}, x^j = (x_{1j}, \dots, x_{Nj})$ the vector of assignment variables related to the median in $j \in \mathcal{M}$ and $w = (w_1, \dots, w_N)$ the vector of vertex weights. Consider the CPMP linear relaxation, expressed as follows:

$$\min \sum_{j \in \mathcal{M}} (d^j, 0)^T (x^j, y_j)$$

s.t.

$$\sum_{j \in \mathcal{M}} (x^j, y_j) = (\mathbf{1}, p)$$

$$(x^j, y_j) \in \Omega^j \qquad \forall j \in \mathcal{M}$$

where $\Omega^j = \left\{(x^j, y_j) \in \Re_+^{N+1} \mid x_{ij} \leq 1 \ \forall i \in \mathcal{N}, y_j \leq 1, w^T x^j \leq Q_j y_j\right\}$. The optimal value of this linear relaxation is a lower bound for the CPMP. To improve this bound, we replace each polyhedron Ω^j with the convex hull of its integer points:

$$\min \sum_{j \in \mathcal{M}} (d^j, 0)^T (x^j, y_j)$$

s.t.

$$\sum_{j \in \mathcal{M}} (x^j, y_j) = (\mathbf{1}, p)$$

$$(x^j, y_j) \in \text{conv}(\Omega^j) \qquad \forall \text{j} \in \mathcal{M}$$

Because Ω^j is the polyhedron of the linear relaxation of a binary knapsack problem (see [19] for a classical reference), which is known not to have the integrality property, its extreme points can have fractional coordinates; therefore, the lower bound computed after the convexification of each Ω^j can be stronger than that of the linear relaxation of the CPMP (and our experiments reported in Table 1 show that this is actually the case).

Because the y variables are bounded, and because the weights and the capacities are nonnegative, every polyhedron Ω^j is bounded, and every solution $(x^j, y_j) \in \text{conv}(\Omega^j)$ can be

TABLE 1. Comparison between different lower bounds.

N	p	CPMP (a)	SG (b)	LMP (c)
50	5	1.21%	1.14%	0.99%
	12	4.17%	3.20%	2.88%
	16	3.90%	2.81%	2.39%
	20	4.44%	4.04%	1.73%
100	10	1.26%	1.06%	0.90%
	25	3.84%	3.28%	2.90%
	33	3.43%	2.98%	2.67%
	40	3.84%	3.16%	2.12%
150	15	0.52%	0.58%	0.30%
	37	2.31%	1.93%	1.65%
	50	3.00%	2.54%	2.26%
	60	3.36%	2.85%	2.03%
200	20	0.64%	0.64%	0.44%
	50	4.89%	4.34%	3.93%
	66	5.38%	4.69%	4.32%
	80	5.48%	4.65%	3.22%

obtained as a convex combination of the L_j+1 extreme points of $\text{conv}(\Omega^j)$. We indicate the set of the extreme points with $\{(0,0),(\bar{x}^1,1),\ldots,(\bar{x}^{L_j},1)\}$. Therefore, we have

$$(x^j,y_j)=\sum_{k\in Z^j}(\bar{x}^k,\bar{y}^k)z_k^j,$$

$$\sum_{k\in Z^j}z_k^j=1,\quad z_k^j\in\Re_+\quad\forall k\in Z^j \quad (4)$$

where each $k\in Z^j$ is the index of an extreme point of Ω^j. Substituting expression (4) in the formulation of the linear relaxation of the CPMP we obtain

$$\min\sum_{j\in\mathcal{M}}(d^j,0)^T\sum_{k\in Z^j}(\bar{x}^k,\bar{y}^k)z_k^j$$

s.t.

$$\sum_{j\in\mathcal{M}}\sum_{k\in Z^j}(\bar{x}^k,\bar{y}^k)z_k^j=(\mathbf{1},p)$$

$$\sum_{k\in Z^j}z_k^j=1\qquad\forall j\in\mathcal{M}$$

$$z_k^j\in\Re_+\qquad\forall j\in\mathcal{M}\quad\forall k\in Z^j$$

Therefore, the decomposition and convexification of the linear relaxation of the CPMP yields the following linear master problem (LMP):

$$LMP)\ \min\sum_{j\in\mathcal{M}}\sum_{k\in Z^j}\left(\sum_{i\in\mathcal{N}}d_{ij}x_i^k\right)z_k^j$$

s.t.

$$\sum_{j\in\mathcal{M}}\sum_{k\in Z^j}x_i^kz_k^j\geq 1\qquad\forall i\in\mathcal{N} \quad (5)$$

$$-\sum_{j\in\mathcal{M}}\sum_{k\in Z^j}z_k^j\geq -p \quad (6)$$

$$-\sum_{k\in Z^j}z_k^j\geq -1\qquad\forall j\in\mathcal{M} \quad (7)$$

$$z_k^j\in\Re_+\qquad\forall j\in\mathcal{M},\quad\forall k\in Z^j.$$

Each column of this model corresponds to a feasible cluster, that is, an assignment of vertices to a median, that satisfies the capacity constraint. Each cluster k is described by assignment coefficients x_i^k equal to 1 if and only if vertex $i\in\mathcal{N}$ belongs to cluster $k\in Z^j$ of median $j\in\mathcal{M}$. A binary variable z_k^j is associated with each cluster. Constraints (5) guarantee that each vertex is assigned to at least one median; constraint (6) implies that the total number of selected clusters is at most p; constraints (7) impose that no more than one cluster is associated to the same median.

In model (5)–(7) all equality constraints have been replaced by inequalities, and hereafter we briefly discuss the correctness and usefulness of the substitutions.

Partitioning constraints (1) can be replaced by covering constraints (5) because all distances are nonnegative and therefore it always exists as an optimal solution in which no user is assigned more than once.

Because there is no fixed cost associated to the medians, if an optimal solution exists with $p'<p$ medians, there is also an equivalent solution with p medians, $p-p'$ of which have empty clusters. Therefore, equality constraint (3) can be replaced by the inequality $\sum_{j\in\mathcal{M}}y_j\leq p$. This yields the constraint $\sum_{j\in\mathcal{M}}\sum_{k\in z^j}y^kz_k^j\leq p$ in our reformulation. However, empty clusters affect neither the objective function value nor the satisfaction of constraints (5), and therefore, we can make the assumption that no empty cluster (i.e., a cluster with $y^k=0$) appears in LMP. For this reason the constraint $\sum_{j\in\mathcal{M}}\sum_{k\in z^j}y^kz_k^j\leq p$ can be restated as $\sum_{j\in\mathcal{M}}\sum_{k\in z^j}z_k^j\leq p$, and constraints (7) are stated as inequalities.

Each set Z^j of feasible clusters contains an exponential number of elements. Because LMP cannot be solved directly because of the exponential number of its columns, column generation (see [10]) is applied: a restricted linear master problem (RLMP), defined by a relatively small subset of columns is considered and solved to optimality; then, a search is performed for new columns of negative reduced cost, and if any such column is found, it is inserted into the formulation and the RLMP is solved again. When no columns of negative reduced cost exist, the optimal solution of the RLMP is also optimal for the LMP, and its value is a valid dual bound to be used in a branch-and-bound framework.

When solving LMP by column generation, the set covering formulation above has at least two advantages compared to the equivalent set partitioning formulation: first, the generation of feasible solutions is easier, and therefore the primal simplex algorithm can be always easily initialized with a feasible basis; second, all dual variables are nonnegative, and this restricts the dual space and speeds up the convergence. A drawback is that the LMP may have an optimal fractional solution, in which some user is covered more than once; in this case, some minor modifications to the primal bounding procedure and to the branching scheme have to be made (see Section 3).

2.2. The Pricing Problem

Let $\lambda\in\mathcal{R}_+^N,\eta\in\mathcal{R}_+$ and $\mu\in\mathcal{R}_+^M$ be the vectors of nonnegative dual variables corresponding to constraints (5), (6), and (7), respectively; the reduced cost of column $k\in Z^j$ is

$$\tau^k(\lambda,\eta,\mu)=\sum_{i\in\mathcal{N}}d_{ij}x_i^k-\sum_{i\in\mathcal{N}}\lambda_ix_i^k+\eta+\mu_j$$

To find columns with negative reduced cost, one must solve a pricing problem for each median $j\in\mathcal{M}$:

$$\min\sum_{i\in\mathcal{N}}(d_{ij}-\lambda_i)x_i^k+\eta+\mu_j$$

s.t.

$$\sum_{i\in\mathcal{N}}w_ix_i^k\leq Q_j$$

$$x_i^k\in\{0,1\}\qquad\forall i\in\mathcal{N}$$

and this requires the solution of the following binary knapsack problem:

$$KP_j) \max \tau_j = \sum_{i \in \mathcal{N}} (\lambda_i - d_{ij}) x_i^k$$

s.t.

$$\sum_{i \in \mathcal{N}} w_i x_i^k \leq Q_j$$

$$x_i^k \in \{0,1\} \qquad \forall i \in \mathcal{N}$$

Therefore, the computational effectiveness of our branch-and-price algorithm mainly relies upon that of the algorithms used for the iterated solution of the master problem (i.e., the simplex algorithm) and the binary knapsack subproblem. One of our main motivations for applying the branch-and-price technique to the CPMP is the existence of very effective algorithms and reliable implementations for both linear programming and the binary knapsack problem.

2.3. Lagrangean Relaxation

The lower bound obtained from the LMP can also be obtained through the Lagrangean relaxation of semi-assignment constraints (1) of the CPMP formulation:

$$LR) \min \omega_{LR} = \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{M}} d_{ij} x_{ij} + \sum_{i \in \mathcal{N}} \lambda_i \left(1 - \sum_{j \in \mathcal{M}} x_{ij}\right)$$

s.t.

$$\sum_{i \in \mathcal{N}} w_i x_{ij} \leq Q_j y_j. \qquad \forall j \in \mathcal{M} \tag{8}$$

$$\sum_{j \in \mathcal{M}} y_j = p \tag{9}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in \mathcal{N}, \forall j \in \mathcal{M}$$

$$y_j \in \{0,1\} \qquad \forall j \in \mathcal{M}$$

The Lagrangean multipliers in LR correspond to the dual variables λ in the LMP and the Lagrangean subproblem can be decomposed in the same M binary knapsack problems as the pricing subproblem in the column generation approach (for the equivalence between Dantzig-Wolfe decomposition and Lagrangean relaxation the reader is referred to mathematical programming textbooks like [23]). Therefore, column generation can be used as a method alternative to subgradient optimization to update the Lagrangean multipliers.

2.4. Benchmark Instances and Lower Bound Comparison

We considered a testbed made of four classes of instances, named α, β, γ and δ. Each class consists of forty instances: 20 of them are taken from the literature ([1, 17, 24]) and concern graphs with 50 and 100 vertices and $p = \frac{N}{10}$; the other 20 instances were randomly generated on graphs with cardinality 150 (15 medians) and 200 (20 medians): vertex coordinates were drawn from a uniform distribution between 1 and 100 and vertex demands were drawn from a uniform distribution between 1 and 20. Each d_{ij} coefficient was taken as the Euclidean distance computed from the coordinates of vertices i and j, rounded down to the nearest integer. In random instances all capacities were fixed to 120. In both the random and the literature instances, the set $\mathcal{N}$ of vertices and the set $\mathcal{M}$ of candidate medians coincide; hence, $N = M$. In our experiments we could observe that the behavior of the different algorithms may be significantly influenced by the ratio between p and N. This means that a sound test set for the CPMP cannot be limited to instances where the ratio between p and N is constant. Therefore, the same 40 instances were solved with different number of medians: $p = \frac{N}{10}$ in class α, $p = \lfloor \frac{N}{4} \rfloor$ in class β, $p = \lfloor \frac{N}{3} \rfloor$ in class γ and $p = \lfloor \frac{2N}{5} \rfloor$ in class δ. The overall capacity was preserved in all classes by setting $Q_j = \lceil 12 \frac{N}{p} \rceil \forall j \in \mathcal{M}$. In all instances in our test set the medians have the same capacity and $d_{ij} = 0$ whenever user i and median j coincide.

All tests presented in this article were done on an Intel Pentium IV 1600 MHz PC with 512 MB RAM in Linux environment (RedHat 7.2, kernel 2.4.19). The algorithms were coded in C++, compiled with gcc/g++ version 2.96 with the "−O3" optimization level. We imposed some resource limitation to every test: computation was halted after 1 hour or in case of memory overflow. ILOG CPLEX 6.5 was used as an LP solver in the branch-and-price algorithm.

In Table 1 we compare (a) the bound obtained from the CPMP linear relaxation strengthened by inequalities $x_{ij} \leq y_i$ and tightened with clique, cover and GUB-cover inequalities automatically generated by CPLEX; (b) the bound obtained from Lagrangean relaxation and subgradient optimization (see Section 4.1 for details); (c) the bound obtained from the LMP. As a measure of the quality of these relaxations we indicate the average gap between each lower bound (LB) and the best known upper bound (UB), that it $\frac{UB-LB}{UB}$. The upper bounds we used are often the optimal solution values.

Bound (a) is systematically worse than the others on all classes and all sizes. Lower bound (b) is found by subgradient optimization halted after 300 iterations. Therefore, it does not exactly correspond to lower bound (c), obtained from the LMP, that corresponds to solving the Lagrangean dual to optimality. Lower bound (c) is definitely dominating both (a) and (b); this can be interpreted as a measure of the effectiveness of the convexification of the capacity constraints.

3. A BRANCH-AND-PRICE-ALGORITHM

3.1. Branching

The choice of the branching rule in branch-and-price algorithms is crucial because the addition of constraints may change the structure of the pricing problem. We have tested two branching strategies, outlined hereafter.

3.1.1. Strategy 1: Branching on Binary Variables. This branching strategy is similar to the one used by Pirkul in his Lagrangean-based branch-and-bound algorithm for the capacitated concentrator location problem [25] and to the method by Diaz and Fernández for the single-source capacitated plant location problem [6]. Intuitively, fixing location

variables y has much a stronger effect than fixing assignment variables x: when a location variable is fixed to 0 many assignment variables can be also fixed to 0; moreover, once p location variables have been fixed to 1, the CPMP reduces to a generalized assignment problem. Therefore, it is effective to fix y variables early in the exploration of the search tree.

A candidate median j^* is selected and two possibilities are considered: in one branch all columns k with $k \in Z^{j^*}$ are discarded (that is the candidate median is fixed as "not used"); in the other branch the equality constraint $\sum_{k \in Z^{j^*}} z_k^{j^*} = 1$ is inserted into the RLMP (that is the candidate median is fixed as "used"). In both cases the structure of the pricing problem is not affected. The branching variable j^* is selected among the candidate medians not yet fixed as the one with minimum value of τ_j (that is the most unpromising one) and the branch in which the median is not used is explored first, with a depth-first policy.

When p medians have been fixed as "used" or $N - p$ have been fixed as "not used," branching is performed by fixing variables x, that is assigning the vertices to the selected medians. Assigning vertex i^* to median j^* corresponds to fixing $x_{i^*}^k = 1$ in the pricing subproblem KP_{j^*} and $x_{i^*}^k = 0$ in all $KP_j, j \neq j^*$. The branching vertex i^* is the one with largest number of fractional assignments to different medians in the optimal solution of the LMP (instead, in Pirkul's algorithm the vertex with maximum weight w_i is chosen).

In each successor node of the search tree the branching vertex i^* is assigned to a different median, so that p branches are considered (in random order).

We used this branching strategy in conjunction with depth-first search policy, because this allows to exploit the structure of the LMP (namely, the columns and the optimal basis) of each predecessor node in the search tree to solve the LMP of its first successor without explicitly storing such information for each node. Moreover depth-first search policy quickly produces good feasible solutions, that can be useful to prune the search tree.

3.1.2. Strategy 2: Branching on Semiassignment Constraints. This branching strategy is similar to that used by Savelsbergh is his branch-and-price algorithm for the GAP [28]. For each vertex $i \in \mathcal{N}$ we consider the set $\mathcal{M}_i$ of candidate medians $j \in \mathcal{M}$ for which there is a fractional assignment x_{ij} in the optimal solution of the RLMP. In Savelsbergh's algorithm the set $\mathcal{M}_i$ is partitioned into two subsets $\mathcal{M}_i^- = \{j \in \mathcal{M} : x_{ij} > 0,\ j \leq j^*\}$ and $\mathcal{M}_i^+ = \{j \in \mathcal{M} : x_{ij} > 0,\ j > j^*\}$ by choosing j^* in such a way that $|\mathcal{M}_i^-| = \lceil \frac{|\mathcal{M}_i|}{2} \rceil$. We elaborated on this idea, aiming at a balanced partition: the elements in $\mathcal{M}_i$ are sorted by nonincreasing value of fractional assignment and they are inserted alternately in $\mathcal{M}_i^-$ and in $\mathcal{M}_i^+$. The set of candidate medians to which vertex i is not assigned is also partitioned into two subsets $\widehat{\mathcal{M}}_i^-$ and $\widehat{\mathcal{M}}_i^+$ of balanced cardinality. In [28], the author proposed to choose j^* as close as possible to $\frac{M}{2}$ with the constraint of leaving at least one fractional variable on each side, but we found that our choice of j^* produces better results. The vertex i^* selected for branching is the one for which $|\mathcal{M}_i|$ is maximum. In case of ties we select the vertex for which $\sum_{j \in \mathcal{M}_{i^*}^-} \sum_{k \in Z^j} x_{i^*}^k z_k^j$ is closest to $\frac{1}{2} \sum_{j \in \mathcal{M}_{i^*}} \sum_{k \in Z^j} x_{i^*}^k z_k^j$. Then we branch on the original constraint $\sum_{j \in \mathcal{M}} x_{i^*j} = 1$ by setting $\sum_{j \in \mathcal{M}_{i^*}^- \cup \widehat{\mathcal{M}}_{i^*}^-} x_{i^*j} = 0$ in one branch and $\sum_{j \in \mathcal{M}_{i^*}^+ \cup \widehat{\mathcal{M}}_{i^*}^+} x_{i^*j} = 0$ in the other. The addition of these constraints does not change the structure of the knapsack problems; it only reduces their size.

Our experiments showed that branching strategy 2 is definitely more effective than strategy 1: only 57 of the 160 instances considered were solved to proven optimality with strategy 1, while 92 instances were solved with strategy 2. Moreover, on the instances solved with both methods, the algorithm with strategy 2 was faster. For this reason the computational results reported in Tables 3 to 6 are referred to strategy 2.

3.2. Primal Bound

The problem of finding a feasible solution to the CPMP is $\mathcal{NP}$-complete: the reduction from Partition [15] is the same as for the GAP (see [19]). To compute approximate solutions to the CPMP we modified the MTHG algorithm [18], that was devised by Martello and Toth for the GAP. Once defined suitable coefficients f_{ij} as a measure of the desirability of assigning each job i to each machine j, the MTHG algorithm goes through two steps. In the first step all jobs are sorted in decreasing order of a regret value and then they are assigned one at a time to their "most desired" machine. The regret is defined as the difference between the coefficient $f_{ij'}$ referred to the most desirable machine and the coefficient $f_{ij''}$ referred to the second most desirable one. In the second step, provided that all jobs have been assigned without exceeding capacities, the solution is improved by single-job exchanges in a local search fashion. If some job remains unassigned the algorithm fails.

Jörnsten and Näsberg [13] proposed a similar algorithm in which all jobs are assigned even if capacity constraints are violated. Afterwards, local search steps are executed to possibly achieve feasibility, followed by further local search steps to improve the solution.

In our algorithm we first choose p candidate medians (step 1); then the vertices are assigned to the medians in the same way as in the MTHG algorithm, that is without exceeding capacities (step 2); if some vertex remains unassigned, local search iterations are performed to produce a feasible solution (step 3); if this step succeeds, a final local search tries to improve the solution (step 4). A synthetic description of each step follows; the complete pseudocode is reported in the appendix.

STEP 1. *Medians selection.* If a CPMP instance with non-uniform capacities is feasible, obviously it has at least one feasible solution in which the p most capable vertices are the medians. However, in the CPMP instances considered in the literature all candidate medians have the same capacity. In this

case, following Savelsbergh [28], the desirability coefficients have been defined as

$$f_{ij} = \sum_{k \in Z^j} x_i^k z_k^j \quad \forall i \in \mathcal{N} \quad \forall j \in \mathcal{M}$$

to produce an integer solution similar to the optimal solution of the RLMP. Then the p vertices with the highest values of ψ_j are chosen as medians, where

$$\psi_j = \sum_{i \in \mathcal{N}} f_{ij} \quad \forall j \in \mathcal{M}$$

STEP 2. *Direct assignment.* In step 2 our algorithm is identical to that of Martello and Toth: the vertices are assigned to the selected medians in decreasing order of the regret value. Step 2 terminates as soon as a vertex is encountered, which cannot be assigned to any median.

STEP 3. *Assignment through exchanges.* In this step all best-fit l-exchanges are evaluated: an unassigned vertex i replaces a vertex k of smaller weight in a cluster C_j whenever the capacity constraint allows for such an exchange. The replaced element k is immediately inserted in another cluster if possible, as in step 2; otherwise it remains unassigned. The algorithm chooses i, j, and k so that the residual capacity of C_j is minimized.

STEP 4. *Solution improvement.* Two different neighborhoods are explored in this final local search step. At each iteration one element is shifted from a cluster to another or two elements belonging to different clusters are swapped.

At the root node the evaluation of the primal bound is done after each column generation iteration; in all the other nodes of the search tree it is done at the end of the column generation routine.

3.3. Columns Management

At each iteration the algorithm inserts into the RLMP all columns with negative reduced cost that have been found. After the termination of the column generation algorithm, it removes from the RLMP all columns whose reduced cost is higher than a threshold, that is a function of the gap between the best incumbent primal solution and the optimal solution of the LMP. In particular, we set the threshold equal to the ratio between the primal-dual gap and the number of medians p. The removal is also performed during the execution of the column generation algorithm every time the number of columns exceeds a limit (set to 3000 in our experiments). In this case, the dual bound of the father node is used in the computation of the threshold.

In general there is no guarantee that the columns removed from the RLMP in one node of the search tree will not belong to the optimal solution of another node. Therefore, it is useful to store removed columns in a pool and to scan it before running the pricing algorithm. Because every column is related to a particular candidate median, the pool is partitioned into M subsets. If any column with negative reduced cost is found in the pool, it is inserted into the RLMP.

To keep the pool size limited, the columns are erased from the pool when their reduced cost is nonnegative for a certain number of consecutive evaluations (three in our experiments).

To achieve feasibility the RLMP is initialized at each node with a dummy column of cost $\sum_{i \in \mathcal{N}} \max_{j \in \mathcal{M}} \{d_{ij}\}$, corresponding to an infeasible cluster, that covers all vertices and uses none of the free medians (those whose corresponding y variable has not been fixed). The structure of the dummy column is the following: all its coefficients in constraints (5) are set to 1; the coefficient in constraint (6) is set to 0; the coefficients in constraints (7) are set to -1 if the corresponding y variable has been fixed to 1, and they are set to 0 otherwise.

To improve the performance of the column generation algorithm at the root node, we also generate 20 initial solutions with the same primal heuristic illustrated above. Ten solutions are computed using $f_{ij} = -d_{ij}$ and 10 using $f_{ij} = 1/d_{ij}$ for $i \neq j$ and setting f_{ii} to a very large value. In both cases the set of medians in step 1 is chosen at random with uniform probability distribution. The columns generated in this way are inserted even if the corresponding solutions are infeasible because of violations of the partitioning constraints.

In each node of the search tree the RLMP is initialized with the columns of the most recently solved node plus some additional columns taken from the pool with the following procedure: for every node of the search tree we store the optimal values of the dual variables; for each successor node we use the dual values of its predecessor to compute the reduced costs of all columns in the pool; if any column is found with negative reduced cost, it is added to the initial formulation of the RLMP in the successor node.

3.4. Lower Bound and Termination

The equivalence between Dantzig-Wolfe decomposition and Lagrangean relaxation is exploited both for variable fixing purposes and in the termination test.

At each iteration t of column generation the current values of the dual variables λ^t are used as Lagrangean multipliers to compute a valid lower bound:

$$\omega_{LR}^t = -\sum_{j \in \mathcal{M}^{LR}} \tau_j^t + \sum_{i \in \mathcal{N}} \lambda_i^t$$

where $\mathcal{M}^{LR} \subseteq \mathcal{M}$ is the set of vertices with the p maximum values of τ_j^t. In this way a sequence of valid lower bounds is computed during column generation, and this allows to fix variables or even to prune the current node of the search tree before column generation is over.

When the gap between the optimal value of the RLMP at iteration t and the best incumbent Lagrangean lower bound $\omega_{LR}^{*t} = \max\{\omega_{LR}^1, \ldots, \omega_{LR}^t\}$ is smaller than a predefined threshold (set to 10^{-4} in our tests), the column generation algorithm is terminated and ω_{LR}^{*t} is kept as the final lower bound. This allows to reduce the typical and undesired tailing-off effect in the column generation routine. This technique is not used at the root node, where column generation is

iterated until no more columns with negative reduced cost can be identified. In fact, the set of dual values obtained during the last column generation iterations can be useful to perform effective variable fixing tests; moreover, the structure of quasioptimal LMP solutions can be exploited by the primal heuristic to obtain tight bounds.

Experimental results show that this termination criterion reduces the computation time and the number of column generation iterations in nonroot nodes. We made a comparison between the performances of our algorithm with and without the use of the Lagrangean lower bound on the set of instances with $N = 50$ of the four classes and we observed a reduction of about 8.5% in the overall CPU time required and a reduction of about 2.5% in the average number of column generation iterations needed in each nonroot node of the search tree.

3.4.1. Subgradient Optimization. It is also possible to better exploit the relationship between column generation and Lagrangean relaxation outlined above to improve the dual variables via subgradient optimization [12] after each column generation iteration. Starting with the current optimal values of the dual variables λ^t, 100 subgradient iterations are executed. The step parameter is initialized to 2 and halved after every 10 iterations in which $\omega_{LR}^t \leq \omega_{LR}^{t-1}$, that is the current lower bound has not been improved with respect to the previous iteration.

At each subgradient iteration we also compute a primal solution using a Lagrangean heuristic: medians are chosen according to the values of the Lagrangean penalties; when partitioning constraints are not violated, we use the same assignments that appear in the solution of the RLMP; the allocation of the other vertices is done as in the MTHG algorithm, with desirability coefficients $f_{ij} = -d_{ij}$; then a local search step is performed, as proposed by Mulvey and Beck [20]: after every primal bound evaluation, the optimal median for every cluster is recomputed and a new primal bound is evaluated with the new set of medians; this process is iterated until no more changes in the medians occur or a limit of five iterations is reached.

This combination of column generation with subgradient optimization yielded significant improvements at the root node, in terms of reduced number of column generation iterations, reduced computational time to achieve convergence, increased number of variables fixed and quality of the primal bound. However, this improvement did not reduce either the size of the search tree or the time required by the algorithm to complete the enumeration. Furthermore, this method was too time-consuming to be used at each node of search tree.

3.4.2. Variable Fixing. Given a solution of the Lagrangean relaxation LR, let ω_{LR} be its value and let $j^{WI} \in \operatorname{argmin}_{j \in \mathcal{M}}^{LR}\{\tau_j\}$ be the vertex with minimum τ_j value which is median and $j^{BO} \in \operatorname{argmax}_{j \notin \mathcal{M}^{LR}}\{\tau_j\}$ be the vertex with maximum τ_j, value which is not a median (WI stands for "worst in," BO for "best out"). Let also υ^* be a primal bound. If it does exist $j \in \mathcal{M}^{LR}$ such that $\lceil \omega_{LR} + \tau_j - \tau_{j^{BO}} \rceil \geq \upsilon^*$, then y_j can be fixed to 1. Analogously, if it does exist $j \notin \mathcal{M}^{LR}$ such that $\lceil \omega_{LR} - \tau_j + \tau_{j^{WI}} \rceil \geq \upsilon^*$, then y_j can be fixed to 0 (this also implies $x_{ij} = 0 \; \forall i \in \mathcal{N}$).

Once the τ_j values have been computed, this variable fixing step takes $O(M)$ time and it may reduce the problem size considerably. In our experiments variable fixing was done at each iteration of the subgradient optimization algorithm at the root node, and only at the end of column generation at the other nodes in the search tree.

The effectiveness of variable fixing test at the root node can be appreciated from the computational results reported in Tables 3 to 6 in columns fix.med.

3.5. Pricing Algorithm

In the pricing step we solve binary knapsack problem instances to optimality by a modified version of Pisinger's MINKNAP algorithm [26], that combines dynamic programming with bounding and reduction techinques to dynamically adjust the core. Yet MINKNAP was devised for knapsack problems with integer coefficients, while in our pricing subproblems the dual variables, as well as the multipliers in Lagrangean relaxation, can be fractional. Instead of scaling the coefficients, that implies the computation of the determinant of a matrix and can be very time-consuming, we relaxed the bounding test so that the optimal solution computed by the modified algorithm may differ from optimality by at most $N_{KP}\varepsilon$, where N_{KP} is the number of variables left outside the core and ε is a very small positive parameter. Although this technique gives slightly worse dual bounds, we found that it has a clear computational advantage compared to the scaling approach. In particular, we considered values of ε in the range between 10^{-6} and 10^{-12}: for values higher than 10^{-6} the approximation is not tight; for values lower than 10^{-12} numerical problems may arise; in between these values we obtained tight approximations, and we observed that the computation time does not change significantly.

In general, it is also possible to generate columns with negative reduced cost by solving the pricing problem approximately. This is usually done when the pricing problem is difficult to solve to optimality. However, our computational experience showed that this is not worth doing in this case, because the time required to solve the binary knapsack is negligible compared with that needed to solve the LMP.

Column generation can be also speeded up by multiple pricing: instead of inserting into the RLMP only the optimal column for each candidate median, if any is found with negative reduced cost, it is worth adding more (suboptimal) columns to enlarge the search space for the linear programming algorithm. This is particularly useful at the root node, when the column pool is still empty and the set of available columns may be small.

To this purpose we exploit the subgradient optimization algorithm and we insert into the RLMP the set of columns corresponding to each solution of the LR for which the Lagrangean lower bound improves upon the best incumbent Lagrangean lower bound.

Another method we devised to obtain promising columns consists of applying local exchanges to the columns found at each column generation iteration. For each candidate median, we consider the optimal cluster, that is the cluster described by the column with the minimum reduced cost, and we generate all feasible clusters obtained by the exchange of one vertex in the cluster with one vertex outside the cluster. All columns generated in this way are added to the RLMP, provided that their reduced cost is negative and that it is no more than 10% far away from the optimal one.

In Table 2, we present the computational results when (a) multiple pricing is not used, (b) multiple columns are generated via local exchanges, and (c) multiple columns are generated via subgradient optimization. For each policy we report the number of iterations of the column generation algorithm (CGRoot), the number of columns generated (ColsRoot) and the time spent (TimeRoot) at the root node. In the same way for the search tree nodes we report the average number of column generation iterations (CGTree), the number of columns generated (ColsTree) and the time spent to complete the enumeration process (TimeTree). Finally, we report the number of nodes explored (Eval. Nodes).

In case (c) the number of column generation iterations and the time spent at the root node are inferior. Moreover, the CPU time reported in case (c) includes the time spent for the Lagrangean heuristic and the variable fixing routine. However, in case (c), we often observed an increase in the computation time required to complete the overall enumeration process. This can be explained by at least two reasons: first, when the optimum of the LMP is degenerate, the solution obtained in case (c) has slightly more fractional variables; second, when the algorithm generates less columns at the root node, the pool is scarcely useful and more columns must be generated later during the exploration of other nodes.

3.6. Experimental Results

We report in Tables 3, 4, 5, and 6 detailed results on the computational behavior of the branch-and-price algorithm when subgradient-based multiple pricing is used. These tables, one for each class of instances, are composed by two horizontal blocks: the first block contains experimental results for the root node, while the second block refers to the whole search tree.

TABLE 2. Comparison between different multiple pricing methods.

N	p	CGRoot	ColsRoot	TimeRoot (s)	CG Tree	ColsTree	TimeTree (s)	Eval. nodes
(a) No multiple pricing								
50	5	107.3	4566.0	1.38	4.27	51.02	207.05	1304.6
	12	77.7	3589.4	0.43	3.76	58.01	93.10	2770.0
	16	73.7	3422.1	0.33	3.64	60.77	19.17	13,707.7
	20	71.4	3331.3	0.30	2.84	47.69	35.92	1727.0
100	10	168.8	15,166.6	12.83	10.98	259.55	214.56	1336.5
	25	137.9	13,099.0	4.68	6.59	222.57	622.46	19,984.2
	33	132.6	12,680.8	3.93	4.83	175.83	505.28	21,048.5
	40	130.4	12,527.7	3.46	3.50	128.70	619.98	20,357.2
Average		112.48	8547.86	3.42	5.05	125.52	289.69	10,279.46
(b) Multiple pricing via local search								
50	5	80.7	25,458.8	10.04	5.48	67.90	228.75	1288.0
	12	64.5	10,921.6	1.30	4.71	75.25	83.31	2318.0
	16	62.9	8684.4	0.79	3.61	62.66	16.46	13,230.3
	20	61.4	7067.6	0.49	2.55	44.80	50.91	2324.6
100	10	133.5	129,155.0	146.71	10.08	237.93	418.48	1187.3
	25	117.3	50,237.6	15.05	6.35	214.95	510.33	18,263.3
	33	116.0	39,770.0	9.31	5.31	194.48	328.28	18,751.4
	40	114.3	33,000.8	6.32	3.71	138.41	436.83	17,172.5
Average		93.83	38,036.98	23.75	5.23	129.55	259.17	9316.93
(c) Multiple pricing via subgradient optimization								
50	5	8.9	3526.7	0.77	12.65	109.90	189.11	1204.6
	12	7.0	2139.1	0.55	6.93	95.97	90.45	2495.6
	16	6.3	2274.6	0.51	5.34	80.00	15.46	13,450.5
	20	5.4	2341.6	0.51	4.06	62.68	49.40	2188.2
100	10	12.8	8299.3	3.53	20.76	416.74	543.60	1248.7
	25	7.2	4887.1	2.46	8.16	258.21	115.67	18,858.7
	33	6.6	5222.8	2.33	6.56	235.11	467.95	18,065.1
	40	5.3	4612.3	1.83	5.56	199.96	795.43	17,850.5
Average		7.44	4162.94	1.56	8.75	182.32	283.39	9420.24

TABLE 3. Branch-and-price with subgradient-based multiple pricing—Class α.

	Instance		Root									Search tree							
N	p		CG it.	cols	UB	LB	gap	UB gap	LB gap	time (s)	fix.med.	CG it.	cols	F.UB	F.LB	gap	time (s)	avg fix.med.	ev. nodes
50	5	cpmp01	13	2706	713	705	1.13%	0.00%	1.12%	0.60	28	16.00	139.84	713	713	0.00%	4.58	2.73	33
		cpmp02	3	896	740	740	0.00%	0.00%	0.00%	0.15	0	0.00	0.00	740	740	0.00%	0.16	0.00	1
		cpmp03	13	2755	751	749	0.27%	0.00%	0.27%	0.66	40	40.25	254.25	751	751	0.00%	2.43	0.00	5
		cpmp04	4	1571	651	651	0.00%	0.00%	0.00%	0.28	0	0.00	0.00	651	651	0.00%	0.31	0.00	1
		cpmp05	5	1441	664	664	0.00%	0.00%	0.00%	0.54	0	0.00	0.00	664	664	0.00%	0.56	0.00	1
		cpmp06	3	1598	778	778	0.00%	0.00%	0.00%	0.30	0	0.00	0.00	778	778	0.00%	0.33	0.00	1
		cpmp07	10	2650	787	779	1.03%	0.00%	1.11%	0.59	22	14.00	167.00	787	787	0.00%	7.13	3.06	37
		cpmp08	11	2835	820	772	6.22%	0.00%	5.89%	0.86	2	20.63	210.05	820	820	0.00%	1904.73	3.61	11965
		cpmp09	10	2575	715	713	0.28%	0.00%	0.34%	0.51	30	10.00	74.00	715	715	0.00%	1.00	4.00	5
		cpmp10	9	1829	829	818	1.34%	0.00%	1.34%	0.78	17	10.57	97.99	829	829	0.00%	7.96	3.26	109
		Average	8.1	2085.6			1.03%	0.00%	1.01%	0.53	13.9	11.14	94.31			0.00%	192.92	1.67	1215.8
100	10	cpmp11	11	5573	1007	1002	0.50%	0.10%	0.49%	2.16	40	19.50	414.58	1006	1006	0.00%	19.08	10.08	25
		cpmp12	11	4896	969	959	1.04%	0.31%	0.78%	1.73	20	17.04	358.63	966	966	0.00%	93.51	7.13	211
		cpmp13	13	5568	1026	1022	0.39%	0.00%	0.43%	2.94	66	38.86	809.07	1026	1026	0.00%	30.46	5.00	15
		cpmp14	11	4880	985	972	1.34%	0.31%	1.04%	2.15	24	17.06	351.92	982	982	0.00%	232.13	6.47	425
		cpmp15	12	5623	1092	1081	1.02%	0.09%	0.97%	2.55	28	17.86	311.71	1091	1091	0.00%	407.73	5.21	721
		cpmp16	8	3940	955	952	0.32%	0.10%	0.28%	1.55	60	16.70	266.50	954	954	0.00%	5.70	4.14	11
		cpmp17	13	4603	1034	1026	0.78%	0.00%	0.84%	2.26	33	17.06	249.75	1034	1034	0.00%	113.91	5.03	283
		cpmp18	10	5053	1046	1032	1.36%	0.29%	1.06%	1.96	25	21.25	476.07	1043	1043	0.00%	877.95	6.35	1181
		cpmp19	13	5567	1036	1027	0.88%	0.48%	0.46%	2.66	27	13.57	329.41	1031	1031	0.00%	18.03	5.18	45
		cpmp20	11	4906	1022	974	4.93%	1.59%	3.22%	2.83	0	19.04	474.55	1006	993	1.31%	—	8.87	9128
		Average	11.3	5060.9			1.25%	0.33%	0.96%	2.28	32.3	19.79	404.22			0.13%	199.83	6.35	1204.5
150	15	cpmp21	10	6576	1294	1282	0.94%	0.47%	0.48%	4.02	26	14.97	526.78	1288	1288	0.00%	81.86	8.36	105
		cpmp22	12	8969	1256	1248	0.64%	0.00%	0.65%	6.14	39	20.13	505.33	1256	1255	0.08%	—	6.00	3182
		cpmp23	11	7898	1279	1278	0.08%	0.00%	0.09%	5.29	90	8.50	230.00	1279	1279	0.00%	6.89	0.00	3
		cpmp24	11	6461	1220	1219	0.08%	0.00%	0.13%	5.22	100	22.50	536.00	1220	1220	0.00%	8.35	0.00	3
		cpmp25	10	6135	1193	1189	0.34%	0.00%	0.35%	5.15	68	29.81	918.35	1193	1193	0.00%	125.71	6.60	53
		cpmp26	11	7240	1269	1259	0.79%	0.40%	0.41%	5.67	20	16.63	641.36	1264	1264	0.00%	184.92	16.11	121
		cpmp27	10	7838	1330	1312	1.37%	0.53%	0.85%	5.54	15	17.58	599.31	1323	1320	0.23%	—	12.26	3139
		cpmp28	11	7200	1237	1231	0.49%	0.32%	0.20%	4.65	45	26.70	1030.20	1233	1233	0.00%	20.72	19.75	11
		cpmp29	5	5850	1219	1219	0.00%	0.00%	0.00%	7.36	0	0.00	0.00	1219	1219	0.00%	7.56	0.00	1
		cpmp30	13	8536	1205	1201	0.33%	0.33%	0.06%	5.41	53	61.50	4413.75	1201	1201	0.00%	34.76	12.75	5
		Average	10.4	7270.3			0.51%	0.20%	0.32%	5.45	45.6	21.83	940.11			0.03%	58.85	8.18	662.3
200	20	cpmp31	11	9913	1379	1373	0.44%	0.07%	0.43%	10.20	49	31.01	1847.13	1378	1378	0.00%	1003.81	13.10	223
		cpmp32	12	8687	1429	1410	1.35%	0.00%	1.34%	12.48	12	24.16	1803.95	1429	1419	0.70%	—	7.82	1149
		cpmp33	15	11,154	1383	1362	1.54%	1.17%	0.38%	11.66	6	23.01	1659.16	1367	1367	0.00%	1420.90	21.10	557
		cpmp34	12	11,092	1385	1375	0.73%	0.00%	0.75%	13.34	44	25.67	1046.43	1385	1383	0.14%	—	11.76	1425
		cpmp35	10	8917	1442	1431	0.77%	0.35%	0.44%	8.75	23	19.77	920.21	1437	1437	0.00%	2662.5	16.06	1265
		cpmp36	11	9638	1385	1379	0.44%	0.22%	0.27%	8.41	53	26.77	1611.16	1382	1382	0.00%	188.59	11.21	65
		cpmp37	11	10,360	1458	1455	0.21%	0.00%	0.26%	7.90	84	32.94	1730.03	1458	1458	0.00%	109.22	14.73	33
		cpmp38	12	9379	1400	1373	1.97%	1.01%	0.99%	10.13	0	26.52	2195.10	1386	1381	0.36%	—	21.93	1100
		cpmp39	11	8568	1389	1370	1.39%	1.09%	0.31%	7.98	9	26.45	1998.51	1374	1374	0.00%	222.67	26.75	81
		cpmp40	10	8967	1432	1414	1.27%	1.13%	0.20%	8.50	3	18.79	1351.40	1416	1416	0.00%	167.56	19.45	87
		Average	11.5	9667.5			1.01%	0.50%	0.54%	9.93	28.3	25.51	1616.31			0.12%	825.04	16.39	598.5
		Overall average	10.33	6021.08			95%	0.26%	0.71%	4.55		19.57	763.74			0.07%	319.16		920.28

For the root node we report the number of column generation iterations needed to reach optimality (CG it.), the number of columns generated (cols), the best upper bound found (UB), the value of the LMP relaxation (LB), the gap between the upper and lower bounds, that is $\frac{UB-LB}{LB}$ (gap). Moreover, we report the gap between the upper bound at the root node and the best solution found during the exploration of the search tree $\left(\text{UB gap} = \frac{UB-F.UB}{F.UB}\right)$ and the gap between the initial LMP relaxation and the best solution found $\left(\text{LB gap} = \frac{F.UB-LB}{F.UB}\right)$. The former evaluates the quality of the heuristic solution found by column generation, the latter the quality of the LMP relaxation. Finally, we include the CPU time spent (time) and the number of candidate medians fixed to 0 by the variable fixing tests (fix.med.) at the root node.

For the search tree we report the average number of iterations of the column generation algorithm (CG it.) and the average number of generated columns in each node of the search tree (cols), the final upper and lower bounds (F.UB and F.LB), the approximation obtained (gap), the CPU time spent in the optimization (the tests for which computation exceeded time limit are marked with a dash), the average number of variable fixing tests succeeded in each node of the search tree (avg.fix.med.) and the number of nodes evaluated in the search tree (ev. nodes).

The number of column generation iterations and the number of generated columns at the root node decrease as the ratio $\frac{P}{N}$ increases. This is especially true moving from class α to class β. We explained this phenomenon with the following observation: when the ratio is high, there are several medians

TABLE 4. Branch-and-price with subgradient-based multiple pricing—Class β.

Instance			Root									Search tree							
N	p		CG it.	cols	UB	LB	gap	UB gap	LB gap	time (s)	fix.med.	CG it.	cols	F.UB	F.LB	gap	time (s)	avg.fix.med	ev. nodes
50	12	cpmp01	4	1285	387	374	3.48%	1.04%	2.52%	0.15	1	7.05	94.36	383	383	0.00%	5.75	5.25	165
		cpmp02	6	1361	420	409	2.69%	1.94%	0.92%	0.26	1	7.75	142.25	412	412	0.00%	1.86	2.57	37
		cpmp03	5	1201	405	399	1.50%	0.00%	1.67%	0.24	3	6.76	90.20	405	405	0.00%	4.38	4.46	107
		cpmp04	6	1618	385	365	5.48%	0.26%	4.97%	0.33	0	8.08	102.44	384	384	0.00%	308.15	4.21	8075
		cpmp05	4	880	429	419	2.39%	0.00%	2.38%	0.21	4	6.47	81.06	429	429	0.00%	11.19	4.61	359
		cpmp06	5	1302	485	467	3.85%	0.62%	3.28%	0.28	1	7.97	118.55	482	482	0.00%	73.69	3.70	2633
		cpmp07	5	1235	445	425	4.71%	0.00%	4.64%	0.33	0	7.74	97.50	445	445	0.00%	318.60	3.61	8431
		cpmp08	6	1199	407	393	3.56%	0.99%	2.55%	0.34	2	6.70	100.49	403	403	0.00%	18.82	4.09	657
		cpmp09	5	1227	452	423	6.86%	3.67%	3.11%	0.41	0	6.83	94.47	436	436	0.00%	23.68	5.42	623
		cpmp10	5	1309	466	443	5.19%	1.08%	4.09%	0.41	1	6.51	93.69	461	461	0.00%	113.35	4.44	3607
		Average	5.1	1261.7			3.97%	0.96%	3.01%	0.30	1.3	7.19	101.50			0.00%	87.95	4.24	2469.4
100	25	cpmp11	6	3101	549	529	3.78%	0.92%	2.77%	1.01	0	7.73	235.57	544	544	0.00%	2963.49	8.58	25,757
		cpmp12	5	2190	508	496	2.42%	0.79%	1.59%	0.80	2	6.94	169.11	504	504	0.00%	99.66	8.80	1017
		cpmp13	5	2392	569	535	6.36%	2.15%	4.12%	0.76	0	8.56	337.38	557	546	2.01%	—	2.25	32,375
		cpmp14	5	2938	556	530	4.91%	2.21%	2.67%	0.79	0	9.81	287.92	544	541	0.55%	—	6.91	28,074
		cpmp15	5	2611	586	573	2.27%	0.51%	1.77%	0.90	3	7.56	196.19	583	583	0.00%	231.80	9.04	1707
		cpmp16	6	2566	543	521	4.22%	0.93%	3.32%	1.08	0	8.25	299.15	538	530	1.51%	—	3.20	36,101
		cpmp17	5	2352	551	536	2.80%	1.66%	1.14%	0.96	1	6.37	154.73	542	542	0.00%	6.60	9.17	53
		cpmp18	8	4408	508	501	1.40%	0.00%	1.38%	1.33	10	10.89	342.38	508	508	0.00%	47.78	7.88	179
		cpmp19	5	2879	562	531	5.84%	2.00%	3.69%	0.85	0	8.60	320.10	551	547	0.73%	—	7.75	31,511
		cpmp20	4	1930	581	523	11.09%	1.57%	8.63%	1.04	0	7.98	300.49	572	537	6.52%	—	0.06	32,135
		Average	5.4	2736.7			4.51%	1.27%	3.11%	0.95	1.6	8.27	264.30			1.13%	669.87	6.36	18,890.9
150	37	cpmp21	6	4793	682	675	1.04%	0.15%	0.98%	2.01	4	10.36	380.63	681	681	0.00%	838.88	11.00	2685
		cpmp22	5	2703	680	646	5.26%	1.80%	3.40%	2.21	0	8.43	463.76	668	654	2.14%	—	0.94	13,447
		cpmp23	7	4599	682	643	6.07%	1.79%	4.17%	2.88	0	8.81	486.25	670	652	2.76%	—	0.31	12,794
		cpmp24	6	4884	597	587	1.70%	0.51%	1.21%	2.01	2	8.83	361.28	594	594	0.00%	2365.36	11.56	8527
		cpmp25	6	3893	636	628	1.27%	1.11%	0.29%	1.63	0	6.41	338.03	629	629	0.00%	24.07	7.50	103
		cpmp26	6	4953	680	647	5.10%	4.13%	1.06%	1.86	0	8.66	392.74	653	653	0.00%	188.63	13.15	599
		cpmp27	6	4138	771	714	7.98%	2.12%	5.51%	2.70	0	7.93	435.58	755	723	4.43%	—	0.00	13,355
		cpmp28	5	3785	654	633	3.32%	1.55%	1.84%	1.58	0	8.75	472.12	644	641	0.47%	—	10.97	14,962
		cpmp29	6	5180	670	641	4.52%	3.24%	1.29%	1.98	0	7.09	309.75	649	649	0.00%	472.24	17.00	2135
		cpmp30	7	5092	634	619	2.42%	0.48%	1.90%	2.24	0	8.77	418.25	631	628	0.48%	—	9.79	17,376
		Average	6.0	4402.0			3.87%	1.69%	2.17%	2.11	0.6	8.40	405.84			1.03%	777.84	8.22	8598.3
200	50	cpmp31	5	5421	748	712	5.06%	1.63%	3.29%	3.58	0	7.83	550.64	736	718	2.51%	—	0.10	7376
		cpmp32	5	4647	908	800	13.50%	3.42%	8.97%	4.42	0	6.34	346.10	878	806	8.93%	—	0.00	6050
		cpmp33	6	5397	726	694	4.61%	0.14%	4.32%	4.61	0	8.82	642.66	725	701	3.42%	—	0.00	7024
		cpmp34	6	4867	854	779	9.63%	0.00%	8.81%	5.64	0	7.40	402.77	854	785	8.79%	—	0.00	6187
		cpmp35	6	6955	763	728	4.81%	0.66%	4.07%	4.26	0	7.79	539.26	758	733	3.41%	—	0.00	7504
		cpmp36	7	7652	708	693	2.16%	0.71%	1.53%	4.96	0	8.13	586.80	703	697	0.86%	—	3.77	7682
		cpmp37	5	5700	785	744	5.51%	3.97%	1.54%	2.89	0	8.72	655.21	755	749	0.80%	—	3.77	7898
		cpmp38	5	4365	775	730	6.16%	0.00%	5.81%	4.20	0	7.89	531.62	775	735	5.44%	—	0.00	6134
		cpmp39	7	7417	736	713	3.23%	1.80%	1.44%	4.10	0	8.87	595.13	723	720	0.42%	—	13.79	6916
		cpmp40	6	6377	796	738	7.86%	0.38%	6.99%	5.74	0	7.83	454.32	793	745	6.44%	—	0.00	6192
		Average	5.8	5879.8			6.25%	1.27%	4.68%	4.44	0.0	7.96	530.45			4.10%	—	2.14	6896.3
		Overall average	5.58	3570.05			4.65%	1.30%	3.24%	1.95		7.96	325.52			1.57%	511.88		9213.73

with similar allocation pattern. The reduced cost of the corresponding columns is almost the same, and many columns are inserted into the RLMP at each column generation iteration. On the opposite, the performances of the primal heuristic worsen as the ratio increases, as it is harder to find an optimal set of medians. Instances of class γ are the hardest to solve for the branch-and price algorithm: the average gap between the upper and lower bounds after 1 hour of computation and the number of unsolved instances are higher than for the other classes. A similar behavior has been observed for the uncapacitated version of the problem (see [4]). Finally, we ran our algorithm without time limit. It was able to solve to optimality 15 more instances, and to reduce the average gap between the best known upper and lower bounds below1% before running out of memory.

3.6.1. Large Scale Instances. We tried to use our algorithm on even larger instances: to obtain benchmark instances similar to those presented before, we used the instances pmed-38, pmed-39, and pmed-40 for the uncapacitated p-median problem from the OR Library, that involve 900 vertices, fixing the number of medians to 90. We generated random weights as described in the previous section, and set the capacity of each candidate median to 120. Our algorithm solved the root problem in less than 40 minutes, producing solutions whose primal-dual gap was less than 2.6%, 2.4%, and 3.0%. No significant improvement was observed in these bounds after some hours of computation. However this was expected: no CPMP instance of this dimension has been solved so far, although larger instances have been solved for the uncapacitated version of the problem. This

TABLE 5. Branch-and-price with subgradient-based multiple pricing—Class γ.

Instance			Root									Search tree							
N	p		CG it.	cols	UB	LB	gap	UB gap	LB gap	time (s)	fix.med.	CG it.	cols	F.UB	F.LB	gap	time (s)	avg.fix.med	ev. nodes
50	16	cpmp01	4	1537	305	296	3.04%	2.35%	0.67%	0.16	2	4.67	82.50	298	298	0.00%	0.42	1.00	7
		cpmp02	3	814	338	328	3.05%	0.60%	2.38%	0.13	2	5.19	85.08	336	336	0.00%	3.51	2.92	143
		cpmp03	4	1412	317	309	2.59%	0.96%	1.59%	0.15	0	4.80	79.50	314	314	0.00%	0.58	4.40	11
		cpmp04	4	1280	303	297	2.02%	0.00%	2.29%	0.21	5	4.93	64.51	303	303	0.00%	2.23	2.91	71
		cpmp05	5	1555	351	346	1.45%	0.00%	1.42%	0.25	2	5.06	79.44	351	351	0.00%	0.94	4.38	19
		cpmp06	4	1019	403	387	4.13%	3.33%	0.90%	0.25	0	5.25	93.19	390	390	0.00%	0.75	1.00	17
		cpmp07	5	1747	362	357	1.40%	0.28%	1.11%	0.30	5	4.36	69.29	361	361	0.00%	3.19	3.17	95
		cpmp08	5	1237	363	323	12.38%	2.54%	8.86%	0.34	0	7.64	146.61	354	349	1.43%	—	4.87	127066
		cpmp09	4	1010	387	366	5.74%	3.75%	1.88%	0.27	0	4.16	62.05	373	373	0.00%	9.34	4.08	407
		cpmp10	4	1042	399	376	6.12%	2.31%	3.79%	0.32	0	5.93	84.96	390	390	0.00%	143.78	2.98	5137
		Average	4.2	1265.3			4.19%	1.61%	2.49%	0.24	1.6	5.20	84.71			0.14%	18.30	3.17	13,297.3
100	33	cpmp11	4	2165	420	406	3.45%	1.45%	1.97%	0.56	0	6.27	217.97	414	414	0.00%	97.42	6.72	1053
		cpmp12	5	4854	402	375	7.20%	2.81%	4.16%	0.75	0	8.11	303.72	391	387	1.03%	—	6.11	34,995
		cpmp13	5	4169	446	441	1.13%	0.00%	1.23%	0.75	3	6.67	209.77	446	446	0.00%	30.25	6.09	203
		cpmp14	4	1672	455	434	4.84%	1.79%	2.91%	0.70	0	7.19	270.52	447	443	0.90%	—	6.19	39,581
		cpmp15	4	2686	517	470	10.00%	9.07%	0.98%	0.68	0	6.29	230.82	474	474	0.00%	135.43	2.82	1317
		cpmp16	4	2483	454	431	5.34%	0.44%	4.77%	0.72	0	6.49	265.22	452	440	2.73%	—	0.77	41,818
		cpmp17	5	3931	447	424	5.42%	3.71%	1.78%	0.81	0	6.46	210.82	431	431	0.00%	120.12	6.06	855
		cpmp18	5	2211	462	431	7.19%	0.00%	6.78%	1.06	0	6.64	269.58	462	442	4.52%	—	0.37	39,287
		cpmp19	5	3675	471	431	9.28%	5.84%	3.22%	0.87	0	6.39	189.70	445	445	0.00%	1399.44	7.08	12,989
		cpmp20	4	2772	470	451	4.21%	2.17%	1.96%	0.88	0	5.91	176.56	460	460	0.00%	847.31	7.81	9321
		Average	4.5	3061.8			5.81%	2.73%	2.98%	0.78	0.3	6.64	234.47			0.92%	438.33	5.00	18,141.9
150	50	cpmp21	6	4249	612	581	5.34%	0.00%	5.13%	3.46	0	6.45	363.75	612	588	4.08%	—	0.00	15,748
		cpmp22	4	2694	590	540	9.26%	2.08%	6.60%	2.06	0	7.05	412.41	578	548	5.47%	—	0.00	15,096
		cpmp23	5	4655	590	551	7.08%	3.87%	3.02%	2.52	0	7.31	448.29	568	559	1.61%	—	1.53	15,158
		cpmp24	4	2917	522	496	5.24%	3.37%	1.95%	1.54	0	7.14	414.21	505	503	0.40%	—	8.08	15,747
		cpmp25	4	3652	512	486	5.35%	4.92%	0.58%	1.21	0	6.64	439.24	488	488	0.00%	13.09	1.42	59
		cpmp26	5	4893	547	526	3.99%	1.30%	2.65%	1.67	0	6.66	383.68	540	537	0.56%	—	9.27	17,439
		cpmp27	5	5124	610	568	7.39%	5.35%	2.01%	2.17	0	7.84	398.79	579	578	0.17%	—	10.40	14,258
		cpmp28	4	2385	511	499	2.40%	1.59%	0.80%	1.16	0	4.87	247.27	503	503	0.00%	72.69	4.76	483
		cpmp29	5	5378	554	530	4.53%	0.00%	4.46%	1.77	0	6.52	376.30	554	535	3.55%	—	0.03	17,533
		cpmp30	5	4010	521	488	6.76%	3.17%	3.50%	1.87	0	7.02	399.80	505	495	2.02%	—	1.10	18,879
		Average	4.7	3995.7			5.73%	2.56%	3.07%	1.94	0.0	6.75	388.37			1.79%	42.89	3.66	13,040.0
200	66	cpmp31	5	5247	579	572	1.22%	0.70%	0.68%	3.27	0	6.01	413.83	575	574	0.17%	—	8.81	11,898
		cpmp32	4	5238	833	700	19.00%	7.07%	10.13%	3.67	0	5.08	282.11	778	705	10.35%	—	0.00	8084
		cpmp33	4	4901	654	600	9.00%	0.00%	8.39%	4.31	0	6.07	383.52	654	605	8.10%	—	0.00	7349
		cpmp34	6	5678	749	684	9.50%	0.00%	8.74%	6.74	0	6.71	489.37	749	691	8.39%	—	0.00	6745
		cpmp35	6	5718	631	593	6.41%	1.94%	4.29%	5.25	0	6.67	540.47	619	598	3.51%	—	0.00	8535
		cpmp36	5	5341	608	570	6.67%	0.00%	6.33%	4.76	0	6.58	456.59	608	574	5.92%	—	0.00	7025
		cpmp37	4	5787	655	617	6.16%	0.00%	5.88%	3.00	0	6.28	507.32	655	621	5.48%	—	0.00	6898
		cpmp38	5	4592	633	596	6.21%	4.11%	2.08%	3.86	0	6.53	524.49	608	601	1.16%	—	2.34	9655
		cpmp39	4	5743	609	574	6.10%	2.35%	3.60%	3.58	0	6.43	523.50	595	578	2.94%	—	0.00	7995
		cpmp40	4	3513	657	608	8.06%	1.55%	6.04%	4.11	0	6.44	469.29	647	613	5.55%	—	0.00	7761
		Average	4.7	5175.8			7.83%	1.77%	5.62%	4.25	0.0	6.28	459.05			5.16%	—	1.11	8194.5
		Overall average	4.53	3374.65			5.89%	2.17%	3.54%	1.80		6.22	291.65			2.00%	166.51		13,168.43

puts in evidence that the capacity constraints actually make the problem harder.

4. BENCHMARKS AND EXPERIMENTAL COMPARISONS

4.1. Benchmark Algorithms

4.1.1. General Purpose Solver. We solved the CPMP instances with CPLEX 6.5, using the formulation of the CPMP presented in Section 2, tightened with the inequalities $x_{ij} \leq y_j$. All the parameters were kept at the default values. These include automatic dynamic generation of clique, cover, and GUB-cover inequalities, best-bound-first search strategy, and a relative and absolute optimality tolerance of 0.01% and 10^{-6}, respectively.

4.1.2. Lagrangean Relaxation. Lagrangean relaxation has been successfully applied to many combinatorial optimization problems; hence, it is a good benchmark for other methods. Pirkul's branch-and-bound algorithm for the capacitated concentrators location problem [25], based on the Lagrangean relaxation of the partitioning constraints, can be easily adapted to the CPMP. Also, Baldacci et al. [1] compared their algorithm with an adaptation of Pirkul's algorithm. Following [25] and [1], we implemented a branch-and-bound algorithm based on the Lagrangean relaxation of

TABLE 6. Branch-and-price with subgradient-based multiple pricing—Class δ.

Instance			Root								Search tree							
N	p		CG it.	cols	UB	LB	gap	UB gap	LB gap	time (s)	fix.med.	CG it.	cols	F.UB	F.LB	gap	time (s)	avg. fix. med	ev. nodes
50	20	cpmp01	3	1066	266	259	2.70%	0.00%	2.74%	0.13	2	4.91	74.95	266	266	0.00%	1.69	3.33	57
		cpmp02	4	2107	300	293	2.39%	0.67%	1.85%	0.31	4	4.44	67.60	298	298	0.00%	3.43	1.91	79
		cpmp03	3	1000	352	307	14.66%	13.18%	1.29%	0.21	0	4.06	71.38	311	311	0.00%	0.66	0.75	17
		cpmp04	3	784	277	276	0.36%	0.00%	0.54%	0.15	6	2.69	40.75	277	277	0.00%	0.44	0.43	17
		cpmp05	4	1550	360	355	1.41%	1.12%	0.42%	0.38	1	4.00	54.00	356	356	0.00%	0.57	3.67	5
		cpmp06	4	1400	370	367	0.82%	0.00%	0.81%	0.26	6	3.00	8.50	370	370	0.00%	0.35	0.00	3
		cpmp07	4	1356	359	357	0.56%	0.28%	0.28%	0.36	6	3.83	62.83	358	358	0.00%	0.60	3.60	7
		cpmp08	4	914	322	298	8.05%	3.21%	4.78%	0.33	0	4.44	73.65	312	312	0.00%	44.90	2.90	2025
		cpmp09	3	739	417	404	3.22%	1.21%	2.09%	0.29	0	3.86	64.30	412	412	0.00%	11.65	1.90	539
		cpmp10	3	920	502	442	13.57%	9.61%	3.68%	0.39	0	4.48	77.35	458	458	0.00%	426.68	2.65	19,429
		Average	3.5	1183.6			4.77%	2.93%	1.85%	0.28	2.5	3.97	59.53			0.00%	49.10	2.11	2217.8
100	40	cpmp11	4	3381	446	406	9.85%	7.47%	2.21%	0.89	0	6.45	249.48	415	413	0.48%	—	5.89	38,912
		cpmp12	4	2035	424	365	16.16%	11.58%	4.14%	1.04	0	5.93	237.71	380	373	1.88%	—	1.88	42,328
		cpmp13	4	3955	424	405	4.69%	2.91%	1.74%	0.74	0	4.54	147.86	412	412	0.00%	430.11	4.05	4449
		cpmp14	4	2230	474	413	14.77%	12.59%	1.97%	0.92	0	5.29	172.20	421	421	0.00%	352.47	3.72	3731
		cpmp15	3	1924	526	489	7.57%	6.05%	1.44%	0.68	0	5.50	178.90	496	496	0.00%	737.53	5.60	9075
		cpmp16	4	3423	435	425	2.35%	1.64%	0.86%	0.91	0	4.26	147.66	428	428	0.00%	95.71	2.95	1205
		cpmp17	4	1848	475	430	10.47%	7.95%	2.27%	0.95	0	5.85	190.92	440	440	0.00%	1588.28	5.14	18,091
		cpmp18	5	3857	464	434	6.91%	3.11%	3.69%	1.13	0	5.75	191.39	450	450	0.00%	666.00	4.82	6675
		cpmp19	5	2901	466	440	5.91%	3.56%	2.43%	1.57	0	5.25	197.79	450	450	0.00%	3587.27	4.90	40,217
		cpmp20	5	5076	523	476	9.87%	7.61%	2.19%	1.81	0	5.19	180.20	486	486	0.00%	533.02	4.71	4773
		Average	4.2	3063.0			8.86%	6.45%	2.30%	1.06	0.0	5.40	189.41			0.24%	998.80	4.36	16,945.6
150	60	cpmp21	6	4780	591	545	8.44%	7.07%	1.28%	3.23	0	5.81	367.01	552	552	0.00%	333.66	2.18	1481
		cpmp22	4	4519	659	588	12.07%	9.65%	2.21%	2.46	0	6.06	385.59	601	595	1.01%	—	2.40	18,434
		cpmp23	3	3160	597	540	10.56%	4.01%	5.99%	1.59	0	5.68	353.92	574	548	4.74%	—	0.00	17,100
		cpmp24	4	2386	506	475	6.53%	1.61%	4.80%	2.01	0	5.91	367.48	498	480	3.75%	—	0.02	17,603
		cpmp25	4	4910	478	428	11.68%	9.63%	1.90%	1.77	0	5.91	321.61	436	434	0.46%	—	7.28	21,154
		cpmp26	4	2433	534	506	5.53%	3.89%	1.65%	2.70	0	5.36	329.25	514	511	0.59%	—	4.34	16,901
		cpmp27	5	2838	772	715	7.97%	0.39%	7.09%	3.47	0	5.74	363.25	769	722	6.51%	—	0.00	14,311
		cpmp28	4	4844	485	463	4.75%	2.75%	2.03%	1.58	0	5.39	292.07	472	469	0.64%	—	5.39	23,304
		cpmp29	4	4988	524	488	7.38%	6.07%	1.21%	1.30	0	5.24	303.62	494	494	0.00%	1976.75	6.63	11,337
		cpmp30	5	7058	461	435	5.98%	3.83%	2.06%	1.96	0	5.84	335.76	444	441	0.68%	—	6.93	19,715
		Average	4.3	4191.6			8.09%	4.89%	3.02%	2.21	0.0	5.69	341.95			1.84%	1155.21	3.52	16,134.0
200	80	cpmp31	4	7505	567	528	7.39%	3.66%	3.50%	3.76	0	5.82	397.49	547	533	2.63%	—	0.04	7983
		cpmp32	4	3796	926	781	18.57%	9.20%	7.96%	5.80	0	5.94	500.34	848	788	7.61%	—	0.00	7657
		cpmp33	4	4155	575	548	4.93%	1.05%	3.72%	3.86	0	5.22	403.86	569	551	3.27%	—	0.02	8852
		cpmp34	6	6058	1043	835	24.91%	20.02%	4.01%	9.57	0	6.14	513.67	869	838	3.70%	—	0.00	6097
		cpmp35	4	4175	588	542	8.49%	3.34%	4.75%	4.57	0	5.44	453.69	569	545	4.40%	—	0.00	8076
		cpmp36	4	6777	581	538	7.99%	0.00%	7.42%	3.53	0	5.66	436.61	581	543	7.00%	—	0.00	7576
		cpmp37	5	7722	634	580	9.31%	3.59%	5.28%	5.64	0	5.65	437.58	612	584	4.79%	—	0.00	6686
		cpmp38	5	6375	635	587	8.18%	7.26%	0.85%	5.33	0	5.96	489.10	592	590	0.34%	—	10.51	8422
		cpmp39	4	3591	578	528	9.47%	0.00%	8.66%	4.95	0	5.20	366.94	578	532	8.65%	—	0.00	5775
		cpmp40	4	4433	632	571	10.68%	3.61%	6.40%	3.91	0	5.49	410.60	610	575	6.09%	—	0.00	6401
		Average	4.4	5458.7			10.99%	5.17%	5.26%	5.09	0.0	5.65	440.99			4.85%	—	1.06	7352.5
		Overall average	4.10	3474.23			8.18%	4.86%	3.11%	2.16		5.18	257.97			1.73%	737.37		10,662.48

the semiassignment constraints presented in Section 3. The Lagrangean dual problem is solved by subgradient optimization. At most 300 subgradient iterations are executed at the root node, at most 50 iterations at the other nodes of the first level search tree (where branching is done on location variables) and at most 15 iterations at the nodes of the second level branching tree (where branching is done on assignment variables). Primal bounds are computed at each subgradient iteration by Pirkul's procedure Heur2.

We also implemented alternative and faster bounding techniques proposed in [25], namely evaluating the dual bound using the best Lagrangean penalties found at the predecessor node in the search tree, and solving the linear relaxation of the knapsack subproblems using a "good" set of multipliers.

More details on the adaptation of Pirkul's algorithm to the CPMP can be found in [3], where an alternative formulation is also discussed, consisting of the relaxation of constraints (1) and (3).

4.2. *Algorithms Comparison*

In Tables 7 and 8, for every class of instances the column "υ^*(gap)" reports the optimal value if optimality was proven; otherwise it reports the gap between the best primal and dual bounds obtained. As indicated in Subsection 2.4, computation was halted after 1 hour or in case of memory overflow. If computation exceeded these resource limitations, the "time" column is marked with a dash and the type of resource exceeded is indicated in the "status" column. The

TABLE 7. CPLEX 6.5.

Instance		Class α			Class β			Class γ			Class δ		
		v^* (gap)	time (s)	status	v^* (gap)	time (s)	status	v^* (gap)	time (s)	status	v^* (gap)	time (s)	status
$N = 50$	cpmp01	713	4.98		383	7.26		298	2.07		266	3.96	
	cpmp02	740	0.48		412	4.34		336	16.46		298	7.31	
	cpmp03	751	31.15		405	27.41		314	8.98		311	20.27	
	cpmp04	651	1.43		384	453.59		303	98.69		277	8.43	
	cpmp05	664	5.92		429	85.84		351	14.52		356	22.81	
	comp06	778	0.68		482	70.51		390	23.66		370	6.39	
	cpmp07	787	50.02		445	380.05		361	45.99		358	19.69	
	cpmp08	820	154.10		403	131.38		353	571.27		312	230.11	
	cpmp09	715	26.31		436	290.84		373	174.40		412	804.70	
	cpmp10	829	127.49		461	551.10		390	530.10		458	646.66	
		0.000%	40.256	10	0.000%	200.232	10	0.000%	148.614	10	0.000%	177.033	10
$N = 100$	cpmp11	1006	423.55		544	969.41		414	80.13		415	696.14	
	cpmp12	966	142.38		504	2045.81		391	1691.34		377	1092.72	
	cpmp13	1026	50.01		555	1788.78		446	100.56		412	365.01	
	cpmp14	982	929.99		544	2279.67		447	1897.32		421	1624.19	
	cpmp15	1091	1111.13		583	700.60		474	323.27		496	1158.77	
	cpmp16	954	176.70		(550;529)	—	(a)	(455;447)	—	(a)	428	1261.00	
	cpmp17	1034	577.67		542	261.80		431	46.31		440	2612.34	
	cpmp18	1043	1129.22		508	223.17		456	3223.04		450	690.38	
	cpmp19	1031	557.86		551	3538.42		445	2107.70		450	1607.81	
	cpmp20	(1061;995)	—	(a)	(611;533)	—	(a)	460	1925.53		(—;458)	—	(a)
		0.663%	869.952	9	1.860%	1900.980	8	0.179%	1499.630	9	(0.000%)	1234.262	9
$N = 150$	cpmp21	1288	481.72		(1292;672)	—	(a)	(760;581)	—	(a)	(1466;539)	—	(a)
	cpmp22	(1256;1251)	—	(a)	(876;645)	—	(a)	(1364;535)	—	(a)	(—;575)	—	(a)
	cpmp23	1279	665.76		(706;653)	—	(a)	(1652;542)	—	(a)	(1474;535)	—	(a)
	cpmp24	(1254;1213)	—	(a)	594	1235.17		(1241;492)	—	(a)	(491;484)	—	(a)
	cpmp25	1193	430.84		629	204.04		488	1330.24		(1044;427)	—	(a)
	cpmp26	1264	1502.08		653	886.45		(750;528)	—	(a)	512	589.29	
	cpmp27	1323	3463.23		(1540;701)	—	(a)	(1967;560)		(a)	(—;668)	—	(a)
	cpmp28	1233	455.58		644	473.86		503	138.68		471	1199.79	
	cpmp29	1219	202.37		649	390.83		(545;544)	—	(a)	494	347.70	
	cpmp30	1201	156.56		630	896.90		(1234;487)	—	(a)	444	163.33	
		0.378%	1462.320	8	25.588%	1861.870	6	98.966%	3052.720	2	(61.680%)	2102.900	4
$N = 200$	cpmp31	(1439;1371)	—	(a)	(1545;710)	—	(a)	575	396.79		(726;526)	—	(a)
	cpmp32	(1472;1411)	—	(a)	(—;776)	—	(a)	(-;680)	—	(a)	(—;722)	—	(a)
	cpmp33	(1406;1363)	—	(a)	(781;701)	—	(a)	(1492;594)	—	(a)	(—;545)	—	(a)
	cpmp34	(2466;1372)	—	(a)	(—;762)	—	(a)	(832;666)	—	(a)	(—;776)	—	(a)
	cpmp35	(1494;1431)	—	(a)	(1462;723)	—	(a)	(-;585)	—	(a)	(1781;533)	—	(a)
	cpmp36	1382	1089.73		701	2949.48		(662;573)	—	(a)	(—;534)	—	(a)
	cpmp37	1458	936.61		753	1809.92		(1341;614)	—	(a)	(1464;578)	—	(a)
	cpmp38	(—;1370)	—	(a)	(1533;729)	—	(a)	(646;606)	—	(a)	(—;576)	—	(a)
	cpmp39	1374	976.39		(835;712)	—	(a)	(610;584)	—	(a)	(563;536)	—	(a)
	cpmp40	1416	2878.57		(1484;736)	—	(a)	(678;614)	—	(a)	(—;563)	—	(a)
		(10.731%)	2707.790	4	(57.554%)	3368.600	2	(41.440%)	3285.100	1	(107.613%)	3697.240	0

(a) = computation exceeded time limit; (b) = out of memory.

last row of each block in the tables reports the average gap, the average computation time (neglecting the tests that exceeded resource limitations), and the number of problems solved to proven optimality.

From the examination of the gaps between the primal solution, the optimum and the lower bound at the root node, the branch-and-price algorithm shows a good behavior also when it is used as a heuristic. This is especially true for instances in class α, where the best solution found is on the average 0.26% from optimality after only 4.55 seconds. For all sizes considered CPLEX could solve to optimality more instances in class β than the other algorithms.

The Lagrangean approach shows a completely different behavior, because it is competitive even for large instances but only for small values of the $\frac{p}{N}$ ratio. When the number of medians increases, its performances significantly worsen. This is easy to explain: for given N, when p is higher the first level branching tree grows larger, because it is necessary to fix more location variables to reach a leaf node.

On the 40 instances with $N = 200$, the primal solutions found by branch-and-price after 1 hour were consistently better than those found by any of the other algorithms considered. CPLEX was not able to find any feasible solution within the time limit for 11 of these 40 instances. For the remaining 29 instances, the average approximation error, computed with respect to the best known lower bound, was 2.64% for the branch-and-price algorithm, while the approximation error yielded by the Lagrangean relaxation algorithm was 7.57% and that given by CPLEX more than 40%.

TABLE 8. Branch-and-bound with Lagrangean relaxation.

Instance		Class α			Class β			Class γ			Class δ		
		v^* (gap)	time (s)	status	v^* (gap)	time (s)	status	v^* (gap)	time (s)	status	v^* (gap)	time (s)	status
$N = 50$	cpmp01	713	0.17		383	12.35		298	3.4		266	440.37	
	cpmp02	740	0.06		412	1.19		336	271.86		298	2423.56	
	cpmp03	751	0.13		405	8.85		314	18.49		311	77.54	
	cpmp04	651	0.07		384	910.32		303	85.28		277	256.79	
	cpmp05	664	0.09		429	308.33		351	22.15		356	216.23	
	cpmp06	778	0.09		482	256.78		390	272.26		370	39.49	
	cpmp07	787	0.80		445	715.14		361	107.85		358	42.29	
	cpmp08	820	17.54		403	153.58		(365;322)	—	(a)	(313;290)	—	(a)
	cpmp09	715	0.92		436	91.26		373	165.14		(418;365)	—	(a)
	cpmp10	829	2.85		461	647.89		(390;368)	—	(a)	(481;403)	—	(a)
		0.000%	2.272	10	0.000%	310.569	10	1.933%	118.304	8	4.181%	499.467	7
$N = 100$	cpmp11	1006	3.26		(544;527)	—	(a)	(439;405)	—	(a)	(435;400)	—	(a)
	cpmp12	966	15.26		(509;496)	—	(a)	(394;374)	—	(a)	(393;362)	—	(a)
	cpmp13	1026	1.32		(567;534)	—	(a)	(451;439)	—	(a)	(425;402)	—	(a)
	cpmp14	982	47.97		(548;529)	—	(a)	(481;433)	—	(a)	(442;411)	—	(a)
	cpmp15	1091	32.33		(593;570)	—	(a)	(481;468)	—	(a)	(504;482)	—	(a)
	cpmp16	954	8.17		(558;518)	—	(a)	(453;430)	—	(a)	(430;423)	—	(a)
	cpmp17	1034	42.49		542	1673.37		(439;424)	—	(a)	(795;427)	—	(a)
	cpmp18	1043	39.94		508	977.96		(493;429)	—	(a)	(500;429)	—	(a)
	cpmp19	1031	18.92		(562;531)	—	(a)	(473;430)	—	(a)	(498;437)	—	(a)
	cpmp20	1005	3377.63		(597;514)	—	(a)	(483;448)	—	(a)	(1630;463)	—	(a)
		0.000%	358.729	10	4.936%	1325.660	2	7.196%	—	0	40.554%	—	0
$N = 150$	cpmp21	1288	227.11		(683;674)	—	(a)	(626;580)	—	(a)	(585;543)	—	(a)
	cpmp22	1256	1100.95		(665;643)	—	(a)	(599;537)	—	(a)	(992;582)	—	(a)
	cpmp23	1279	62.43		(711;639)	—	(a)	(659;548)	—	(a)	(1235;528)	—	(a)
	cpmp24	1220	288.73		(603;585)	—	(a)	(533;495)	—	(a)	(518;471)	—	(a)
	cpmp25	1193	29.55		(634;627)	—	(a)	(499;484)	—	(a)	(461;427)	—	(a)
	cpmp26	1264	48.11		(653;646)	—	(a)	(565;526)	—	(a)	(530;504)	—	(a)
	cpmp27	1323	1795.07		(837;711)	—	(a)	(631;566)	—	(a)	(2859;694)	—	(a)
	cpmp28	1233	74.16		(647;630)	—	(a)	(526;498)	—	(a)	(493;462)	—	(a)
	cpmp29	1219	11.70		(656;641)	—	(a)	(565;529)	—	(a)	(505;488)	—	(a)
	cpmp30	1201	11.61		(641;619)	—	(a)	(515;487)	—	(a)	(472;434)	—	(a)
		0.000%	364.942	10	4.762%	—	0	8.758%	—	0	56.609%	—	0
$N = 200$	cpmp31	1378	482.48		(767;711)	—	(b)	(590;570)	—	(a)	(590;527)	—	(b)
	cpmp32	(1447;1404)	—	(a)	(1580;789)	—	(b)	(1699;694)	—	(b)	(—;732)	—	(b)
	cpmp33	(1385;1360)	—	(a)	(765;693)	—	(b)	(751;598)	—	(a)	(662;547)	—	(a)
	cpmp34	(1385;1372)	—	(a)	(1137;770)	—	(b)	(915;677)	—	(a)	(—;788)	—	(b)
	cpmp35	1437	2982.44		(801;725)	—	(b)	(636;592)	—	(a)	(630;540)	—	(b)
	cpmp36	1382	100.30		(717;690)	—	(b)	(615;569)	—	(a)	(574;537)	—	(a)
	cpmp37	1458	259.51		(783;743)	—	(a)	(679;615)	—	(b)	(668;578)	—	(b)
	cpmp38	(1390;1369)	—	(a)	(839;730)	—	(b)	(645;595)	—	(b)	(764;581)	—	(b)
	cpmp39	1374	106.60		(736;712)	—	(a)	(620;573)	—	(b)	(600;528)	—	(b)
	cpmp40	1416	1300.01		(842;735)	—	(b)	(712;606)	—	(b)	(1366;568)	—	(b)
		0.738%	871.89	6	21.882%	—	0	26.908%	—	0	(32.217%)	—	0

(a) = computation exceeded time limit; (b) = out of memory.

4.3. The Algorithm of Baldacci, Hadjicostantinou, Maniezzo and Mingozzi

In [1] Baldacci et al. proposed an algorithm (in the remainder we call it BHMM for short) that is able to prove optimality of the solution found or to provide a valid dual bound, and therefore an *a posteriori* approximation guarantee.

BHMM adopts a three steps approach: a primal-dual bound gap is computed by a procedure H1, that relies on Lagrangean relaxation (the same described above) and subgradient optimization. The dual bound obtained is strengthened by a procedure H2, that uses the linear relaxation of the CPMP reformulation as a set partitioning problem, whose columns are the feasible clusters whose reduced costs do not exceed the gap found in H1; the reduced costs are evaluated using the best set of multipliers encountered. To reduce the problem size only the best Δ clusters ($\Delta = 2000$ in the implementation of [1]) for every candidate median are considered. When a new dual bound (and the corresponding dual solution) is obtained, a new set of columns is computed and an integer solution is obtained using CPLEX (procedure EHP); that solution can be nonoptimal, but a dual bound based on the reduced cost of the best cluster discarded can either prove optimality or provide information on the approximation obtained.

BHMM may behave as an optimization algorithm or as a heuristic depending on the value of parameter Δ, which is user-controlled. Unfortunately, it is not known how Δ must be chosen to have an *a priori* optimality guarantee: once a value of Δ has been guessed, it is possible that BHMM

terminate providing a suboptimal solution together with a lower bound. If the user wants to find the optimal solution, he must guess a larger value of Δ and try again, and the procedure must be repeated until optimality is reached or the available computing resources are exhausted. Obviously, larger values of Δ imply larger amounts of computing time. On the contrary, our branch-and-price algorithm does certainly yield the optimal solution, provided that the program is not aborted for insufficient computing resources. Hence, when a difficult CPMP instance must be solved to optimality one has two alternatives: either tentatively tuning Δ according to the available resources and running BHMM in the hope that the solution will be provably optimal or running the branch-and-price algorithm in the hope that the computing resources will be sufficient. Even if the two methods are of different nature from a theoretical viewpoint, we tried to make a significant experimental comparison because at the best of our knowledge they represent the state of the art to solve the CPMP. To this purpose we set Δ to the same value indicated in [1], that is $\Delta = 2000$, which is large enough to often obtain provably optimal solutions and small enough to make the algorithms comparable from the viewpoint of computing time.

Moreover, we had to address two other points, concerning the formulation of the problem and the initialization of BHMM. In [1], the authors used a slightly less general

TABLE 9. Comparison between BHMM and branch-and-price.

		Problem			BHMM + bionomic bound		Branch-and-price	
N	P	Instance	Bionomic bound	Optimum	ν^* (gap)	time (s)	ν^* (gap)	time (s)
50	5	ccpx1	713	713	713	40.25	713	4.58
		ccpx2	740	740	720	40.08	740	0.16
		ccpx3	751	751	751	40.16	751	2.43
		ccpx4	651	651	651	40.07	651	0.31
		ccpx5	664	664	664	40.15	664	0.56
		ccpx6	778	778	778	40.08	778	0.33
		ccpx7	787	787	787	41.24	787	7.13
		ccpx8	820	820	(820;790)	—	820	1904.73
		ccpx9	715	715	715	40.28	715	1.00
		ccpx10	829	829	829	44.41	829	7.96
		Average			0.38%	40.74	0.00%	2.72
100	10	ccpx11	1006	1006	1006	42.20	1006	19.08
		ccpx12	966	966	966	133.83	966	93.51
		ccpx13	1026	1026	1026	40.62	1026	30.46
		ccpx14	982	982	982	125.23	982	232.13
		ccpx15	1091	1091	1091	90.04	1091	407.73
		ccpx16	954	954	954	40.95	954	5.70
		ccpx17	1034	1034	1034	57.67	1034	113.91
		ccpx18	1043	1043	(1043;1040)	179.14	1043	877.95
		ccpx19	1031	1031	1031	43.48	1031	18.03
		ccpx20	1013	1005	(1013;985)	—	(1006;993)	—
		Average			0.31%	71.75	0.13%	115.07
150	15	ccpx21	1290	1283	(1283;1279)	3952.79	1283	4119.65
		ccpx22	1292	1291	1291	206.04	1291	450.46
		ccpx23	1220		(1219;1192)	—	(1216;1207)	—
		ccpx24	1236	1235	(1235;1230)	470.59	1235	5143.87
		ccpx25	1189	1188	1188	40.72	1188	3.94
		ccpx26	1228	1227	1227	40.63	1227	4.37
		ccpx27	1270	1269	1269	208.40	1269	730.71
		ccpx28	1181	1180	1180	49.49	1180	391.48
		ccpx29	1260		(1259;1248)	6389.09	(1262;1250)	—
		ccpx30	1243	1241	(1241;1236)	2291.95	(1241;1239)	—
		Average			0.57%	109.05	0.19%	316.19
200	20	ccpx31	1447	1446	1446	294.68	1446	3827.76
		ccpx32	1352		(1351;1336)	—	(1358;1344)	—
		ccpx33	1391	1390	1390	40.45	1390	10.47
		ccpx34	1395		(1394;1362)	—	(1393;1374)	—
		ccpx35	1401		(1400;1382)	—	(1401;1389)	—
		ccpx36	1384	1382	(1382;1378)	2643.33	(1382;1380)	—
		ccpx37	1399		(1398;1370)	—	(1388;1379)	—
		ccpx38	1462		(1461;1438)	—	(1476;1447)	—
		ccpx39	1427		(1426;1421)	1013.62	(1426;1425)	—
		ccpx40	1393	1392	1392	41.20	1392	12.94
		Average			0.91%	125.44	0.62%	1283.72

formulation of the CPMP, imposing that a vertex hosting a selected median must belong to its cluster, that is x_{jj} variables were used instead of y_j as location variables. From a modeling viewpoint this assumption is a significant restriction in capacitated problems: it can change the optimal solution (this is the case of instance ccpx-7, class δ, for example) and it can even inhibit the existence of feasible solutions. For this reason we decided to address the more general formulation, in which y_j variables are introduced. However, from an algorithmic viewpoint the difference is negligible: we could easily adapt BHMM to the more general formulation, and we did not observe any significant change in its performances.

The second key issue to make a significant experimental comparison is initialization. The computational results reported in [1] were obtained by initializing procedure H1 with very tight primal bounds: the authors used solutions given by a so-called "bionomic" algorithm described in [17], and these solutions are often optimal; the initialisation was made by taking such values increased by 1 as initial upper bounds. However, the computing time spent in the computation of so good initial solutions took 10 minutes on an Intel Pentium 166 MHz PC, as reported in [17], and it was not considered in the presentation of the computational results in [1]. On the contrary, our branch-and-price algorithm does not require the knowledge of quasi-optimal solutions in advance, because upper bounds are generated inside the column generation procedure.

Therefore, to make a fair comparison between the two techniques, we did the following. First of all, we initialized BHMM as in [1] and we compared the outcome with the results reported by Baldacci et al., to validate our reimplementation of their code, which had not been made available to us. The observed computing times well correspond to those reported in [1] when they are scaled by a factor 10.8, which is the ratio between the speed of our hardware and that of the machine cited in [1], according to the LINPACK benchmark [8].

Then we compared the BHMM algorithm and the branch-and-price algorithm under the usual hypothesis that nothing is known in advance: this means that the branch-and-price algorithm started from scratch with no initialization, while the BHMM algorithm was initialized with the best solution given by the bionomic algorithm (the values were taken from [1], without increasing them by 1) and its computing time was increased by a corresponding amount, equal to 10 minutes divided by the scale factor given by the LINPACK benchmark: for the machine used for the experiments reported in [17] such factor is about 15. The comparison is reported in Table 9, and it was necessarily limited to the instances of class α for which the initial value given by the bionomic algorithm was known. As in [1], computation was halted after 1 hour for the tests on instances with $N = 50$ and $N = 100$, and after 2 hours on instances with $N = 150$ and $N = 200$. As in the previous tables, the column "υ^* (gap)" reports the optimal value if optimality was proven, otherwise the best primal and dual bounds are shown and the "time" column is marked with a dash. For each instance, in the columns "Bionomic bound" and "Optimum" the primal bound given by the bionomic algorithm and the optimal value (if known) are indicated. When the BHMM algorithm terminated without proving optimality, we report both the best primal and dual bounds and the time required. CPLEX 6.5 was used both as an LP solver and a IP solver in the BHMM algorithm.

For instances with $N = 50$, both algorithms performed well: BHMM was slower, due to the time needed by the bionomic algorithm to reach a good primal bound. Branch-and-price solved to optimality one instance more than BHMM. For instances with $N = 100$ and $N = 150$ BHMM was faster, but branch-and-price proved the optimality of more instances and produced smaller primal-dual gaps. For instances with $N = 200$, both BHMM and branch-and-price solved to optimality only three instances over 10. Once again, BHMM was faster, but branch-and-price gave tighter approximations.

4.4. Concluding Remarks

The computational results presented in Sections 3 and 4 show that the performances of all algorithms we have considered are strongly affected both by the size of the instance, that is, the number of vertices, and by the value of the $\frac{p}{N}$ ratio. In addition, the algorithms show complementary behaviors. CPLEX is particularly effective is solving the smaller instances and those of class β. However, larger problems are still too big to be efficiently managed by a general-purpose solver: no feasible solution was found for several of the instances with $N = 200$ after 1 hour of computation.

The Lagrangean-based branch-and-bound works fine for small values of $\frac{p}{N}$ (class α), but its performance worsens quickly as this ratio increases. In fact, both the quality of the lower bound and the effectiveness of the branching policy are affected by high $\frac{p}{N}$ ratios. The branch-and-price algorithm shows a much more stable behavior: it is effective on small problems, where no significant difference can be observed within the four classes of instances, still giving tight upper and lower bounds for the larger instances. A feasible solution is always found within a few percentage points from optimality.

This suggests to use branch-and-price as an approximation method for large CPMP problems: as an alternative approach, we investigated the behavior of the BHMM algorithm initialized with upper bounds given by the bionomic metaheuristic. When both methods complete the computation within the time limit, the BHMM algorithm is on average faster. However, branch-and-price consistently yields smaller gaps between upper and lower bounds, and is able to prove the optimality of a larger number of instances.

Acknowledgments

We are grateful to Marco Trubian, who inspired this work with his invaluable suggestions, and to Roberto Baldacci for providing us the test instances. The comments of three anonymous referees were very useful to improve both the

content and the presentation of the article. We also acknowledge the support of ACSU (Associazione Cremasca Studi Universitari) to the Operations Research Laboratory of our department, where this work was done.

APPENDIX

Primal heuristic:

Input: $z^k \ \forall k \in Z$
(the solution of LRMP)
Output: $C^j \ \forall j \in \mathcal{M}$
(the set of clusters of a feasible solution for CPMP)

(Step 1: medians selection)
$f_{ij} = \sum_{k \in Z^j} x_i^k z_k^j \ \forall i \in \mathcal{N} \ \forall j \in \mathcal{M}$
$\psi_j = \sum_{i \in \mathcal{N}} f_{ij} \ \forall j \in \mathcal{M}$
$\bar{\mathcal{M}} := \emptyset$
for p times do
  $j^* := \text{argmax}_{j \in \mathcal{M} \setminus \bar{\mathcal{M}}} \{\psi_j\}$
  $\bar{\mathcal{M}} := \bar{\mathcal{M}} \cup \{j^*\}$

(Step 2: direct assignment)
forall $j \in \bar{\mathcal{M}}$ do $q_j := Q_j$
$\mathcal{N}_W := \mathcal{N}$
$C^1 := C^2 := \ldots := C^M := \emptyset$
while $\mathcal{N}_W \neq \emptyset$ do
  (evaluation of regret values)
  forall $i \in \mathcal{N}_W$ do
    $\mathcal{M}^i = \{j | j \in \bar{\mathcal{M}}, q_j \geq w_i\}$
    if $\mathcal{M}^i = \emptyset$ then GOTO step 3
    $j_i' := \text{argmax}_{j \in \mathcal{M}^i} \{f_{ij}\}$
    if $\mathcal{M}^i = \{j_i'\}$ then $D_i := +\infty$
    else
      $j_i'' := \text{argmax}_{j \in \mathcal{M}^i, j \neq j_i'} \{f_{ij}\}$
      $D_i := f_{ij'} - f_{ij''}$
  (assignment of the node with highest regret)
  $i^* := \text{argmax}_{i \in \mathcal{N}_W} \{D_i\}$
  $j^* := j_{i^*}'$
  $C^{j^*} := C^{j^*} \cup \{i^*\}$
  $\mathcal{N}_W := \mathcal{N}_W \setminus \{i^*\}$
  $q_{j^*} := q_{j^*} - w_{i^*}$

(Step 3: assignment through exchanges)
while $\mathcal{N}_W \neq \emptyset$ do
  forall $i \in \mathcal{N}_W$ do
    (Evaluation of the set of clusters in which node i could be inserted)
    $L_i = \{j \mid \exists k \in C^j \mid q_j + w_k \geq w_i, w_k < w_i, j \in \bar{\mathcal{M}}\}$
    forall $j \in L_i$ do
      $l(i,j) := \min_{k \in C^j} \{(q_j + w_k) | q_j + w_k \geq w_i, w_k < w_i\}$
      $k(i,j) := \text{argmin}\{l(i,j)\}$
  if $\bigcup_{i \in \mathcal{N}_W} L_i \neq \emptyset$ then
    (Shifting that minimizes the residual capacity of a median)
    $(i^*, j^*) := \text{argmin}_{i \in \mathcal{N}_W, j \in L_i} \{l(i,j) - w_i\}$
    $C^{j^*} := C^{j^*} \setminus \{k(i^*, j^*)\}; C^{j^*} := C^{j^*} \cup \{i^*\}$
    $q_{j^*} := l(i^*, j^*) - w_{i^*}$
    $\mathcal{N}_W := \mathcal{N}_W \setminus \{i^*\}$
    if $\exists j \in \bar{\mathcal{M}} \mid q_j \geq w_{k(i^*,j^*)}$ then
      $C^j := C^j \cup \{k(i^*, j^*)\}$
      $q_j := q_j - w_{k(i^*,j^*)}$
    else $\mathcal{N}_W := \mathcal{N}_W \cup \{k(i^*, j^*)\}$
  else FAIL

(Step 4: solution improvement)
if $\mathcal{N}_W = \emptyset$ then
  (definition of a dummy node Ω)
  $w_\Omega = 0; \ d_{\Omega_j} = 0 \ \forall_j \in \mathcal{M}$
  forall $j \in \bar{\mathcal{M}}$, forall $i \in C^j$ do $s_i := j$
  do
    forall $i \in \mathcal{N}$ do
      forall $j \in \bar{\mathcal{M}}$ do
        $E_i^j = \{k \in C^j \cup \{\Omega\} | q_j + w_k \geq w_i, q_{s_i} + w_i \geq w_k\}$
      $g_i := \min_{(j \in \bar{\mathcal{M}}, k \in E_i^j)} \{d_{ij} - d_{is_i} + d_{ks_i} - d_{kj}\}$
      $(j_i', k_i') := \text{argmin}_{(j \in \bar{\mathcal{M}}, k \in E_i^j)} \{d_{ij} - d_{is_i} + d_{ks_i} - d_{kj}\}$
    $i^* := \text{argmin}_{i \in \mathcal{N}} \{g_i\}; \ j^* := j_{i^*}'; \ k^* := k_{i^*}'$
    if $(g_{i^*} < 0)$ then
      $q_{s_{i^*}} := q_{s_{i^*}} + w_{i^*} - w_{k^*}$
      $q_{j^*} := q_{j^*} - w_{i^*} + w_{k^*}$
      if $(k^* \neq \Omega)$ then
        $C^{j^*} := C^{j^*} \setminus \{k^*\}; \ C^{s_{i^*}} := C^{s_{i^*}} \cup \{k^*\}$
        $s_{k^*} := s_{i^*}$
      $C^{s_{i^*}} := C^{s_i} \setminus \{i^*\}; \ C^{j^*} := C^{j^*} \cup \{i^*\};$
      $s_{i^*} := j^*$
  while $g_{i^*} < 0$

REFERENCES

[1] R. Baldacci, E. Hadjiconstantinou, V. Maniezzo, and A. Mingozzi, A new method for solving capacitated location problems based on a set partitioning approach, Comput Operat Res 29 (2002), 365–386.

[2] J.E. Beasley, A note on solving large p-median problems, Eur J Operat Res 21 (1985), 270–273.

[3] A. Ceselli, Algoritmi branch and bound e branch and price per il problema delle p-mediane con capacità (in Italian), Dipartimento di Technologie dell'Informazione, Università degli Studi di Milano, Master degree thesis 2002.

[4] N. Christofides and J.E. Beasley, A tree search algorithm for the p-median problem, Eur J Operat Res 10 (1981), 196–204.

[5] G. Cornueojols, M.L. Fisher, and G.L. Nemhauser, Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms, Manage Sci 23 (1977), 789–810.

[6] J.A. Diaz, and E. Fernández, A branch-and-price algorithm for the single source capacitated plant location problem, J Operat Res Soc 53 (2002), 728–740.

[7] J.A. Diaz and E. Fernández, Hybrid scatter scatter search and path relinking for the capacitated p-median problem, Eur J of Operat Res, to appear.

[8] J.J. Dongarra, Performance of various computers using standard linear equations software, Technical Report. University of Tennessee, 2003 (available at http://www.netlib.org/benchmark/performance.ps).

[9] R.D. Galvão, A dual-bounded algorithm for the p-median problem, Operat Res 28 (1979), 5.

[10] P.C. Gilmore and R.E. Gomory, A linear programming approach to the cutting stock problem, Operat Res 9 (1961), 849–859.

[11] P. Hanjoul and D. Peeters, A comparison of two dual-based procedures for solving the p-median problem, Eur J Operat Res 20 (1985), 387–396.

[12] M. Held, P. Wolfe, and H.P. Crowder, Validation of subgradient optimization, Mathe Program 6 (1974), 62–88.

[13] K. Jörnsten and M. Näsberg, A new lagrangean relaxation approach to the generalized assignment problem, Eur J Operat Res 27 (1986), 313–323.

[14] O. Kariv and S.L. Hakimi, An algorithmic approach to network location problems. Part 2. The p-median., SIAM J App Math 37 (1979), 539–560.

[15] R.M. Karp, "Reducibility among combinatorial problems," R.E. Miller, J.W. Thatcher (editors), Complexity of computer computations, Plumunu Press, New York 1972, pp. 85–103.

[16] L. Lorena and E. Senne, A column generation approach to capacitated p-median problems, Compu and Operat Res 31 (2004), 863–876.

[17] V. Maniezzo, A. Mingozzi, and R. Baldacci, A bionomic approach to the capacitated p-median problem, J of Heuristics 4 (1998), 263–280.

[18] S. Martello and P. Toth, "An algorithm for the generalized assignment problem," Operational Research '81, J.P. Brans (editor), North-Holland, Amsterdam, (1981), pp. 589–603.

[19] S. Martello and P. Toth, Knapsack problems—Algorithms and computer implementations, John Wiley and Sons, New York, 1989.

[20] J.M. Mulvey and P. Beck, Solving capacitated clustering problems, Eur J Operat Res 18 (1984), 339–348.

[21] S.C. Narula, U.I. Ogbu, and H.M. Samuelsson, An algorithm for the p-median problem, Operat Res 25 (1977), 4.

[22] M. Labbé, D. Peeters, and J.F. Thisse, "Location on networks, in network routing," Handbooks in OR and MS, Elsevier Science B.V., Amsterdam, 1995, vol 8.

[23] G.L. Nemhauser and L.A. Wolsey, Integer and combinatorial optimization, Wiley-Interscience, New York, 1988.

[24] I.H. Osman and N. Christofides, Capacitated clustering problems by hybrid Simulated annealing and tabu search, Int Trans Operat Res 13 (1994), 317–336.

[25] H. Pirkul, Efficient algorithms for the capacitated concentrator location problem, Comput Operat Res 14 (1987), 197–208.

[26] D. Pisinger, A minimal algorithm for the 0-1 knapsack problem, Operat Res 46 (1995), 758–767.

[27] G.T. Ross and R.M. Soland, Modeling facility location problems as generalized assignment problems, Manage Sci 24 (1977), 3.

[28] M. Savelsbergh, A branch-and-price algorithm for the generalized assignment problem, Operat Res 45 (1997), 6.