OPERATIONS RESEARCH
Vol. 54, No. 6, November–December 2006, pp. 1172–1184
ISSN 0030-364X | EISSN 1526-5463 | 06 | 5406 | 1172

informs

DOI 10.1287/opre.1060.0323

A Branch-and-Price Algorithm for the Multilevel Generalized Assignment Problem

Alberto Ceselli, Giovanni Righini
Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano, via Bramante 65, 26013, Crema, Italy
{ceselli@dti.unimi.it, righini@dti.unimi.it}

The multilevel generalized assignment problem (MGAP) is a variation of the generalized assignment problem, in which agents can execute tasks at different efficiency levels with different costs. We present a branch-and-price algorithm that is the first exact algorithm for the MGAP. It is based on a decomposition into a master problem with set-partitioning constraints and a pricing subproblem that is a multiple-choice knapsack problem. We report on our computational experience with randomly generated instances with different numbers of agents, tasks, and levels; and with different correlations between cost and resource consumption for each agent-task-level assignment. Experimental results show that our algorithm is able to solve instances larger than those of the maximum size considered in the literature to proven optimality.

Subject classifications: integer programming: branch-and-price; production-scheduling: generalized assignment.
Area of review: Optimization.
History: Received May 2004; revisions received August 2005, October 2005; accepted October 2005.

1. Introduction

The multilevel generalized assignment problem (MGAP) is a variation of the well-known generalized assignment problem (GAP). The GAP consists of assigning tasks to agents with limited capacity, so that each task is assigned to an agent and a capacity constraint is satisfied for each agent. In the MGAP, each task-agent assignment can be made at different levels, implying both different costs (or revenues) and different amounts of resource used.

The MGAP arises in the context of large manufacturing systems: It was first described in Glover et al. (1979) as a task allocation problem in a real manufacturing environment. The problem arises when machines performing manufacturing operations on jobs can work at different "levels:" This means that the same job can be executed, for instance, with more or less accuracy, in more or less time, or with a larger or smaller energy consumption. Obviously, the outcome in terms of product quality or added value also depends on the level on which the manufacturing operations have been done. Levels may also represent different lot sizes, as in the original paper by Glover et al. Besides its application in production-planning contexts, due to its combinatorial structure the MGAP can also appear as a subproblem in other contexts, such as load balancing in clusters for high-performance computing, multifacility location, and multivehicle routing problems. For this reason, we here prefer the general terms "task" and "agent" instead of "job" and "machine," which are more specific to production-scheduling optimization. Because it is a generalization of the GAP, the MGAP is $\mathcal{NP}$-hard, and even the problem of determining whether a feasible solution exists is $\mathcal{NP}$-complete.

Laguna et al. (1995) proposed a tabu search algorithm for the MGAP. They reported on results obtained with instances involving up to 40 tasks, four agents, and four efficiency levels. More recently, French and Wilson (2002) presented two heuristic algorithms tested on larger instances with up to 200 tasks, 30 agents, and five efficiency levels. No ad hoc algorithm has been presented so far for the exact optimization of the MGAP. The only attempts to obtain optimal solutions have been made with general-purpose optimization packages, but the very large number of binary variables allows us to solve only problem instances of small size. Osorio and Laguna (2003) proposed adding logic cuts to strengthen the initial formulation; in this way, they could solve problem instances with up to 60 tasks, 30 agents, and two levels to optimality using CPLEX.

Branch and price is an effective mathematical programming technique to solve optimization problems like the GAP and the MGAP, requiring the partition of a set of elements into constrained subsets. A branch-and-price algorithm for the GAP was presented in Savelsbergh (1997).

In this paper, we present a branch-and-price algorithm based on a decomposition of the MGAP into a master problem and a pricing subproblem; the former is a set-partitioning problem, while the latter is a multiple-choice knapsack problem. We illustrate a branching strategy that is both effective at improving the dual bound and compatible with the combinatorial structure of the pricing subproblem. Our algorithm could solve problem instances larger than those of the maximum size considered in the literature (400 tasks, 80 agents, four levels). We compared the branch-and-price algorithm with CPLEX with and without logic cuts, solving random instances with different correlations between cost coefficients and resource requirements.

This paper is organized as follows: In §2, we introduce the basic formulation of the MGAP and its set-partitioning reformulation. In §3, we describe the branch-and-price algorithm and discuss some implementation details. In §4, we present our experimental results, and draw some conclusions.

2. Formulations

Consider a set of agents $\mathcal{N} = \{1, \ldots, N\}$ and a set of tasks $\mathcal{M} = \{1, \ldots, M\}$, such that each task must be assigned to an agent. Each agent $i \in \mathcal{N}$ can execute each task $j \in \mathcal{M}$ at different efficiency levels $k \in \mathcal{K} = \{1, \ldots, K\}$. Following Laguna et al. (1995), we formulate the MGAP as a minimization problem.

$$\min \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{M}} \sum_{k\in\mathcal{K}} c_{ijk} x_{ijk} \tag{1}$$

$$\text{s.t.} \sum_{i\in\mathcal{N}} \sum_{k\in\mathcal{K}} x_{ijk} = 1 \quad \forall j \in \mathcal{M}, \tag{2}$$

$$\sum_{j\in\mathcal{M}} \sum_{k\in\mathcal{K}} a_{ijk} x_{ijk} \leqslant b_i \quad \forall i \in \mathcal{N}, \tag{3}$$

$$x_{ijk} \in \{0, 1\} \quad \forall i \in \mathcal{N}, \forall j \in \mathcal{M}, \forall k \in \mathcal{K}. \tag{4}$$

This model will be indicated as the "natural formulation" of the MGAP. Binary variables x are assignment variables: $x_{ijk} = 1$ if and only if task $j \in \mathcal{M}$ is assigned to agent $i \in \mathcal{N}$ at level $k \in \mathcal{K}$. Each task $j \in \mathcal{M}$ implies a resource consumption $a_{ijk} \geqslant 0$ when it is assigned to agent $i \in \mathcal{N}$ at level $k \in \mathcal{K}$; each agent $i \in \mathcal{N}$ has an amount b_i of available resource. Each agent-task-level assignment implies a cost $c_{ijk} \geqslant 0$. Set-partitioning constraints (2) impose that each task is assigned to one agent at one efficiency level. Capacity constraints (3) impose the resource restriction for each agent. The objective is to minimize the sum of allocation costs.

Usually the following assumption holds in real cases: For each agent-task pair (i, j) and for each two different levels k and h with $k < h$, we assume that $a_{ijk} < a_{ijh}$ and $c_{ijk} > c_{ijh}$. When this property does not hold, some assignments are dominated, and the corresponding variables can be fixed to zero by a trivial preprocessing. It is clear that the correlation between the coefficients plays an important role in making an instance easy or hard to solve. This is explained in more detail in §4. Consider the relaxation in which constraints (2) are replaced by

$$\sum_{i\in\mathcal{N}} \sum_{k\in\mathcal{K}} x_{ijk} \geqslant 1 \quad \forall j \in \mathcal{M}. \tag{5}$$

Any feasible solution of (1), (5), (3), (4) in which some task is assigned more than once can be transformed into a feasible solution of (1), (2), (3), (4) by simply deleting the assignments in excess, and this does not increase the value of the objective function. Therefore, this relaxation has the same optimal value of the natural formulation.

We introduce here an alternative formulation of the MGAP, which is viable for a branch-and-price approach. Let a duty d for agent i be an assignment of tasks to agent i, that is a vector $\mathbf{x}_i^d = (x_{i11}^d, \ldots, x_{iMK}^d)$, where each component x_{ijk}^d takes value one if task j is assigned to agent i at efficiency level k, and zero otherwise. Let $\mathcal{D}_i = \{x_i^1, \ldots, x_i^{D_i}\}$ be the set of all feasible duties for agent $i \in \mathcal{N}$, i.e., the set of vectors $(x_{i11}^d, \ldots, x_{iMK}^d)$ such that

$$\sum_{j\in\mathcal{M}} \sum_{k\in\mathcal{K}} a_{ijk} x_{ijk}^d \leqslant b_i,$$

$$\sum_{k\in\mathcal{K}} x_{ijk}^d \leqslant 1 \quad \forall j \in \mathcal{M},$$

$$x_{ijk}^d \in \{0, 1\} \quad \forall j \in \mathcal{M} \ \forall k \in \mathcal{K}.$$

Let z_i^d be a binary variable indicating whether a duty $d \in \mathcal{D}_i$ is selected for agent $i \in \mathcal{N}$. The MGAP can be reformulated as follows:

$$\min \sum_{i\in\mathcal{N}} \sum_{d\in\mathcal{D}_i} \left(\sum_{j\in\mathcal{M}} \sum_{k\in\mathcal{K}} c_{ijk} x_{ijk}^d \right) z_i^d \tag{6}$$

$$\text{s.t.} \sum_{i\in\mathcal{N}} \sum_{d\in\mathcal{D}_i} \left(\sum_{k\in\mathcal{K}} x_{ijk}^d \right) z_i^d = 1 \quad \forall j \in \mathcal{M}, \tag{7}$$

$$\sum_{d\in\mathcal{D}_i} z_i^d = 1 \quad \forall i \in \mathcal{N}, \tag{8}$$

$$z_i^d \in \{0, 1\} \quad \forall i \in \mathcal{N} \ \forall d \in \mathcal{D}_i. \tag{9}$$

We remark that each x_{ijk}^d term represents a constant in this model; therefore, expressions (6) and (7) are linear. In this master problem (MP for short), constraints (7) guarantee that each task is assigned to one agent, and constraints (8) guarantee that one duty is selected for each agent. Both of them can be replaced by inequalities. Partitioning constraints (7) can be relaxed into covering constraints

$$\sum_{i\in\mathcal{N}} \sum_{d\in\mathcal{D}_i} \left(\sum_{k\in\mathcal{K}} x_{ijk}^d \right) z_i^d \geqslant 1 \quad \forall j \in \mathcal{M} \tag{10}$$

for the same reason outlined above. Because the empty duty (i.e., a duty with no assignments) is always feasible for each agent, we can replace constraints (8) with

$$\sum_{d\in\mathcal{D}_i} z_i^d \leqslant 1 \quad \forall i \in \mathcal{N}. \tag{11}$$

We consider a master problem with inequality constraints (10) and (11) instead of (7) and (8) because this makes it easier for the simplex algorithm to find feasible solutions when solving its linear relaxation.

In general, each set $\mathcal{D}_i$ includes an exponential number of assignments, and therefore the master problem has an exponential number of variables. We solve the linear relaxation of the master problem (LMP for short) by column

generation: We consider a restricted linear master problem (R-LMP) including only some subsets $\mathscr{D}'_i$ of columns, that is,

$$\min \sum_{i\in\mathscr{N}}\sum_{d\in\mathscr{D}'_i}\left(\sum_{j\in\mathscr{M}}\sum_{k\in\mathscr{K}} c_{ijk}x^d_{ijk}\right)z^d_i \tag{12}$$

$$\text{s.t.} \sum_{i\in\mathscr{N}}\sum_{d\in\mathscr{D}'_i}\left(\sum_{k\in\mathscr{K}} x^d_{ijk}\right)z^d_i \geqslant 1 \quad \forall j\in\mathscr{M}, \tag{13}$$

$$-\sum_{d\in\mathscr{D}'_i} z^d_i \geqslant -1 \quad \forall i\in\mathscr{N}, \tag{14}$$

$$z^d_i \geqslant 0 \quad \forall i\in\mathscr{N}\ \forall d\in\mathscr{D}'_i, \tag{15}$$

where constraints $z^d_i \leqslant 1$ have been removed because they are implied by constraints (14).

Let $\lambda\in\mathbb{R}^M_+$ and $\mu\in\mathbb{R}^N_+$ be the vectors of nonnegative dual variables corresponding to constraints (13) and (14), respectively. The reduced cost of duty d for agent i is

$$\bar{r}^d_i = \sum_{j\in\mathscr{M}}\sum_{k\in\mathscr{K}} c_{ijk}x^d_{ijk} - \sum_{j\in\mathscr{M}}\lambda_j\left(\sum_{k\in\mathscr{K}} x^d_{ijk}\right)+\mu_i.$$

To find columns with negative reduced cost, we must solve the following pricing problem for each agent $i\in\mathscr{N}$:

$$\min \ \bar{r}^d_i = \sum_{j\in\mathscr{M}}\sum_{k\in\mathscr{K}}(c_{ijk}-\lambda_j)x^d_{ijk}+\mu_i \tag{16}$$

$$\text{s.t.} \sum_{j\in\mathscr{M}}\sum_{k\in\mathscr{K}} a_{ijk}x^d_{ijk} \leqslant b_i \tag{17}$$

$$\sum_{k\in\mathscr{K}} x^d_{ijk} \leqslant 1 \quad \forall j\in\mathscr{M}, \tag{18}$$

$$x^d_{ijk}\in\{0,1\} \quad \forall j\in\mathscr{M},\ \forall k\in\mathscr{K}, \tag{19}$$

that is, a multiple-choice knapsack problem (MCKP). Although $\mathscr{NP}$-hard, the MCKP is well solvable in practice (see Martello and Toth 1990, Pisinger 1995), and this is a reason that makes our column generation approach to the MGAP particularly appealing.

The main computational advantage of the reformulation presented above is that the bound given by the linear relaxation of Model (6), (10), (11), (9) dominates that given by the linear relaxation of Model (1), (5), (3), (4). This is due to the convexification of constraints

$$\sum_{j\in\mathscr{M}}\sum_{k\in\mathscr{K}} a_{ijk}x^d_{ijk} \leqslant b_i,$$

$$\sum_{k\in\mathscr{K}} x^d_{ijk} \leqslant 1 \quad \forall j\in\mathscr{M}.$$

The MCKP polyhedra defined by these constraints do not possess the integrality property because the MCKP is $\mathscr{NP}$-hard. Therefore, their convexification yields a lower bound that is guaranteed to be greater than or equal to the linear programming lower bound (Martin 1999). In our experiments (see Table 1, §4), the lower bound provided by the reformulation used in our branch-and-price algorithm was actually tighter than the linear programming bound of the original model, which is used by general-purpose MIP solvers.

3. A Branch-and-Price Algorithm

3.1. Lower Bound and Termination

We exploit the equivalence between Lagrangean relaxation and Dantzig-Wolfe decomposition in the column generation termination test: The terms μ_i in the pricing problem (16)–(19) are not relevant in the definition of the optimal solution; hence, at each iteration t of column generation the current values of the dual variables $\boldsymbol{\lambda}^t$ are used as multipliers to compute a valid lower bound:

$$\omega^t = -\sum_{i\in\mathscr{N}}\tau^t_i + \sum_{j\in\mathscr{M}}\lambda^t_j,$$

where τ^t_i is the optimal value of the pricing subproblem for agent i. In this way, a sequence of lower bounds is computed during column generation. This often allowed us to prune the current node of the search tree even before column generation was over. When the gap between the optimal value of the R-LMP at iteration t and the best incumbent lower bound is smaller than a predefined threshold, the column generation algorithm is terminated and the best incumbent is kept as the final lower bound. This is useful to avoid undesired tailing-off effects in the column generation algorithm. In our experiments, we fixed the threshold to 10^{-6}.

3.2. Branching Strategy

One of the most challenging aspects in the design of a branch-and-price algorithm is the choice of the branching strategy: Besides partitioning the solution space, a good branching strategy must make infeasible the current optimal solution of the R-LMP, and it must not change the structure of the pricing subproblem. Many authors have addressed the issue of the design of effective branching strategies in branch-and-bound and branch-and-price algorithms. We refer the reader to Land and Doig (1960) and Barnhart et al. (1998) for detailed treatments of the subject.

We have devised a ternary branching rule that consists of selecting a task j^*, which has a fractional assignment to two or more agents in the optimal solution of the R-LMP. We forbid some of the assignments in each of two new subproblems, and we assign a task to a particular agent in the third subproblem.

For each task $j\in\mathscr{M}$, we consider the set M_j of agents for which there is a fractional assignment

$$f_{ij} = \sum_{d\in\mathscr{D}'_i}\sum_{k\in\mathscr{K}} x^d_{ijk}z^d_i$$

in the optimal solution of the R-LMP, and we select the agent $i^*_j = \arg\max_{i\in\mathscr{N}}\{f_{ij}\}$ corresponding to the highest fractional assignment for task j. The set $M_j\backslash\{i^*_j\}$ is partitioned into two subsets M^-_j and M^+_j in the following way: The agents in $M_j\backslash\{i^*_j\}$ are sorted by nonincreasing values of f_{ij}, and they are inserted alternately in M^+_j and M^-_j.

Agent i_j^* is inserted in both M_j^+ and M_j^-. In this way, we compute a heuristic solution to a subset sum problem to obtain a balanced partition of the agents in M_j. The idea is to fix the same number of variables in each branch, while trying to keep a balanced partition.

The set of agents to which task j is not assigned, $\widehat{M}_j$, is also partitioned into two subsets $\widehat{M}_j^- = \{i \in \mathcal{N}\colon f_{ij} = 0, i \leqslant \tilde{\imath}\}$ and $\widehat{M}_j^+ = \{i \in \mathcal{N}\colon f_{ij} = 0, i > \tilde{\imath}\}$, where $\tilde{\imath}$ is chosen in such a way that $|\widehat{M}_j^-| = \lceil |\widehat{M}_j|/2 \rceil$.

The task j^* selected for branching is the one for which $|M_j|$ is maximum. In case of ties, we select the task for which the partition obtained is most balanced, that is, $|\sum_{i \in M_j^-} f_{ij} - \sum_{i \in M_j^+} f_{ij}| / \sum_{i \in M_j} f_{ij}$ is minimum.

Then, we branch on j^* by setting

- $\sum_{i \in M_{j^*}^- \cup \widehat{M}_{j^*}^-} \sum_{k \in \mathcal{K}} x_{ij^*k} = 0$ in the first branch,
- $\sum_{i \in M_{j^*}^+ \cup \widehat{M}_{j^*}^+} \sum_{k \in \mathcal{K}} x_{ij^*k} = 0$ in the second branch, and
- $\sum_{k \in \mathcal{K}} x_{i_{j^*}^* j^* k} = 1$ in the third branch.

The addition of constraints in the first and second branches forbids some assignments, but it does not change the structure of the pricing problem. The constraint in the third branch is handled in a similar way: We forbid the assignment of task j^* to all agents but $i_{j^*}^*$, and we state as equality the j^*th constraint of set (18) in the pricing problem for agent i_j^*. This does not change the structure of the pricing problem.

We adopt a mixed search strategy. The third branch is always explored first, in a depth-first search fashion. This allows us to quickly reoptimize the LMP and to search for good primal solutions deep in the search tree. The subproblems in the first and second branches are stored as open nodes. Whenever an integer solution is found, or the dual bound for the subproblem exceeds the value of the best incumbent primal solution, the node with the lowest dual bound is retrieved from the open nodes list in a best-bound-first search fashion. A set of experiments showed that this branching strategy is more effective than a standard two-branches rule: First, the depth-first exploration of the third branch helps in quickly finding tight primal bounds; moreover, the assignment of the task to the most desirable agent is forbidden in both of the first two branches, and this helps in tightening the corresponding bounds.

When all the variables in a relaxed solution have integer values, the optimal task-level assignment is computed solving a MCKP for each agent.

3.3. Column Generation

Pricing Algorithm. We solve the binary MCKP to optimality by a modified version of Pisinger's algorithm (see Pisinger 1995) which combines dynamic programming with bounding and reduction techniques. This algorithm was devised for the MCKP with integer coefficients, while in our pricing subproblems the dual variables (as well as the multipliers in Lagrangean relaxation) can be fractional. Therefore, we modified the algorithm in a way similar to that described in Ceselli and Righini (2005) and Ceselli (2003), that is, by relaxing the bounding tests so that the solution computed by the algorithm may differ from optimality by at most a very small positive value ($n \cdot 10^{-9}$ in our experiments, where n is the number of variables left outside the core). Because the classical formulation of the MCKP has equality constraints, we add a set of N dummy elements, each appearing in a constraint of the set (18), corresponding to items with zero resource consumption and zero cost.

Column Management. At each iteration of the column generation algorithm, all columns that are generated with a negative reduced cost are inserted into the R-LMP.

Whenever the number of columns exceeds a limit, we remove columns from the R-LMP. According to statistical results (see §4.2), this limit was set to 3,000 in our experiments. The removal criterion depends on three different tests on the reduced cost of each column, so three types of removable columns are considered.

- A column is *red* if its reduced cost exceeds the gap between the best incumbent feasible solution and the lowest lower bound among all the open nodes of the search tree. In this case, the column cannot belong to an optimal solution of any node of the search tree, and therefore it is deleted.
- A column is *yellow* if its reduced cost exceeds the gap between the current R-LMP value and the Lagrangean lower bound. In this case, the column cannot belong to the optimal solution of the current node; the column is deleted from the R-LMP and is stored in a yellow pool P_y.
- A column is *green* if its reduced cost exceeds the same gap as above, divided by N. In this case, the column can belong to the optimal solution of the current node; it is removed from the R-LMP and is stored in a green pool P_g.

Because every column is related to a particular agent, each pool is partitioned into N subpools. The green pool P_g is scanned before executing the pricing algorithm, also at the node of the search tree in which the deletion has occurred: If any column with negative reduced cost is found, it is inserted into the R-LMP. The columns in the yellow pool P_y are considered for reinsertion only in subsequent nodes of the search tree.

Finally, to avoid an excessive growth of the pools, the columns are erased from the pool when their reduced cost is nonnegative for a certain number of consecutive evaluations. This parameter was tuned to a value of six in our experiments (see §4.2).

Initialization. To guarantee that a feasible solution of the R-LMP exists in each node of the search tree, a dummy column is inserted into the initial R-LMP; it corresponds to a duty in which all tasks are executed by a dummy agent with infinite capacity. The cost of such a column is set to a very high value, that is, $\sum_{j \in \mathcal{M}} \max_{i \in \mathcal{N}, k \in \mathcal{K}} \{c_{ijk}\}$. Moreover, 11 sets of columns are inserted into the initial R-LMP at the root node, corresponding to primal solutions produced by heuristic algorithms. A detailed description of this initialization is reported in §3.4.

To obtain a warm start, in each nonroot node the R-LMP is initialized with the feasible columns of the most recently solved node, plus all columns from the pools that have negative reduced cost when they are evaluated with the optimal dual values of the father node.

3.4. Primal Bounds

The problem of finding a feasible solution to the MGAP is $\mathcal{NP}$-complete. Nevertheless, we search for feasible solutions at every node of the search tree because the availability of good primal bounds may considerably reduce the overall computing time needed to reach a provably optimal solution.

We devised a fast rounding heuristic, and implemented both heuristic algorithms, MGAPH1 and MGAPH2, proposed by French and Wilson (2002). Furthermore, we propose a new local search neighborhood (that we call *SHIFT*) and a modification of the local search technique used for MGAPH2 (that we call *SWAP*). In the following paragraphs, we outline these algorithms and two local search techniques. Then, we describe how each heuristic is used in the branch-and-price algorithm.

For a formal description of MGAPH1 and MGAPH2, we refer to the original paper, while the complete pseudocodes of the rounding heuristic and the local search algorithms are reported in the appendix.

Rounding Heuristic. Let $f_{ij} = \sum_{d \in \mathcal{D}_i} (\sum_{k \in \mathcal{K}} x^d_{ijk}) z^d_i$ be the (possibly fractional) assignment of task j to agent i corresponding to the fractional R-LMP solution defined by the z^d_i variables. First, each task is assigned to the agent for which f_{ij} is maximum. Let C_i be the resulting set of tasks assigned to agent i. Second, for every agent i an MCKP with integer coefficients is solved to optimality by the algorithm of Pisinger (see Pisinger 1995). If a feasible solution can be found for each agent, a primal bound for MGAP is obtained; otherwise, the heuristic fails. We observed that in almost all cases, this method finds a feasible solution.

MGAPH1. This algorithm consists of two steps. First, a superoptimal integer solution is built with a greedy approach. For each task i, the agent-level assignment (i, j, k) with the lowest c_{ijk} is selected, possibly violating some capacity constraint. Then, a local search for feasible solutions is performed, shifting tasks from overloaded agents to agents with enough residual resources. The shift corresponding to the minimum increase in the solution value per resource consumption unit is iteratively selected.

MGAPH2. Consider $g_{ijk} = \sum_{d \in \mathcal{D}_i} x^d_{ijk} z^d_i$. In a construction step, a value $r_j = g_{i'jk'} - g_{i''jk''}$ is computed for each task, where (i', j, k') and (i'', j, k'') are the first and the second agent-level assignments for task j that do not violate capacity constraints with the highest g_{ijk} values. The task with the highest r_j is selected, and the (i', j, k') assignment is made. If, due to capacities, a task is found that cannot be assigned to any agent, the heuristic fails. Otherwise, a local search is performed, considering a neighborhood made of all solutions that can be obtained from the current one by swapping two tasks assigned to different agents.

Local Search. First, we propose a *SHIFT* procedure: The neighborhood of the current solution is made of all solutions that can be obtained by shifting a task from an agent to another or from an efficiency level to another. As detailed in the appendix, we consider in turn each task, each agent, and each efficiency level on that agent. The elements in each set are considered in random order. Whenever an improving move is found, it is immediately performed with a first-improve policy. The cost of this local search step is $O(NMK)$. Second, we modified the pairwise swap neighborhood of French and Wilson (2002), obtaining a procedure that we call *SWAP* in the following way. We consider all solutions that can be obtained by swapping two tasks assigned to different agents, or by swapping the efficiency levels of two tasks assigned to the same agent. Only the best improving swap is performed at each iteration. The cost of this improving phase is $O(M^2K^2)$ for each iteration.

Our rounding heuristic, coupled with the exploration of the *SHIFT* neighborhood, was run at every column generation step. It yielded good upper bounds even in the earlier iterations with a low computational cost, and this was useful also to drive the column removal routine. MGAPH2 was used once for each node in the search tree, using the optimal R-LMP solution. We modified the local search step of MGAPH2 in the following way: First, we explore the *SHIFT* neighborhood and the first-improving shift is made, until no more improving shifts can be found. We then explore the *SWAP* neighborhood and the best improving swap is made until no more improving swaps can be found. The exploration of the *SHIFT* and the *SWAP* neighborhoods is iterated until no more improving moves can be made. The neighborhood *SWAP* is considered only at the root node. The rounding heuristic and the MGAPH1 algorithm were used in the initialization of the R-LMP: We chose to generate 25 sets of columns using the former, randomly drawing each f_{ij} in the interval $[0, 1)$ (see §4.2). The details on how the random values were generated are reported in §4. Also, columns corresponding to infeasible solutions were added to the R-LMP.

4. Experimental Analysis

4.1. Test Instances

We tested the branch-and-price algorithm on three classes of instances. Classes C and D are generated using random generators as described by Martello and Toth for the GAP, and extended to the MGAP, while Class E is generated as proposed by Laguna et al. (1995).

- **Class C**: *uncorrelated resource consumption and cost.* a_{ijk} is taken as a random integer from a uniform distribution in $[5, \ldots, 25]$.

c_{ijk} is taken as a random integer from a uniform distribution in $[1, \ldots, 40]$.

$b_i = 0.8 \sum_{i \in \mathcal{N}} \sum_{k \in \mathcal{K}} a_{ijk}/(NK)$.

- **Class D:** *strongly correlated resource consumption and cost.*

a_{ijk} is taken as a random integer from a uniform distribution in $[1, \ldots, 100]$.

c_{ijk} is taken as a random integer from a uniform distribution in $[101 - a_{ijk}, \ldots, 121 - a_{ijk}]$.

$b_i = 0.8 \sum_{i \in \mathcal{N}} \sum_{k \in \mathcal{K}} a_{ijk}/(NK)$.

- **Class E:** *resource consumption and cost correlated through an exponential distribution.*

a_{ijk} is randomly generated as $1 - 10 \ln[\text{random}(0, 1]]$ rounded to the nearest integer with probability p, $a_{ijk} = \infty$ (that is, the assignment of task j to agent i at level k is forbidden) with probability $1 - p$.

c_{ijk} is randomly generated as $1{,}000/a_{ijk} - 10 \cdot \text{random}(0, 1]$ rounded to the nearest integer.

$b_i = \max\{0.8 \sum_{i \in \mathcal{N}} \sum_{k \in \mathcal{K}} a_{ijk}/(NK), \max_{j,k}\{a_{ijk}\}\}$.

By "random(0, 1]" we mean a random rational value uniformly drawn in the interval $(0, 1]$. Such a random value was generated by drawing a random signed integer and dividing this by the constant value "INT_MAX" (on our machine, four bytes are used for signed integers, and "INT_MAX" is set to $2^{31} - 1$). We generated 215 test instances in the following way. For each class, we generated two problem sets, with $M = 100$ and $M = 200$ tasks. For $M = 100$, we considered a number of agents N equal to 10, 20, and 30 and a number of levels K equal to 3, 4, and 5; for $M = 200$, we considered a number of agents N equal to 15 and 30 and a number of levels K equal to 4 and 5. For the instances in Class E with $M = 100$, the probability p of allowing an agent-task-level assignment was fixed to 1.0, while for the instances in Class E with $M = 200$, the case $p = 0.8$ and the case $p = 1.0$ have been considered. Each combination is reported in the first four columns of the tables reported in this section, and consists of five instances; hence, each row of the tables reports the average results for these five instances.

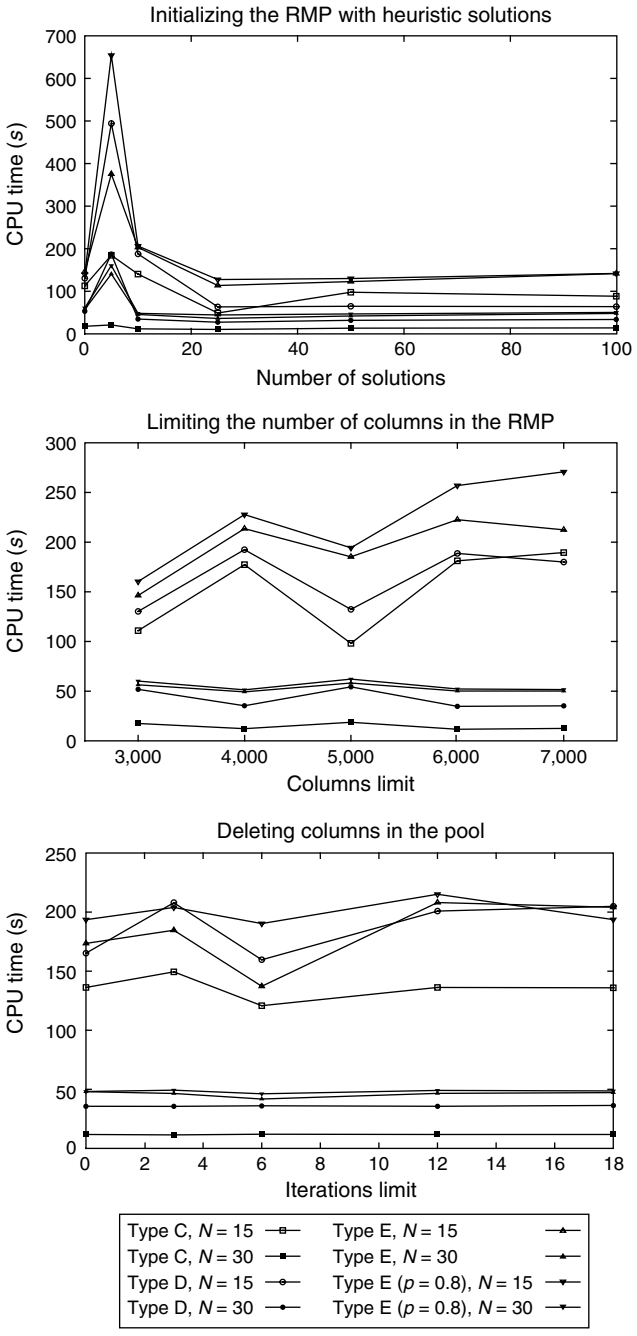
As reported in §2, dominated assignments can be found by simple preprocessing tests. In fact, about 50% of the binary variables were fixed to zero for the Class C instances, about 11% for Class D, and about 8% for Class E. As expected, the percentage of fixed variables increases as the number of levels K increases.

The branch-and-price algorithm was coded in C++ and compiled under Linux OS with gcc version 2.96 with full optimizations. CPLEX 6.5.3 was used as an LP solver. All tests were run on a PC equipped with an Intel P4 1,600 MHz CPU and 512 MB RAM. Each test was stopped in case of memory overflow or whenever a time-out of two hours was exceeded.

4.2. Parameter Tuning

As discussed in the previous sections, three parameters affect the computational performance of the algorithm. They are: the number of columns generated to populate the initial RMP, the threshold on the number of columns in the RMP that triggers the columns removal routine, and the number of iterations in which a column is kept in the columns pool. To tune these parameters, we considered the CPU time needed to solve the root problem of the instances involving 200 tasks. In Figure 1, we include three charts, in which different settings for each parameter are compared. In each chart, the values in the abscissae correspond to the different settings of the parameter, and the values in the ordinates are the corresponding computation time.

Figure 1. Tuning the algorithm parameters.



We observed that the computation time did not change significantly for different numbers of levels. Therefore, in each chart we present eight series, considering each combination of the correlation type and the number of agents, and reporting the average results on the corresponding instances. First, we tried to initialize the RMP with the dummy column only, or with columns corresponding to 5, 10, 25, and 100 primal solutions. It is worth noting that the worst performance corresponds to the initialization of the RMP with only a few columns: When considered as constraints in the dual problem, these columns restrict the solution space, leaving the dual variables free to assume misleading values. Setting this parameter to 25 seems to be the most appropriate choice. Second, we tried to set the columns-removal threshold to 3,000, 4,000, 5,000, 6,000, and 7,000. Even if sometimes a setting to 5,000 columns would be better (instances of type C, with 15 agents), fixing the threshold to 3,000 gives the best average CPU time. Third, we tuned the iterations limit on the columns pool by setting it to zero, which means not managing a pool of removed columns, 3, 6, 12, and 18. The value of six was found to be the best. A slight increase in the computation time was observed moving from zero to three, which reflects the overhead for managing the pool.

4.3. Computational Results

In the first set of tests, the quality of the bound given by the LMP relaxation is compared to the bound given by the LP relaxation, which is used by CPLEX and other general-purpose solvers. In Table 1 we report, for both methods, the average integrality gap (int. gap)—that is, the gap $(v^* - \omega)/v^*$ between the relaxation value ω and the optimal (or the best-known) integer solution value v^*; and the average number of fractional agent-level assignments for each task (fract.)—that is, the ratio between the number of nonzero variables in an optimal fractional solution and M, the number of tasks.

Both relaxations yield a rather tight bound, but the LMP relaxation clearly dominates the LP relaxation, also from an experimental point of view. Nevertheless, the time required to obtain the LMP bound is one order of magnitude (sometimes even two orders) higher than the time required to compute the LP relaxation.

The ratio between the LMP integrality gap and the LP integrality gap increases as the ratio M/N decreases and decreases as K increases. This was expected because the M/N value represents the average number of tasks assigned to the same agent. As reported by Martello and Toth (1990), at most two variables can assume fractional values in the linear relaxation of an MCKP problem; these two fractional variables must belong to the same multiple-choice constraint (18). Hence, when a high number of tasks is assigned to the same agent in a fractional solution, several task-level assignments take an integer value, and the convexification of constraints (17)–(18) has a minor effect. Second, a high K corresponds to a high number of binary variables in the same multiple-choice constraint. This does not affect the number of variables with a fractional value; thus, following the previous argumentation, the convexification has a minor effect. On the opposite, for both relaxations the number of nonzero variables in a fractional solution increases as the ratio M/N decreases. On the average, the LMP solution has more fractional assign-

Table 1. Comparison between LP relaxation and LMP relaxation.

Instances				Linear relaxation		LMP relaxation	
Correlation	N	M	K	Int. gap (%)	Fract.	Int. gap (%)	Fract.
C	10	100	3	2.35	1.10	0.34	1.23
	10	100	4	2.41	1.10	0.45	1.33
	10	100	5	1.82	1.09	0.39	1.22
	20	100	3	4.59	1.19	0.38	1.31
	20	100	4	3.32	1.19	0.40	1.43
	20	100	5	3.10	1.19	0.40	1.41
	30	100	3	5.96	1.29	0.34	1.41
	30	100	4	3.73	1.28	0.35	1.50
	30	100	5	0.36	1.29	0.00	1.83
Avg. (C)				3.07	1.19	0.34	1.41
D	10	100	3	0.12	1.10	0.06	1.73
	10	100	4	0.12	1.10	0.09	1.71
	10	100	5	0.06	1.10	0.06	2.03
	20	100	3	0.67	1.20	0.60	2.45
	20	100	4	0.53	1.20	0.52	2.67
	20	100	5	0.43	1.20	0.43	3.32
	30	100	3	1.06	1.30	0.91	2.81
	30	100	4	0.56	1.30	0.55	2.87
	30	100	5	0.50	1.29	0.50	3.37
Avg. (D)				0.45	1.20	0.41	2.55
E	10	100	3	0.16	1.10	0.02	1.43
	10	100	4	0.13	1.10	0.00	1.13
	10	100	5	0.11	1.09	0.00	1.21
	20	100	3	0.65	1.19	0.03	1.71
	20	100	4	0.56	1.20	0.02	1.78
	20	100	5	0.63	1.20	0.03	1.77
	30	100	3	1.86	1.29	0.07	2.32
	30	100	4	1.64	1.30	0.08	2.20
	30	100	5	1.53	1.29	0.07	2.13
Avg. (E)				0.81	1.19	0.04	1.74
C	15	200	4	1.07	1.08	0.23	1.25
	15	200	5	1.00	1.07	0.20	1.26
	30	200	4	0.19	1.14	0.09	2.07
	30	200	5	0.00	1.15	0.00	2.74
Avg. (C)				0.57	1.11	0.13	1.83
D	15	200	4	0.15	1.08	0.15	2.56
	15	200	5	0.11	1.07	0.11	3.13
	30	200	4	0.66	1.15	0.66	4.13
	30	200	5	0.46	1.15	0.46	4.71
Avg. (D)				0.34	1.11	0.34	3.63
E ($p = 0.8$)	15	200	4	0.08	1.01	0.01	1.31
	15	200	5	0.07	1.01	0.00	1.32
	30	200	4	0.31	1.01	0.01	1.60
	30	200	5	0.29	1.01	0.01	1.58
Avg. (E, p 0.8)				0.19	1.01	0.01	1.45
E ($p = 1.0$)	15	200	4	0.07	1.07	0.01	1.32
	15	200	5	0.06	1.07	0.01	1.25
	30	200	4	0.27	1.14	0.01	1.42
	30	200	5	0.27	1.14	0.01	1.57
Avg. (E, p 1.0)				0.17	1.11	0.01	1.39

ments than the LP solution. This is especially evident for instances in Class D.

In Table 2, we report computational results for our method. The table consists of two horizontal blocks. The first refers to the root node and the second to the nodes of the search tree. We indicate $\bar{v}$ the value of the best primal solution found at the root node, v^* the optimal solution value (or the best-known primal bound, when optimality was not proved), $\bar{\omega}$ the optimal LMP value at the root node, and ω^* the dual bound at the end of computation. In the "root node" block, we report the value $(\bar{v} - v^*)/v^*$, which indicates how far the initial primal bound is from optimality (primal gap); the value $(\bar{v} - \lceil\bar{\omega}\rceil)/\lceil\bar{\omega}\rceil$—that is, the gap between the primal and dual bound at the root node (p.d. gap); the number of column generation iterations needed to reach a LMP optimum (iter.), the number of generated columns (cols); and the time spent at the root node. In the whole search tree block we indicate the number of nodes evaluated (ev. nodes), the gap between the primal and dual bound at the end of computation $(v^* - \lceil\omega^*\rceil)/\lceil\omega^*\rceil$ (gap), the number of instances solved to proven optimality (opt), and the overall time spent (time) in the exploration of the whole search tree (including the root node).

By looking at the rightmost three columns of the root node block, it can be noticed that the number of column generation iterations and the time required to solve the relaxation decrease as the number of agents increases. In fact, the insertion of new columns in the R-LMP is governed by an "all-negative" policy: A new column with negative reduced cost can be found for each agent. A larger number of agents means a larger number of columns added to the R-LMP at each column generation iteration, and thus faster convergence.

By analyzing the computational results for the whole search tree, it is clear that instances in Class C can be solved very easily: Optimality was proven at the root node for all instances but one. This was expected because resource consumption and costs are not correlated, and an optimal solution can quickly be found by choosing assignments with low resource consumption and low cost.

All the instances in Class E were solved to optimality in a few minutes: Often a very tight primal bound was found at the root node, and the gap between primal and dual bounds was closed by exploring a few nodes of the search tree. This shows that our branching rule is effective for these kinds of instances.

Effect of Symmetries. Instances in Class D are indeed the hardest ones. A singular effect was experimentally observed: Tight primal and dual bounds were obtained at the root node, but the gap could not be closed after many branching steps. We explain this phenomenon with the following observation. Let the *efficiency* of an assignment (i, j, k) be the ratio $1/a_{ijk}c_{ijk}$. Because resource consumption and cost are strongly correlated, several assignments have a similar efficiency. Suppose that a task j is fractionally assigned to agent i in the optimal LMP solution of a node in the search tree, and a branching operation forbids this fractional assignment. Even if agents are not identical, an equivalent solution is likely to exist, in which the fractional part of task j is assigned to a different agent i' at the same efficiency, maybe by shifting to agent i the fractional part of another task j' previously allocated to agent i'. Hence, a fractional optimal solution after branching would be a simple rearrangement of the tasks between the agents. This effect is mitigated in Class E instances because the correlation between resource consumption and cost is not linear.

This further motivates the depth-first search feature of our branching rule: The gap can be closed just by finding the optimal integer solution, which is more likely to be found deep in the search tree than at the root node, as fixing task-agent assignments to one has a strong effect in the construction of an integer solution.

Large-Scale Instances. To test our algorithm on a more challenging testbed, we considered the set of instances presented by Yagiura et al. (1998). These instances correspond to GAP problems generated with C, D, and E correlation types, in which up to 1,600 tasks must be assigned to up to 80 agents. We considered the problems involving the assignment of 400, 900, and 1,600 tasks. To obtain a set of corresponding MGAP instances, we put the tasks on two, three, and four levels for the instances with 400, 900, and 1,600 tasks, respectively, and adjusted the capacity of the agents by dividing each value by two, three, and four correspondingly. We imposed no time limitation on these tests. The results obtained by our branch and price on these modified instances are reported in Table 3. In the first horizontal block of this table, we report the original GAP instance ID and the number of agents, tasks, and levels of the corresponding MGAP instance. The second block corresponds to the optimization status at the root node. We indicate how far the primal solution found at the root node is with respect to the best-known primal solution, the gap between the primal and dual solutions and the time spent. The third block refers to the overall branching tree and contains the gap between the primal and dual bounds at the end of computation, the number of explored nodes, and the time required to obtain a proven optimal solution. The last two columns were marked with a dash when the process ran out of memory. We observed the same behavior of the previous analysis: Branch-and-price was able to solve all the instances with correlation type C and E; on the other hand, it failed to solve to proven optimality the instances with correlation type D, even though the gap between primal and dual bounds was very small.

Further Testing. As discussed in the previous paragraph, instances in Class D remain challenging. While it is computationally easy to find a very tight primal bound, it is hard to identify an optimal integer solution. We also measured the Hamming distance between the primal solution found by our heuristics at the root node and the best primal

Table 2. Experimental results for the branch-and-price algorithm.

Instances				Root node					Whole search tree			
Corr.	N	M	K	Primal gap (%)	P.d. gap (%)	Iter.	Cols.	Time(s)	Ev. nodes	Gap (%)	Opt.	Time(s)
C	10	100	3	0.00	0.00	166.20	1,702.40	5.19	0.00	0.00	5	5.19
	10	100	4	0.00	0.00	152.80	1,684.40	5.14	0.00	0.00	5	5.14
	10	100	5	0.00	0.00	150.00	1,619.40	4.77	0.00	0.00	5	4.77
	20	100	3	0.00	0.00	49.20	1,315.20	1.62	0.00	0.00	5	1.62
	20	100	4	0.00	0.00	42.40	1,222.40	1.58	0.00	0.00	5	1.58
	20	100	5	0.00	0.00	42.40	1,196.20	1.61	0.00	0.00	5	1.61
	30	100	3	0.00	0.00	25.20	1,329.00	1.18	0.00	0.00	5	1.18
	30	100	4	0.98	0.16	21.20	1,242.60	1.26	1.20	0.00	5	1.60
	30	100	5	0.00	0.00	14.80	1,142.80	1.28	0.00	0.00	5	1.28
Avg. (C)				0.11	0.02	73.80	1,383.82	2.62	0.13	0.00	45	2.66
D	10	100	3	0.46	0.49	161.00	1,741.80	6.92	6,110.00	0.02	3	2,102.17
	10	100	4	0.54	0.57	162.20	1,763.20	7.49	6,669.80	0.00	4	3,324.51
	10	100	5	0.54	0.54	123.20	1,473.00	6.20	3,133.80	0.00	5	1,678.22
	20	100	3	1.62	1.76	58.80	1,527.60	3.36	10,217.60	0.12	0	—
	20	100	4	1.52	1.59	50.40	1,469.60	3.38	9,584.20	0.07	0	—
	20	100	5	1.58	1.63	41.20	1,325.00	3.21	11,828.80	0.04	1	6,608.36
	30	100	3	1.63	1.80	37.20	1,700.40	2.78	7,775.00	0.15	0	—
	30	100	4	1.37	1.51	34.00	1,680.40	2.95	9,740.80	0.13	0	—
	30	100	5	1.16	1.22	28.00	1,576.60	2.94	10,950.20	0.06	0	—
Avg. (D)				1.16	1.24	77.33	1,584.18	4.36	8,445.58	0.07	13	3,428.31
E	10	100	3	0.41	0.42	181.80	1,933.60	7.78	66.00	0.00	5	63.52
	10	100	4	0.04	0.04	192.60	2,026.40	9.06	1.20	0.00	5	10.73
	10	100	5	0.28	0.28	189.00	1,982.00	9.02	4.20	0.00	5	14.40
	20	100	3	1.08	1.10	65.20	1,617.40	3.51	123.60	0.00	5	53.89
	20	100	4	0.52	0.54	65.00	1,620.80	3.87	173.40	0.00	5	80.72
	20	100	5	0.95	0.97	68.00	1,636.60	4.39	136.80	0.00	5	68.59
	30	100	3	1.84	1.95	32.40	1,519.00	2.43	1,698.00	0.00	5	597.99
	30	100	4	1.03	1.10	33.00	1,567.40	2.91	711.60	0.00	5	274.41
	30	100	5	0.92	0.98	34.40	1,585.60	3.38	478.80	0.00	5	190.87
Avg. (E)				0.79	0.82	95.71	1,720.98	5.15	377.07	0.00	45	150.57
Avg. $M = 100$				0.68	0.69	82.28	1,562.99	4.04	2,940.93	0.02	103	1,193.85
C	15	200	4	0.00	0.00	357.80	2,548.20	87.46	0.00	0.00	5	87.46
	15	200	5	0.00	0.00	356.00	2,131.40	54.79	0.00	0.00	5	54.79
	30	200	4	0.00	0.00	47.00	2,024.60	10.15	0.00	0.00	5	10.15
	30	200	5	0.00	0.00	27.60	1,579.00	10.73	0.00	0.00	5	10.73
Avg. (C)				0.00	0.00	197.10	2,070.80	40.78	0.00	0.00	20	40.78
D	15	200	4	0.72	0.75	182.20	2,160.00	68.73	5,278.20	0.02	0	—
	15	200	5	0.53	0.54	162.00	2,767.00	58.26	4,111.60	0.01	3	1,429.00
	30	200	4	1.19	1.25	63.60	2,659.00	29.70	5,289.00	0.06	0	—
	30	200	5	0.94	0.96	60.00	2,545.00	25.18	7,500.80	0.02	0	—
Avg. (D)				0.85	0.88	116.95	2,532.75	45.47	5,544.90	0.03	3	1,429.00
E ($p = 0.8$)	15	200	4	0.37	0.37	428.80	2,779.80	123.18	16.20	0.00	5	287.66
	15	200	5	0.38	0.38	412.40	1,627.80	104.06	30.00	0.00	5	433.27
	30	200	4	0.64	0.65	128.80	1,711.80	45.11	148.20	0.00	5	419.52
	30	200	5	0.41	0.41	137.40	1,720.40	45.87	47.40	0.00	5	206.73
Avg. (E, 0.8)				0.45	0.45	276.85	1,959.95	79.55	60.45	0.00	20	336.79
E ($p = 1.0$)	15	200	4	0.42	0.42	390.00	2,219.00	106.92	53.40	0.00	5	975.23
	15	200	5	0.23	0.23	405.00	1,710.20	148.19	58.80	0.00	5	1,037.15
	30	200	4	0.41	0.41	128.00	1,633.20	44.52	46.80	0.00	5	199.19
	30	200	5	0.28	0.28	130.40	1,689.80	44.47	86.40	0.00	5	271.30
Avg. (E, 1.0)				0.33	0.34	263.35	1,813.05	86.03	61.35	0.00	20	620.71
Avg. $M = 200$				0.41	0.42	213.56	2,094.14	62.96	1,416.68	0.01	63	606.82

Table 3. Testing branch-and-price on large-size instances.

Instances					Root node			Whole search tree		
ID	Corr.	N	M	K	Primal gap (%)	P.d. gap (%)	Time(s)	Final gap (%)	Ev. nodes	Time(s)
c10400	C	10	200	2	43.51	43.51	643.21	0.00	6	757.69
c20400		20	200	2	0.64	0.69	111.75	0.00	30	278.82
c40400		40	200	2	0.54	0.54	20.35	0.00	12	50.25
c15900		15	300	3	0.09	0.09	4,514.97	0.00	6	4,854.12
c30900		30	300	3	0.10	0.10	372.94	0.00	90	2,078.95
c60900		60	300	3	0.00	0.00	26.40	0.00	0	31.95
c201600		20	400	4	0.07	0.07	5,381.14	0.00	3	5,877.77
c401600		40	400	4	0.00	0.00	203.58	0.00	0	213.42
c801600		80	400	4	0.00	0.00	43.99	0.00	0	61.22
Avg.					4.99	5.00	1,257.59	0.00	16.33	1,578.24
d10400	D	10	200	2	0.46	0.49	526.17	0.03	—	—
d20400		20	200	2	0.51	0.58	86.22	0.07	—	—
d40400		40	200	2	1.33	1.48	23.48	0.15	—	—
d15900		15	300	3	0.63	0.65	2,576.19	0.02	—	—
d30900		30	300	3	1.10	1.19	301.49	0.09	—	—
d60900		60	300	3	0.93	1.05	101.30	0.12	—	—
d201600		20	400	4	0.00	0.76	10,606.39	0.76	—	—
d401600		40	400	4	0.00	0.74	649.28	0.74	—	—
d801600		80	400	4	1.00	1.07	218.10	0.07	—	—
Avg.					0.66	0.89	1,676.51	0.23	—	—
e10400	E	10	200	2	1.41	1.42	670.55	0.00	804	17,473.80
e20400		20	200	2	0.21	0.21	94.91	0.00	3	121.89
e40400		40	200	2	1.96	1.97	27.38	0.00	39	151.86
e15900		15	300	3	0.03	0.03	4,732.69	0.00	15	6,607.22
e30900		30	300	3	0.85	0.85	350.93	0.00	72	1,759.14
e60900		60	300	3	0.02	0.02	130.60	0.00	12	313.62
e201600		20	400	4	0.15	0.15	12,446.65	0.00	3	12,996.30
e401600		40	400	4	0.02	0.02	875.21	0.00	45	3,188.50
e801600		80	400	4	0.17	0.17	276.39	0.00	213	2,685.78
Avg.					0.54	0.54	2,178.37	0.00	134.00	5,033.12

solution encountered while exploring the search tree. Even if a small improvement is made in the solution value, an average distance higher than 100 and 250 is observed for Class D instances with $M = 100$ and $M = 200$, respectively (for Class E, such distance is about 45 and 65). The symmetry in the LMP optimal solutions suggests the presence of symmetries also in the integer optimal solutions. However, even though symmetric optima could exist, which are less far apart from the initial solution, it would still be hard to cover such a large distance with standard local search methods.

Finally, we remark that the number of efficiency levels does not affect the performance of our method: Branch-and-price was able to solve instances involving up to 30 efficiency levels. Although no problem with so high a number of efficiency levels is addressed in the literature, these kinds of instances could arise as a discretization of some nonlinear resource consumption function.

4.4. Benchmark Algorithms

General Purpose Solver. As a first term of comparison, we present the behavior of ILOG CPLEX 6.5.3 when used as an IP solver to optimize the MGAP. CPLEX uses a branch-and-cut approach, automatically generating cliques, cover, and GUB-cover inequalities to strengthen the LP relaxation. All internal parameters were kept at their default values. These include a relative optimality tolerance of 0.01%. While instances in Class C were handled quite easily, CPLEX was able to solve only a small number of instances in Class E, and no instance in Class D, mainly due to memory overflow problems.

Logic Cuts. Logic cuts are a class of inequalities, analogous to cover cuts, that can be obtained in a preprocessing phase from the capacity constraints (3). A similar generation of valid cover cuts in a preprocessing phase has recently been adopted by Nauss (2003) for the case of the GAP. The use of logic cuts to improve the performances of a general-purpose solver for the MGAP is discussed by

Osorio and Laguna (2003). As proposed by the authors, we added all nondominated 1-cuts to the model of MGAP and used CPLEX to solve the new formulation. The 1-cuts can be generated in linear time with a simple procedure.

According to the computational experience described by Osorio and Laguna (2003), all CPLEX parameters were kept at their default values, except for the search policy parameter, whose setting was changed from "best-bound-first" to "up-branch-first."

The introduction of logic cuts yielded a substantial improvement to the performances of CPLEX for small instances: The size of the search tree was strongly reduced, avoiding the memory overflow problems. However, the introduction of these inequalities enlarged the dimension of the problem considerably, and the computation time was substantially increased.

To assess the effectiveness of the heuristics and local search methods used for branch-and-price, we tried to incorporate them into the optimization process of CPLEX. We kept the same settings used for branch-and-price: We ran MGAPH2 and considered the SWAP neighborhood only at the root node, while MGAPH1 with the SHIFT neighborhood was used once at each node of the branching tree.

Experimental Results. In Table 4, we compare the performances of the four methods. The table is divided into four horizontal blocks; each block refers to a solution strategy, which is indicated in the leading row. For each method, we report the average time to complete computation (when optimality was proven), the number of instances solved to proven optimality, and the average gap between the primal and dual bounds, when computation exceeded resource limitation. As outlined before, even if the introduction of logic cuts solves the memory problems, the convergence of the solver is slow. The embedding of the primal heuristics yielded a further improvement to the performances of CPLEX, but the overall behavior of the method was still the same. This test highlighted that the bounding and branching techniques are of key importance in the algorithmic success of branch-and-price. Branch-and-price clearly outperformed the other three methods: This is especially evident for instances in Class E, where CPLEX was able to prove the optimality of only nine of the instances with $M = 100$, the introduction of logic cuts allowed to solve two more instances, while our algorithm always provided the optimal solution. Instances in Class D are difficult for both CPLEX and our algorithm. However, our method allows us to obtain a smaller gap between the final upper and lower bounds.

A Hard Instance. Laguna et al. (1995) described a hard instance involving 30 tasks, eight agents, and three efficiency levels. The best solution found by their tabu search method after 120.7 seconds (on a DECstation 5000/120 MHz) has a cost equal to 691,634. French and Wilson (2002) tested their heuristics on this instance, finding solutions of cost equal to 714,608 and 703,912 within 0.2 and 0.1 seconds (on a HP9000/700 MHz), with MGAPH1 and MGAPH2, respectively. Osorio and Laguna (2003), using CPLEX with logic cuts and allowing a relative optimality tolerance of 1.0%, found a solution of value 690,624 in 36 hours of computation (on a machine equipped with a Pentium 100 MHz CPU). We started branch-and-price from scratch; that means we did not exploit information about the previously known upper bounds. It solved this instance to proven optimality in about three hours of computation, confirming the value of the optimal solution to be 690,624.

4.5. Conclusions

In this paper, we have described a branch-and-price algorithm for the MGAP. At the best of our knowledge, this is the first exact method proposed in the literature for the MGAP. It compares favorably against a state-of-the-art general-purpose MIP solver, and it was able to prove optimality for a very hard MGAP instance that had not been solved to proven optimality so far. Its success mainly relies upon the tightness of the set-covering reformulation, the existence of an effective algorithm to solve the MCKP, and a branching rule that does not affect the combinatorial structure of the pricing subproblem. Following an up-branch-first search strategy, the exact optimization algorithm presented here is also suitable for approximation purposes when very large-scale MGAP instances are tackled.

Appendix

Rounding Heuristic

Input: $z_i^d \ \forall i \in \mathcal{N} \ \forall d \in \mathcal{D}'_i$
(*the solution of R-LMP*)
Output: $i(j) \in \mathcal{N}$ (agent assignment) and
$k(j) \in \mathcal{K}$ (level assignment) $\forall j \in \mathcal{M}$
(*a feasible solution for the MGAP*)

(*Initialization*)
forall $i \in \mathcal{N}$ do
$C^i := \varnothing$; $q_i := b_i$

(*Step* 1: *task-agent assignment*)

$$f_{ij} = \sum_{d \in \mathcal{D}'_i} \left(\sum_{k \in \mathcal{K}} x_{ijk}^d \right) z_i^d \ \forall i \in \mathcal{N} \ \forall j \in \mathcal{M}$$

forall $j \in \mathcal{M}$ do
$I_j := \{i \mid q_i \geqslant \min_{k \in \mathcal{K}} \{a_{ijk}\}\}$
if $I_j = \varnothing$ then FAIL
else
$i(j) := \arg\max_{i \in I_j} \{f_{ij}\}$
$C^{i(j)} := C^{i(j)} \cup \{j\}$; $q_{i(j)} := q_{i(j)} - \min_{k \in \mathcal{K}} \{a_{i(j)jk}\}$

(*Step* 2: *task-level assignment*)
forall $i \in \mathcal{N}$ do
Solve to optimality the MCKP:

Table 4. Comparison between CPLEX 6.5.3 and branch-and-price.

Instances				CPLEX 6.5.3			CPLEX + Logic cuts			CPLEX + LC + Heurs			Branch-and-price		
Corr.	N	M	K	Gap (%)	Opt.	Time(s)	Gap (%)	Opt.	Time(s)	Gap (%)	Opt.	Time(s)	Gap (%)	Opt.	Time(s)
C	10	100	3	0.00	5	3.61	0.00	5	6.56	0.00	5	4.10	0.00	5	5.19
	10	100	4	0.00	5	8.20	0.00	5	11.00	0.00	5	8.16	0.00	5	5.14
	10	100	5	0.00	5	5.44	0.00	5	7.50	0.00	5	6.60	0.00	5	4.77
	20	100	3	0.00	5	4.49	0.00	5	7.92	0.00	5	6.60	0.00	5	1.62
	20	100	4	0.00	5	6.25	0.00	5	7.96	0.00	5	7.65	0.00	5	1.58
	20	100	5	0.00	5	12.64	0.00	5	15.03	0.00	5	14.06	0.00	5	1.61
	30	100	3	0.00	5	5.89	0.00	5	6.99	0.00	5	6.47	0.00	5	1.18
	30	100	4	0.00	5	8.08	0.00	5	13.55	0.00	5	12.24	0.00	5	1.60
	30	100	5	0.00	5	9.88	0.00	5	11.43	0.00	5	13.19	0.00	5	1.28
Avg. (C)				0.00	45	7.16	0.00	45	9.77	0.00	45	8.78	0.00	45	2.66
D	10	100	3	0.64	0	—	0.58	0	—	0.24	0	—	0.02	3	2,102.17
	10	100	4	0.57	0	—	0.50	0	—	0.28	0	—	0.00	4	3,324.51
	10	100	5	0.66	0	—	0.52	0	—	0.22	0	—	0.00	5	1,678.22
	20	100	3	1.48	0	—	1.32	0	—	0.90	0	—	0.12	0	—
	20	100	4	1.33	0	—	1.30	0	—	0.89	0	—	0.07	0	—
	20	100	5	1.28	0	—	0.99	0	—	0.89	0	—	0.04	1	6,608.36
	30	100	3	1.87	0	—	1.68	0	—	1.32	0	—	0.15	0	—
	30	100	4	1.86	0	—	1.68	0	—	1.39	0	—	0.00	0	—
	30	100	5	1.51	0	—	1.46	0	—	1.18	0	—	0.06	0	—
Avg. (D)				1.24	0	—	1.12	0	—	0.81	0	—	0.07	13	3,428.31
E	10	100	3	0.14	2	791.06	0.14	4	3,820.26	0.03	4	708.37	0.00	5	63.52
	10	100	4	0.06	3	521.65	0.08	3	3,191.33	0.02	4	714.58	0.00	5	10.73
	10	100	5	0.05	4	466.01	0.04	4	2,278.66	0.00	5	1,734.20	0.00	5	14.40
	20	100	3	1.77	0	—	1.49	0	—	0.86	0	—	0.00	5	53.89
	20	100	4	1.52	0	—	1.71	0	—	0.97	0	—	0.00	5	80.72
	20	100	5	1.58	0	—	2.60	0	—	2.04	0	—	0.00	5	68.59
	30	100	3	3.90	0	—	4.02	0	—	3.32	0	—	0.00	5	597.99
	30	100	4	2.97	0	—	3.57	0	—	3.33	0	—	0.00	5	274.41
	30	100	5	3.18	0	—	3.04	0	—	2.55	0	—	0.00	5	190.87
Avg. (E)				1.69	9	592.91	1.86	11	3,096.75	1.46	13	1,052.38	0.00	45	150.57
Avg. $M = 100$				0.98	54	300.04	0.99	56	1,553.26	0.76	58	530.58	0.02	103	1,193.85
C	15	200	4	0.00	5	91.86	0.00	5	93.90	0.00	5	123.76	0.00	5	87.46
	15	200	5	0.00	5	82.64	0.00	5	131.47	0.00	5	103.48	0.00	5	54.79
	30	200	4	0.00	5	273.34	0.00	5	385.82	0.00	5	273.05	0.00	5	10.15
	30	200	5	0.39	4	243.10	0.00	5	586.54	0.00	5	143.33	0.00	5	10.73
Avg. (C)				0.10	19	172.73	0.00	20	299.44	0.00	20	160.91	0.00	20	40.78
D	15	200	4	0.63	0	—	0.54	0	—	0.29	0	—	0.02	0	—
	15	200	5	0.60	0	—	0.48	0	—	0.22	0	—	0.01	3	1,429.00
	30	200	4	0.91	0	—	0.99	0	—	0.79	0	—	0.06	0	—
	30	200	5	0.80	0	—	0.83	0	—	0.64	0	—	0.02	0	—
Avg. (D)				0.73	0	—	0.71	0	—	0.49	0	—	0.03	3	1,429.00
E	15	200	4	0.17	0	—	0.37	0	—	0.17	0	—	0.00	5	287.66
p 0.8	15	200	5	0.17	0	—	0.64	0	—	0.16	0	—	0.00	5	433.27
	30	200	4	1.17	0	—	2.15	0	—	1.86	0	—	0.00	5	419.52
	30	200	5	1.10	0	—	2.26	0	—	2.03	0	—	0.00	5	206.73
Avg. (E, p 0.8)				0.65	0	—	1.36	0	—	1.05	0	—	0.00	20	336.79
E	15	200	4	0.25	0	—	0.85	0	—	0.66	0	—	0.00	5	975.23
p 1.0	15	200	5	0.24	0	—	0.89	0	—	0.80	0	—	0.00	5	1,037.15
	30	200	4	1.06	0	—	2.11	0	—	2.28	0	—	0.00	5	199.19
	30	200	5	1.17	0	—	1.97	0	—	2.30	0	—	0.00	5	271.30
Avg. (E, p 1.0)				0.68	0	—	1.45	0	—	1.51	0	—	0.00	20	620.71
Avg. $M = 200$				0.54	19	172.73	0.88	20	299.44	0.76	20	160.91	0.01	63	606.82

$$\min \sum_{j \in C^i} \sum_{k \in K} c_{ijk} x_{ijk}$$
$$\text{s.t.} \sum_{j \in C^i} \sum_{k \in \mathcal{K}} a_{ijk} x_{ijk} \leqslant b_i$$
$$\sum_{k \in \mathcal{K}} x_{ijk} = 1 \quad \forall j \in C^i$$
$$x_{ijk} \in \{0, 1\} \quad \forall j \in C^i, \forall k \in \mathcal{K}$$
forall $j \in C^i$ do $k(j) := (k \mid x_{ijk} = 1)$

Local Search

(*Shift Neighborhood*)
forall $j \in \mathcal{M}$ do
  $i' :=$ (null element); $v := c_{i(j)jk(j)}$
  forall $i \in \mathcal{N}$, forall $k \in \mathcal{K}$ do
    if ($c_{ijk} < v$ AND $q_i \geqslant a_{ijk}$) then
      $i' := i$; $k' := k$; $v := c_{ijk}$
  if ($i' \neq$ (null element)) then
    $q_{i(j)} := q_{i(j)} + a_{i(j)jk(j)}$
    $q_{i'} := q_{i'} - a_{i'jk'}$
    $i(j) := i'$; $k(j) := k'$

(*Swap Neighborhood*)
forever do
  $i' :=$ (null element); $v := 0$
  forall $js, jd \in \mathcal{N}$ do
    forall $ks, kd \in \mathcal{K}$ do
      $is := i(js)$; $id := i(jd)$
      $\Delta := (c_{id\,js\,ks} + c_{is\,jd\,kd}) - (c_{is\,js\,k(js)} + c_{id\,jd\,k(jd)})$
      $\delta_s := a_{is\,jd\,kd} - a_{is\,js\,k(js)}$
      $\delta_d := a_{id\,js\,ks} - a_{id\,jd\,k(jd)}$
      if ($\Delta < v$ AND $\delta_s \leqslant q_{is}$ AND $\delta_d \leqslant q_{id}$) then
        $v := \Delta$;
        $j' := js$; $j'' := jd$
        $k' := ks$; $k'' := kd$
  if ($i' =$ (null element)) then BREAK
  $q_{i(j')} := q_{i(j')} - a_{i(j')j'k(j')} + a_{i(j')j''k''}$
  $q_{i(j'')} := q_{i(j'')} - a_{i(j'')j''k(j'')} + a_{i(j'')j'k'}$
  SWAP($i(j'), i(j'')$); $k(j') := k'$; $k(j'') := k''$

Acknowledgments

The authors are grateful to Manuel Laguna and Maria Auxilio Osorio for providing their test instances. The authors also acknowledge the kind support of ACSU (Associazione Cremasca Studi Universitari) to the OR Lab at the Dipartimento di Technologie dell'Informazione of the Università di Milano degli Studi, where this work was done.

References

Barnhart, C., E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, P. H. Vance. 1998. Branch-and-price. Column generation for solving huge integer programs. *Oper. Res.* **46**(3) 316–329.

Ceselli, A. 2003. Two exact algorithms for the capacitated p-median problem. *4OR* **1**(4) 319–341.

Ceselli, A., G. Righini. 2005. A branch-and-price algorithm for the capacitated p-median problem. *Networks* **45**(3) 125–142.

French, A. P., J. M. Wilson. 2002. Heuristic solution methods for the multilevel generalized assignment problem. *J. Heuristics* **8** 143–153.

Glover, F., J. Hultz, D. Klingman. 1979. Improved computer-based planning techniques. Part II. *Interfaces* **9**(4) 12–20.

Laguna, M., J. P. Kelly, J. L. Gonzalez-Velarde, F. Glover. 1995. Tabu search for the multilevel generalized assignment problem. *Eur. J. Oper. Res.* **82** 176–189.

Land, A. H., A. G. Doig. 1960. An automatic method for solving discrete programming problems. *Econometrica* **28** 497–520.

Martello, S., P. Toth. 1990. *Knapsack Problems—Algorithms and Computer Implementations*. Wiley, New York.

Martin, R. K. 1999. *Large Scale Linear and Integer Optimization*. Kluwer Academic Publishers, Norwell, MA.

Nauss, R. M. 2003. Solving the generalized assignment problem: An optimizing and heuristic approach. *INFORMS J. Comput.* **15**(3) 249–266.

Osorio, M. A., M. Laguna. 2003. Logic cuts for multilevel generalized assignment problems. *Eur. J. Oper. Res.* **151** 238–246.

Pisinger, D. 1995. A minimal algorithm for the multiple-choice knapsack problem. *Eur. J. Oper. Res.* **83**(2) 392–410.

Savelsbergh, M. 1997. A branch-and-price algorithm for the generalized assignment problem. *Oper. Res.* **45**(6) 831–841.

Yagiura, M., T. Yamaguchi, T. Ibaraki. 1998. A variable depth search algorithm with branching search for the generalized assignment problem. *Optim. Methods Software* **10** 419–441.