Algorithmic Operations Research Vol.2 (2007) 129–136

Alternative Decomposition Based Approaches for Assigning Disjunctive Tasks

Salim Haddadi [a]

[a]University of the 8th of May, 1945, Department of Computer Science, Guelma, Algeria

Omar Slimani [b]

[b]Badji Mokhtar University, Applied Mathematics Department, Annaba, Algeria

Abstract

We consider a special linear assignment problem where some tasks are grouped, and in each group the tasks are 'disjunctive' (in the sense that at most one of them could be performed). We show this problem NP-hard. Then we present and compare two alternative decomposition based algorithms on randomly generated instances.

Key words: Assignment, Benders decomposition, Branch-and-Bound, heuristic, lagrangian decomposition.

1. Introduction

We consider a special linear assignment problem. An instance of this problem consists of a set $I = \{i_1, \ldots, i_m\}$ of m agents, a set $J = \{j_1, \ldots, j_n\}$ of n tasks, p subsets $S_1, \ldots, S_p$ of J, and an $m \times n$-matrix (p_{ij}) which gives the profit of assigning task j to agent i. The tasks grouped in each subset $S_k \subset J$ are 'disjunctive' or conflicting in the sense that at most one of them could be performed. A task can be realized by at most one agent, and an agent can perform at most one task. Now, the goal is to assign the tasks to the agents so as to maximize the total profit. Let $K = \{k_1, \ldots, k_p\}$ be the set of the indices of the groups of disjunctive tasks. The mathematical model of our problem, called assignment problem of disjunctive tasks (APDT), is

$$\max \sum_{i\in I}\sum_{j\in J} p_{ij}x_{ij} \tag{1}$$

$$\sum_{j\in J} x_{ij} \le 1 \qquad i \in I \tag{2}$$

$$\sum_{j\in S_k}\sum_{i\in I} x_{ij} \le 1 \qquad k \in K \tag{3}$$

$$x_{ij} \in \{0,1\} \qquad i \in I, j \in J \tag{4}$$

where x_{ij}, as usual, is a binary variable that indicates whether task j is assigned to agent i. Constraints (2) ensure that each agent can perform at most one task, and constraints (3) specify that, in each subset $S_k \subset J$, at most one task can be performed. Problem APDT is thus an integer programming problem with $m \times n$ binary variables and $m + p$ linear inequalities. Since $x_{ij} = 0, i \in I, j \in J$, is a trivial feasible assignment, and since the objective function value is bounded from above by $\sum_{i\in I}\sum_{j\in J} p_{ij}$, problem APDT has always an optimal solution.

Note that if the subsets $S_k, k \in K$, were disjoint, or if their pairwise intersection was restricted to one task, problem APDT would be easy. However, as we shall see, the problem is NP-hard for general instances.

Claim 1 *Problem APDT is NP-hard.*

Proof: Set $y_j = \sum_{i\in I} x_{ij}$ and $p_{ij} = 1, i \in I, j \in J$. It follows from (3) and (4) that $y_j \in \{0,1\}$. So, problem

---

⋆ This paper is part of a doctoral dissertation prepared by the second author.

Email: Salim Haddadi [s_haddadi@hotmail.com], Omar Slimani [Slimani_omar_2005@hotmail.com].

APDT restricted to (1), (3) and (4) reads

$$\max \sum_{j \in J} y_j$$
$$\sum_{j \in S_k} y_j \leq 1 \qquad k \in K$$
$$y_j \in \{0,1\} \qquad j \in J$$

a maximum cardinality set packing problem which is NP-hard. Thence problem APDT is at least as hard.

The problem APDT has a very interesting application since it also models a real-life practical problem, the combinatorial auction problem (CAP) (see [9]). Therefore, any algorithm for APDT should solve the CAP. Furthermoore, the APDT might be interesting from theoretical as well as from algorithmic point of view (see references [1,2] for analogous extensions of the standard linear assignment problem). Nevertheless, our sole purpose here is to use it as a tool for: 1) designing two alternative (Branch-and-Bound and Benders like) algorithms; 2) comparing them from computing time point of view; 3) showing they have completely reverse behavior according to the density of the instance (which will be defined later).

In fact, the simple trick used in the proof above permits to transform APDT into a mixed binary linear program with $m \times n$ continue variables and only n binary ones. Let us rewrite problem APDT as follows (call R the resulting problem)

$$\max \sum_{i \in I} \sum_{j \in J} p_{ij} x_{ij}$$
$$\sum_{j \in J} x_{ij} \leq 1 \qquad i \in I \tag{5}$$
$$\sum_{i \in I} x_{ij} = y_j \qquad j \in J \tag{6}$$
$$\sum_{j \in S_k} y_j \leq 1 \qquad k \in K \tag{7}$$
$$x_{ij} \geq 0 \qquad i \in I, j \in J \tag{8}$$
$$y_j \in \{0,1\} \qquad j \in J \tag{9}$$

For fixed binary y_j's, constraints (5) and (6) become flow constraints on the bipartite graph with node set $I \cup J$. Therefore, we can relax the integrality constraints on variables x_{ij}. Furthermore, from the constraints (5) we have that $x_{ij} \leq 1$ are always valid. Problems R and APDT are thus equivalent.

Now, consider a third problem, called P, which is problem R with constraints (6) replaced by

$$\sum_{i \in I} x_{ij} \leq y_j \qquad j \in J$$

Claim 2 *Problems P and R are equivalent.*

Proof: Let $\mathcal{F}_{\mathrm{P}}$ and $\mathcal{F}_{\mathrm{R}}$ be respectively the sets of feasible solutions of problems P and R. Clearly, problem P is a weaker formulation since $\mathcal{F}_{\mathrm{R}} \subset \mathcal{F}_{\mathrm{P}}$. Therefore it suffices to show that from any optimal solution of problem P we can extract an optimal solution of problem R. Assume (x, y) is an optimal solution of problem P and set $z_j = \sum_{i \in I} x_{ij}, j \in J$. Clearly (x, z) is a feasible solution for problem R with the same objective function value as (x, y). Therefore (x, z) is optimal.

2. An heuristic for APDT

Another way to describe the relationship between the disjunctive tasks, is to define the sets $E_j = \{k \in K \,|S_k \ni j\}, j \in J$. We have $j \in S_k \Longleftrightarrow k \in E_j$ and $\sum_{k=1}^{p} |S_k| = \sum_{j=1}^{n} |E_j|$ ($|.|$ stands for set cardinality). For convenience, assume $|E_j| \neq \varnothing, j \in J$ (it is easy to deal with a task which does not belong to any group of disjunctive tasks). In fact, problem APDT consists of seeking for a maximum weight matching in the bipartite graph (I, J) (the weights are the p_{ij}'s) with the additional requirement that the set $J' \subset J$ of the vertices of the matching must satisfy $E_i \cap E_j = \varnothing, i, j \in J', i \neq j$. This observation leads to the 'natural' heuristic described in what follows. Obviously, this heuristic guarantees the achievement of a feasible assignment for problem APDT but not an optimal one. Set

$$d_j = \max_{i \in I} p_{ij}$$

Algorithm 1 *(1) Solve the set packing problem*

$$\max \sum_{j \in J} d_j y_j$$
$$\sum_{j \in S_k} y_j \leq 1 \qquad k \in K$$
$$y_j \in \{0,1\} \qquad j \in J$$

Let $J' = \{j \in J \,|y_j = 1\}$.

(2) Solve the maximum weight matching on the bipartite graph with node set $(I \cup J')$.

Example Consider the following instance of APDT: $m = 3, n = 5, p = 3, S_1 = \{1,2,5\}, S_2 = \{1,2,4\}, S_3 = \{2,3,4\}$ and the matrix of the p_{ij}'s

$$\begin{pmatrix} 13 & 3 & 39 & 75 & 39 \\ 71 & 66 & 3 & 62 & 74 \\ 63 & 47 & 47 & 97 & 90 \end{pmatrix}$$

First, we have to solve the set packing problem

$$\begin{array}{lllllll} \max & 71y_1 & +66y_2 & +47y_3 & +97y_4 & +90y_5 & \\ & y_1 & +y_2 & & & +y_5 & \leq 1 \\ & y_1 & +y_2 & & +y_4 & & \leq 1 \\ & & y_2 & +y_3 & +y_4 & & \leq 1 \\ & y_1, & y_2, & y_3, & y_4, & y_5 & \in \{0,1\} \end{array}$$

whose optimal solution is $y_1 = y_2 = y_3 = 0$ and $y_4 = y_5 = 1$. So $J' = \{4,5\}$. Solving the maximum weight matching on the bipartite graph with node set (I, J') results in assigning task 4 to agent 3 and task 5 to agent 2 with a total profit of $97 + 74 = 171$.

3. Benders decomposition algorithm

Benders decomposition is a natural way to tackle our mixed binary linear problem P. This method is well presented in [7] in a more general framework. So, we shall merely present the algorithm.

For fixed y_j's, consider the problem (SP)

$$\begin{aligned} \max & \sum_{i\in I}\sum_{j\in J} p_{ij}x_{ij} \\ & \sum_{j\in J} x_{ij} \leq 1 \qquad i \in I \\ & \sum_{i\in I} x_{ij} \leq y_j \qquad j \in J \\ & x_{ij} \geq 0 \qquad i \in I, j \in J \end{aligned}$$

(which can be seen as a 'maximum profit' flow problem on the bipartite graph with node set (I, J)) whose dual problem is

$$\begin{aligned} \min & \sum_{i\in I} u_i + \sum_{j\in J} y_j v_j \\ & u_i + v_j \geq p_{ij} \quad i \in I, j \in J \\ & u_i, v_j \geq 0 \quad i \in I, j \in J \end{aligned}$$

each of which having an optimal solution. Since the feasible region of problem SP is nonempty and bounded, it follows that the feasible region of its dual has no extreme rays. So we need only consider extreme points of the dual of SP.

Algorithm 2 *Benders decomposition algorithm*

Input *Integers* m, n, p, *sets* $S_1, \ldots, S_p$ *and matrix* (p_{ij})

Output *optimal assignment* $\overline{x}$ *and profit* $\overline{p}$

$s \longleftarrow 0$

$\{$*Let* $\overline{y}^{(0)}$ *be the optimal (or approximated) solution of the set packing problem obtained by the heuristic*$\}$

Repeat

Solve *the problem SP*

$$\begin{aligned} \max & \sum_{i\in I}\sum_{j\in J} p_{ij}x_{ij} \\ & \sum_{j\in J} x_{ij} \leq 1, \quad i \in I \\ & \sum_{i\in I} x_{ij} \leq \overline{y}_j^{(s)}, \quad j \in J \\ & x_{ij} \geq 0 \quad i \in I, j \in J \end{aligned}$$

$\{$*Let* $\overline{x}$ *be the optimal solution,* $\overline{u}^{(s+1)}, \overline{v}^{(s+1)}$ *be the optimal dual solution and* $\overline{p}$ *be the optimal function value*$\}$

Add *to the master problem (MP)*

$$\begin{aligned} & \max \ z \\ & \sum_{j\in S_k} y_j \leq 1 \qquad k \in K \\ & z - \sum_{j\in J} \overline{v}_j^{(t)} y_j \leq \sum_{i\in I} \overline{u}_i^{(t)} \qquad t = 1, \ldots, s \\ & z \geq 0 \\ & y_j \in \{0,1\} \qquad j \in J \end{aligned}$$

the cut

$$z - \sum_{j\in J} \overline{v}_j^{(s+1)} y_j \leq \sum_{i\in I} \overline{u}_i^{(s+1)}$$

Solve *problem MP*

$\{$*Let* $\overline{y}^{(s+1)}$ *be the optimal solution and* $\overline{z}$ *the corresponding objective function value*$\}$

$s \longleftarrow s + 1$

until $\overline{p} = \overline{z}$

Example (continued) Recall that $\overline{y}_1^{(0)} = \overline{y}_2^{(0)} = \overline{y}_3^{(0)} = 0$ and $\overline{y}_4^{(0)} = \overline{y}_5^{(0)} = 1$ was the optimal solution of the set packing problem corresponding to the instance of this example obtained by the heuristic. Solving problem SP gives the optimal dual solution $\overline{u}_1^{(1)} = 0$, $\overline{u}_2^{(1)} = 6$, $\overline{u}_3^{(1)} = 22$, $\overline{v}_1^{(1)} = 65$, $\overline{v}_2^{(1)} = 60$, $\overline{v}_3^{(1)} = 39$, $\overline{v}_4^{(1)} = 75$, $\overline{v}_5^{(1)} = 68$ and $\overline{p} = 171$. The problem MP to be solved (after adding the Benders cut) is

$$\begin{aligned}
&\max z\\
&\quad y_1 + y_2 + y_5 \le 1\\
&\quad y_1 + y_2 + y_4 \le 1\\
&\quad y_2 + y_3 + y_4 \le 1\\
&\quad z - 65y_1 - 60y_2 - 39y_3 - 75y_4 - 68y_5 \le 28\\
&\quad y_1,\ y_2,\ y_3,\ y_4,\ y_5 \in \{0,1\}\\
&\quad z \ge 0
\end{aligned}$$

whose optimal solution is $\overline{y}_1^{(1)} = \overline{y}_2^{(1)} = \overline{y}_3^{(1)} = 0$ and $\overline{y}_4^{(1)} = \overline{y}_5^{(1)} = 1$ with $\overline{z} = 171$. Since both the optimal function values of problems SP and MP are equal, the algorithm terminates. The optimal solution of APDT, as discovered by the heuristic, is thus to assign task 4 to the third agent and task 5 to the second.

4. Lagrangian decomposition and Branch-and-Bound

It is tempting to try to reduce problem P to a flow problem, of course by adding some kind of side constraints, since problem P is NP-hard. Having this objective in mind, we go to split every variable $y_j, j \in J$, of problem P into $|E_j|$ copies with one special representative, noted $y_j^{\delta_j}$, as we shall see. Set $\delta_j = \min\{k\,|k \in E_j\}, j \in J$. Now, let us rewrite problem P as follows (call Q the resulting problem)

$$\begin{aligned}
&\max \sum_{i\in I}\sum_{j\in J} p_{ij}x_{ij}\\
&\quad \sum_{j\in J} x_{ij} \le 1 \quad i \in I\\
&\quad \sum_{i\in i} x_{ij} \le y_j^{\delta_j} \quad j \in J\\
&\quad \sum_{j\in S_k|k=\delta_j} y_j^{\delta_j} + \sum_{j\in S_k|k\ne\delta_j} y_j^k \le 1 \quad k \in K\\
&\quad y_j^{\delta_j} = y_j^k \quad k \in E_j, |E_j| \ge 2, k \ne \delta_j, j \in J \qquad (10)\\
&\quad x_{ij} \ge 0 \quad i \in I, j \in J\\
&\quad y_j^k \in \{0,1\} \quad k \in E_j, j \in J
\end{aligned}$$

For each task $j \in J$, there are $|E_j| - 1$ 'coupling' constraints (10). Though problem Q is another weaker representation of problem APDT, it is easy to see that problems Q and APDT are equivalent.

It is not at all obvious, but problem Q is an integer flow problem with homologous arcs which is known to be NP-hard (see [4]). To be convinced, let us construct the corresponding network. The node set is $I \cup J \cup K \cup \{s_1, s_2\}$, where s_1, s_2 are the source and sink of the network. There are $m+n+p+2$ nodes altogether. The arc set is $U = U_1 \cup U_2 \cup U_3 \cup U_4 \cup U_5 \cup \{(s_2, s_1)\}$ where $U_1 = \{(s_1, i)|i \in I\}$, $U_2 = \{(i,j)|i \in I, j \in J\}$, $U_3 = \{(k, s_2)|k \in K\}$, $U_4 = \{(j, \delta_j)|j \in J\}$ (δ_j is an element of K) and $U_5 = \{(s_1, k)|k \in E_j, |E_j| \ge 2, k \ne \delta_j, j \in J\}$. Let us count the number of arcs. Since $|U_1| = m$, $|U_2| = m \times n$, $|U_3| = p$, $|U_4| = n$, $|U_5| = \sum_{j=1}^n |E_j| - n = \sum_{k=1}^p |S_k| - n$ (which depends on the groups of disjunctive tasks), there are $m \times n + m + p + \sum_{k=1}^p |S_k| + 1$ arcs. This is a multigraph since there are repeated arcs in U_5. Let us call $a_u, b_u, c_u, u \in U$, respectively the lower and upper bound on the capacity of arc u, and the unit cost to go over it, which are defined as follows $a_u = 0, u \in U$,

$$b_u = \begin{cases} 1 & u \in U\backslash\{(s_2, s_1)\} \\ \infty & u = (s_2, s_1) \end{cases}$$

and

$$c_u = \begin{cases} 0 & u \in U\backslash U_2 \\ p_{ij} & u \in U_2 \end{cases}$$

Provided that $|E_j| \ge 2$, an arc of the form (j, δ_j) in U_4 has $|E_j| - 1$ homologous arcs in U_5 which are of the form $(s_1, k), k \in E_j, k \ne \delta_j$. The arcs are homologous in the sense that they must carry the same value of the flow (either all 0's or all 1's). Before going any further, let us return to our previous example.

Example (continued) We have $E_1 = \{1,2\}$, $E_2 = \{1,2,3\}$, $E_3 = \{3\}$, $E_4 = \{2,3\}$, $E_5 = \{1\}$, $\delta_1 = \delta_2 = \delta_5 = 1$, $\delta_3 = 3$ and $\delta_4 = 2$. Here, task 3 (and 5) belongs to only one group, so there are no coupling constraints corresponding to these two tasks, and therefore no corresponding homologous arcs in the network. Solving APDT amounts to seeking for a maximum profit integer flow φ in the network of figure 1 (a triple in front of each arc u gives the values a_u, b_u, c_u)) with the requirements $\varphi_{u_1} = \varphi_{u_4}$, $\varphi_{u_2} = \varphi_{u_5}$, $\varphi_{u_2} = \varphi_{u_6}$, $\varphi_{u_3} = \varphi_{u_7}$.

We use Branch-and-Bound (see [5]) with 'largest-upper-bound-next' strategy to solve problem Q. At each node of the decision tree, a local upper bound is computed by solving the relaxed problem, which is a flow

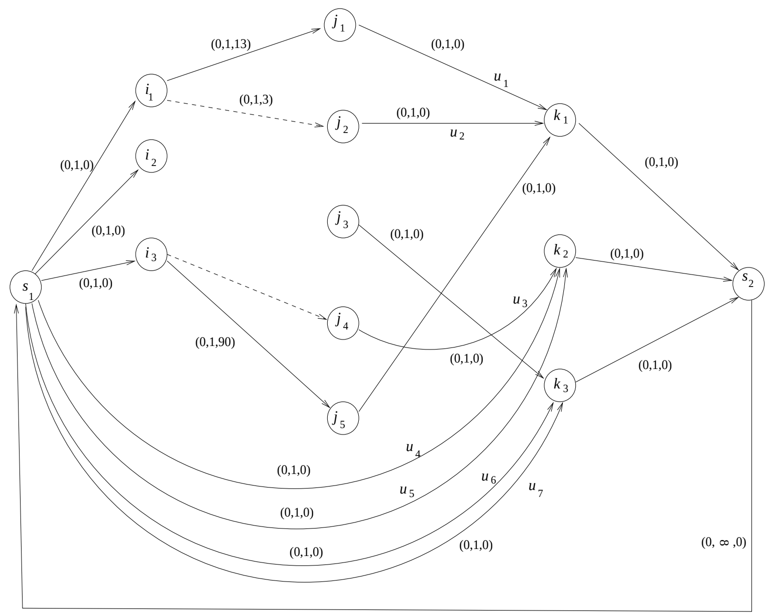

Fig. 1. Network of the example. Multiple arcs are labeled.

problem as already shown,

$$\begin{aligned}
&\max \sum_{i\in I}\sum_{j\in J} p_{ij}x_{ij}\\
&\sum_{j\in J} x_{ij} \le 1 \quad i\in I\\
&\sum_{i\in i} x_{ij} \le y_j^{\delta_j} \quad j\in J\\
&\sum_{j\in S_k|k=\delta_j} y_j^{\delta_j} + \sum_{j\in S_k|k\ne\delta_j} y_j^k \le 1 \quad k\in K\\
&x_{ij}\ge 0 \quad i\in I, j\in J\\
&\alpha_j \le y_j^k \le \beta_j \quad k\in E_j, j\in J
\end{aligned}$$

where the values of $\alpha_j, \beta_j, j \in J$, are decided by the branching procedure as we shall see, with $0 \le \alpha_j \le \beta_j \le 1$. At the root node we begin with $\alpha_j = 0, \beta_j = 1, j \in J$.

The heuristic provides a first lower bound with which we begin. Then at each node, after solving the relaxed problem, we consider the restriction of the flow φ on the arc set U_2. If it constitutes a feasible assignment for problem APDT and improves the current lower bound, then we update.

It hardly happens that the relaxed coupling constraints (10) are satisfied while solving the relaxed problem. When it does not, we select the bundle of homologous arcs (which are then neither all 0's nor all 1's) for which the sum of the values of the flow through the homologous arcs is maximum. Let j^* be the label of this bundle, we compute

$$j^* = \arg\max_{j\in J, |E_j|\ge 2}\left[\varphi_{(j,\delta_j)} + \sum_{k\in E_j, k\ne\delta_j}\varphi_{(s_1,k)}\right]$$

The current node is separated into two nodes and we set in the first $\alpha_{j^*} = 1$ and in the second $\beta_{j^*} = 0$, to enforce the satisfaction of the coupling constraints relative to task j^*.

A node is fathomed either because all the coupling constraints are satisfied, or because the flow is infeasible (recall the changes in the bounds of the capacity of the homologous arcs), or because the local upper bound is not greater than the lower bound.

Example (finished) At the root node, we solve the maximum profit flow on the network of figure 1. The optimal profit is 210 since the flow is null on all of the edges of the bipartite subgraph (I, J) but $\varphi(i_1, j_3) = \varphi(i_2, j_5) = \varphi(i_3, j_4) = 1$. Let us consider the relaxed constraints of the homologous arcs. We have $\varphi_{u_1} = \varphi_{u_4} = 0$, $\varphi_{u_2} = \varphi_{u_5} = \varphi_{u_6} = 0$, but $\varphi_{u_3} = 1$ and $\varphi_{u_7} = 0$. To enforce the satisfaction of the constraint $\varphi_{u_3} = \varphi_{u_7}$ we act as in figure 2. Recall that a lower bound (171) is provided by the heuristic. So, the two children of the root node are fathomed, and the optimality of the feasible assignment obtained by the heuristic is proven.

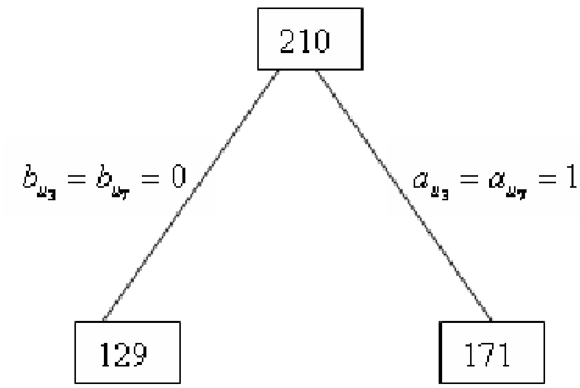

Fig. 2. Decision tree of the example

5. Computational experience

Both of the two algorithms were coded in C (with the 'gcc' compiler of Linux) and run on a compatible PC with a Pentium IV (2.4 GHz) processor. We used a Network Simplex algorithm to solve problem SP (see [3]), the Out-of-Kilter algorithm to solve the relaxed problem in the BaB algorithm (see [6]), and the 'opbdp' software [8] to solve the master problem MP.

The algorithms were tested on a set of 45 randomly generated instances. We fixed, once for all, the number of agents ($m = 20$) and the number of tasks ($n = 50$). The reason of this choice (of n) is that the 'opbdp' software we used to solve the master problem MP slows down for larger values of n. On the other hand, since it is natural that the amount of time spent by any of the two algorithms will depend on a lot of parameters (at least on m, n, p, the subsets S_k, the coefficients p_{ij}), it would be interesting to fix some of the parameters and compare the algorithms from the point of view of the crucial ones. Here, two parameters are important: the number p of groups of disjunctive tasks and their 'density'. Observe that the subsets $S_1, \cdots, S_p$ can be described by giving a binary $p \times n$-matrix whose rows are the incidence vectors of the subsets. The term density should then be understand as the density of this binary matrix, which is expressed in percentage as

$$\sum_{k=1}^{p} |S_k| \div (n \times p) \times 100$$

The coefficients p_{ij} are randomly generated in the range [1,10]. In the first set of experiments, we fixed $p = 15$ and generated and run five instances for each value of the density from 5, 10, 15, 20, 25%. In the second, the density is fixed to 15% and p varies in $\{5, 10, 15, 20, 25\}$. The experimental results are recorded respectively in tables 1 and 2, and summarized in figures 2 and 3. In each of the two tables, the second column refers to the profit realized by the approximated solution obtained by the heuristic, while the third column gives the optimal profit. The three following columns concern the BaB algorithm and the last two, the Benders algorithm. They refer respectively to the number of coupling constraints (10), the size of the decision tree, the time (in seconds) spent by the BaB algorithm, the number of generated cuts, and the computing time of the Benders algorithm.

The heuristic seems to be very effective since it fails only four times to discover the optimal solution. In our implementation, we solved the set packing problem using 'opbdp', but instead, we may approximate it in order to make the heuristic polynomial time.

For fixed p (see figure 3), we notice that the BaB algorithm generates larger decision trees and, consequently, spends more computing time as the density increases, while the Benders algorithm has a complete reverse behavior. This is easily interpreted regarding the BaB algorithm. The more the density increases, the more the groups of disjunctive tasks overlap, the more the number of coupling constraints (10) increases. But the question to know why the Benders algorithm converges quickly when the density of the $p \times n$-matrix (which is a submatrix of the master problem MP) increases is rather puzzling. The integral feasible domain (call it D) of the master problem is included in the unit hypercube of $\mathbb{R}^n$. Does the binary matrix contain much 'more information' when its density increases to the point where the domain D is cut deeply off. An open question ?

When the density is fixed (see figure 4), the BaB algorithm continue to behave as before, the previous interpretation remaining valid. What is surprising is that the Benders algorithm seems to be insensitive to the variation of the number p.

Fine tuning the two algorithms (by improving the bounding procedure, the branching strategy, using commercial software to solve the master problem, and so on), will certainly result in speeding each of them up. However, we think it cannot go to the point where the form of the observed curves overturns.

6. Conclusion

We posed a combinatorial optimisation problem and proved its NP-hardness. Then we proposed a heuristic and two exact solution procedures from alternative decompositions. The computational experience performed

Table 2

Comparison of the two algorithms by increased density

			BaB			Benders	
Density	Heu	Opt	Const.	Nodes	Time	Cuts	Time
5%	135	135	9	1	0.01	8	76.47
	135	135	16	1	0.02	19	181.86
	134	135	12	7	0.08	14	105.96
	126	126	14	7	0.08	9	45.87
	133	135	9	3	0.04	7	65.97
10%	108	108	42	63	0.72	23	27.03
	126	126	33	35	0.45	4	1.86
	108	108	42	97	0.90	15	10.96
	99	99	38	99	0.88	7	4.11
	107	107	36	163	1.64	13	18.92
15%	98	98	62	149	1.67	1	0.34
	99	99	62	91	0.93	6	1.93
	99	99	66	439	4.22	7	2.19
	108	108	62	201	2.11	3	0.96
	97	98	76	1201	12.58	5	1.56
20%	81	81	104	649	6.78	2	0.54
	80	80	101	4185	37.85	6	1.61
	72	72	121	709	6.29	3	0.74
	79	79	99	1137	11.94	2	0.52
	81	81	111	537	5.75	3	0.80
25%	45	45	166	1981	21.21	1	0.25
	63	63	141	385	3.69	3	0.74
	62	62	129	3213	31.36	3	0.88
	71	71	139	865	8.90	1	0.24
	54	54	150	8455	78.83	3	0.75

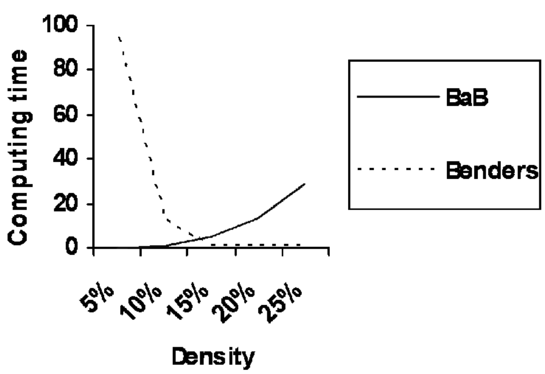

Fig. 3. Summary of table 1 (mean values)

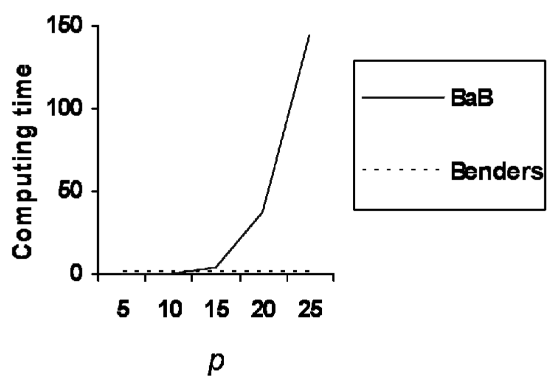

Fig. 4. Summary of table 2 (mean values)

on a set of randomly generated instances points out that the Benders algorithm should be preferred unless the number p of groups of disjunctive tasks is small enough in relation to the number of agents n and the density is sufficiently small ($\leq 15\%$).

Acknowledgements

The authors would like to thank Peter Barth, Max Planck Institut für Informatik, Saarbrüken, Germany, for letting the software ‘opbdp’ freely available.

Table 4

Comparison of the two algorithms by increased number of groups of disjunctive tasks

			BaB			Benders	
p	Heu	Opt	Const.	Nodes	Time	Cuts	Time
5	45	45	8	1	0.01	1	0.39
	45	45	11	1	0.01	3	1.32
	42	42	7	1	0.01	2	0.97
	44	44	13	1	0.01	3	1.24
	45	45	7	1	0.01	2	0.77
10	81	81	36	19	0.22	5	2.30
	90	90	37	1	0.01	6	2.26
	82	83	36	21	0.19	1	0.38
	90	90	40	1	0.01	5	2.18
	89	89	42	1	0.01	1	0.31
20	108	108	99	2155	25.21	2	0.66
	99	99	95	5317	53.52	7	2.30
	90	90	105	6043	65.14	2	0.63
	98	99	85	1991	19.55	2	0.58
	90	90	104	2439	24.48	1	0.32
25	81	81	136	24567	234.86	3	0.87
	81	81	143	9665	93.37	1	0.26
	71	71	139	20257	193.74	1	0.27
	80	81	133	12217	116.74	7	1.82
	80	80	159	8131	85.75	7	1.75

References

[1] Aboudi, R., and G.L. Nemhauser, *Some Facets for an Assignment Problem with Side Constraints*. Operations Research 39 (1981) 244-250.

[2] Aggarwal, V., *A Lagrangian-Relaxation Method for the Constrained Assignment Problem*. Computers and Operations Research 12 (1985) 97-106.

[3] Chvátal, V., *Linear Programming*. Freeman (1983).

[4] Garey, M.R., and D.S. Johnson, Computers and intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co. (1979).

[5] Garfinkel, R.S., and G.L. Nemhauser, Integer Programming. Wiley (1972).

[6] Gondran, M., and M. Minoux, Graphes et Algorithmes. Eyrolles (1979).

[7] Lasdon, L. S., Optimization Theory for Large Systems. MacMillan (1970).

[8] opbdp, free software available from ftp.mpi-sb.mpg.de:/pub/ guide/ staff/ barth/ opbdp/ opbdp.tar.Z.

[9] Vries de, S., and R. Vohra, Combinatorial Auctions: A Survey, INFORMS Journal on Computing 15 (2003) 284-309.

Received 16 January, 2007; revised 29 March 2007; accepted 4 May 2007