Exact Solutions to a Class of Stochastic Generalized Assignment Problems

Maria Albareda-Sambola[*]
Maarten H. van der Vlerk[†]
and Elena Fernández[‡]

February 2002

Abstract

This paper deals with a stochastic Generalized Assignment Problem with recourse. Only a random subset of the given set of jobs will require to be actually processed. An assignment of each job to an agent is decided *a priori*, and once the demands are known, reassignments can be performed if there are overloaded agents.

We construct a convex approximation of the objective function that is sharp at all feasible solutions. We then present three versions of an exact algorithm to solve this problem, based on branch and bound techniques, optimality cuts, and a special purpose lower bound. Numerical results are reported.

Keywords: Generalized assignment, integer recourse, convex approximation

Mathematics Subject Classification: 90C15, 90C11

1 Introduction

Pairing problems constitute a vast family of problems in Combinatorial Optimization. Different versions of these problems have been studied since the mid-fifties due both to their many applications and to the challenge of understanding their combinatorial nature. The range of problems in this group is very wide. Some can be easily solved in polynomial time, whereas others are extremely difficult. The simplest one is the Assignment Problem, that can be easily solved by the Hungarian Algorithm [Kuh55]. Matching Problems appear when the underlying graph is no longer bipartite and are much more involved, although they can still be solved in polynomial time [Edm65]. On the other

[*]Dept. d'Estadística I Investigació Operativa, Universitat Politècnica de Catalunya, e-mail: maria.albareda@upc.es. Partially supported by grant AP97-43739460 from the Spanish MEC and CICYT grant TIC-2000 1750 006 03.

[†]Department of Econometrics & OR, University of Groningen, e-mail: m.h.van.der.vlerk@eco.rug.nl. The research of this author has been made possible by a fellowship of the Royal Netherlands Academy of Arts and Sciences.

[‡]Dept. d'Estadística I Investigació Operativa, Universitat Politècnica de Catalunya, e-mail: e.fernandez@upc.es. Partially supported by CICYT grant TIC-2000 1750 006 03.

extreme, the Generalized Assignment Problem is a very difficult combinatorial optimization problem that is $\mathcal{N}P$-hard.

In general terms, the Assignment Problem consists of finding the best assignment of items to agents according to a predefined objective function. Among its many applications, we mention the assignment of tasks to workers, of jobs to machines, of fleets of aircraft to trips, or the assignment of school buses to routes. However, in most practical applications, each agent requires a quantity of some limited resource to process a given job. Therefore, the assignments have to be made taking into account the resource availability of each agent. The problem derived from the classical Assignment Problem by taking into account these capacity constraints is known as the Generalized Assignment Problem (GAP). Among its many applications, we find assignment of variable length commercials into time slots, assignment of jobs to computers in a computer network ([Bal72]), distribution of activities to the different institutions when making a project plan ([ZR88]), etc. Besides these applications, it also appears as a subproblem in a variety of combinatorial problems like Vehicle Routing ([FJ81]) or Plant Location ([RS77, Día01]). A survey of exact and heuristic algorithms to solve the GAP can be found in [CW92]. More recently, Savelsbergh ([Sav97]) proposed a Branch and Price algorithm, and different metaheuristic approaches have been proposed by other authors ([BC97, DF01, YTI98, YIG99]).

As mentioned before, the GAP is a very challenging problem, not only because it is $\mathcal{N}P$-hard, but also because the decision problem to know if a given instance is feasible is $\mathcal{N}P$-complete [MT90]. In fact, the GAP is in practice even more difficult, since most of its applications have a stochastic nature. Stochasticity can be due to two different sources. On the one hand, it appears when the actual amount of resource needed to process the jobs by the different agents is not known in advance. This happens, for instance, when assigning software development tasks to programmers; the time needed for each task is not known *a priori*. Similarly, the actual running times of jobs are not known when they are assigned to processors. This is due to the fact that the actual running times of jobs depend on the overall load of the system. In all these cases, the amount of resource consumed when assigning tasks to agents should be modeled with continuous random variables.

The second source of stochasticity is uncertainty about the presence or absence of individual jobs. In such cases, there is a set of potential jobs, but only a subset of them will have to be actually processed. This subset is not known when the assignment has to be decided. This is the case of emergency services, or the assignment of repairmen to machines. In this situation, the resource requirement of each job can be modeled as a random variable with Bernoulli distribution. This kind of stochasticity has also been considered in other problems, such as stochastic routing problems ([BSL88, LLM94])

The stochasticity in the GAP studied in this paper is of the latter type. The problem will be modeled as a stochastic programming problem with recourse.

In the last decades, Stochastic Programming has become an increasingly studied area in optimization. Problems with recourse (that is, problems where an adaptation or recourse action is allowed once the values of stochastic parameters are known) constitute one of the most studied issues (see [KW94, Pr95, BL97] for a general overview). An extensive bibliography on stochastic programming can be found in [Vle01].

More recently, stochasticity has also been considered in integer programs (see e.g. the surveys [SSV96, SV97, KV99]). Starting with the work by Yudin & Tsoy [YT74], several approaches have been proposed to deal with integrality in stochastic programs (see [Wol80, LL92, LV93, KSV96, CS99, SSV98, NER98, ATS00, RS01]).

We are aware of only two papers addressing any type of stochastic assignment problems. Mine et al. [MFIS83] present a heuristic for an assignment problem with stochastic side constraints. Albareda & Fernández [AF00] propose some model-based heuristics for the same stochastic GAP considered in the present work. To the best of our knowledge, no results on exact algorithms for any type of stochastic GAP are known.

In this paper, we consider the following stochastic GAP: After the assignment is decided, each job requires to be processed with some known probability. In this context, we can think that jobs are customers requiring some service with a demand distributed as a Bernoulli random variable. The problem is modeled as a recourse problem. Assignments of customers to agents are decided *a priori*. Once the actual demands are known, if the capacity of an agent is violated, some of its assigned jobs are reassigned to underloaded agents at a prespecified cost. For a given instance of the problem, the overall demand of the customers requiring service can be bigger than the total capacity. In this case, part of the customers are lost, and a penalty cost is paid.

The objective is to minimize the expected total costs. The costs consist of two terms: the assignment costs, and the expected penalties for reassigned and/or lost customers. This problem has relatively complete recourse, i.e., the second-stage problem is feasible for any *a priori* assignment and any realization of the demand. Additionally, due to the assumption that demands are binary, we can build a model for the second-stage problem where only the right-hand side contains non-deterministic elements.

We propose three versions of an exact algorithm for this stochastic GAP. All versions have in common that the (nonconvex, discontinuous) recourse function is replaced by a convex approximation, which is exact at all binary first-stage solutions. Next, a cutting procedure is used to iteratively generate a partial description of this convex approximation of the recourse function, following the ideas presented in [VW69] and in [LL92]. Integrality of the first-stage variables is addressed using branch and bound techniques.

The first version of the algorithm has the structure of a branch and cut algorithm. At each node, cuts are iteratively added until no more violated valid constraints are found. Then, if the current solution is not integer, branching is performed.

However, in the considered stochastic GAP, the separation problem to find the new cut to be added is highly time consuming because it requires the evaluation of the convex approximation of the recourse function. To reduce the number of evaluations of this function, a second version has been designed where, at each iteration, an integer problem is solved using branch and bound. Once an integer assignment is found, the associated cut is computed and added. The algorithm terminates when the integer solution found does not violate the associated constraint. This approach makes a great effort to find integer solutions even at the earliest stages when the information about the recourse function is rather poor.

A third strategy has been designed as a tradeoff between the other two. The idea is to avoid the excessive number of evaluations of the recourse function as well as to reduce the time invested to reach integrality. At each node of the enumeration tree, an associated cut is computed. If the solution of the node is integer, the cut is added and the current problem is reoptimized. Otherwise, branching is performed and the new cut is added to the descendant nodes.

All three algorithms compute the same upper and lower bounds initially. The upper bound is obtained with the heuristics presented in [AF00]. The lower bound is found by solving a family of stochastic linear subproblems with the L-shaped algorithm.

The remainder of this paper is organized as follows: In Section 2 the two-stage problem is modeled and its properties are studied. In particular, we introduce a convex approximation of the non-convex recourse function. The algorithms are described in Section 3. After that, Section 4 briefly discusses the computation of the upper bound, and defines the lower bound. In Section 5 some strategies to improve the performance of the algorithms are discussed. Finally, computational experiences and conclusions are presented in Sections 6 and 7 respectively.

2 The model

The Generalized Assignment Problem (GAP) consists in finding the cheapest assignment of a set of jobs to a set of agents such that each job is assigned exactly to one agent and capacity constraints on some resource are satisfied.

Let I and J be the index sets of agents and jobs, respectively, with $|I| = ns$ and $|J| = nt$. We define, for $i \in I$ and $j \in J$,

c_{ij} is the cost of assigning job j to agent i;

q_{ij} is the amount of resource needed by agent i to perform task j;

b_i is the resource capacity of agent i ($i \in I$),

and

x_{ij} takes the value 1 if job j is assigned to agent i, and 0 otherwise.

Using this notation, the GAP is modeled as:

$$\begin{array}{lll}
\textbf{Min} & \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} & \\
\text{s.t.} & \sum_{i\in I} x_{ij} = 1 & j \in J \\
 & \sum_{j\in J} q_{ij}x_{ij} \leqslant b_i & i \in I \\
 & x_{ij} \in \{0,1\} & i \in I,\ j \in J
\end{array}$$

Model (2.1)

Most often, however, the assignment of jobs to agents must be decided before the actual values of the demands for resource capacity $q_{ij}, i \in I, j \in J$, are known, so that the above model is no longer valid.

In this paper, the possibility of reassigning some of the jobs once the values of the vector q are known, incurring costs $K_j, j \in J$ is considered.

Assuming that the values of $q_{ij}, i \in I, j \in J$ are agent independent (that is, for each $j \in J$, $q_{ij} = q_j \quad \forall i \in I$) and that they are Bernoulli random variables, the problem can be formulated using the following recourse model:

$$\begin{array}{lll} \textbf{Min} & \mathcal{Q}(x) & \\ \text{s.t.} & \displaystyle\sum_{i \in I} x_{ij} = 1 & j \in J \\ & x_{ij} \in \{0,1\} & i \in I, j \in J \end{array}$$

Model (2.2)

where[1] $\mathcal{Q}(x) := \mathbb{E}_q v(x,q)$ is the recourse function, and $v(x,q)$ is the value of

$$\begin{array}{lll} \textbf{Min} & \displaystyle\sum_{i \in I}\sum_{j \in J} c_{ij} y_{ij} + \sum_{j \in J} K_j z_j & \\ \text{s.t.} & y_{ij} + z_j \geqslant q_j x_{ij} & i \in I, j \in J \\ & \displaystyle\sum_{i \in I} y_{ij} \geqslant q_j & j \in J \\ & \displaystyle\sum_{j \in J} y_{ij} \leqslant b_i & i \in I \\ & y_{ij} \in \{0,1\} & i \in I, j \in J \\ & z_j \in \{0,1\} & j \in J \end{array}$$

Model (2.3)

In the first stage, each job is provisionally assigned to an agent. The second-stage problem determines the final assignment pattern once the demands are known. Variables y_{ij} ($(i,j) \in I \times J$) are defined like x_{ij}, and z_j ($j \in J$) point out those jobs with nonzero demand that have been reassigned. The first group of constraints (from now on, flag constraints) set z_j to 1 if job j has nonzero demand and it is not assigned to the same agent it was assigned to *a priori*. The other constraints ensure that all jobs with nonzero demand are assigned to an agent indeed and that capacities of the agents are not violated, respectively.
Remark *It would be consistent with the presentation of* Model (2.2) *to use equalities in the second set of constraints of* Model (2.3). *However, in view of Proposition 2.2 below, we prefer the equivalent formulation using inequalities.*

This is a two-stage recourse model with binary variables in both stages. In addition to difficulties caused by integrality of the first-stage variables, this means that the recourse function $\mathcal{Q}$ is non-convex in general.

Indeed, since the parameters q are discretely distributed, this function is lower semicontinuous but discontinuous in general [Sch93]. Moreover, evaluation of $\mathcal{Q}$ for a given x calls for solving many second-stage problems, which are binary programming problems here. Since these second-stage problems are not easily solvable, this is computationally very demanding. In the next section we show how to overcome these problems by redefining the second-stage problem.

[1]$\mathbb{E}$ denotes mathematical expectation.

2.1 Convex approximation of the recourse function

First we introduce an alternative formulation of the second-stage problem for GAP. Subsequently, we show that this new formulation has a nice mathematical property, which allows to drop the integer restrictions on the second-stage variables.

Proposition 2.1 *Given a feasible solution x of* Model (2.2) *and q a* 0-1 *vector,* Model (2.3) *is equivalent to*

$$\begin{array}{lll}
\textbf{Min} & \displaystyle\sum_{i\in I}\sum_{j\in J} c_{ij}y_{ij} + \sum_{i\in I}\sum_{j\in J} K_j z_{ij} & \\
s.t. & y_{ij} + z_{ij} \geqslant q_j x_{ij} & i\in I, j\in J \\
 & \displaystyle\sum_{i\in I} y_{ij} \geqslant q_j & j\in J \\
 & \displaystyle\sum_{j\in J} y_{ij} \leqslant b_i & i\in I \\
 & y_{ij}\in\{0,1\} & i\in I, j\in J \\
 & z_{ij}\in\{0,1\} & i\in I, j\in J
\end{array}$$

Model (2.4)

Proof. Let (y^*, z^*) be an optimal solution of Model (2.4). From the fact that for a feasible point x each job is assigned *a priori* to exactly one agent it follows that for a fixed $j\in J$ at most one of the variables z^*_{ij} $(i\in I)$ takes value 1 in the optimum and the others are 0. So, taking $\tilde{z}_j = \max_{i\in I} z^*_{ij}$, $(y^*, \tilde{z})$ is a feasible point for Model (2.3) with the same value as (y^*, z^*).

Similarly, given an optimal solution (y^*, z^*) of Model (2.3), a feasible solution $(y^*, \tilde{z})$ of Model (2.4) with the same objective value can be built as follows:

If $z^*_j = 0$, take $\tilde{z}_{ij} = 0, \forall i\in I$.

If $z^*_j = 1$, then, using that $\sum_{i\in I} x_{ij} = 1$, only one of the flag constraints is tight, say (i_1, j). Set $\tilde{z}_{i_1 j} = 1$ and $\tilde{z}_{ij} = 0, \forall i \neq i_1$.

Thus, both models lead to the same optimal value. $\square$

Proposition 2.2 *The matrix defining the feasible region of* Model (2.4) *is totally unimodular (TU).*

Proof.

The structure of the matrix is

$$A = \left(\begin{array}{c|c} I_{ns\cdot nt} & I_{ns\cdot nt} \\ \hline M & \mathbb{O} \end{array}\right),$$

where M is the matrix of a transportation problem, which is known to be TU. By Proposition III.2.1 in [NW88], the matrices

$$M_1 = \left(\begin{array}{c} I_{ns\cdot nt} \\ \hline M \end{array}\right) \quad \text{and} \quad M_2 = \left(\begin{array}{c|c} M_1 & I_{ns\cdot nt+ns+nt} \end{array}\right)$$

are also TU. Finally, since A is a submatrix of M_2, it is TU as well. $\square$

Since the matrix defining the feasibility region of Model (2.4) is totally unimodular, its linear relaxation will have integer optimal solutions whenever the right-hand side is integral; moreover, in that case, the optimal values of Model (2.4) and its relaxation are equal.

Assuming that the capacities b_i, $i \in I$, are integral, the right-hand side of Model (2.4) is integral for every feasible (i.e., binary) first-stage solution of GAP. Thus, using Propositions 2.1 and 2.2, we redefine the function $v(x, q)$ as the optimal value of the LP problem

$$\begin{array}{lll}
\textbf{Min} & \sum_{i\in I}\sum_{j\in J} c_{ij}y_{ij} + \sum_{j\in J} K_j z_j & \\
\text{s.t.} & y_{ij} + z_j \geqslant q_j x_{ij} & i \in I, j \in J \\
& \sum_{i\in I} y_{ij} \geqslant q_j & j \in J \\
& \sum_{j\in J} y_{ij} \leqslant b_i & i \in I \\
& y_{ij} \in [0,1] & i \in I, j \in J \\
& z_j \geqslant 0 & j \in J
\end{array}$$

Model (2.5)

As shown above, this function coincides with the previously defined $v(x, q)$ in all feasible vectors x, for any demand vector q. This is not true for fractional vectors x, but this causes no problems since such x are not feasible anyway. The advantages of defining v as the value function of a second-stage problem with continuous variables are two-fold: (i) the evaluation of v can be done (much) faster; (ii) the mathematical properties of v are much nicer. In particular, the new function v is a convex function of x. As we will discuss next, these properties carry over to the recourse function $\mathcal{Q}$, which is defined as the expectation of v.

2.2 Properties of the recourse function

Before introducing our algorithm, we present some well-known properties of the recourse function $\mathcal{Q}$. Discussion on these and other properties in a general context can be found in the textbooks [BL97, KW94, Pr95].

Remark *Assuming that*

$$nt \leqslant B \tag{2.1}$$

where $B = \sum_{i\in I} b_i$, *it follows that* Model (2.2) *is a two-stage program with relatively complete recourse. Assuming in addition that* $K_j \geqslant 0$, $j \in J$, *it follows that, for every realization of the demand vector* q, *the second-stage value function* v *is finite for all* $x \in R^{ns\cdot nt}$.

If (2.1) *is not satisfied, relatively complete recourse is obtained by introducing a dummy agent* $0 \in I$ *with enough capacity in the second stage, who handles excess demand at a unit penalty cost* P. *That is, this dummy agent has assignment costs* $c_{0j} = P - K_j$, $j \in J$, *since any* a posteriori *assignment to it will also* *induce the corresponding reassignment penalty K_j; it makes no sense, however, to pay both penalties at the same time.*

Proposition 2.3 *The function $\mathcal{Q}$ is finite and convex. Moreover, since the random parameters are discretely distributed, $\mathcal{Q}$ is a polyhedral function.*

Proposition 2.4 *Let S be the index set of scenarios (realizations). For $s \in S$, let p^s be the probability of scenario s and q^s the corresponding demand vector, so that*

$$\mathcal{Q}(x) = \sum_{s\in S} p^s v(x, q^s), \quad x \in R^{ns\cdot nt}.$$

Let $\lambda(x, q^s)$ be a vector of dual prices for the first set of constraints in Model (2.5) *for the pair (x, q^s). Then $u(x)$,*

$$u(x) = \sum_{s\in S} p^s \lambda(x, q^s)\, diag(\bar{q}_1^s, \ldots, \bar{q}_{nt}^s),$$

is a subgradient of $\mathcal{Q}$ at x. Here, $\bar{q}_j^s$ is a vector of $|J|$ components, all equal to the demand of customer j in scenario s.

3 Algorithms

Van Slyke and Wets used these properties of the recourse function for continuous two-stage programs to develop the so-called L-shaped algorithm [VW69]. This algorithm exploits the structure of the recourse function within a cutting plane scheme similar to Benders' partitioning algorithm.

The basic idea of that algorithm is to substitute the objective function $\mathcal{Q}(x)$ of Model (2.2) by a continuous variable θ and include a constraint of the form

$$\theta \geqslant \mathcal{Q}(x)$$

in the definition of the feasible region.

This constraint is initially relaxed and successively approximated by a set of linear cuts:

$$\theta \geqslant \alpha + \beta x,$$

which are called optimality cuts.

In this work we will use two kinds of optimality cuts. From Propositions 2.3 and 2.4 we derive cuts of the form:

$$\theta \geqslant \mathcal{Q}(\bar{x}) + \langle x - \bar{x}, u(\bar{x})\rangle, \tag{3.2}$$

where $\langle\cdot,\cdot\rangle$ denotes the usual inner product. We will call (3.2) *∂-optimality cuts* to distinguish them from the L-L-optimality cuts defined below.

Integrality in two-stage programming was first addressed in [Wol80] where programs with binary first-stage variables are considered. In that paper, an implicit enumeration scheme is presented which includes the generation of optimality cuts at every feasible solution generated. More recently, Laporte and Louveaux [LL92] presented a branch and cut algorithm, also for two-stage recourse problems with binary first-stage variables. They introduced a class of optimality cuts that are valid at all binary first-stage solutions, for general second-stage variables. The structure of these cuts, given a binary vector $\bar{x}$, is

$$\theta \geqslant (\mathcal{Q}(\bar{x}) - L)\left(\sum_{\bar{x}_i=1} x_i - \sum_{\bar{x}_i=0} x_i\right) - (\mathcal{Q}(\bar{x}) - L)(\sum_i \bar{x}_i - 1) + L \tag{3.3}$$

where L is a global lower bound of $\mathcal{Q}$. See [CT98] for a generalization of these results.

We will refer to (3.3) as *L-L-optimality cuts*. In addition, for our GAP problem ∂-optimality cuts can be used due to the equivalent reformulation of the second-stage problem using continuous second-stage variables, as derived in Section 2.1.

We will combine both kinds of optimality cuts. Note that both types of cuts are sharp at the (binary) solution at which they are generated.

The three versions of the algorithm presented in this work deal with integrality of the first-stage variables by means of a branch and bound scheme, and approach the recourse function by successively adding optimality cuts. The difference between them is the order in which these operations are done.

BFCS (Branch first, cut second) seeks for violated cuts and adds them once an integer solution is found.

BCS (Branch and cut simultaneously) adds violated cuts at each node of the tree, if they exist, and branches if there is any non-integer variable.

CFBS (Cut first, branch second) iteratively adds cuts to the problem at a node until no more violated cuts are found and then it branches.

Using the reformulation presented in Section 2, the recourse function $\mathcal{Q}$ of our problem is convex. Consequently, ∂-optimality cuts generated at fractional vectors x are valid for all feasible (binary) solutions. Hence, they can be generated at any node of a branch-and-cut scheme. On the other hand, by definition L–L-optimality cuts can only be generated at binary vectors.

3.1 BFCS

The BFCS (Branch first, cut second) version of the algorithm operates on binary subproblems IP_K, defined as

$$\begin{array}{lll}
\textbf{Min} \ \ \theta & & \\
\text{s.t.} & \displaystyle\sum_{i \in I} x_{ij} = 1 & i \in I \\
& \alpha_l^k + \beta_l^k x \leqslant \theta & k = 1, \ldots, K, l = 1, 2 \\
& x_{ij} \in \{0, 1\} & i \in I, j \in J
\end{array}$$

Model (3.1)

where the constraints in the second set are the optimality cuts generated so far.

Then the algorithm is defined as follows:

1. Find a lower bound $\underline{\theta}$
2. Find an initial assignment x^0. Let $K := 1$.
3. Find (α_1^K, β_1^K) and (α_2^K, β_2^K) that define the L-L-optimality cut and the ∂-optimality cut associated to x^{K-1}, respectively.
4. Using a linear integer programming solver, find (x^K, θ^K)

   as a solution of IP_K.
5. Evaluate $r^K = \mathcal{Q}(x^K)$.
6. If $r^K = \underline{\theta}$, let $\theta^K := r^K$, go to 8
7. If $r^K > \theta^K$, $K := K + 1$ and go to 2.
8. x^K is an optimal solution with recourse cost θ^K.

Remark *Observe that IP_K will always be feasible, so that step* 3 *is well defined.*

Remark *If an assignment is found in step* 3 *in two different iterations, say $K_1 < K_2$, then the K_1-th optimality cuts will ensure that $r^{K_2} \leqslant \theta^{K_2}$ and the algorithm will end. Thus, by finiteness of the set of integer assignments, the algorithm terminates in a finite number of steps.*

Remark *Following this scheme, the recourse function is always evaluated at binary points so that, in this case, L–L-optimality cuts as well as ∂-optimality cuts are appropriate. Computational experiments have indicated that information provided by these optimality cuts is supplementary, so that it is worthwhile to add both kinds of cuts at each iteration.*

3.2 BCS

The BCS (branch and cut simultaneously) version of the algorithm operates on LP subproblems in which some of the binary variables are fixed. Given a pair of disjoint subsets of $I \times J$, $S = (S^0, S^1)$, we define the problem $P_{K,S}$ as:

$$
\begin{array}{lll}
\textbf{Min} \quad \theta & & \\
\text{s.t.} & \displaystyle\sum_{i \in I} x_{ij} = 1 & i \in I \\
& \alpha^k + \beta^k \cdot x \leqslant \theta & k = 1, \ldots, K \\
& x_{ij} = 0 & (i,j) \in S^0 \\
& x_{ij} = 1 & (i,j) \in S^1 \\
& x_{ij} \in [0,1] & \text{otherwise}
\end{array}
$$

Model (3.2)

The proposed algorithm can now be described as follows. Note that below we generate ∂-optimality cuts at non-integer solutions. In this respect our algorithm differs from the integer L-shaped algorithm presented in [LL92], because there cuts are only generated in nodes where integer solutions are obtained.

1. $\mathcal{L} := \{(\emptyset, \emptyset)\}$ and $K := 1$.
2. Compute a lower bound, $\underline{\theta}$.
3. Find a feasible assignment x^0, compute (α^1, β^1) (associated L-L cut).

   Let $\bar{\theta} := \mathcal{Q}(x^0)$ and $x^* := x^0$.
4. If $\mathcal{L}$ is empty, go to 8.

   Otherwise, take S from $\mathcal{L}$ following a last-in-first-out policy, and solve $P_{K,S}$ to obtain (x^K, θ^K)
5. If $\theta^K > \bar{\theta}$ or $P_{K,S}$ is infeasible go to 4.

   Otherwise find $r^K = \mathcal{Q}(x^K)$.
6. If $r^K = \underline{\theta}$ and $x^K \in \mathbb{Z}^n$, let $x^* := x^K$, $\bar{\theta} := r^K$ and go to 8.
7. There are four possibilities:

	$r^K \leqslant \theta^K$	$r^K > \theta^K$
$x^K \in \mathbb{Z}^n$	• If $r^K < \bar{\theta}$, $x^* := x^K$, $\bar{\theta} := \theta^K$. • Go to 4.	• Find[1] $(\alpha^{K+1}, \beta^{K+1})$, • $K := K + 1$, • Add S to $\mathcal{L}$. • Go to 4.
$x^K \notin \mathbb{Z}^n$	• Take $x_{ij}^K \notin \mathbb{Z}$. • Add $(S^0 \cup \{(i,j)\}, S^1)$, $(S^0, S^1 \cup \{(i,j)\})$ to $\mathcal{L}$. • Go to 4.	• Find[2] $(\alpha^{K+1}, \beta^{K+1})$ • K:=K+1 • Take $x_{ij}^K \notin \mathbb{Z}$. • Add $(S^0 \cup \{(i,j)\}, S^1)$, $(S^0, S^1 \cup \{(i,j)\})$ to $\mathcal{L}$. • Go to 4.

8. An optimal solution is given by x^* and its recourse cost is $\bar{\theta}$.

[1] *L–L* cut
[2] ∂ cut

3.3 CFBS

This scheme is an adaptation of the branch-and-cut method developed for 0-1 mixed convex programming in [SM99] to our concrete problem.

It is similar to BCS but now branching is only done when the solution at hand does not violate any optimality cut.

```
1. $\mathcal{L} := \{(\emptyset, \emptyset)\}$ and $K = 1$.

2. Find a lower bound $\underline{\theta}$

3. Generate a feasible assignment $x^0$ and the corresponding $\partial$-optimality cut $(\alpha^1, \beta^1)$.

   $\bar{\theta} := \mathcal{Q}(x^0)$ and $x^* := x^0$.

4. If $\mathcal{L}$ is empty, go to 9.

   Otherwise, take $S$ from $\mathcal{L}$ following a LIFO policy

5. Solve $P_{K,S}$ to obtain $(x^K, \theta^K)$.

6. If $\theta^K > \bar{\theta}$ or $P_{K,S}$ is infeasible go to 4.

   Otherwise find $r^K = \mathcal{Q}(x^K)$.

7. If $r^k > \theta^K$,

   let $K := K + 1$, compute $(\alpha^K, \beta^K)$   ($\partial$ cut)

   If $x^k \in \mathbb{Z}^n$, let K:=K+1, compute $(\alpha^K, \beta^K)$   ($L$–$L$ cut)

   go to 4.

8. If $x^k \in \mathbb{Z}^n$ go to 8.

   Otherwise,

       Take $x^K_{ij} \notin \mathbb{Z}$.
       Add $(S^0 \cup \{(i,j)\}, S^1)$, $(S^0, S^1 \cup \{(i,j)\})$ to $\mathcal{L}$ .
       Go to 4.

9. The optimal solution is given by $x^*$ and its value is $\bar{\theta}$.
```

4 Lower and upper bounding

All three versions of the algorithm start by computing a lower bound for the problem and finding an initial feasible solution, which also provides an initial upper bound. The quality of both the upper and the lower bound is crucial for the behavior of the algorithm. A good upper bound will restrict the size of the search tree, whereas the quality of the lower bound determines the impact of the L-L-optimality cuts.

4.1 Lower bound

In combinatorial optimization, lower bounding is frequently achieved by means of linear or Lagrangian relaxations. In this paper, we strengthen the linear relaxation in the following way.

Given a fixed customer $j_0 \in J$, we know that in the optimal solution it will be assigned *a priori* to exactly one agent $i \in I$; that is, in the optimal vector x, x_{ij_0} will be 1 for exactly one $i \in I$. With z^* denoting the optimal value of the SGAP problem, and, for $i \in I$ defining L_{ij_0} as the value of the convex continuous recourse problem

$$
\begin{array}{lll}
\textbf{Min} & \mathcal{Q}(x) & \\
\text{s.t.} & \sum_{i\in I} x_{ij} = 1 & j \in J \\
& x_{ij} \in [0,1] & i \in I, j \in J \\
& x_{i_0 j_0} = 1 &
\end{array}
$$

Model (4.1)

for at least one $i_0 \in I$ it holds that $L_{i_0,j_0} \leqslant z^*$. Thus, for a fixed j_0, we have

$$\min_{i\in I} L_{ij_0} \leqslant z^* \tag{4.4}$$

Since (4.4) holds for any $j_0 \in J$, we obtain the valid lower bound

$$LB := \max_{j\in J} \min_{i\in I} L_{ij}.$$

In the computational experiments reported in this paper, LB has been computed using the L-shaped algorithm proposed in [VW69].

Obviously, the computation of this lower bound requires many evaluations of the function $\mathcal{Q}$. As we will illustrate in Section 6, these calculations can be done efficiently, due to the reformulation of the second-stage problem as proposed in Section 2.1, which allows to evaluate $\mathcal{Q}$ as the expected value function of an LP instead of an IP problem.

4.2 Upper bound

Any assignment of each customer to an agent gives rise to an upper bound for the problem. In this paper we have taken as initial solution the one given by approach B from [AF00]. It consists of taking the assignment found as the optimum of the following deterministic approximation of the problem:

$$
\begin{array}{lll}
\textbf{Min} & \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} & \\
\text{s.t.} & \sum_{i\in I} x_{ij} = 1 & j \in J \\
& \sum_{j\in J} x_{ij} \leqslant \tilde{b}_i & i \in I \\
& x_{ij} \in \{0,1\} & i \in I, j \in J
\end{array}
$$

Model (4.2)

where

$$\tilde{b}_i = \left\lceil b_i \frac{nt}{\sum_{k=1}^{nt} b_k} \right\rceil .$$

That is, auxiliary capacities are defined, proportional to the original ones, that are just big enough to serve one unit of demand to each customer. The solution of this problem is expected to be good, especially for instances where the capacity constraint is tight.

5 Improving strategies

Evaluations of the recourse function $\mathcal{Q}$ are highly time consuming, even though our reformulation allows its computation by solving LP problems instead of integer problems. Therefore, some strategies directed at reducing the number of evaluations needed to solve a problem have been considered.

5.1 Initial cuts

The first-stage problem contains no information about the capacities of the agents and the probability of demand. Consequently, until a few cuts are generated, the solutions of problems IP_K and $P_{K,S}$ are almost meaningless.

The first attempt to improve the behavior of all three versions of the algorithm, has been to consider a few feasible assignments and generate the corresponding cuts, before any optimization is performed.

To choose such assignments, several options have been considered: obtained at random, by solving linear approximations of the problem, and so on. The choice that gave better results was to consider, for each agent, the assignment of all jobs to it. This leads to an initial set of ns optimality cuts. In this phase, ∂–optimality cuts have been used.

5.2 Integrality cuts

The branch and cut scheme has been reinforced with cover cuts derived from the optimality cuts and a global upper bound for $\mathcal{Q}$.

These cover cuts are lifted using the methodology presented in [Wol98]. The order of variable lifting is decisive for the results. In this work lexicographical order has been chosen.

6 Computational results

Computations have been made on a SUN sparc station 10/30 with 4 hyperSPARC processors at 100 MHz., SPECint95 2.35. To solve the problems IP_K and $P_{K,S}$ CPLEX 6.0 was used.

6.1 Problem statistics

All three versions of the algorithm have been tested on a set of 46 problems.

The number of agents ranges from 2 to 4, and the number of customers from 4 to 15. Demand probabilities have been selected from the set

$$\{0.1,\ 0.2,\ 0.4,\ 0.6,\ 0.8,\ 0.9\},$$

and capacities have been chosen in such a way that the ratio of total capacity to total demand ranges from 0.5 to 2.

In principle, the assignment costs c_{ij} have been generated randomly. However, in some cases we have modified the realizations in order to obtain more interesting instances.

6.2 Evaluation of $\mathcal{Q}$

Efficient evaluation of the function $\mathcal{Q}$ is crucial for both the computation of the lower bound LB as well as for all variants of the proposed algorithm.

Due to the reformulation of the second-stage problem as proposed in Section 2.1, which allows to evaluate $\mathcal{Q}$ as the expected value function of an LP instead of an IP problem, these calculations can be done much faster.

In support of this claim, Table 1 shows average CPU times (for each group of instances with given dimensions) for evaluating $\mathcal{Q}$ using both formulations. For each instance, $\mathcal{Q}$ was evaluated in the corresponding optimal binary solution, and also in a randomly generated fractional solution (that is feasible with respect to the first set of constraints in Model (2.2)).

dim	binary x		fractional x	
	$\mathcal{Q} \sim$ IP	$\mathcal{Q} \sim$ LP	$\mathcal{Q} \sim$ IP	$\mathcal{Q} \sim$ LP
2×4	0.09	0.02	0.46	0.03
4×8	2.37	0.31	14.08	0.52
3×7	1.08	0.07	10.21	0.24
3×8	2.46	0.16	20.96	0.37
3×9	5.41	0.30	122.61	0.75
3×10	11.79	0.63	160.04	1.22
3×15	468.39	24.79	16000[2]	75.08
4×7	1.50	0.09	11.57	0.29
4×8	3.21	0.19	24.64	0.77

Table 1: Average CPU times for evaluating $\mathcal{Q}$ at binary and fractional solutions.

Table 1 shows that by using the reformulation of $\mathcal{Q}$ as proposed in Section 2.1, the evaluation times are reduced by a factor up to almost 19 for binary arguments. This effect is even stronger when the argument is fractional, because then it is harder to find integer solutions to Model (2.3). In the latter case the speedup factor exceeds 200 for the 3 instance.

As we will see below, solving an instance from our set of test problems typically takes several hundreds of evaluations of $\mathcal{Q}$.

[2]Time limit reached.

6.3 Numerical results

First, the behavior of the bounding procedures is studied. Results are shown in Table 2. The first five columns show index numbers, dimensions, total capacities, and probabilities of demand of the problems, respectively. Then percent deviations of the upper and the lower bound from the optimal solution value are reported, as well as the gap between both bounds, relative to the lower bound. The CPU time (in seconds) required to compute both bounds appears in the last column.

The relative deviations of the bounds are also shown in Figure 1.

#	ns	nt	total cap.	p	upper dev.	lower dev.	gap	bounding time
1	2	4	2	0.2	0.00	2.82	2.90	0.55
2	2	8	4	0.2	0.50	5.24	6.06	43.43
3	3	7	7	0.1	0.00	0.00	0.00	3.64
4	3	7	7	0.2	0.00	0.00	0.00	7.15
5	3	7	7	0.4	0.00	0.00	0.00	4.93
6	3	7	7	0.6	0.00	0.00	0.00	5.26
7	3	7	7	0.8	0.00	0.00	0.00	4.64
8	3	7	7	0.9	0.00	0.00	0.00	5.51
9	3	7	7	0.1	0.00	0.00	0.00	5.38
10	3	7	7	0.2	0.00	0.00	0.00	2.98
11	3	7	7	0.4	0.00	0.00	0.00	7.79
12	3	7	7	0.6	0.00	0.00	0.00	7.34
13	3	7	7	0.8	0.00	0.00	0.00	6.89
14	3	7	7	0.9	0.00	0.00	0.00	6.10
15	3	8	8	0.8	0.00	0.00	0.00	19.37
16	3	8	6	0.8	0.25	4.20	4.65	84.97
17	3	8	9	0.8	10.22	0.00	10.22	17.77
18	3	8	6	0.6	3.42	6.95	11.14	67.55
19	3	8	15	0.8	33.63	0.00	33.63	6.85
20	3	9	9	0.1	1.95	0.00	1.95	57.75
21	3	9	9	0.2	1.44	0.00	1.44	54.16
22	3	9	9	0.4	0.00	0.00	0.00	56.61
23	3	9	9	0.6	0.00	0.00	0.00	42.97
24	3	9	9	0.8	0.00	0.00	0.00	36.49
25	3	9	9	0.9	0.00	0.00	0.00	36.83
26	3	10	9	0.8	1.22	23.97	33.13	193.04
27	3	10	10	0.4	1.53	0.11	1.64	110.53
28	3	15	14	0.4	0.75	0.00	0.75	6906.27
29	4	7	6	0.1	0.00	0.00	0.00	44.96
30	4	7	6	0.2	0.00	0.27	0.27	65.37
31	4	7	6	0.4	0.00	4.23	4.42	122.91
32	4	7	6	0.6	0.00	7.87	8.54	99.77
33	4	7	6	0.8	0.00	6.35	6.78	67.05
34	4	7	6	0.9	0.00	4.84	5.08	111.74
35	4	8	7	0.1	0.00	0.00	0.00	95.10
36	4	8	7	0.2	0.00	0.23	0.23	146.14
37	4	8	7	0.4	0.00	3.84	3.99	210.44

#	ns	nt	total cap.	p	upper dev.	lower dev.	gap	bounding time
38	4	8	7	0.6	0.00	7.84	8.51	306.18
39	4	8	7	0.8	0.02	6.63	7.12	234.86
40	4	8	7	0.9	0.09	4.96	5.31	192.87
41	4	8	7	0.1	0.00	0.00	0.00	112.36
42	4	8	7	0.2	0.00	0.30	0.30	117.29
43	4	8	7	0.4	0.00	4.44	4.65	158.55
44	4	8	7	0.6	0.03	8.79	9.68	146.13
45	4	8	7	0.8	0.15	6.64	7.28	309.64
46	4	8	7	0.9	0.00	4.94	5.20	149.43

Table 2: Relative deviations of bounds and CPU time (in seconds).

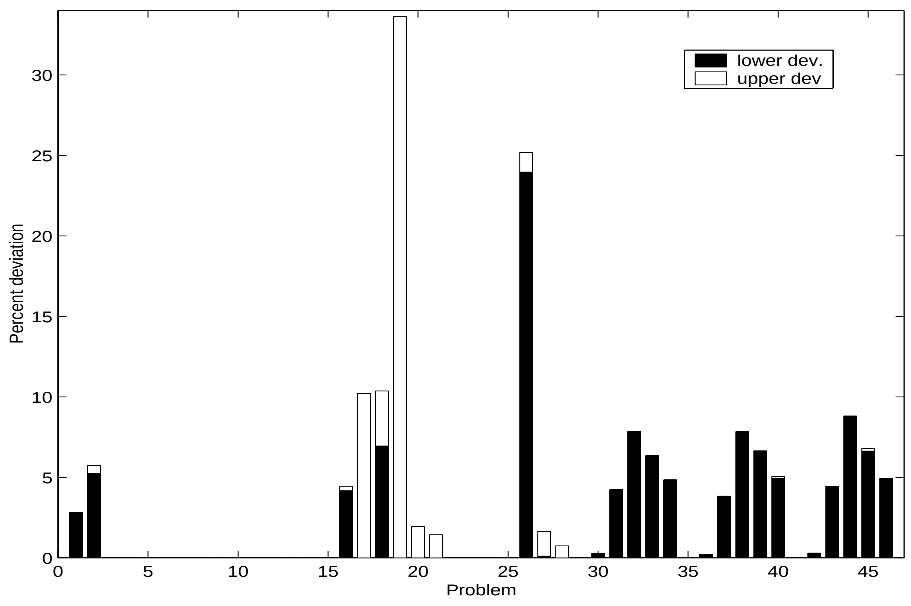

Figure 1: Percent deviations (summed) of the bounds.

The results in Table 2 indicate that the CPU times needed to complete the bounding phase are reasonable.

Due to its dimensions, the bounding time for problem 28 is considerably higher than for the other instances. Its relatively large number of customers affects the computation of the lower bound in two ways. First, the number of subproblems Model (4.1) to be solved is higher (45 subproblems, compared to e.g. 32 for 4×8 instances). Secondly, as shown in Table 1, the evaluation of $\mathcal{Q}$ is much more time consuming in this case since the number of realizations of the demand vector grows exponentially with the number of customers (32768 realizations, instead of e.g. 1024 for the problems with 10 customers).

As can be seen in Table 2 and in Figure 1, for many of the test problems the gap between the upper and the lower bound is zero, so that an optimal solution is already identified at the bounding phase. For such problems, Table 3 shows the CPU time needed for bounding in the BCS column; for all other instances it shows the total CPU time needed to find an optimal solution, for each version of the algorithm.

#	*ns*	*nt*	cap.	*p*	BCS	CFBS	BFCS
1	2	4	2	0.2	0.76	0.67	0.80
2	2	8	4	0.2	91.04	221.29	368.22
3	3	7	7	0.1	3.64	—	—
4	3	7	7	0.2	7.15	—	—
5	3	7	7	0.4	4.93	—	—
6	3	7	7	0.6	5.26	—	—
7	3	7	7	0.8	4.64	—	—
8	3	7	7	0.9	5.51	—	—
9	3	7	7	0.1	5.38	—	—
10	3	7	7	0.2	2.98	—	—
11	3	7	7	0.4	7.79	—	—
12	3	7	7	0.6	7.34	—	—
13	3	7	7	0.8	6.89	—	—
14	3	7	7	0.9	6.10	—	—
15	3	8	8	0.8	19.37	—	—
16	3	8	6	0.8	2226.80	24054.04	335800.11
17	3	8	9	0.8	19.51	17.91	16.99
18	3	8	6	0.6	402.36	4903.32	18313.17
19	3	8	15	0.8	8.05	6.97	8.69
20	3	9	9	0.1	63.92	58.22	43.40
21	3	9	9	0.2	59.65	54.64	58.13
22	3	9	9	0.4	56.61	—	—
23	3	9	9	0.6	42.97	—	—
24	3	9	9	0.8	36.49	—	—
25	3	9	9	0.9	36.87	—	—
26	3	10	9	0.8	1156.66	5803.34	6581.60
27	3	10	10	0.4	122.02	113.48	119.97
28	3	15	14	0.4	7384.26	6933.17	*
29	4	7	6	0.1	44.96	—	—
30	4	7	6	0.2	68.42	67.53	67.46
31	4	7	6	0.4	264.90	1216.38	283.16
32	4	7	6	0.6	990.97	16039.32	14914.99
33	4	7	6	0.8	1889.68	25202.24	89437.42
34	4	7	6	0.9	2294.23	22155.11	66.22
35	4	8	7	0.1	95.10	—	—
36	4	8	7	0.2	160.36	148.85	182.15
37	4	8	7	0.4	492.42	2923.12	804.82
38	4	8	7	0.6	4957.58	110683.83	143552.59
39	4	8	7	0.8	15085.77	*	1119394.47
40	4	8	7	0.9	16552.54	*	183.49

#	*ns*	*nt*	cap.	*p*	BCS	CFBS	BFCS
41	4	8	7	0.1	112.36	—	—
42	4	8	7	0.2	120.11	113.41	139.71
43	4	8	7	0.4	389.54	1860.08	351.94
44	4	8	7	0.6	5558.37	101081.14	*
45	4	8	7	0.8	17704.89	107036.85	*
46	4	8	7	0.9	15188.75	112888.18	*

Table 3: CPU times in seconds for each version of the algorithm. (* denotes an unsolved instance; — denotes an instance solved in the bounding phase.)

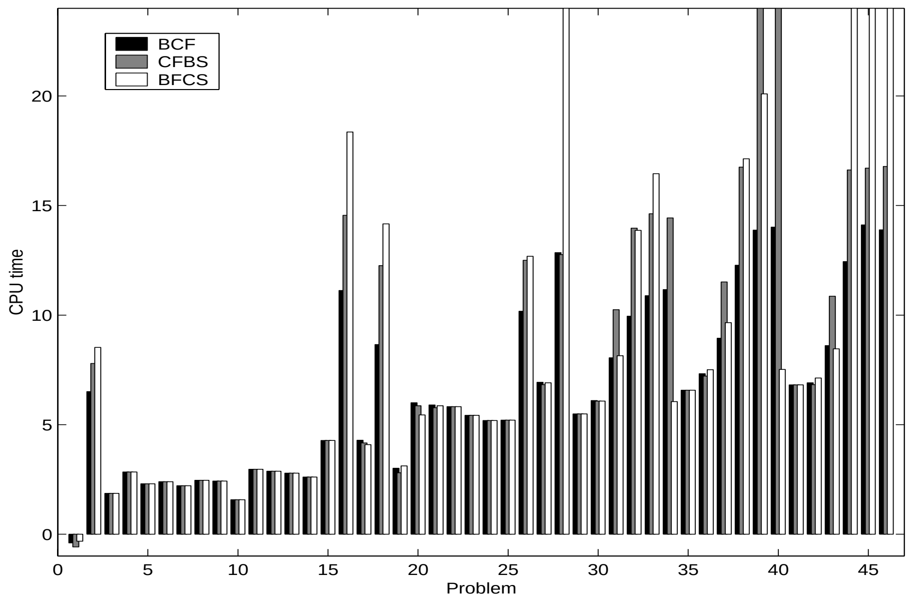

Figure 2: Logarithm of CPU times of each version of the algorithm, for each problem instance. (A value of 24 denotes an unsolved instance.)

For most of the test problems, an optimal solution was found by each version of the algorithm within the prespecified limits on the number of the evaluations of the recourse function. The limit was 5000 in the case of BFS. For BFCS, since many cuts are expected to be added at each node of the tree, the limit was raised to 15000. On the other hand, since BFCS requires the solution of an integer problem before each evaluation of the recourse function, the limit was set to only 1000 in this case.

The CFBS version of the algorithm did not solve problem instances 39 and 40. The same happened with the BFCS version on four different instances: numbers 28 and 44 – 46. The BCS version of the proposed algorithm solved all problem instances. Moreover, as can be seen from Table 3 and Figure 2, BCS is an order of magnitude faster on several hard instances (problem number 34 being a remarkable exception).

Hardness of the problem instances is not determined by their dimensions alone. For example, we see that the solution times were very different for problems 16 and 19 (both 3×8) and for problems 42 and 46 (both 4×10). In the case of problems 16 and 19, the only difference between them is the tightness of the capacity constraint, with total capacities of 6 and 15, respectively. Similarly, problems 42 and 46 differ only in their respective probabilities of demand, which are 0.2 and 0.9.

Finally, we compare the performance of the proposed (versions of the) algorithm to a naive complete enumeration. For an instance with ns agents and nt jobs, there are $P = ns^{nt}$ possible assignments of jobs to agents. For each assignment, the recourse function $\mathcal{Q}$ needs to be evaluated.

For each version of the proposed algorithm, the number of evaluations of the recourse function used was in general substantially smaller than the total number of possible assignments, as shown in Table 4.

dim.	bounding	#	BCS	CFBS	BFCS	P
2×4	22.00	1	27.00	27.00	27.00	16
4×8	88.00	1	171.00	185.00	369.00	256
3×7	54.25	0	—	—	—	2187
3×8	113.60	4	634.50	451.00	2341.50	6561
3×9	98.00	2	95.50	97.00	95.00	19683
3×10	154.00	2	433.00	195.50	1213.00	59049
3×15	199.00	1	203.00	*	202.00	14348907
4×7	280.17	5	952.20	293.60	3754.40	16384
4×8	334.17	10	2028.20	515.13*	6565.50*	65536

Table 4: Evaluations of the recourse function.

For each dimension, the second column contains the average number of evaluations of $\mathcal{Q}$ needed in the bounding phase. In the subsequent columns, only problems which were not already solved in the bounding phase have been considered. The third column contains the number of such problems. The average total number of evaluations of $\mathcal{Q}$ (bounding phase, initial cuts, and cuts generated by the algorithm) that each version of the algorithm required to find the optimum are reported in the next three columns; a star denotes that at least one of the problems was not solved. The last column shows the number of function evaluations needed in a complete enumeration, which is equal to the number of possible assignments.

7 Conclusions

This paper considers a stochastic Generalized Assignment Problem (GAP) where the demand for a certain resource of each job is not known in advance, but can be modeled as a Bernoulli random variable with known success rate. Jobs are interpreted as customers that may or may not require a service.

An assignment of jobs to agents is designed *a priori*, but can be modified once the actual demands are known. Different penalties are paid for reassigning jobs and for leaving unprocessed jobs with positive demand. The objective is to minimize the total expected costs.

This GAP is modeled as a stochastic problem with recourse. The recourse function reflects the expected cost associated to each possible *a priori* assignment.

Due to the binary variables in the defining second-stage problem, this function is non-convex in general. In this paper, we construct a convex approximation of our particular recourse function that is sharp at all feasible points. Based on this approximation, we propose an exact algorithm to solve the problem. Integrality of the first-stage variables is addressed using branch and bound techniques, and the objective function is iteratively approximated using two different kinds of cuts.

Three versions of the algorithm are proposed and tested. In the first version (CFBS), the main effort is devoted to getting a rich approximation of the recourse function, while the third one (BFCS), gives priority to integrality of the current solutions. The second version (BCS) is a tradeoff between the other two.

The computational experiences show that the performance of BCS is in general the best one, in particular for the harder instances of GAP.

This suggests that it is more efficient to tackle integrality and stochasticity at the same level, instead of considering these two factors separately. Indeed, the two solution schemes that give priority to one single factor, appear to waste a lot of time exploring solutions that are not optimal because of the other factor.

This paper also presents a lower bound for the GAP in consideration. A stochastic subproblem is derived from each possible assignment of a single customer and its linear relaxation is solved using the L-shaped algorithm. The actual lower bound is obtained from the solutions to these subproblems. This lower bound has proved to be very tight in the computational experiments.

One of the heuristics presented in [AF00] is used to obtain an initial solution, as well as to provide the algorithm with a suitable upper bound.

CPU times required to obtain both bounds are relatively small for all but one instance. In 43% of the cases, the bounding phase was sufficient to identify an optimal solution. On the other hand, there are also a few instances with a large gap between the bounds.

For those instances where the bounding phase did not already provide an optimal solution, the exact algorithm gave satisfactory results. Times required to solve such instances were reasonable, taking into account the high difficulty of the problem. The success of the algorithm is mostly due to the approximation of the recourse function that we use. It is much faster to evaluate than the original recourse function and moreover it is convex, which allows to obtain good approximations using a cutting plane procedure.

References

[AF00] Maria Albareda Sambola and Elena Fernández Aréizaga. The stochastic generalised assignment problem with Bernoulli demands. *TOP*, 8(2):165–190, 2000.

[ATS00] S. Ahmed, M. Tawarmalani, and N.V. Sahinidis. A finite branch and bound algorithm for two-stage stochastic integer programs. World Wide Web, `http://archimedes.scs.uiuc.edu/papers/sip.pdf`, 2000.

[Bal72] V. Balachandran. An integer generalized transportation problem for optimal job assignment in computer networks. Technical Report 43-72-3, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, 1972.

[BC97] J.E. Beasley and P.C. Chu. A genetic algorithm for the generalised assignment problem. *Computers and Operations Research*, 24:17–23, 1997.

[BL97] J.R. Birge and F.V. Louveaux. *Introduction to Stochastic Programming.* Wiley, New York, 1997.

[BSL88] Oded Berman and David Simchi-Levi. Finding the optimal a priori tour and location of a traveling salesman with nonhomogeneous customers. *Transportation Science*, 22(2):148–154, 1988.

[CPL00] CPLEX Optimization, Inc. Using the CPLEX callable library. Manual, 2000.

[CS99] Claus C. Carøe and Rüdiger Schultz. Dual decomposition in stochastic integer programming. *Operations Research Letters*, 24(1-2):37–45, 1999.

[CT98] Claus C. Carøe and Jørgen Tind. L-shaped decomposition of two-stage stochastic programs with integer recourse. *Mathematical Programming*, 83(3):451–464, 1998.

[CW92] Dirk G. Cattrysse and Luck N. Van Wassenhove. A survey of algorithms for the generalized assignment problem. *European Journal of Operational Research*, 60:260–272, 1992.

[DF01] Juan A. Díaz and Elena Fernández. A tabu search heuristic for the generalized assignment problem. *European Journal of Operational Research*, 132:22–38, 2001.

[Día01] Juan A. Díaz. Algorithmic approaches for the single source capacitated plant location problem, 2001. PhD Thesis, Universitat Politècnica de Catalunya, Barcelona, June, 2001.

[Edm65] J. Edmonds. Maximum matching and a polyhedron with 0-1 vertices. *Journal of Research of the National Bureau of Standards*, B69:125–130, 1965.

[FJ81] M.L. Fisher and Jaikumar. A generalized assignment heuristic for the large scale vehicle routing. *Networks*, 11:109–124, 1981.

[KSV96] W.K. Klein Haneveld, L. Stougie, and M.H. van der Vlerk. An algorithm for the construction of convex hulls in simple integer recourse programming. *Annals of Operations Research*, 64:67–81, 1996. Stochastic programming, algorithms and models (Lillehammer, 1994).

[Kuh55] H.W. Kuhn. The hungarian method for the assignment problem. *Naval Res. Logist. Quart.*, 2:83–97, 1955.

[KV99] W.K. Klein Haneveld and M.H. van der Vlerk. Stochastic integer programming: general models and algorithms. *Annals of Operations Research*, 85:39–57, 1999.

[KW94] P. Kall and S. W. Wallace. *Stochastic Programming.* Wiley, Chichester etc., 1994.

[LL92] Gilbert Laporte and François V. Louveaux. The integer L-shaped method for stochastic integer programs with complete recourse. *Operations Research Letters*, 13:133–142, 1992.

[LLM94] G. Laporte, F. Louveaux, and H. Mercure. An exact solution for the a priori optimization of the probabilistic traveling salesman problem. *Operations Research*, 39:71–78, 1994.

[LV93] F.V. Louveaux and M.H. van der Vlerk. Stochastic programming with simple integer recourse. *Mathematical Programming*, 61:301–325, 1993.

[MFIS83] H. Mine, M. Fukushima, K. Ishikawa, and I. Sawa. An algorithm for the assignment problem with stochastic side constraints. *Memoirs of the Faculty of Engineering*, XLV(part 4), 1983.

[MT90] S. Martello and P. Toth. *Knapsack problems, algorithms and computer implementations.* Wiley, 1990.

[NER98] V.I. Norkin, Yu.M. Ermoliev, and A. Ruszczyński. On optimal allocation of indivisibles under uncertainty. *Operations Research*, 46(3):381–395, 1998.

[NW88] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization.* Wiley, New York, 1988.

[Pr95] A. Prékopa. *Stochastic Programming.* Kluwer Academic Publishers, Dordrecht, 1995.

[RS77] G.T. Ross and R.M. Soland. Modeling facility location problems as generalized assignment problems. *Management Science*, 24(3):345,357, 1977.

[RS01] W. Römisch and R. Schultz. Multistage stochastic integer programs: An introduction. Stochastic Programming E-Print Series, `http://dochost.rz.hu-berlin.de/speps/`, 2001.

[Sav97] M.W.P. Savelsbergh. A Branch-and-Price algorithm for the Generalized Assignment Problem. *Operations Research*, 45:831–841, 1997.

[Sch93] R. Schultz. Continuity properties of expectation functions in stochastic integer programming. *Mathematics of Operations Research*, 18:578–589, 1993.

[SM99] Robert A. Stubbs and Sanjay Mehrotra. A branch-and-cut method for 0-1 mixed convex programming. *Mathematical Programming*, 86(3):515–532, 1999.

[SSV96] R. Schultz, L. Stougie, and M. H. van der Vlerk. Two-stage stochastic integer programming: a survey. *Statistica Neerlandica*, 50(3):404–416, 1996.

[SSV98] R. Schultz, L. Stougie, and M.H. van der Vlerk. Solving stochastic programs with complete integer recourse: A framework using Gröbner Bases. *Mathematical Programming*, 83(2):229–252, 1998.

[SV97] L. Stougie and M. H. van der Vlerk. Stochastic integer programming. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, chapter 9, pages 127–141. Wiley, 1997.

[Vle01] M. H. van der Vlerk. Stochastic programming bibliography. World Wide Web, `http://mally.eco.rug.nl/biblio/stoprog.html`, 1996-2001.

[VW69] R. Van Slyke and R. Wets. L-shaped linear programs with applications to optimal control and stochastic linear programs. *SIAM journal on Applied Mathemathics*, 17:638–663, 1969.

[Wol80] Richard D. Wollmer. Two stage linear programming under uncertainty with 0-1 integer first stage variables. *Mathematical Programming*, 19:279–288, 1980.

[Wol98] Laurence A. Wolsey. *Integer programming.* John Wiley & Sons Inc., New York, 1998.

[YIG99] M. Yagiura, T. Ibaraki, and F. Glover. An ejection chain approach for the Generalized Assignment Problem. Technical Report 99013, Departament of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, 1999.

[YT74] D. B. Yudin and E. V. Tsoy. Integer-valued stochastic programming. *Engrg. Cybernetics*, 12:1–8, 1974.

[YTI98] M. Yagiura, T. Tamaguchi, and T. Ibaraki. A Variable Depth Search algorithm with branching search for the Generalized Assignment Problem. *Optimization Methods and Software*, 10:419–441, 1998.

[ZR88] V.A. Zimokha and M.I. Rubinshtein. R&D planning and the generalized assignment problem. *Automation and Remote Control*, 49:484–492, 1988.