Chapter 4

Stochastic Generalized Assignment Problem*

Introduction

Pairing problems constitute a vast family of problems in CO. Different versions of these problems have been studied since the mid 1950s 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 (Kuhn 1955). 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 (Edmonds 1965). On the other extreme, the Generalized Assignment Problem is a very difficult CO problem that is $\mathcal{NP}$-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 (Balachandran 1972), distribution of activities to the different institutions when making a project plan (Zimokha and Rubinshtein 1988), etc. Besides these applications, it also appears as a subproblem in a variety of combinatorial problems like Vehicle Routing (Fisher and Jaikumar 1981) or Plant Location (Ross and Soland 1977; Díaz 2001). A survey of exact and heuristic algorithms to solve the GAP can be found in Cattrysse and Van Wassenhove (1992). More recently, (Savelsbergh 1997) proposed a Branch and Price algorithm, and different metaheuristic approaches have been proposed by other authors (Chu and Beasley 1997; Díaz and Fernández 2001; Yagiura, Tamaguchi, and Ibaraki 1998; Yagiura, Ibaraki, and Glover 1999).

As mentioned before, the GAP is a very challenging problem, not only because it is $\mathcal{NP}$-hard, but also because the decision problem to know if a given instance is feasible is $\mathcal{NP}$-complete (Martello and Toth 1990). In fact, the GAP is in practice even more difficult, since most of its applications have

*The contents of this chapter areincluded in Albareda-Sambola and Fernández (2000) and Albareda-Sambola, van der Vlerk, and Fernández (2002).

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 (Berman and Simchi-Levi 1988; Laporte, Louveaux, and Mercure 1994)

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

We are aware of only two papers addressing any type of stochastic assignment problems. Mine, Fukushima, Ishikawa, and Sawa (1983) present a heuristic for an assignment problem with stochastic side constraints. The second paper (Albareda-Sambola and Fernández 2000), partially contained in this chapter, proposes some heuristics based on approximation models for the SGAP with Bernoulli demands.

To the best of our knowledge, no results on exact algorithms for any type of stochastic GAP are known.

In this chapter, we consider the following SGAP: 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 SGAP. 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 Van Slyke and Wets (1969) and in Laporte and Louveaux (1992). 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 SGAP, 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 versions of the exact algorithm compute the same upper and lower bounds initially. The upper bound is obtained with the heuristics presented in Section 4.2. The lower bound is found by solving a family of stochastic linear subproblems with the L-shaped algorithm.

The remainder of this chapter is organized as follows: In Section 4.1 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 proposed heuristics are presented in 4.2 and the exact algorithms are described in Section 4.3. After that, Section 4.4 gives a description of the lower bound. In Section 4.5 some strategies to improve the performance of the algorithms are discussed. Finally, computational experiences and conclusions are presented in Sections 4.6 and 4.7 respectively.

4.1 Problem description and modeling

The GAP consists of 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 $|I| = n_s$ and $|J| = n_t$. We define, for $i \in I$ and $j \in J$,

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

d_{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 $B = \sum_{i \in I} b_i$

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:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} \\
\text{subj. to} \quad & \sum_{i\in I} x_{ij} = 1 \qquad\qquad j\in J \\
& \sum_{j\in J} d_{ij}x_{ij} \leqslant b_i \qquad\quad i\in I \\
& x_{ij}\in\{0,1\} \quad\ i\in I,\ j\in J.
\end{aligned}$$

Most often, however, the assignment of jobs to agents must be decided before the actual values of the demands for resource capacity $d_{ij}, i \in I, j \in J$, are known, so that the above model is no longer valid. In what follows, ξ denotes the random vector of demands, and we assume that, once the values of the vector ξ become available, it is possible to reassign some of the jobs, incurring costs $K_j, j \in J$.

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

$$\begin{aligned}
\text{(FSP)} \quad \text{minimize} \quad & \mathcal{Q}(x) \\
\text{subj. to} \quad & \sum_{i\in I} x_{ij} = 1 \quad j\in J \\
& x_{ij}\in\{0,1\} \quad i\in I, j\in J,
\end{aligned}$$

where[*] $\mathcal{Q}(x) := \mathbb{E}_\xi v(x,\xi)$ is the recourse function, and

$$\begin{aligned}
\text{(SSP)} \quad v(x,\xi) = \text{minimize} \quad & \sum_{i\in I}\sum_{j\in J} c_{ij}y_{ij} + \sum_{j\in J} K_j z_j \\
\text{subj. to} \quad & y_{ij} + z_j \geqslant q_j x_{ij} \qquad i\in I, j\in J \\
& \sum_{i\in I} y_{ij} \geqslant \xi_j \qquad\qquad j\in J \\
& \sum_{j\in J} y_{ij} \leqslant b_i \qquad\qquad i\in I \\
& y_{ij}\in\{0,1\} \qquad\qquad i\in I, j\in J \\
& z_j \in\{0,1\} \qquad\qquad j\in J.
\end{aligned}$$

At 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 ($j \in J$) denote 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 actually assigned to an agent and that capacities of the agents are not violated, respectively.

[*]$\mathbb{E}$ stands for mathematical expectation.

FSP is a two-stage recourse model with binary variables in both stages. Thus, in addition to difficulties caused by integrality of the first-stage variables, the recourse function $\mathcal{Q}$ is non-convex in general. Indeed, since the parameters ξ are discretely distributed, it is lower semicontinuous but discontinuous in general (Schultz 1993). Moreover, evaluation of $\mathcal{Q}$ for a given x calls for solving many second-stage problems, which, in this case, are binary programming problems. Since these second-stage problems are not easily solvable, this is computationally very demanding. In the next subsection we show how to overcome these problems by redefining the second-stage problem.

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 4.1.1. *Given a feasible solution x of FSP and ξ a 0-1 vector, SSP is equivalent to*

$$
\begin{array}{lll}
\text{(SSP2)} \quad \textit{minimize} & \sum_{i\in I}\sum_{j\in J} c_{ij}y_{ij} + \sum_{i\in I}\sum_{j\in J} K_j z_{ij} & \\
\textit{subj. to} & y_{ij} + z_{ij} \geqslant \xi_j x_{ij} & i \in I, j \in J \\
& \sum_{i\in I} y_{ij} \geqslant \xi_j & j \in J \\
& \sum_{j\in J} y_{ij} \leqslant b_i & i \in I \\
& y_{ij} \in \{0,1\} & i \in I, j \in J \\
& z_{ij} \in \{0,1\} & i \in I, j \in J
\end{array}
$$

Proof. Let (y^*, z^*) be an optimal solution to SSP2. From the fact that for a feasible point x each job is assigned *a priori* to exactly one agent it follows that, in the optimal solution, for a fixed $j \in J$ at most one of the variables z^*_{ij} $(i \in I)$ takes value 1 and the others are 0. So, $\tilde{z}_j = \max_{i\in I} z^*_{ij}$, $(y^*, \tilde{z})$ is a feasible point for SSP, with the same objective function value as (y^*, z^*).

Similarly, given an optimal solution (y^*, z^*) of SSP, a feasible solution $(y^*, \tilde{z})$ of SSP2 with the same objective value can be built as follows:

- If $z^*_j = 0$, take $\tilde{z}_{ij} = 0, \forall i \in I$.
- If $z^*_j = 1$, because of the structure of x, only one of the flag constraints is tight, say (i_1, j). Then, 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. □

Proposition 4.1.2. *The matrix defining the feasible region of* SSP2 *is totally unimodular (TU).*

Proof.

The structure of the matrix is:

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

where M is the matrix of a transportation problem, which is known to be TU. By Proposition III.2.1 in Nemhauser and Wolsey (1988), the matrices

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

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 SSP2 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 SSP2 and its relaxation are equal.

Assuming that the capacities b_i, $i \in I$, are integral, the right-hand side of SSP3 is integral for every feasible (i.e., binary) first-stage solution to the GAP. Thus, using Propositions 4.1.1 and 4.1.2, we redefine the function $v(x, \xi)$ as the optimal value of the LP problem

$$\begin{array}{lll}
\text{(SSP3)} \quad \text{minimize} & \displaystyle\sum_{i \in I} \sum_{j \in J} c_{ij} y_{ij} + \sum_{j \in J} K_j z_j & \\
& y_{ij} + z_j \geqslant \xi_j x_{ij} & i \in I, j \in J \\
& \displaystyle\sum_{i \in I} y_{ij} \geqslant \xi_j & j \in J \\
& \displaystyle\sum_{j \in J} y_{ij} \leqslant b_i & i \in I \\
& y_{ij} \in [0, 1] & i \in I, j \in J \\
& z_j \geqslant 0 & j \in J
\end{array}$$

As stated in Proposition 4.1.1, this function coincides with the previously defined $v(x, \xi)$ in all feasible vectors x, for any demand vector ξ. 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 below, these properties carry over to the recourse function $\mathcal{Q}$, which is defined as the expectation of v.

Remark 4.1.1. Observing that SSP3 is simply the LP relaxation of the original second-stage SSP, the question rises whether the constraint matrix of this problem is TU itself. As we show below, this is not the case.

Consider the submatrix of SSP formed by columns corresponding to variables y_{00}, y_{10} and z_0, and the two rows of the first group concerning variables y_{00} and y_{10}, and the row in the second group concerning job 0. This submatrix equals

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

with determinant -2, so that it is not TU. This submatrix occurs in any instance with at least one job and two agents; matrices of all non-trivial instances are therefore not TU.

Consequently, there is no guarantee that SSP3 yields the correct optimal value for all integer right-hand sides. However, Propositions 4.1.1 and 4.1.2 imply that SSP3 does give the correct value for all *relevant* right-hand sides, i.e., those corresponding to feasible first-stage solutions.

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 (Birge and Louveaux 1997; Kall and Wallace 1994; Prékopa 1995).

Remark 4.1.2. Assuming that

$$n_t \leqslant B \tag{4.1}$$

it follows that FSP 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 ξ, the second-stage value function v is finite for all $x \in R^{n_s \cdot n_t}$.

If (4.1) is not satisfied, relatively complete recourse is obtained by introducing a dummy agent $0 \in I$ with enough capacity in the second stage, which 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 4.1.3. *The function $\mathcal{Q}$ is finite and convex. Moreover, since the random parameters are discretely distributed, $\mathcal{Q}$ is a polyhedral function.*

Proposition 4.1.4. *Let S be the index set of scenarios (realizations). For $s \in S$, let p^s be the probability of scenario s and ξ^s the corresponding demand vector, so that*

$$\mathcal{Q}(x) = \sum_{s \in S} p^s v(x, \xi^s), \quad x \in R^{n_s \cdot n_t}. \tag{4.2}$$

Let $\lambda(x, \xi^s)$ be a vector of dual prices for the first set of constraints in SSP3 for the pair (x, ξ^s). Then $u(x)$,

$$u(x) = \sum_{s \in S} p^s \lambda(x, \xi^s) diag(\bar{\xi}^s_1, \ldots, \bar{\xi}^s_{n_t}), \tag{4.3}$$

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

4.2 Heuristic Solutions for the Stochastic Generalized Assignment Problem

In this section we present three heuristics for the SGAP under consideration. All heuristics consist in solving an auxiliary deterministic model that approximates FSP.

Heuristic A: Chance Constrained Model

The first of the proposed approximation models is a chance constrained model. Since the sum of Bernoulli *i.i.d.* variables is a Binomial, and we know its distribution, it is easy to find the deterministic equivalent for problems where sums of ξ variables appear in probabilistic constraints.

Penalties for reassignments are likely to have higher values than the assignment costs, so, it is desirable to have solutions where a small number of reassignments is likely to be needed. We propose a model where capacity violations are only allowed to occur with a prespecified probability α. If a plant i has k customers assigned, the probability that reassignments are needed is given by:

$$\mathbb{P}[\text{More than } b_i \text{ among } k \text{ customers have demand}] = \sum_{s=b_i+1}^{k} \binom{k}{s} p^s (1-p)^{k-s}.$$

Thus, by taking

$$K(b_i, p, \alpha) = \max \left\{ k \in \mathbb{Z} \mid \sum_{s=b_i+1}^{k} \binom{k}{s} p^s (1-p)^{k-s} \leqslant \alpha \right\},$$

we can find good assignments as the solutions of

$$\begin{aligned}
\text{minimize} \quad & \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} && && (4.4)\\
\text{subj. to} \quad & \sum_{i\in I} x_{ij} \geqslant 1 && j \in J && (4.5)\\
& \sum_{j\in J} x_{ij} \leqslant K(b_i, p, \alpha) && i \in I, j \in J && (4.6)\\
& x_{ij} \in \{0,1\} && i \in I, j \in J. && (4.7)
\end{aligned}$$

Note that, since $K(b_i, p, \alpha)$ takes integer values, and demands are binary, integrality constraints can be dropped in the resolution of the problem.

The choice of the parameter α will determine the quality of this model as an approximation of FSP. High values of α can lead to bad assignments, in the sense of capacity constraints violations, whereas the problem can become infeasible for small values. Since feasibility is easily checkable for model (4.4)-(4.7), the strategy we propose is to initially set α to a small value, and increase it successively until the model becomes feasible.

Heuristic B: Transportation Model

In general terms, the model proposed as the basis of Heuristic A uses auxiliary capacities for the plants that guarantee the feasibility of the assignment problem, but that are tight enough as to avoid unbalanced workloads for the agents. What we use as an alternative to Heuristic A is to solve a similar model, but using simpler auxiliary capacities. The values we propose for the new capacities are

$$\tilde{b}_i = \left\lceil b_i \frac{n_t}{B} \right\rceil$$

That is, we propose to take capacities proportional to the real ones, but large enough as to globally satisfy the demands of all the customers.

The model to solve now is (4.4)-(4.7) where constraints (4.6) have been replaced by

$$\sum_{j\in J} x_{ij} \leqslant \tilde{b}_i, \quad i \in I.$$

Note that, again, total unimodularity of the constraints matrix and integrality of the right hand side allow us to solve the problem without taking into account the integrality constraints (4.7) explicitly.

Heuristic C: Generalized Assignment Problem

Both approaches presented above for the stochastic GAP try to avoid reassignments as much as possible while taking into account the *a priori* assignment costs. However,once all necessary reassignments have been done, the actual assignment costs can differ significantly from the original ones. What we propose next is to give priorities to the customers in such a way that those customers with wider ranges of assignment costs have more stable assignments.

Let δ_j be the ratio between the range of assignment costs corresponding to customer j, and the total range of costs,

$$\delta_{j_0} = \frac{\max\{c_{ij_0} | i \in I\} - \min\{c_{ij_0} | i \in I\}}{\max\{c_{ij} | i \in I, j \in J\} - \min\{c_{ij} | i \in I, j \in J\}},$$

and R the expected ratio between the aggregated demand and the total capacity.

$$R = \frac{p\, n_t}{B}.$$

In problems with small values of R there is a low probability that capacity violations occur. So, reassignments are not likely to be needed too often. On the contrary, when R takes large values, one can expect that reassignments will be often carried out, and it is in those cases that stability of customers j with large values of δ_j can be profitable.

Priorities are given to the customers by assigning them different demands, depending on their associated range ratio δ_j. We use the parameter R to scale the influence of δ in function of the structure of the problem. The demands we use are

$$d_j = 1 + R\,\delta_j,$$

where a minimum value 1 is set to prevent degenerate problems with customers whose demand is too close to 0.

Again, the capacities of the plants must be replaced by suitable values, for the resulting problem to be feasible. This time, however, feasibility is not as easy to state as it was in the former cases, where customers demands where always set to 1. As in the former model, we chose capacities proportional to the original ones but this time, the scaling factor is a bit more involved. We take, for each plant, the ratio

$$r_i = n_t \frac{b_i}{B},$$

as an indicator of the number of customers it should assume. Then, we define the auxiliary capacities

$$\tilde{b}_i = \rho\, d\, r_i,$$

where d is the average demand of the customers, and ρ is a scaling factor used to ensure feasibility.

Using these definitions for the demands and the capacities, we can find promising *a priori* assignments by solving the problem

$$\text{minimize} \quad \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \tag{4.8}$$

$$\text{subj. to} \quad \sum_{i \in I} x_{ij} = 1 \qquad j \in J \tag{4.9}$$

$$\sum_{j \in J} d_j x_{ij} \leqslant \tilde{b}_i \qquad i \in I, j \in J \tag{4.10}$$

$$x_{ij} \in \{0,1\} \qquad i \in I, j \in J. \tag{4.11}$$

Algorithm 4.1 shows a simple heuristic used to find suitable values for the parameter ρ.

Algorithm 4.1 Set_ρ

```
Initialization: $\rho \leftarrow 1, StopCriterion \leftarrow$ **false**
**while** (**not** $StopCriterion$) **do**
    $\tilde{b}_i \leftarrow \rho d r_i, i \in I, \tilde{b}_0 \leftarrow \infty$
    Use a first fit decreasing heuristic to find
     an assignment of jobs in $J$ to agents in $I \cup \{0\}$
    **if** (Agent 0 is used) **then**
        $\rho \leftarrow \rho + 0.01$
    **else**
        $StopCriterion \leftarrow$ **true**
    **end if**
**end while**
```

Note that (4.8)-(4.11) is the model of a deterministic GAP. So, the computational effort required to solve it exactly is considerable. Since it is already an approximation of the problem we want to solve, it is sensible to solve it approximately by using a heuristic. After testing some existing heuristics as well as the exact resolution, we have chosen the simple heuristic depicted in Algorithm 4.2.

Algorithm 4.2 GAP_rounding

```
Solve the linear relaxation of (4.8)-(4.11) $\longrightarrow \quad x$
**for** ($j \in J$) **do**
    **if** (The demand of $j$ is splitted in $x$) **then**
        Chose $i_0 \in \arg\min\{\frac{c_{ij}}{x_{ij}} \mid x_{ij} > 0\}$
        Assign job $j$ to $i_0$
    **end if**
**end for**
```

4.3 Exact Algorithm Variants

Van Slyke and Wets used the properties of the recourse function for continuous two-stage programs presented in Section 4.1 to develop the so-called L-shaped algorithm (Van Slyke and Wets 1969). 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 FSP by a continuous variable θ and to include a constraint of the form

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

in the definition of the feasible region (4.8)-(4.11). 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 4.1.3 and 4.1.4 we derive cuts of the form:

$$\theta \geqslant \mathcal{Q}(\bar{x}) + \langle x - \bar{x}, u(\bar{x}) \rangle, \tag{4.12}$$

where $\langle \cdot, \cdot \rangle$ denotes the usual inner product. We will call (4.12) *∂-optimality cuts* to distinguish them from the L-L-optimality cuts defined below.

Integrality in two-stage programming was first addressed in (Wollmer 1980) 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 (see Laporte and Louveaux 1992) 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

$$\theta \geqslant (\mathcal{Q}(\bar{x}) - L)\left(\sum_{\bar{x}_i=1} x_i - \sum_{\bar{x}_i=0} x_i\right) - (\mathcal{Q}(\bar{x}) - L)(\sum_i \bar{x}_i - 1) + L \tag{4.13}$$

where L is a global lower bound of $\mathcal{Q}$ (See Carøe and Tind 1998 for a generalization of these results).

We will refer to (4.13) 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 4.1.

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, *i.e.* they are tangent to the recourse function at that point.

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 4.1, 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.

BFCS

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

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

Model (4.1)

where the constraints in the second set are the optimality cuts generated so far. It operates at described in Algorithm 4.3

Algorithm 4.3 BFCS

```
1: Initialization: Find a lower bound $\underline{\theta}$, an initial assignment $x^0$ and let $K \leftarrow 1$.
2: repeat
3:     Find $(\alpha_1^K, \beta_1^K)$ and $(\alpha_2^K, \beta_2^K)$ that define the L-L-optimality cut and the $\partial$-optimality
       cut associated with $x^{K-1}$, respectively.
4:     Using a linear integer programming solver, find $(x^K, \theta^K)$ as a solution of $IP_K$.
5:     Evaluate $r^K \leftarrow \mathcal{Q}(x^K)$.
6: until (not $r^K > \min(\underline{\theta}, \theta^K)$)
7: if $r^K = \underline{\theta}$ then
8:     $\theta^K \leftarrow r^K$
9: end if
10: $x^K$ is an optimal solution with recourse cost $\theta^K$.
```

Remark 4.3.1. Observe that IP_K will always be feasible, so that Line 4 of Algorithm 4.3 is well defined.

Remark 4.3.2. If an assignment is found in Line 4 of this algorithm 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 4.3.3. 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.

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:

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

Model (4.2)

The proposed algorithm is now as described in Algorithm 4.4. Note that here we generate ∂-optimality cuts at non-integer solutions. In this respect our algorithm differs from the integer L-shaped algorithm presented in Laporte and Louveaux (1992), because in that paper, cuts are only generated in nodes where integer solutions are obtained. Algorithm 4.4 outlines this version of the exact algorithm.

CFBS

This scheme is an adaptation of the branch-and-cut method developed for 0-1 mixed convex programming in (Stubbs and Mehrotra 1999) to our specific problem.

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

Algorithm 4.4 BCS

```
$\mathcal{L} \leftarrow \{(\emptyset,\emptyset)\}$ and $K \leftarrow 1$.
Compute a lower bound $\underline{\theta}$.
Find a feasible assignment $x^0$, compute $(\alpha^1, \beta^1)$ (associated L-L cut).
Let $\bar{\theta} \leftarrow \mathcal{Q}(x^0)$ and $x^* \leftarrow x^0$.
repeat
  Take $S \in \mathcal{L}$ following a last-in-first-out (LIFO) policy.
  Solve $P_{K,S} \longrightarrow (x^K, \theta^K)$.
  if not ($\theta^K > \bar{\theta}$ or $P_{K,S}$ is infeasible) then
    Find $r^K \leftarrow \mathcal{Q}(x^K)$.
    if $x^K \in \mathbb{Z}^n$ then
      if $r^K \leqslant \theta^K$ then
        {binary solution and $r^K \leqslant \theta^K$}
        if $r^K < \bar{\theta}$ then
          $x^* \leftarrow x^K$ and $\bar{\theta} \leftarrow \theta^K$.
        end if
      else
        {binary solution and $r^K > \theta^K$}
        Find[a] $(\alpha^{K+1}, \beta^{K+1})$,
        $K \leftarrow K + 1$,
        Add $S$ to $\mathcal{L}$ .
      end if
    else
      {fractional solution and $r^K \leqslant \theta^K$}
      if $r^K \leqslant \theta^K$ then
        Take $x^K_{ij} \notin \mathbb{Z}$.
        Add $(S^0 \cup \{(i,j)\}, S^1)$, $(S^0, S^1 \cup \{(i,j)\})$ to $\mathcal{L}$ .
      else
        {fractional solution and $r^K > \theta^K$}
        Find[b] $(\alpha^{K+1}, \beta^{K+1})$,
        $K \leftarrow K + 1$,
        Take $x^K_{ij} \notin \mathbb{Z}$,
        Add $(S^0 \cup \{(i,j)\}, S^1)$,$(S^0, S^1 \cup \{(i,j)\})$ to $\mathcal{L}$ .
      end if
    end if
  end if{not ($\theta^K > \bar{\theta}$ or $P_{K,S}$ is infeasible)}
until $\mathcal{L}$ is empty or ($r^K = \underline{\theta}$ and $x^K \in \mathbb{Z}^n$)
if $r^K = \underline{\theta}$ and $x^K \in \mathbb{Z}^n$ then
  Let $x^* \leftarrow x^K$ and $\bar{\theta} \leftarrow r^K$.
  An optimal solution is given by $x^*$ and its recourse cost is $\bar{\theta}$.
end if
```

[a] *L–L* cut
[b] ∂ cut

Algorithm 4.5 BFCS

```
$\mathcal{L} \leftarrow \{(\emptyset, \emptyset)\}$ and $K \leftarrow 1$.
Compute a lower bound $\underline{\theta}$.
Generate a feasible assignment $x^0$ and the corresponding $\partial$-optimality cut, $(\alpha^1, \beta^1)$.
$\bar{\theta} \leftarrow \mathcal{Q}(x^0)$ and $x^* \leftarrow x^0$.
repeat
  Take $S$ from $\mathcal{L}$ following a last-in-first-out (LIFO) policy.
  $Fathom \leftarrow$ false
  repeat
    Solve $P_{K,S}$ to obtain $(x^K, \theta^K)$.
    $NewCut \leftarrow$ false
    if $\theta^K > \bar{\theta}$ or $P_{K,S}$ is infeasible then
      $Fathom \leftarrow$ true.
      Exit.
    end if
    Find $r^K \leftarrow \mathcal{Q}(x^K)$.
    if $r^k > \theta^K$ then
      {Optimality cuts needed.}
      $K \leftarrow K + 1$, compute $(\alpha^K, \beta^K)$ ($\partial$ cut).
      $NewCut \leftarrow$ true
      if $x^k \in \mathbb{Z}^n$ then
        $K \leftarrow K + 1$, compute $(\alpha^K, \beta^K)$ ($L$–$L$ cut).
        $NewCut \leftarrow$ true
      end if
    end if
  until not $NewCut$
  if not $Fathom$ then
    if $x^k \in \mathbb{Z}^n$ then
      Exit.
    else
      {Branching.}
      Take $x_{ij}^K \notin \mathbb{Z}$.
      Add $(S^0 \cup \{(i,j)\}, S^1)$, $(S^0, S^1 \cup \{(i,j)\})$ to $\mathcal{L}$ .
    end if
  end if
until $\mathcal{L}$ is empty
if $x^k \in \mathbb{Z}^n$ then
  The optimal solution is given by $x^*$ and its value is $\bar{\theta}$.
end if
```

4.4 Lower Bound for the Stochastic Generalized Assignment Problem

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.

In CO, lower bounding is frequently achieved by means of linear or Lagrangian relaxations. Here, 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, and, for $i \in I$ defining L_{ij_0} as the value of the convex continuous recourse problem

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

Model (4.3)

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

$$\min_{i\in I} L_{ij_0} \leqslant z^*. \tag{4.14}$$

Since (4.14) 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 here, LB has been computed using the L-shaped algorithm proposed in Van Slyke and Wets (1969).

Obviously, the computation of this lower bound requires many evaluations of the function $\mathcal{Q}$. As we will illustrate in Section 4.6, these calculations can be done efficiently, due to the reformulation of the second-stage problem as proposed in Section 4.1, which allows to evaluate $\mathcal{Q}$ as the expected value function of an LP instead of an IP problem.

4.5 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, in the three versions of the exact algorithm proposed in Section 4.3 some strategies directed at reducing the number of evaluations needed to solve a problem have been considered.

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 to 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 n_s optimality cuts. In this phase, ∂–optimality cuts have been used.

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 (Wolsey 1998). The order of variable lifting is decisive for the results. In this work lexicographical order has been chosen.

4.6 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.

Problem Statistics

All three versions of the algorithm have been tested on a set of 46 randomly generated instances.

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.

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 4.1, 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 4.1 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 FSP).

Table 4.1: Average CPU times for evaluating $\mathcal{Q}$ at binary and fractional solutions.

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[†]	75.08
4×7	1.50	0.09	11.57	0.29
4×8	3.21	0.19	24.64	0.77

Table 4.1 shows that by using the reformulation of $\mathcal{Q}$ as proposed in Section 4.1, 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 of SSP. In the latter case the speedup factor exceeds 200 for the 3×15 instance.

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

Computational Results

First, the behavior of the bounding procedures is studied. Results are shown in Table 4.2. The first five columns show index numbers, dimensions, total capacities, and probabilities of demand of the instances, respectively. We consider the upper bound given by the heuristic B of Section 4.2 and the lower bound presented in Section 4.4. 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.

Table 4.2: Relative Deviations of Bounds and CPU Times (in Seconds).

#	n_s	n_t	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

[†]Time limit reached.

Bounds' Deviations and CPU Times

#	n_s	n_t	total cap.	p	upper dev.	lower dev.	gap	bounding time
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
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

The relative deviations of the bounds are also shown in Figure 4.1.

The results in Table 4.2 indicate that the CPU times needed to complete the bounding phase are

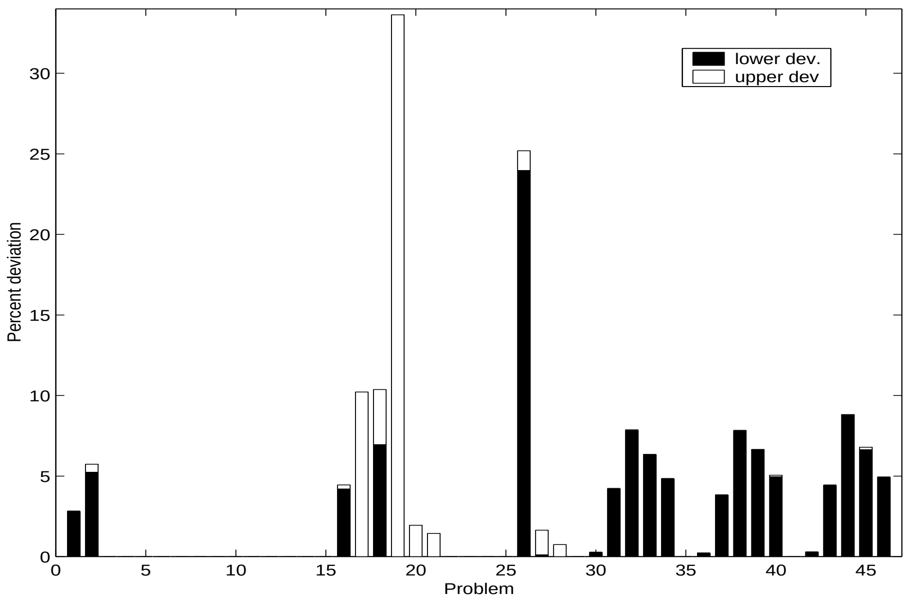

Figure 4.1: Percent deviations (summed) of the bounds.

reasonable.

Due to its dimensions, the bounding time for Instance 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 (4.3) to be solved is higher (45 subproblems, compared to e.g. 32 for 4×8 instances). Secondly, as shown in Table 4.1, 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 instances with 10 customers).

As can be seen in Table 4.2 and in Figure 4.1, for many of the test instances 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 instances, Table 4.3 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.

Table 4.3: CPU Times in Seconds for Each Version of the Algorithm. (* Denotes an Unsolved Instance; — Denotes an Instance Solved in the Bounding Phase.)

#	n_s	n_t	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	—	—

CPU Times For the Exact Algorithm

#	n_s	n_t	cap.	p	BCS	CFBS	BFCS
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
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	*

For most of the test instances, 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 BCS. For CFBS, 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.

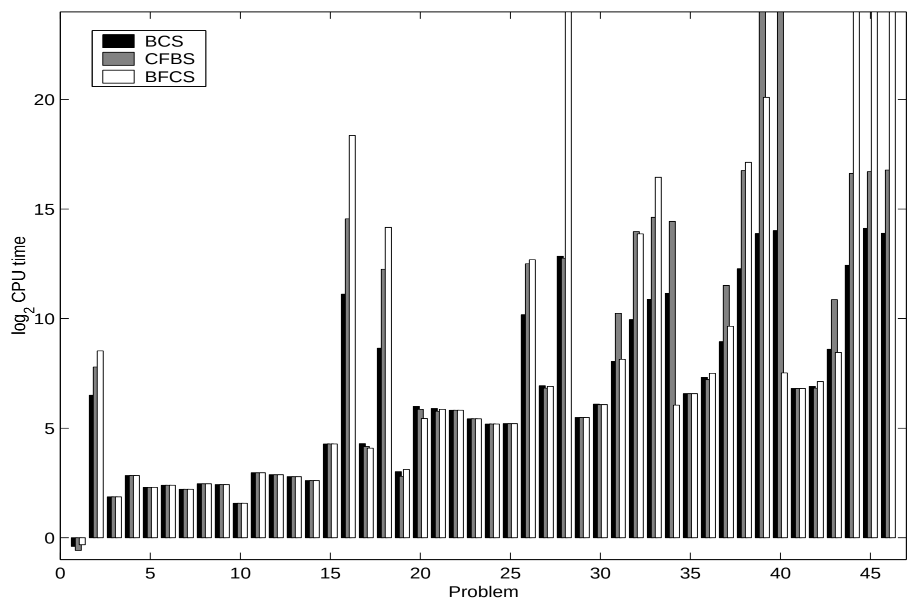

Figure 4.2: Logarithm of CPU times of each version of the algorithm, for each problem instance. (A value of 24 denotes an unsolved instance.)

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 4.3 and Figure 4.2, BCS is an order of magnitude faster on several hard instances (Instance 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 Instances 16 and 19 (both 3×8) and for Instances 42 and 46 (both 4×10). In the case of Instances 16 and 19, the only difference between them is the tightness of the capacity constraint, with total capacities of 6 and 15, respectively. Similarly, Instances 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 n_t jobs, there are $P = {n_s}^{n_t}$ 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 4.4.

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 instances which were not already solved in the bounding phase have been considered. The third column contains the number of such instances. 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 instances 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.

Table 4.4: Evaluations of the Recourse Function.

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

4.7 Conclusions

In this chapter, we consider a SGAP 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 modelled as a stochastic problem with recourse. The recourse function reflects the expected cost associated with 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.

We also present three heuristics, all based on different deterministic models used to approximate our specific GAP. The solution obtained with one of them is used as the starting point for the three versions of the exact algorithm, and the upper bound it provides reduces substantially the size of the exploration trees.

A lower bound for the GAP under consideration is developed as well. A stochastic linear sub-problem 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.

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.