A two-phase path-relinking algorithm for the Generalized Assignment Problem

Laurent Alfandari[*], Agnès Plateau[†], Pierre Tolla[‡]

June 2002

Abstract We propose a two-phase Path-Relinking (PR) heuristic for the NP-hard Generalized Assignment Problem. The first phase of the algorithm combines LP-solving, cuts and local search for obtaining a population of solutions with guided diversity. The second phase applies the PR template to the population built at phase one. Both feasible and infeasible solutions are inserted in the reference set during combinations of PR. The trade-off between feasibility and infeasibility is controlled through a penalty coefficient for infeasibility which is dynamically updated throughout the search for maintaining a balance between feasible and infeasible solutions in the reference set. Numerical experiments on classical testbed instances of the OR-library show the practical efficiency of the method.

Key-words : combinatorial optimization, assignment, heuristics.

1 Problem statement

The Generalized Assignment Problem (GAP) is a very popular NP-hard optimization problem. Given a set of tasks $I = \{1, \ldots, n\}$ and a set of agents $J = \{1, \ldots, m\}$, the GAP consists in assigning each task of I to exactly one agent of J such that the capacity of each agent is not exceeded, while minimizing the total cost of selected assignments. Numerous applications concern job scheduling, production planning and storage space allocation, computer and communication networks, vehicle routing, facility location, where clients are assigned to facilities with restricted delivery capacities, etc. When modeling real-case assignment problems requires constraints that are specific to the firm studied, the GAP often appears as a subproblem of the application. The data of the GAP are, for every task $i \in I$ and every agent $j \in J$, the cost c_{ij} of assigning task i to agent j, the amount a_{ij} of resource consumed by task i when assigned to agent j, and the capacity (i.e. resource availability) b_j of agent j. A feasible solution of the GAP is a mapping $x : I \to J$ such that $\sum_{i:x(i)=j} a_{ij} \leq b_j$ for all $j = 1, \ldots, m$. An optimal solution minimizes objective function $f(x) = \sum_{i=1}^{n} c_{i,x(i)}$. The GAP can also formulate as a 0-1 Integer Linear Program (ILP) with binary variables

[*]ESSEC, B.P.105 F-95021 Cergy-Pontoise Cedex, France, e_mail: alfandari@essec.fr

[†]CEDRIC, CNAM, Paris, France, e_mail: aplateau@cnam.fr

[‡]LAMSADE, Université Paris IX-Dauphine, Place du Mal De Lattre de Tassigny, 75775 Paris Cedex 16, France, e_mail: tolla@lamsade.dauphine.fr

$\tilde{x}_{ij} \in \{0,1\}$ defined by $\tilde{x}_{ij} = 1$ if $x(i) = j$, $\tilde{x}_{ij} = 0$ otherwise:

$$\min f(\tilde{x}) = \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}\tilde{x}_{ij}$$

subject to

$$\sum_{j=1}^{m} \tilde{x}_{ij} = 1 \qquad i = 1,\ldots,n \tag{1}$$

$$\sum_{i=1}^{n} a_{ij}\tilde{x}_{ij} \leq b_j \qquad j = 1,\ldots,m \tag{2}$$

$$\tilde{x}_{ij} \in \{0,1\} \qquad i = 1,\ldots,n, \quad j = 1,\ldots,m \tag{3}$$

Constraints (1) are called semi-assignment constraints, whereas constraints (2) are complicating capacity (or knapsack) constraints. Since the GAP is NP-hard, heuristic solutions are often required for large-scale problems. The paper is organized as follows. Section 2 presents a state of the art on methods for solving or approximating the GAP. Section 3 describes the Path-Relinking algorithm. Numerical experiments on classical testbed instances are reported in section 4, whereas section 5 concludes the paper.

2 State of the art

The GAP has been extensively studied in the literature. The main contributions are reported in the sequel. Note that two versions of the GAP are generally studied : cost-minimisation and profit-maximisation. For reasons of homogenicity we systematically use expression "lower bound" for the value of a relaxed problem in the following review, even though the paper studies the max-GAP.

2.1 Branching methods for exact solving

We give a brief review of papers on this topic; for a comprehensive survey on exact methods for the GAP, we refer to Cattrysse and Van Wassenhove [5]. First, observe that the 0-1 assignment subproblem (i.e. GAP without knapsack constraints (2)) is easily solvable in polynomial time; this lower bound was exploited by Ross and Soland [23] within a depth-first branch & bound algorithm. Two papers propose refined branch & bound methods computing lower bounds by Lagrangian Relaxation (LR) of whether assignment constraints (1) or capacity constraints (2) : these two LR-methods were developped by Fisher, Jaikumar and Van Wassenhove [9], and Guignard and Rosenwein [13], respectively. Savelsbergh [24] also proposed an efficient branch & price algorithm solving a set partitioning reformulation of the problem. This reformulation, which was originally introduced in [6], consists in associating with every agent $j \in J$ all subsets of tasks $I' \subset I$ which do not violate the capacity of j. Due to the exponential number of subsets, i.e. columns, the problem was solved by column generation.
Typically, the above methods were able to find optimal solutions of problems with an order of two hundred tasks. For very large-scale problems, heuristic methods are needed for reaching high-quality solutions in short computational time.

2.2 Heuristics and metaheuristics for approximate solving

We first distinguish heuristic methods using relaxations. The heuristic of Trick ([25]) uses LP-relaxation for fixing variables, and then makes reassignments within a swap neighborhood. Amini and Racer [3] developped a Variable Depth-Search (VDS) heuristic proceeding in two phases : an initial solution and LP-lower bound computation phase and a reassignment phase using both shifts and swaps. The main interest of their paper is that it provides a carefully-designed statistical comparison of methods. The numerical results of [3] were improved by another VDS, the one of Yagiura, Yamaguchi and Ibaraki ([27]). LR-based methods were also proposed for finding good approximate solutions on large-scale GAP instances ([4, 18]). These methods often combine LR with local search.
Let us cite as well an older paper of Martello and Toth [19], who developed a two-phase heuristic where, firstly, assignements are computed in a greedy way so as to minimize an appropriate regret function and, secondly, a descent method is applied to the greedy solution for finding improving reassignments. This heuristic is embedded in a general reduction template; it is often used as a preprocessing phase in other papers. As for metaheuristics, let us cite genetic algorithms (by Wilson [29], Chu and Beasley [7]), tabu search (by Osman [21], Laguna et al. [16], Yagiura et al. [26], Díaz and Fernández [8], Higgins [14]), simulated annealing/tabu search (by Osman [21]). Most of the heuristics reported here share the following principles :

(i) use of both shifts and swaps as valid moves for local search [7, 8, 14, 27]; in effect, restriction to swap moves does not allow to change the number of tasks assigned to each machine. Ejection chains have also been considered in [16, 26].

(ii) use of a measure of infeasibility generally called "fitness" function, which represents the global over-capacity of agents ([29, 7, 16, 26, 27, 14, 8]).

(iii) oscillation between feasibility and infeasibility for a wider diversification of the search ([7, 16, 26, 27, 8, 14]). The underlying idea is that most of the solutions that lie in the neighborhood of a feasible solution are infeasible, so we make it easier to reach this solution by allowing infeasibility. The balance between feasibility and infeasibty is controlled through an appropriate penalty applied to the fitness function. In most papers, infeasible solutions are admitted but the search is guided towards feasibility ([7, 26, 16, 14]) whereas in [8], feasible and infeasible solutions are compared through a unique evaluation function so that no preference is put on feasibility when exploring a neighborhood. Also, the penalty term is often dynamically adapted all along the search ([8, 26]).

Neither scatter search nor path relinking had been applied yet to the GAP at the moment of our first conference on the topic ([1] for max-GAP, and [2] for min-GAP). Recently, Yagiura et al. ([28]) have developed another PR heuristic using Ejection Chains, which presents the best results up to now for the GAP, we mention their work in section 4. Basically, the *path-relinking* metaheuristic, originally introduced by Glover ([11]), is a deterministic search that combines elite solutions of a population and generates paths between them which are likely to contain improving solutions.

3 The Path Relinking algorithm

3.1 Principles of PR

Path-relinking is a generalization of *scatter search*, which performs linear combinations of *elite* solutions (see [11]). Several papers of the literature concern applications of path-relinking to specific combinatorial optimization problems (see for example [12, 17, 22]). In the path-relinking metaheuristic, paths are generated between pairs $(\underline{x}, \bar{x})$ of 0-1 elite solutions picked from an initial population $\mathcal{P}_0$. The best solutions so-found on the paths are then inserted in $\mathcal{P}_0$ in a dynamic process. The initial endpoint $\underline{x}$ of a path is called the *initiating* solution, whereas the final endpoint $\bar{x}$ is the *guiding* solution. In our path-relinking, the common elements or bits (i.e. assignments) between $\underline{x}$ and $\bar{x}$ are never modified throughout the path; the rationale is that keeping elements shared by different good solutions is likely to provide other good - hopefully, improving - solutions. The remaining elements are chosen in a greedy way, which certainly does not allow deep intensification but permits to combine more pairs of solutions. Precisely, the path linking $\underline{x}$ and $\bar{x}$ is a sequence of (feasible or infeasible) solutions

$$\underline{x} = x^0, x^1, x^2, \ldots, x^p, x^{p+1} = \bar{x} \tag{4}$$

where $x^{k+1} \in N^{pr}(x^k)$, with N^{pr} a neighborhood chosen for path-relinking so as to draw nearer and nearer to the structure of the guiding solution $\bar{x}$ according to some appropriate distance d (like the Hamming distance for instance).

Remark that we can easily bound the number p of intermediate solutions in function of $D = d(\underline{x}, \bar{x})$. Denote respectively by $q_1 \in \mathbf{N}$ and $q_2 \in \mathbf{N}$ the minimum and maximum distance associated with a move of N^{pr}. We have $d(x^k, x^{k+1}) \in [q_1, q_2]$ for all $k = 1, \ldots, p$ and $d(x^p, x^{p+1}) = q' \in [1, q_2]$, so

$$D = d(\underline{x}, \bar{x}) = \sum_{k=0}^{p} d(x^k, x^{k+1}) \Longrightarrow p.q_1 + 1 \leq D \leq (p+1)q_2$$

We deduce that

$$\lceil \frac{D}{q_2} - 1 \rceil \leq p \leq \lfloor \frac{D-1}{q_1} \rfloor \tag{5}$$

A natural distance for the GAP, more convenient than the Hamming distance, is defined as

$$d(x, x') = |\{i \in I : x(i) \neq x'(i)\}| \tag{6}$$

It represents the number of jobs that are not assigned to the same agent in x and x'. Now, in order to adapt path-relinking to the GAP, we need to answer several questions:
a. what neighborhood N^{pr} to choose for the path?
b. which criterion for selecting $x^{k+1} \in N^{pr}(x^k)$?
c. which solution(s) of the path to select for insertion?
d. what population to start with?
e. which pairs of the population to select for combination?
f. when to stop the general path-relinking algorithm?
We can distinguish two kinds of issues : issues about how to construct a path (a-c) and how to manage the population (d-f).

3.2 Construction of a path

We call $pr(\underline{x}, \bar{x})$ the procedure that constructs the path linking $\underline{x}$ to $\bar{x}$. For generating the intermediate solutions of pr, we only consider moves involving tasks that are not assigned to the same machine in $\underline{x}$ and $\bar{x}$. Set $I_* = \{i \in I : \underline{x}(i) \neq \bar{x}(i)\}$. The (common) assignments in $I \setminus I_*$ are fixed all along the path, whereas assignments of I_* will change iteratively so as to tend towards the guiding solution $\bar{x}$.

Choice of neighborhood N^{pr}.

We combine two neighborhoods :

(1) a *shift* neighborhood: given a solution x^k

$$N^{pr}_{shift}(x^k) = \{x : \exists i_1 \in I_* \text{ such that } x^k(i_1) \neq \bar{x}(i_1), \\ \forall i \neq i_1,\ x(i) = x^k(i) \quad \wedge\ x(i_1) = \bar{x}(i_1)\}$$

(2) a *swap* neighborhood: given x^k

$$\begin{aligned} N^{pr}_{swap}(x^k) \quad = \quad & \{x : \exists (i_1, i_2, j_1, j_2) \in I_*^2 \times J^2 \text{ such that} \\ & i_1 \neq i_2, j_1 \neq j_2, x^k(i_1) = j_1, x^k(i_2) = j_2, \bar{x}(i_1) = j_2, \bar{x}(i_2) = j_1, \\ & \forall i \neq i_1, i_2, x(i) = x^k(i) \quad \wedge\ x(i_1) = j_2, x(i_2) = j_1\} \end{aligned}$$

These neighborhhods adapt the definition of the classical N_{shift} and N_{swap} neighborhoods for the GAP, so that the new assignment for N^{pr}_{shift} (resp. at least one of the two new assignments for N^{pr}_{swap}) is an assignment of the guiding solution $\bar{x}$. This condition must hold for decreasing at each step the distance $d(x^k, \bar{x})$ (with d as in (6)). The neighborhood N^{pr} is the union

$$N^{pr} = N^{pr}_{shift} \cup N^{pr}_{swap}$$

Finally, note that we do not constrain neighbor $x^{k+1} \in N^{pr}(x^k)$ to be feasible at any time, as we see in the sequel.

Choice of neighbor $x^{k+1} \in N^{pr}(x^k)$.

As in several papers of the literature ([7, 8, 14, 16, 27, 26]), we do not restrict the search to feasible solutions. In order to compare feasible and infeasible solutions with a single criterion, we use a modified evaluation function :

$$v(x, \alpha) = f(x) + \alpha.fitness(x)$$

where $fitness(x)$, defined by

$$fitness(x) = \frac{1}{m} \sum_{j=1}^{m} \max \left(0,\ \frac{1}{b_j} \sum_{i=1}^{n} a_{ij} x_{ij} - 1\right) \tag{7}$$

is a measure of infeasibility and represents the average (scaled) over-capacity of agents, whereas α is a real "penalty" cofficient applied to the fitness function.

Note that, as in [8], no priority is given to feasibility. A feasible solution x satisfies $v(x,\alpha) = f(x)$ for any $\alpha \in \mathbf{R}$ but an infeasible solution x' is prefered to x at any time of the search as soon as $f(x') + \alpha.fitness(x') < f(x)$. This modified function $v(x,\alpha)$ enables to explore much wider areas of the space. Indeed, restricting the search space to feasible solutions often conduces to a less-diversified and smaller set of solutions compared to a search allowing infeasibility. Of course, the performance of the algorithm largely depends on the penalty coefficient α. A high α forces feasibility but makes the search terminate earlier, whereas a low α forces infeasibility and may lead to prefer an infeasible solution to an optimal feasible solution. For this reason, α must be set so that some balance is kept between feasibility and infeasibility. Since an "optimal" α is hard to guess at the beginning of the search, we start with an initial value α_0 of α and we adjust the coefficient throughout the search according to the proportion of feasible solutions in the population. Initial value α_0 is set as follows. Recall that if x is feasible and x' is infeasible, we should consider x and x' equivalent if $f(x) - f(x') = \alpha.fitness(x')$. If x' is such that all machines but one are in under-capacity and the remaining machine is in over-capacity by one unit, in average $fitness(x') = 1/[\sum_{j=1}^{m} b_j]$. So, $\alpha/[\sum_{j=1}^{m} b_j]$ represents the variation of cost ΔC that we are ready to accept for having a single machine in over-capacity by one unit. Starting with $\alpha_0 = \Delta C_0/(\sum_{j=1}^{m} b_j)$ where ΔC_0 is a fixed parameter, α (or ΔC) is dynamically adjusted along the search as explained in next subsection.

Choice of the best solution of a path.

For each path, we select a single intermediate solution of the path to be candidate for insertion in the population. For endpoints $(\underline{x}, \bar{x})$, we note $x^* = pr(\underline{x}, \bar{x})$ this candidate (feasible or infeasible) solution. x^* is computed as follows. First, we select the intermediate solution x^k of the path (different from $\underline{x}$ and $\bar{x}$) with higher value $v(x^k, \alpha)$. Then, we improve x^k by an *descent* method. For a solution x, $descent(x)$ selects iteratively the solution x' with higher value $v(x', \alpha)$ in the neighborhood $N = N_{shift} \cup N_{swap}$ of the current solution, until a local optimum is reached.

Observe that, with notations of (5), the distance associated with neighborhhod N^{pr}_{shift} (resp. N^{pr}_{swap}) is $q_1 = 1$ (resp. $q_2 = 2$), so using (5) the number of solutions generated on a path ranges in interval $[\lceil \frac{D}{2} - 1 \rceil, D-1]$ i.e. is in $O(D)$. The complexity of selecting $x^{k+1} \in N^{pr}(x^k)$ is in $O(D^2)$, so the complexity of computing the whole path from $\underline{x}$ to $\bar{x}$ is in $O(D^3)$.

3.3 Management of the population

Initial population

We start with a population $\mathcal{P}_0 = \{x^1, x^2, \ldots, x^{p_1}\}$ of p_1 feasible or infeasible solutions built as follows. Let (P^1) denote the LP-relaxation of (ILP), and let $\tilde{y}^1$ denote the fractional optimal solution for (P^1). $\tilde{y}^1$ is rounded in a 0-1 solution x by setting $x(i) = arg\max_{j \in J} \tilde{y}^1_{ij}$. The final solution is obtained by applying the *descent* improvement procedure to x, i.e. $x^1 = descent(x)$. Solution x^k

$(k = 2, \ldots, p_1)$ of $\mathcal{P}_0$ is obtained by solving

$$(P^k) = (P^{k-1}/ \quad \sum_{i=1}^{n}\sum_{j=1}^{m} \tilde{x}_{ij}^{k-1} \leq \beta^{k-1} n)$$

where $\beta \in]0,1[$ is a real parameter, rounding the fractional optimal solution $\tilde{y}^k$ of (P^k) and applying *descent*. So, at each LP-solving, we reduce the proportion of assignements of the former 0-1 solution by a fixed reduction coefficient (β) in order to ensure diversification of the population.

Choice of initiating and guiding solutions.

We call "round" a phase of combinations. The first round consists in running $pr(x, x')$ for every pair $(x, x') \in \mathcal{P}_0 \times \mathcal{P}_0$, $x \neq x'$, such that $v(x, \alpha) \leq v(x', \alpha)$. Each best solution of a path $x^* = pr(x, x')$ is inserted in the population if it is a new solution. The new solutions inserted during round 1 form set $\mathcal{Q}_1$. Round 2 consists in running $pr(x, x')$ for every pair $(x, x') \in \mathcal{P}_0 \times \mathcal{Q}_1$ and every pair $(x, x') \in \mathcal{Q}_1 \times \mathcal{Q}_1$, $x \neq x'$, such that $v(x, \alpha) \leq v(x', \alpha)$. The resulting set of new inserted solutions is $\mathcal{Q}_2$. Generally speaking, our Path-Relinking (PR) algorithm is a sequence of rounds $r = 1, 2, 3, \ldots$ such that the set $\mathcal{Q}_r$ of new solutions inserted during round r is defined by:

$$\mathcal{Q}_r = \{x^* = pr(x, x') : \quad (x, x') \in \left(\left(\bigcup_{t=0}^{r-2} \mathcal{Q}_t \right) \times \mathcal{Q}_{r-1} \right) \cup (\mathcal{Q}_{r-1} \times \mathcal{Q}_{r-1}),$$
$$v(x, \alpha) \leq v(x', \alpha), \; x^* \notin \bigcup_{t=0}^{r-1} \mathcal{Q}_t \}$$

with $\mathcal{Q}_0 = \mathcal{P}_0$ and $\mathcal{Q}_{-1} = \emptyset$. So at each round, among all pairs (x, x') of (old, new) and (new,new) solutions, we combine every pair such that the value v of the guiding solution x' is superior or equal to the value of the initiating solution x.

Stopping criterion

We perform path-relinking rounds until we attain a number of solutions equal to $p_2 > p_1$. As soon as the population contains p_2 solutions, the population is called Reference Set (RS) and a new solution x^* generated on a path is inserted in RS if and only if $v(x^*, \alpha) < v(x_{p_2}, \alpha)$, where $x_{p2} = arg \min_{x \in RS} v(x, \alpha)$ is the worst solution of RS (then x_{p_2} is removed from RS for allways keeping exactly p_2 solutions in RS until the end of the search). Rounds of combinations are performed again as described above, combining each pair of (old,new) and (new,new) solutions in RS. The whole process is stopped whenever no new solution enters RS at the end of a round, i.e. when all solutions x^* generated on paths verify $v(x^*, \alpha) \geq v(x_{p2}, \alpha)$. This stopping criterion elegantly formulates a natural principle of evolution according to which the population dies when it cannot give birth any more to children that outperform their parents. No specific parameter - like a maximum number of iterations or computational time - is needed then for ending the search.

Dynamic adjustment of infeasibility penalty coefficient

The general algorithm is composed of the following main phases:
- phase 1 : "starting" phase constructing initial population $\mathcal{P}_0$,
- phase 2a : "growing" PR phase embedding all rounds $1, 2, \ldots, r_{RS}$, where r_{RS} denotes the round when the size of the population reaches value p_2,
- phase 2b : "stabilized" PR phase embedding all rounds $r_{RS}+1, \ldots, r_{RS}+s$, where $r_{RS}+s$ is the last round, i.e. the round during which no generated solution improves the worst solution in RS.

As we mentioned before, we initialize ΔC to ΔC_0 and $\alpha = \Delta C/(\sum_{j=1}^{m} b_j)$, and we adjust α so that the current population contains a given threshold of feasible solutions and infeasible solutions. At the end of phase 1 and at the end of each round r of phase 2 (2a and 2b), if the population $\mathcal{P}_r = \cup_{t=0}^{r} \mathcal{Q}_r$ contains a proportion of feasible solutions p_{feas} that is outside interval $[\rho, 1-\rho]$, where $\rho \in]0, 1[$ is a fixed parameter, then we do $\Delta C \leftarrow \Delta C + var\Delta C$ (resp. $\Delta C \leftarrow \Delta C - var\Delta C$) if $p_{feas} < \rho$ (resp. $p_{feas} > 1-\rho$) and we apply *descent* to each solution of $\mathcal{P}_r$ with new $\alpha = \Delta C/(\sum_{j=1}^{m} b_j)$, until $p_{feas} \in [\rho, 1-\rho]$. Parameter $var\Delta C$ is fixed all along the search.

4 Computational results

We have tested our algorithm for the GAP on well-known instances of the OR-library (available at `http://mscmga.ms.ic.ac.uk/jeb/orlib/gapinfo.html` and `http://www-or.amp.i.kyoto-u.ac.jp/"yagiura/gap/`).

These problems were used to test the Variable Depth Search of Yagiura and al. [27], denoted by VDSY, the Tabu-Search heuristic of Yagiura et al. [26], denoted by TSEC, the genetic algorithm (GA) of [7], the Tabu-Search of [16] for the Multilevel GAP, denoted by TSEC2, and the Tabu Search of Díaz & Fernandez [8], denoted by TSDA. The test-library is composed of $4 \times 6 = 24$ instances of min-GAP, namely the B-C-D-E instances (A-type instances are generally considered as not discriminant). Each of the 4 series B-C-D-E contains 6 test-problems : 3 problems with $n = 100$ and $m \in \{5, 10, 20\}$, and another 3 problems with $n = 200$ and $m \in \{5, 10, 20\}$. For series B and C, values a_{ij} are integers generated from a uniform distribution $U(5, 25)$, whereas costs c_{ij} are integers generated from $U(10, 50)$. Capacities b_j are set to $0.8/m \times \sum_{i \in I} a_{ij}$ for type C-instances. For series D, a_{ij} are integers generated from $U(1, 100)$, costs c_{ij} are set to $c_{ij} = 111 - a_{ij} + s$ where s is an integer generated from $U(-10, 10)$, and $b_j = 0.8/m \times \sum_{i \in I} a_{ij}$. Finally, for type E-problems, $a_{ij} = 1 - 10 \ln x$ where $x = Uniform]0, 1]$, $c_{ij} = 1000/a_{ij} - 10y$ with $y = Uniform[0, 1]$, and $b_j = \max(0.8/m \times \sum_{i \in I} a_{ij}; \max_j a_{ij})$. Test-series B and C are known to be easier to solve than series D and E.

Numerical experiments were carried out on a SunSparc station with an UltraSparc processor 400 Mhz (Sun OS 5.7). The LP-solver used for phase 1 of the method is CPLEX 7.0, and the algorithm was programmed in C language. After some tuning we have set the feasibility/infeasibility threshold ρ to 0.25 and the reduction coefficient for LP-cuts β to 0.95 for all problems. The other parameters

were set as follows for each type of problems:
B-type : $(p_1, p_2) = (8, 20)$, $\Delta C = 10$ and $var\Delta C = 2$;
C-type : $(p_1, p_2) = (8, 20)$, $\Delta C = 25$ and $var\Delta C = 5$;
D-type : $(p_1, p_2) = (8, 20)$, $\Delta C = 30$ and $var\Delta C = 5$;
E-type : $(p_1, p_2) = (12, 25)$, $\Delta C = 300$ and $var\Delta C = 50$;

A summary of the results obtained is presented in the following two tables Table 1 presents for each problem and each heuristic method VDSY, TSEC1, GA, TSEC2, TSDA and PR, the gap from optimum in %, i.e. $100 \times (c^* - c^h)/c^*$, where c^h denotes the heuristic value and c^* is the best known value (known to be optimal when followed by "*"). For all methods but GA and TSDA, the algorithm is executed once for each problem. For GA and TSDA, the algorithm is ran a fixed number of times for every problem (providing different solutions due to randomly-generated parameters), so for GA and TSDA the gap for each problem is indeed an average gap computed over all the executions of the problem. The average gaps per series are indicated below.

Type	m	n	VDSY	TSEC1	GA	TSEC2	TSDA	PR	best	PR
B	5	100	n.a	n.a	0.347	n.a	0.000	0.000	1843	1843
B	10	100	n.a	n.a	0.071	n.a	0.000	0.000	1407	1407
B	20	100	n.a	n.a	0.069	n.a	0.097	0.000	1166	1166
B	5	200	n.a	n.a	0.301	n.a	0.005	0.028	3552	3553
B	10	200	n.a	n.a	0.308	n.a	0.046	0.035	2828	2829
B	20	200	n.a	n.a	0.060	n.a	0.113	0.043	2340	2340
					0.193		0.044	0.021		
C	5	100	0.000	0.000	0.378	0.000	0.000	0.000	1931	1931
C	10	100	0.000	0.071	0.285	0.000	0.043	0.000	1402	1402
C	20	100	0.080	0.080	0.515	0.000	0.284	0.080	1243	1244
C	5	200	0.000	0.029	0.229	0.000	0.034	0.058	3456	3458
C	10	200	0.107	0.214	0.481	0.007	0.105	0.143	2806	2810
C	20	200	0.418	0.209	0.619	0.025	0.139	0.209	2391*	2396
			0.101	0.101	0.418	0.005	0.101	0.082		
D	5	100	0.079	0.519	0.658	0.041	0.163	0.189	6353	6365
D	10	100	0.284	0.772	1.237	0.167	0.511	0.362	6349*	6372
D	20	100	1.275	1.404	1.652	0.387	0.905	1.146	6196*	6267
D	5	200	0.094	0.353	0.652	0.020	0.093	0.031	12743*	12747
D	10	200	0.354	0.667	1.536	0.076	0.277	0.169	12436*	12457
D	20	200	1.435	1.239	1.983	0.166	0.887	0.563	12264*	12333
			0.587	0.826	1.286	0.143	0.473	0.410		
E	5	100	0.000	0.047	n.a	0.003	0.040	0.000	12681	12681
E	10	100	0.069	0.631	n.a	0.000	0.087	0.173	11577	11597
E	20	100	0.747	1.411	n.a	0.055	0.703	1.245	8436	8541
E	5	200	0.048	1.095	n.a	0.000	0.027	0.004	24930	24931
E	10	200	0.167	1.116	n.a	0.004	0.116	0.039	23307	23316
E	20	200	0.429	1.591	n.a	0.022	0.446	0.331	22379	22453
			0.243	0.982		0.014	0.237	0.298		

Table 1: gap from optimal (in %) for B-C-D-E test-series

Table 2 presents the average CPU computational times (in seconds) for each series and each method : $\bar{t}_{best}$ is the CPU average computational time before obtaining the best feasible solution, whereas $\bar{t}_{total}$ is the total CPU running time of the algorithm. For GA and TSDA, the (best or total) time indicated for a problem is the average time over all the executions of the problem.

test series	m	n	VDSY	TSEC1	GA		TSEC2		TSDA		PR	
			$\bar{t}_{total}$	$\bar{t}_{total}$	$\bar{t}_{best}$	$\bar{t}_{total}$	$\bar{t}_{best}$	$\bar{t}_{total}$	$\bar{t}_{best}$	$\bar{t}_{total}$	$\bar{t}_{best}$	$\bar{t}_{total}$
B	5	100	n.a	n.a	126.9	288.2	n.a	n.a	17.5	95.8	10.0	26.3
B	10	100	n.a	n.a	301.0	276.0	n.a	n.a	10.6	97.2	7.3	48.1
B	20	100	n.a	n.a	191.5	617.3	n.a	n.a	47.5	160.0	11.4	32.8
B	5	200	n.a	n.a	439.5	790.0	n.a	n.a	70.5	339.1	121.9	194.6
B	10	200	n.a	n.a	608.4	1027.4	n.a	n.a	154.8	389.8	37.6	180.2
B	20	200	n.a	n.a	518.5	1323.5	n.a	n.a	222.7	465.4	132.7	196.0
C	5	100	150	150	139.1	301.4	0.6	150.0	3.0	83.6	6.6	44.8
C	10	100	150	150	170.6	394.2	3.0	150.0	28.7	103.3	31.5	51.2
C	20	100	150	150	279.9	669.3	21.6	150.0	66.9	130.7	47.5	114.7
C	5	200	300	300	531.2	810.1	3.7	300.0	212.7	403.0	146.1	231.8
C	10	200	300	300	628.6	1046.0	100.5	300.0	212.7	403.0	104.3	174.9
C	20	200	300	300	1095.9	1792.3	137.4	300.0	291.8	498.3	146.4	247.0
D	5	100	150	150	369.9	530.3	62.6	150.0	43.6	106.4	71.5	79.8
D	10	100	150	150	870.2	1094.7	107.2	150.0	83.2	133.4	77.3	92.3
D	20	100	150	150	1746.1	2126.1	111.0	150.0	119.5	167.4	108.8	131.6
D	5	200	300	300	1665.9	1942.8	95.5	300.0	226.8	363.2	318.1	339.3
D	10	200	300	300	2768.7	3189.6	129.2	300.0	205.0	397.4	105.2	258.0
D	20	200	300	300	4878.4	5565.1	120.7	300.0	348.2	515.6	308.7	528.4
E	5	100	150	20000	n.a	n.a	39.9	150	66.5	124.0	96.1	133.4
E	10	100	150	20000	n.a	n.a	31.6	150	88.3	148.2	71.9	82.2
E	20	100	150	20000	n.a	n.a	90.4	150	146.3	197.2	79.4	104.3
E	5	200	300	20000	n.a	n.a	20.0	300	204.7	351.4	184.0	268.2
E	10	200	300	20000	n.a	n.a	34.1	300	233.0	396.4	372.9	493.7
E	20	200	300	20000	n.a	n.a	209.3	300	425.2	585.3	435.8	688.9

Table 2: CPU times (in s.) for the test series B-C-D-E

Naturally, CPU-times compare differently since experiments were performed on different machines (Sun Ultra 2 Model 2200, 200 MHz for VDSY, Silicon Graphics Indigo R4000 100 MHz for GA, Sun Ultra 2 Model 2300, 300 MHz for TSEC2, Sun Sparc 10/30 4×100 MHz with one processor used for TSDA). We can see that, in terms of average gaps, our PR algorithm is only outperformed by the TS of [26] using ejection chains, which give significantly better results. Our results are comparable to TSDA : our average gaps are slightly better on all types of instances except E-type, but the best solutions found by TSDA are more often better than ours (see [8]), which also explains by the fact that the best solution of TSDA is taken over 30 runs of the algorithm whereas ours is ran only once. Also, the trade-off between quality of solutions and computation time should be examined with care as we benefit from a more powerful computer. In [1], we mentioned that Tabu Search should be very effective when combined with Path-Relinking. This was recently done by Yagiura, Ibaraki and Glover in [28], where the solutions found are often equal or very close to the best known solutions for

the C-D-E-type instances. It should be interesting to study in what measure the success of their method depends on whether proper Path-Relinking factors, like the choice of solutions for combination and the management of the population, or other factors specific to the GAP, like the neighborhood chosen (ejection chains, compared to shift/swap moves) or the quality of the dynamic adjustement of the penalty coefficient.

5 Conclusion

We have designed a path-relinking heuristic for the Generalized Assignment Problem that enriches classical principles for this problem (like combination of shifts and swaps, use of a bi-criteria relaxation of the problem aggregating cost and infeasibility through a penalty coefficient) by original diversification of the initial population through LP-cuts, and population-based updating of the penalty coefficient through a controlled balance between feasible and infeasible solutions in the Reference Set. The tests carried out on classical problem-sets of the OR-library show that the algorithm is very efficient in practice. Further developments of the method should focus on improving the diversification of the initial population, the choice of solutions to combine according to their distance, and the quality of the dynamic adjustement of the penalty coefficient in order to enlarge the exploration of the solutions' space.

References

[1] Alfandari L, Plateau A, Tolla P (2001). A two-phase Path-Relinking algorithm for the Generalized Assignement Problem. Proc. 4th Metaheuristics International Conference, Porto, July 2001,175-179.

[2] Alfandari L, Plateau A, Tolla P (2002). A Path-Relinking metaheuristic for the GAP. Presented at the 12th Intl Symp. on Combinatorial Optimization, Paris, April 8-10 2002.

[3] Amini MM, Racer M (1994). A rigorous comparison of alternative solution methods for the generalized assignment problem. *Management Sc.* **40**:868-890.

[4] Barcia P, Jornsten K (1990). Improved lagrangian decomposition : an application to the generalized assignment problem. *EJOR* **46**:84-92.

[5] Cattrysse DG, Van Wassenhove LN (1992). A survey of algorithms for the generalized assignment problem. *EJOR* **60**:260-272.

[6] Cattrysse DG, Salomon, Van Wassenhove LN (1994). A set partitioning heuristic for the generalized assignment problem. *EJOR* **72**:167-174.

[7] Chu PC, Beasley JE (1997). A genetic algorithm for the generalised assignment problem. *Comp. Oper. Res.* 24:260-272.

[8] Díaz J, Fernández E (2001). A tabu search heuristic for the generalized assignment problem. *EJOR* **132:1**:22-38.

[9] Fisher ML, Jaikumar R, Van Wassenhove LN (1986). A multiplier adjustment method for the generalized assignment problem. *Management Sc.* **32:9**: 1095-1103.

[10] Garey MR, Johnson DS (1979). *Computers and intractability. A guide to the theory of NP-completeness.* W. H. Freeman, San Francisco.

[11] Glover F, Laguna M (1997). Tabu search. Kluwer ed.

[12] Glover F, Kelly JP, Laguna M (1995). Genetic algorithms and tabu search: hybrids for optimization. *Computers and OR* **22:1**: 111-134.

[13] Guignard M, Rosenwein M (1989). An improved dual-based algorithm for the generalized assignment problem. *Oper. Res.* **37:4**: 658-663.

[14] Higgins AJ (2001). A dynamic tabu search for large-scale generalised assignment problems. *Computers and OR* **28:10**: 1039-1048.

[15] Karabakal N, Bean JC, Lohmann JR (1992). A steepest descent multiplier adjustment method for the generalized assignment problem. *Report 92-11* , University of Michigan, Ann Arbor.

[16] Laguna M, Kelly JP, Gonzalez-Velarde JL, Glover F (1994). Tabu search for the multilevel generalized assignment problem. *EJOR* **72**:167-174.

[17] Laguna M, Marti R, Campos V (1999). Intensification and diversification with elite tabu search solutions for the linear ordering problem. *Computers & OR* **26**: 1217-1230.

[18] Lorena LAN, Narciso MG (1996). Relaxation heuristics for a generalized assignment problem. *EJOR* **91**: 600-610.

[19] Martello S, Toth P (1981). An algorithm for the generalized assignment problem. J.P. Brans ed., *Oper. Res. '81*, North Holland, Amsterdam: 589-603.

[20] Martello S, Toth P (1987). Linear assignment problems. *Annals of Disc. Math.* **31**: 259-282.

[21] Osman IH (1991). Heuristics for the generalized assignment problem : simulated annealing and tabu search approaches. *OR Spektrum* **17**: 211-225.

[22] Plateau A, Tachat D, Tolla P (2002). A new method for 0-1 Programmning. To appear in *Intl Trans. in Op. Res.*

[23] Ross GT, Soland RM (1975). A branch-and-bound algorithm for the generalized assignment problem. *Math. Prog.* **8**: 91-103.

[24] Savelsbergh M (1997). A branch-and-cut algorithm for the generalized assignment problem. *Oper. Res.* **45:6**: 831-841.

[25] Trick M (1992). A linear relaxation heuristic for the generalized assignment problem. *Oper. Res.* **39**: 137-151.

[26] Yagiura M, Ibaraki T, Glover F (1999). An ejection chain approach for the generalized assignment problem. Technical Report 99013, Graduate Sch. of Informatics, Kyoto University.

[27] Yagiura M, Yamaguchi T, Ibaraki T (1998). A variable depth search algorithm with branching search for the generalized assignment problem. *Optimization Methods and Software* **10**: 419-441.

[28] Yagiura M, Ibaraki T, Glover F (2002). A path relinking approach for the generalized assignement problem. *Proc. International Symposium on Scheduling*, Japan, June 4-6 2002, 105-108.

[29] Wilson JM (1997). A genetic algorithm for the generalised assignment problem. *J. of the Oper. Res. Soc.* **48**: 804-809.