Lower and upper bounds for the non-linear generalized assignment problem

Claudia d'Ambrosio, Silvano Martello, Michele Monaci

Lower and upper bounds for the non-linear generalized assignment problem

Claudia D'Ambrosio[1] Silvano Martello[2] Michele Monaci[2]

Abstract

We consider a non-linear version of the Generalized Assignment Problem, a well-known strongly $\mathcal{NP}$-hard combinatorial optimization problem. We assume that the variables are continuous and that objective function and constraints are defined by non-linear functions of the variables. A mathematical model is introduced and used to derive upper bounds on the optimal solution value. We present constructive heuristics, obtained from decomposition and non-linear programming tools, and a binary linear programming model that provides approximate solutions. By combining the various methods and a local search framework, we finally obtain a hybrid heuristic approach. Extensive computational experiments show that the proposed methods outperform the direct application of non-linear solvers and provide high quality solutions in a reasonable amount of time.
Keywords. Non-linear generalized assignment problem, Upper bounds, Heuristic algorithms, Computational experiments.

1 Introduction

The *Generalized Assignment Problem* (GAP) is a classical combinatorial optimization problem: We are given a set M of *agents* and a set N of *tasks*. Assigning a task j to an agent i produces a *profit* p_{ij} and requires an amount w_{ij} of resource (*weight*). Each agent i has a maximum resource *capacity* c_i. The objective is to find a maximum profit assignment of each task to at most one agent so that the sum of the resources allocated to each agent does not exceed her capacity. Formally, the GAP can be defined as the

[1]LIX CNRS (UMR7161), École Polytechnique, 91128 Palaiseau Cedex, France.
Email: dambrosio@lix.polytechnique.fr
[2]DEI "Guglielmo Marconi", Alma Mater Studiorum University of Bologna, 40136 Bologna, Italy.
Email: {silvano.martello,michele.monaci}@unibo.it

following binary linear program:

$$\max \sum_{i \in M} \sum_{j \in N} p_{ij} x_{ij} \tag{1}$$

$$\sum_{j \in N} w_{ij} x_{ij} \leq c_i \qquad i \in M \tag{2}$$

$$\sum_{i \in M} x_{ij} \leq 1 \qquad j \in N \tag{3}$$

$$x_{ij} \in \{0, 1\} \qquad i \in M, j \in N, \tag{4}$$

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

The GAP is known to be strongly $\mathcal{NP}$-hard (see, e.g., Martello and Toth [10]) and very difficult to solve in practice.

In this paper, we consider the *Non-Linear Generalized Assignment Problem* (NLGAP), defined as a GAP in which

- variables x_{ij} are continuous;
- the total profit and the amount of resource allocated to each agent are given by non-linear, nondecreasing, separable functions of the variables;
- each task $j \in N$ has a maximum availability a_j.

The mathematical formulation of the NLGAP is then

$$\max \sum_{i \in M} \sum_{j \in N} f_{ij}(x_{ij}) \tag{5}$$

$$\sum_{j \in N} g_{ij}(x_{ij}) \leq c_i \qquad i \in M \tag{6}$$

$$\sum_{i \in M} x_{ij} \leq a_j \qquad j \in N \tag{7}$$

$$x_{ij} \geq 0 \qquad i \in M, j \in N, \tag{8}$$

where x_{ij} denotes the amount of task j that is assigned to agent i ($i \in M, j \in N$). Objective function (5) maximizes the overall profit produced by the assignments. The knapsack-like capacity constraints (6) limit the agent workloads, while constraints (7) impose the task availabilities.

We assume that profit and weight functions are twice continuously differentiable, and that the evaluation of each term $f_{ij}(x_{ij})$ and $g_{ij}(x_{ij})$ can be performed in constant time. We also assume that, given any non-negative value c, the solution of $g_{ij}(x_{ij}) = c$ takes a time bounded by a constant independent of the input size, i.e., that the same complexity applies to the computation of $g_{ij}^{-1}(\cdot)$. Note that we do not make assumptions on convexity or concavity of the involved functions.

Provided all functions can be represented using finite length input, model (5)-(8) has polynomial size (nm variables and $m + n$ constraints). On the one hand, the continuous

nature of the variables makes the problem easier than the classical GAP; on the other, the non-linearity of the objective function and of the capacity constraints make it much more difficult.

Note that, in general, the classical complexity analysis does not apply to nonlinear optimization. Indeed: (i) the input functions might not be easily encodable by finitely many bits; (ii) solutions of the problem might involve irrational numbers (see, e.g., Hochbaum [8]). The former issue is typically tackled by assuming that the evaluation of f and g at a point is provided by an oracle and by performing a complexity analysis based on the number of oracle calls made by the algorithm, together with the number of elementary operations. The latter point remains instead critical, even when considering an *approximate solution*, i.e., a feasible solution that approximates the optimal one within a given sub-optimality error. The strong $\mathcal{NP}$-hardness of approximation is a common complexity approach for continuous optimization, like, e.g., in Tillmann [16], Chen et al. [2], and Chen, Ye, and Wang [3].

Other classical interpretations of the problem (in its linear or non-linear version) consider, instead of tasks and agents, items and knapsacks (as the problem can be seen as a generalization of the *Multiple Knapsack Problem*, see again [10]), or items and bins (in particular, when *all* the items have to be assigned), or customers and facilities (see Sharkey and Romeijn [13]), or jobs and parallel machines (see Shmoys and Tardos [14]).

We are not aware of any previous work on the NLGAP. Some variants of the problem have however been studied in the literature. In 1989, Mazzola [11] considered GAP problems with non-linear capacity interaction, i.e., a case in which the weight functions are expressed as a multilinear (or polynomial) function of the variables. An application of this problem is the *hierarchical production planning problem*, in which groups of products have to be assigned to groups of facilities. The assignment of a group of products to the same facility can give rise to non-linear capacity interaction among them. The case studied in [11] has binary variables and a linear objective function.

Freling et al. [6] modeled a multiperiod single-sourcing problem as a special case of the NLGAP with binary variables, and proposed a branch-and-price algorithm. The same problem was considered by Morales and Romeijn [12], who discussed its computational complexity, reviewed some special cases, and analyzed solution approaches and methods for generating experimental data for purposes of testing.

Lee and Ma [9] were the first to introduce the *Generalized Quadratic Assignment Problem* (GQAP), a variant of the NLGAP with quadratic objective function, linear capacity constraints, and binary variables. The same problem was later considered by Hahn et al. [7], who proposed an algorithm based on an RLT (Reformulation Linearization Technique) dual ascent procedure.

More recently, Srivastava and Bullo [15] considered applications in human-robot interaction, modeled as GAPs (and other packing problems) with sigmoid utilities, linear capacity constraints, and binary variables and proposed approximation algorithms with constant approximation factor.

When profits and weights are independent of the agent, i.e., $f_{ij}(x_{ij}) = f_j(x_{ij})$ and $g_{ij}(x_{ij}) = w_j(x_{ij})$ for each $i \in M$ and $j \in N$, the NLGAP becomes a purely continuous *Multiple Non-Linear Knapsack Problem* (see D'Ambrosio, Martello, and Mencarelli [5]), while in the special case $|M| = 1$ it reduces to a purely continuous *Non-Linear Knapsack Problem* (NLK). The NLK was studied by D'Ambrosio and Martello [4], who proposed a heuristic algorithm that will be used in the next section.

The paper is organized as follows. In the next section we present constructive heuristics, obtained from decomposition and non-linear programming tools. In Section 3 we introduce continuous and binary linear programs and prove that they provide valid relaxations of the problem. An additional binary linear program can be used to produce a feasible solution. In Section 4 we present a local search approach and obtain a hybrid algorithm by combining it with the previous heuristics. Extensive computational experiments are presented in Section 5, showing that the proposed methods outperform the direct application of non-linear solvers and provide high quality solutions in a reasonable amount of time. Conclusions follow in Sections 6.

2 Constructive heuristics from non-linear programs

We introduce in this section two simple heuristic approaches based on the decomposition of the NLGAP into independent single non-linear knapsack subproblems. Recall that in the knapsack interpretation of the NLGAP an agent can be seen as a knapsack and a task as an item. The first heuristic iteratively considers an agent based decomposition. The following procedure separately produces a feasible solution for each agent:

Procedure Agent:
for $j := 1$ **to** n **do** $\overline{a}_j := a_j$;
for $i := 1$ **to** m **do**
  solve the ith NLK instance, defined by knapsack capacity c_i, item profits $f_{ij}(\cdot)$,
    item weights $g_{ij}(\cdot)$, and item availabilities $\overline{a}_j$ $(j \in N)$;
  let y_j $(j \in N)$ be the resulting solution;
  for $j := 1$ **to** n **do** $x_{ij} := y_j$, $\overline{a}_j := \overline{a}_j - y_j$
end for.

The procedure requires the solution of m single non-linear knapsack problems, which are $\mathcal{NP}$-hard, and very difficult to be exactly solved in practice. We can however obtain a fast NLGAP heuristic by computing approximate NLK solutions through the constructive heuristic by D'Ambrosio and Martello [4] as follows.

For the current agent i, let $\bar{u}_{ij}$ $(j \in N)$ denote the maximum amount of task j that can be assigned to i, computed as the minimum between the current task availability and the amount that saturates the agent's capacity:

$$\bar{u}_{ij} := \min(\overline{a}_j, g_{ij}^{-1}(c_i)) \tag{9}$$

and recall that the first term is computed in constant time. The algorithm in [4] is based on a discretization of the solution space induced by an input integer parameter s (number of samplings). For each item j, the sampling step $\Delta_j = \bar{u}_{ij}/s$ is used to discretize profit and weight functions. The algorithm iteratively selects the item and the sample that provide the largest profit to weight ratio, and packs a fraction of such item into the knapsack. The algorithm terminates when no residual capacity remains or all items have been evaluated. Its time complexity is $O(n^2)$ (see [4]).

At each iteration, procedure **Agent** executes the NLK heuristic a constant number of times, with increasing values of parameter s. The first execution is performed with a given value s_0. The following executions use values s_h ($h = 1, 2, \dots$), with $s_h = 2s_{h-1} + 1$. (In this way most of the new sample points are different from those of the previous execution.) It follows that the time complexity of **Agent** is $O(mn^2)$.

The iterative execution of procedure **Agent** for different agent sorting policies results in the following NLGAP heuristic:

Algorithm H-Agent:
$z^* := -\infty$;
sort the agents by nonincreasing capacity c_i;
repeat
  execute **Agent**: let $[x_{ij}]$ be the returned solution and z its value;
  if $z > z^*$ **then** $z^* := z$, $[x^*_{ij}] := [x_{ij}]$;
  for $i := 1$ **to** m **do**
    $\vartheta :=$ random value uniformly distributed in $[-z/m, +z/m]$;
    $r_i := \dfrac{\sum_{j \in N} f_{ij}(x_{ij}) + \vartheta}{c_i}$;
  end for
  sort the agents by nonincreasing r_i values;
until halting condition.

The agents are initially sorted by nonincreasing capacity. Each subsequent iteration sorts them by nonincreasing values of the profit per unit capacity obtained in the previous iteration. In order to avoid cycling, such value is randomly perturbed by a value ϑ depending on the average profit per agent.

If the halting condition is set to a constant number of iterations, the algorithm runs in polynomial time. Each agent sorting takes $O(m \log m)$ time. At each iteration, procedure **Agent** takes $O(mn^2)$ time while the **for** loop requires $O(mn)$ time. The overall time complexity of **H-Agent** is thus $O(m \log m + mn^2)$.

H-Agent iteratively invokes a procedure that considers one agent at a time. The next heuristic adopts a symmetrical strategy. Algorithm **H-Task** makes use of the following procedure, that considers one task at a time and assigns fractions of it to the agents. In this case the current task plays the role of a knapsack, and the agents correspond to items:

Procedure Task:
for $i := 1$ **to** m **do** $\overline{c}_i := c_i$;
for $j := 1$ **to** n **do**
solve the jth NLK instance, defined by knapsack capacity a_j, item profits $f_{ij}(\cdot)$,
unit item weights, and item availabilities $\overline{c}_i$ $(i \in M)$;
let y_i $(i \in M)$ be the resulting solution;
for $i := 1$ **to** m **do** $x_{ij} := y_i$, $\overline{c}_i := \overline{c}_i - g_{ij}(y_i)$
end for.

The NLGAP heuristic iteratively invokes procedure **Task** for different task sorting policies:

Algorithm H-Task:
$z^* := -\infty$;
for $j := 1$ **to** n **do** $r_j := \max_{i \in M} \left\{ \frac{f_{ij}(a_j)}{g_{ij}(a_j)} \right\}$
sort the tasks by nonincreasing r_j values;
repeat
execute **Task**: let $[x_{ij}]$ be the returned solution and z its value;
if $z > z^*$ **then** $z^* := z$, $[x^*_{ij}] := [x_{ij}]$;
for $j := 1$ **to** n **do**
$\vartheta :=$ random value uniformly distributed in $[-z/n, +z/n]$;
$r_j := \dfrac{\sum_{i \in M} f_{ij}(x_{ij}) + \vartheta}{\sum_{i \in M} g_{ij}(x_{ij})}$;
end for
sort the tasks by nonincreasing r_j values;
until halting condition.

(For cases where a denominator takes the value 0, we set the corresponding r_j to zero.)

From the considerations made for **H-Agent** it easily follows that **H-Task** has time complexity $O(n \log n + nm^2)$. The overall time complexity required by the consecutive execution of the two heuristics is thus $O(mn(m+n))$.

Preliminary computational experiments showed that, from a practical point of view, iterating the execution of **Agent** and **Task** for different ordering policies is beneficial for improving the quality of the solutions.

3 Upper and lower bounds from linear programs

In this section we present linear models that produce two relaxations and an approximate solution of the NLGAP.

For each pair (i, j), let u_{ij} be the maximum amount of task j that can be feasibly assigned to agent i, computed as in (9)

$$u_{ij} := \min(a_j, g_{ij}^{-1}(c_i)). \tag{10}$$

Let s be a positive integer parameter. For each pair (i, j) let $\alpha_{ij}^0, \alpha_{ij}^1, \ldots, \alpha_{ij}^s$ be $s + 1$ distinct values (*samples*) such that

$$0 = \alpha_{ij}^0 < \alpha_{ij}^1 < \ldots < \alpha_{ij}^s = u_{ij}. \tag{11}$$

The three models introduced in the following make use of the above samples, but they rely on very different considerations. The one presented in the next section replaces each continuous variable with a *linear* combination of the samples, the next one makes use of *integer* variables associated with the intervals defined by the samples, and the last one computes a lower bound by imposing that each variable can only take the value of a sample.

3.1 An upper bound through linear programming

This relaxation is obtained by overestimating (resp. underestimating) the values of the profit (resp. weight) functions at each sample k ($k = 0, \ldots, s$), namely:

$$\hat{f}_{ij}^k = \begin{cases} f_{ij}(\alpha_{ij}^{k+1}) & \text{if } k < s; \\ f_{ij}(\alpha_{ij}^s) & \text{otherwise} \end{cases} \quad \text{and } \hat{g}_{ij}^k = \begin{cases} g_{ij}(\alpha_{ij}^{k-1}) & \text{if } k \geq 1; \\ 0 & \text{otherwise} \end{cases} \quad (i \in M, j \in N). \tag{12}$$

For each triple (i, j, k), let us introduce a continuous variable $\vartheta_{ij}^k \in [0, 1]$ and consider the linear model

$$(U^{\text{LP}}) \quad \max \sum_{i \in M} \sum_{j \in N} \sum_{k=0}^{s} \hat{f}_{ij}^k \vartheta_{ij}^k \tag{13}$$

$$\sum_{j \in N} \sum_{k=0}^{s} \hat{g}_{ij}^k \vartheta_{ij}^k \leq c_i \qquad i \in M \tag{14}$$

$$\sum_{i \in M} \sum_{k=0}^{s} \alpha_{ij}^k \vartheta_{ij}^k \leq a_j \qquad j \in N \tag{15}$$

$$\sum_{k=0}^{s} \vartheta_{ij}^k = 1 \qquad i \in M, j \in N \tag{16}$$

$$\vartheta_{ij}^k \geq 0 \qquad i \in M, j \in N, k = 0, \ldots, s. \tag{17}$$

Equations (13)-(15) are the counterparts of (5)-(7) obtained through convex combinations (see (16)) of the estimated values at the sample points.

Property 1 *Model* (13)-(17) *is a valid relaxation for the NLGAP* (5)-(8).

Proof. We prove the thesis by showing that, for any feasible solution $[x_{ij}]$ to (5)-(8), there exists a feasible solution $[\vartheta_{ij}^k]$ to (13)-(17) having at least the same objective function value. Consider any pair (i,j). Let ℓ be such that $x_{ij} \in [\alpha_{ij}^\ell, \alpha_{ij}^{\ell+1}]$, i.e., $x_{ij} = \lambda\alpha_{ij}^\ell + (1-\lambda)\alpha_{ij}^{\ell+1}$ for some $\lambda \in [0,1]$. Consider the solution defined, for pair (i,j), by

$$\vartheta_{ij}^k = \begin{cases} \lambda & \text{if } k = \ell; \\ 1-\lambda & \text{if } k = \ell+1; \\ 0 & \text{otherwise.} \end{cases} \tag{18}$$

The corresponding contribution to the left-hand side of (14) is thus $\lambda\hat{g}_{ij}^\ell + (1-\lambda)\hat{g}_{ij}^{\ell+1} \leq g_{ij}(x_{ij})$, where the inequality follows from (12), and hence $[\vartheta_{ij}^k]$ satisfies (14). Similarly, the contribution of pair (i,j) to (15) is $\lambda\alpha_{ij}^\ell + (1-\lambda)\alpha_{ij}^{\ell+1} = x_{ij}$ by definition of ℓ, and hence $[\vartheta_{ij}^k]$ satisfies (15). The definition of ϑ immediately implies (16). Finally, the contribution of (i,j) to (13) is $\lambda\hat{f}_{ij}^\ell + (1-\lambda)\hat{f}_{ij}^{\ell+1}$, which, by (12), is at least equal to $f_{ij}(x_{ij})$. □

As (13)-(17) is a linear program, upper bound U^{LP} can be computed in polynomial time, provided that parameter s is chosen polynomial in the encoding length of the problem instance. As we will see in Section 5, its computation is very fast even for a large number s of samples, allowing a tight approximation of profit and weight functions.

3.2 An upper bound through 0-1 linear programming

The relaxation presented in this section replaces the domain of each continuous variable x_{ij} with the s consecutive intervals

$$(\alpha_{ij}^0, \alpha_{ij}^1], (\alpha_{ij}^1, \alpha_{ij}^2], \ldots, (\alpha_{ij}^{s-1}, \alpha_{ij}^s] \tag{19}$$

induced by the samples, and associates with each interval k ($k = 1, \ldots, s$) the profit of its right extreme and the weight of its left extreme, namely:

$$\widetilde{f}_{ij}^k = f_{ij}(\alpha_{ij}^k) \quad \text{and} \quad \widetilde{g}_{ij}^k = g_{ij}(\alpha_{ij}^{k-1}) \quad (i \in M, j \in N).$$

For each triple (i,j,k), we introduce a binary variable $\psi_{ij}^k \in \{0,1\}$ and consider the 0-1 linear model

$$(U^{\text{LP01}}) \quad \max \sum_{i\in M}\sum_{j\in N}\sum_{k=1}^{s} \widetilde{f}_{ij}^k \psi_{ij}^k \tag{20}$$

$$\sum_{j\in N}\sum_{k=1}^{s} \widetilde{g}_{ij}^k \psi_{ij}^k \leq c_i \qquad i \in M \tag{21}$$

$$\sum_{i\in M}\sum_{k=1}^{s} \alpha_{ij}^{k-1} \psi_{ij}^k \leq a_j \qquad j \in N \tag{22}$$

$$\sum_{k=1}^{s} \psi_{ij}^k = 1 \qquad i \in M, j \in N \tag{23}$$

$$\psi_{ij}^k \in \{0,1\} \qquad i \in M, j \in N, k = 1, \ldots, s. \tag{24}$$

Constraints (23) impose that one interval is assigned to each pair (i, j), while (20)-(22) describe profit, weight and task consumption of the interval assignment.

Property 2 *Model* (20)-(24) *is a valid relaxation for the NLGAP* (5)-(8).

Proof. We prove the thesis by showing that, for any feasible solution $[x_{ij}]$ to (5)-(8), there exists a feasible solution $[\psi_{ij}^k]$ to (20)-(24) having at least the same objective function value. For any pair (i, j),

- let ℓ be such that $x_{ij} \in (\alpha_{ij}^{\ell-1}, \alpha_{ij}^{\ell}]$ ($\ell = 1$ if $x_{ij} = 0$);
- observe that, as f and g are nondecreasing, we have $f_{ij}(x_{ij}) \in [f_{ij}(\alpha_{ij}^{\ell-1}), f_{ij}(\alpha_{ij}^{\ell})]$ and $g_{ij}(x_{ij}) \in [g_{ij}(\alpha_{ij}^{\ell-1}), g_{ij}(\alpha_{ij}^{\ell})]$;
- associate pair (i, j) to segment ℓ, i.e.,
$$\psi_{ij}^k = \begin{cases} 1 & \text{if } k = \ell; \\ 0 & \text{otherwise.} \end{cases} \tag{25}$$
- Recalling that f and g are nondecreasing, the contribution of pair (i, j) to objective function (20), to the left-hand side of (21), and to the left-hand side of (22) are, respectively, $\tilde{f}_{ij}^{\ell} \geq f_{ij}(x_{ij})$, $\tilde{g}_{ij}^{\ell} \leq g_{ij}(x_{ij})$, and $\alpha_{ij}^{\ell-1} \leq x_{ij}$.

The thesis follows. □

3.3 A lower bound through 0-1 linear programming

The samples $[\alpha_{ij}^k]$, see (11), can also be used to define a binary linear program that can produce a heuristic solution to the NLGAP.

For each triple (i, j, k), we sample the profit and weight functions as

$$\bar{f}_{ij}^k = f_{ij}(\alpha_{ij}^k) \quad \text{and} \quad \bar{g}_{ij}^k = g_{ij}(\alpha_{ij}^k) \quad (i \in M, j \in N), \tag{26}$$

and we define a binary variable φ_{ij}^k taking the value one if and only if $x_{ij} = \alpha_{ij}^k$. This leads to the model

$$(L^{0\text{-}1}) \qquad \max \sum_{i \in M} \sum_{j \in N} \sum_{k=0}^{s} \bar{f}_{ij}^k \varphi_{ij}^k \tag{27}$$

$$\sum_{j \in N} \sum_{k=0}^{s} \bar{g}_{ij}^k \varphi_{ij}^k \leq c_i \qquad i \in M \tag{28}$$

$$\sum_{i \in M} \sum_{k=0}^{s} \alpha_{ij}^k \varphi_{ij}^k \leq a_j \qquad j \in N \tag{29}$$

$$\sum_{k=0}^{s} \varphi_{ij}^k = 1 \qquad i \in M, j \in N \tag{30}$$

$$\varphi_{ij}^k \in \{0, 1\} \qquad i \in M, j \in N, k = 0, \ldots, s. \tag{31}$$

The model imposes (see (30)) that, for any pair (i, j), the amount of task j that is assigned to agent i coincide with one of the samples. Equations (27)-(29) describe the corresponding contribution to objective function, agent workload, and task availability, respectively.

Property 3 *Model* (27)-(31) *provides a feasible solution to the NLGAP* (5)-(8).

Proof. Observe that any solution $[\varphi_{ij}^k]$ satisfying (28)-(31) provides a feasible solution $[x_{ij}]$ to the NLGAP. Indeed, (30) imposes that, for each pair (i, j) exactly one sample, say ℓ, is selected, i.e., $x_{ij} = \alpha_{ij}^{\ell}$. Such solution satisfies (6)-(8) and has value $\sum_{i\in M}\sum_{j\in N} f_{ij}(x_{ij}) = \sum_{i\in M}\sum_{j\in N}\sum_{k=0}^{s} \bar{f}_{ij}^k \varphi_{ij}^k$. □

Note that model ($L^{0\text{-}1}$) produces the best solution over the feasible region obtained by discretizing the domain of each variable x_{ij}, and hence it can be expected to provide a good approximation to the problem. Unfortunately, this lower bound cannot be computed efficiently. Indeed,

Property 4 *Problem* ($L^{0\text{-}1}$) *is strongly* $\mathcal{NP}$*-hard.*

Proof. We prove the thesis by transformation from the *Multiple Subset-Sum Problem* (MSSP): Given m identical bins of capacity c and a set of n integer values $w_j \leq c$, select m subsets such that no subset has a total value exceeding c and the sum of the selected integers is a maximum. Formally,

$$\max \sum_{i\in M}\sum_{j\in N} w_j x_{ij} \tag{32}$$

$$\sum_{j\in N} w_j x_{ij} \leq c \qquad i \in M \tag{33}$$

$$\sum_{i\in M} x_{ij} \leq 1 \qquad j \in N \tag{34}$$

$$x_{ij} \in \{0, 1\} \qquad i \in M, j \in N. \tag{35}$$

This problem is known to be strongly $\mathcal{NP}$-hard, see Caprara, Kellerer, and Pferschy [1].

Given an instance of the MSSP, consider an instance of (27)-(31) in which

- $c_i = c$ for all i.
- $s = 1$, i.e., there are only two samples, $\alpha_{ij}^0 = 0$ and $\alpha_{ij}^1 = u_{ij}$ for all i, j;
- $f_{ij}(\cdot)$ and $g_{ij}(\cdot)$ satisfy $f_{ij}(0) = g_{ij}(0) = 0$ and $f_{ij}(u_{ij}) = g_{ij}(u_{ij}) = w_j$ for all i, j;
- $a_j = 1$ for all j,

and observe that the last two definitions imply $u_{ij} = 1$ for all i, j. For such instance, (27)-(31) becomes

$$\max \sum_{i \in M} \sum_{j \in N} \left(0\,\varphi_{ij}^0 + w_j \varphi_{ij}^1\right) \tag{36}$$

$$\sum_{j \in N} \left(0\,\varphi_{ij}^0 + w_j \varphi_{ij}^1\right) \leq c \qquad i \in M \tag{37}$$

$$\sum_{i \in M} \left(0\,\varphi_{ij}^0 + 1\,\varphi_{ij}^1\right) \leq 1 \qquad j \in N \tag{38}$$

$$\varphi_{ij}^0 + \varphi_{ij}^1 = 1 \qquad i \in M, j \in N \tag{39}$$

$$\varphi_{ij}^0, \varphi_{ij}^1 \in \{0, 1\} \qquad i \in M, j \in N. \tag{40}$$

By eliminating the terms with φ_{ij}^0, equation (39) becomes redundant as $\varphi_{ij}^0 = 1 - \varphi_{ij}^1$. We thus obtain a formulation that exactly models the MSSP. □

Although it can be impractical to exactly solve ($L^{0\text{-}1}$), the results in Section 5 show that an ILP solver can quickly provide heuristic solutions of good quality for (27)-(31) and hence for the NLGAP.

4 Post-optimization and hybrid approach

In this section we first introduce a local search algorithm that can be used to improve any feasible solution $[x_{ij}]$ to the NLGAP. We then present an overall hybrid algorithm that combines the constructive heuristics of Section 2 and the 0-1 linear programming approach of Section 3.3.

4.1 Post-optimization

In the following, we denote by $\bar{c}_i$ the current slack (residual capacity) of the i-th constraint (6) and by $\bar{a}_j$ the current slack (residual availability) of the j-th constraint (7). The algorithm makes use of the following four neighborhoods:

1. Preliminary observe that, in general, there is no guarantee that a solution be maximal. This neighborhood contains a solution for each agent i and task j such that $\bar{c}_i > 0$ and $\bar{a}_j > 0$, obtained by increasing x_{ij} to its maximum feasible value. Formally,
   N1(i, j): (a) set $x_{ij} := 0$, and correspondingly update $\bar{c}_i$ and $\bar{a}_j$;
   (b) set $x_{ij} := \min\left(\bar{a}_j,\, g_{ij}^{-1}(\bar{c}_i)\right)$.

2. For each agent i and pair of distinct tasks j and k such that $x_{ij} > 0$ and $\bar{a}_k > 0$, the second neighborhood contains all solutions obtained by decreasing x_{ij} to a certain

value, say Δ ($< x_{ij}$), and increasing x_{ik} to the maximum resulting feasible value. Formally,

N2(i, j, k): (a) set $x_{ij} := \Delta$, $x_{ik} := 0$, and correspondingly update $\bar{c}_i$ and $\bar{a}_k$;
(b) set $x_{ik} := \min\left(\bar{a}_k,\, g_{ik}^{-1}(\bar{c}_i)\right)$.

3. For each task j and pair of distinct agents i and ℓ such that $x_{ij} > 0$ and $\bar{c}_\ell > 0$, the third neighborhood is a "dual" of the second one. It contains all solutions obtained by decreasing x_{ij} to Δ ($< x_{ij}$), and increasing $x_{\ell j}$ to the maximum resulting feasible value. Formally,

   N3(j, i, ℓ): (a) set $x_{ij} := \Delta$, $x_{\ell j} := 0$, and correspondingly update $\bar{c}_\ell$ and $\bar{a}_j$;
   (b) set $x_{\ell j} := \min\left(\bar{a}_j,\, g_{\ell j}^{-1}(\bar{c}_\ell)\right)$.

4. For each pair of distinct agents i and ℓ and pair of distinct tasks j and k such that $x_{ij} > 0$ and $x_{\ell k} > 0$, the last neighborhood contains all solutions obtained by simultaneously decreasing x_{ij} and $x_{\ell k}$ to Δ' ($< x_{ij}$) and Δ'' ($< x_{\ell k}$), respectively, and increasing x_{ik} and $x_{\ell j}$ to their maximum resulting feasible values. Formally,

   N4(i, ℓ, j, k): (a) set $x_{ij} := \Delta'$, $x_{\ell k} := \Delta''$, $x_{ik} := x_{\ell j} := 0$, and correspondingly update $\bar{c}_i$, $\bar{c}_\ell$, $\bar{a}_j$, and $\bar{a}_k$;
   (b) set $x_{ik} := \min\left(\bar{a}_k,\, g_{ik}^{-1}(\bar{c}_i)\right)$ and $x_{\ell j} := \min\left(\bar{a}_j,\, g_{\ell j}^{-1}(\bar{c}_\ell)\right)$.

The Δ values (needed by **N2** and **N3**) are handled as follows. Given a prefixed integer parameter σ, we evaluate, for each x_{ij} variable, σ potential solutions, obtained, for $r = 1, \ldots, \sigma$, by decreasing x_{ij} to $\Delta := r\frac{x_{ij}}{\sigma}$. The best solution is then returned. The values of Δ' and Δ'' needed by **N4** are handled in a similar way.

The four neighborhoods are sequentially explored following a *best-improve* strategy, but the exploration of each neighborhood is not exhaustive, according to the following random criterion. As the neighborhoods have different size, in order to avoid to spend too much time on one of them, thus possibly failing the exploration of others, it was decided to explore a similar number of solutions of each neighborhood. Let ν be an integer parameter representing the desired number of pairs (resp. triples, resp. quadruplets) to examine for **N1** (resp. for **N2** and **N3**, resp. for **N4**): For each pair/triple/quadruplet, the corresponding neighborhood is examined with a probability given by ν over the number of distinct pairs/triples/quadruplets. The overall local search algorithm is then:

Procedure Local Search (ν, σ)**:**
repeat
  for each pair (i, j) **do**
    with probability $\frac{\nu}{nm}$ explore **N1**(i, j);
    possibly update the incumbent solution
  end for;
  if the incumbent solution was updated **then continue**;

for each triple (i, j, k) **do**
with probability $\frac{\nu}{n(n-1)m}$ explore $\mathbf{N2}(i, j, k)$;
possibly update the incumbent solution
end for;
if the incumbent solution was updated **then continue**;
for each triple (j, i, ℓ) **do**
with probability $\frac{\nu}{nm(m-1)}$ explore $\mathbf{N3}(j, i, \ell)$;
possibly update the incumbent solution
end for;
if the incumbent solution was updated **then continue**;
for each quadruplet (i, ℓ, j, k) **do**
with probability $\frac{\nu}{nm(n-1)(m-1)}$ explore $\mathbf{N4}(i, \ell, j, k)$;
possibly update the incumbent solution
end for;
if the incumbent solution was not updated **then return**;
until halting condition.

The procedure terminates when either time/iteration limit occurs or a local optimum is attained. In the latter case, σ is increased and the procedure is re-executed. The increasing is obtained by generating a uniform real number $\rho \in [1, 2]$ and setting $\sigma := \lfloor \rho\sigma \rfloor + 1$ (to guarantee strictly increasing σ values).

Preliminary computational experiments showed that changing the order in which the neighborhoods are explored has no major impact on the quality of the solutions found.

4.2 Hybrid heuristic

Summarizing, the solution methods we have introduced so far are:

- constructive heuristics **H-Agent** and **H-Task** (Section 2);
- 0-1 linear programming approach (Section 3.3);
- post-optimization (Section 4.1).

First observe that post-optimization can be embedded into **H-Agent** and **H-Task** by executing, at each iteration, procedure **Local Search** on the feasible solution produced by inner procedures **Agent** and **Task**, respectively. We denote by **H-Agent*** and **H-Task*** the resulting algorithms.

Their combination leads to the following overall algorithm for the NLGAP:

Procedure Hybrid:
execute **H-Agent*** and **H-Task***, and let $[x_{ij}]$ be the best solution found;
for each $i \in M$ and $j \in N$ **do**
define the α_{ij}^k values $(k = 0, 1, \ldots, s)$ as in (11);
for $k := 0$ **to** s **do** $\varphi_{ij}^k = \begin{cases} 1 & \text{if } k = \arg \max_{h \in \{0,\ldots,s\}} \{\alpha_{ij}^h \leq x_{ij}\}; \\ 0 & \text{otherwise} \end{cases}$
end for;
run an ILP solver on $(L^{0\text{-}1})$, with initial solution $[\varphi_{ij}^k]$;
execute **Local Search** on the solution returned by the solver
end.

5 Computational experiments

The procedures introduced in the previous sections were implemented in C language. Computational experiments were performed on an Intel Xeon E5649 running at 2.53 GHz. To the best of our knowledge, no benchmark for the NLGAP has been proposed in the literature. We thus designed a large benchmark of instances with different characteristics, that is described in the next section. In Section 5.2 we report the outcome of the experiments that compare the performance of our methods with that of state-of-the-art MINLP solvers.

5.1 Benchmarks

The NLGAP can be seen as a generalization of the multiple non-linear knapsack problem. It was then natural to derive benchmark instances from those proposed for such problem. In particular, we started from the instances adopted by D'Ambrosio, Martello, and Mencarelli [5]. Accordingly, we defined the profit functions as

$$f_{ij}(x_{ij}) = \frac{\gamma_{ij}}{1 + \beta_{ij} \mathrm{e}^{-\alpha_{ij}(x_{ij} + \delta_{ij})}}, \qquad (i \in M, j \in N), \tag{41}$$

by uniformly randomly generating α_{ij} in $[0.1, 0.2]$, β_{ij} and γ_{ij} in $[0, 100]$, and δ_{ij} in $[-100, 0]$. In this way each profit function can randomly turn out to be convex, or concave, or non-concave and non-convex.

We generated two families of instances having *Non-linear* and *Linear* weight functions, defined, respectively, as

$$g_{ij}(x_{ij}) = \sqrt{\pi_{ij} x_{ij} + \tau_{ij}} - \sqrt{\tau_{ij}}, \tag{42}$$

with π_{ij} and τ_{ij} uniformly random in $[1, 20]$, and

$$g_{ij}(x_{ij}) = \pi_{ij} x_{ij}, \tag{43}$$

with π_{ij} uniformly random in $[1, 100]$. Note that both functions are concave (the latter is linear), increasing, and take the value zero when $x_{ij} = 0$.

For each family, we produced two sets of instances with different kinds of capacity. Instances with *Similar capacities* have

$$c_i = \rho \sum_{j=1}^{n} g_{ij}(u_j) \quad (i = 1, \ldots, m), \tag{44}$$

ρ being a uniformly random value in $[0.4, 0.6]$. For instances with *Dissimilar capacities*, ρ was generated in $[0.1, 0.9]$.

The two weight functions and the two kinds of capacity produced then four benchmarks, denoted as **NS** (Non-linear weights, Similar capacities), **ND** (Non-linear weights, Dissimilar capacities), **LS** (Linear weights, Similar capacities), and **LD** (Linear weights, Dissimilar capacities). For each benchmark, we considered five values of n (10, 50, 100, 500, and 1000) and three values of m (5, 10, and 20), and generated 20 instances per pair (n, m). The overall test bed includes then 1200 random instances, that are available at `http://or.dei.unibo.it/library/non-linear-generalized-assignment-problem`.

5.2 Experiments

The computational experiments were performed in two phases. First, we computed, for each instance, an upper bound on the optimal solution value. This was obtained by running IBM ILOG CPLEX 12.6.0.0 on

- the linear problem (U^{LP}) of Section 3.1;
- the binary problem (U^{LP01}) of Section 3.2,

for different numbers of samples ($s \in \{5, 10, 50, 100\}$). In all cases, the samples were defined so that the resulting intervals had the same length. A time limit of 300 seconds per execution was imposed. For each instance, the best (lowest) upper bound (U^* in the following) was selected. The upper bound U produced by each computation was then evaluated through the ratio U/U^*.

Table 1 reports average times and ratios for the instances of class NS. Each entry reports the average values over the corresponding 20 instances. For each value of m, an additional row provides the average values over the associated 100 instances, and a final row provides the overall averages.

The linear model was always solved to optimality within comparatively small computing times. The quality of the resulting upper bounds monotonically improves with s (and hence with the CPU time). For $s = 100$, the linear model produces good upper bounds (on average within 5% from the best bound) with reasonable computing times (below 13 seconds on average).

		(U^{LP})								(U^{LP01})							
Size		$s=5$		$s=10$		$s=50$		$s=100$		$s=5$		$s=10$		$s=50$		$s=100$	
m	n	t	U/U^*	t	U/U^*	t	U/U^*	t	U/U^*	t	U/U^*	t	U/U^*	t	U/U^*	t	U/U^*
5	10	0.00	1.676	0.00	1.337	0.01	1.175	0.01	1.159	0.03	1.207	0.06	1.088	0.55	1.008	1.92	1.000
	50	0.01	1.676	0.01	1.230	0.05	1.050	0.10	1.034	0.74	1.218	2.22	1.086	46.29	1.009	77.74	1.000
	100	0.01	1.671	0.02	1.230	0.09	1.046	0.30	1.030	2.06	1.223	10.11	1.087	26.52	1.008	88.82	1.000
	500	0.10	1.637	0.16	1.220	0.88	1.045	5.22	1.030	8.43	1.218	11.39	1.085	61.47	1.008	138.42	1.000
	1000	0.24	1.647	0.43	1.221	2.74	1.044	19.68	1.029	7.95	1.220	14.03	1.085	108.66	1.008	250.21	1.000
	avg.	0.07	1.661	0.12	1.247	0.75	1.072	5.06	1.056	3.84	1.217	7.56	1.086	48.70	1.008	111.40	1.000
10	10	0.00	1.378	0.00	1.218	0.02	1.117	0.03	1.106	0.05	1.114	0.08	1.044	0.56	1.004	2.06	1.000
	50	0.02	1.978	0.02	1.323	0.09	1.068	0.24	1.048	3.38	1.318	38.98	1.111	264.01	1.009	287.87	1.000
	100	0.03	1.937	0.05	1.302	0.22	1.054	0.71	1.035	60.81	1.312	224.18	1.111	300.00	1.010	300.00	1.000
	500	0.22	1.935	0.37	1.300	2.23	1.053	10.99	1.034	285.21	1.317	219.82	1.114	299.28	1.010	300.00	1.205
	1000	0.54	1.937	1.02	1.298	6.79	1.052	37.52	1.032	106.95	1.315	72.90	1.113	298.84	1.211	300.00	1.000
	avg.	0.16	1.833	0.29	1.288	1.87	1.069	9.90	1.051	91.27	1.275	111.18	1.098	232.33	1.049	237.97	1.041
20	10	0.01	1.211	0.01	1.113	0.03	1.050	0.06	1.043	0.05	1.071	0.11	1.032	1.17	1.003	3.53	1.000
	50	0.04	2.208	0.06	1.418	0.22	1.092	0.51	1.067	3.63	1.430	98.12	1.138	300.00	1.009	300.00	1.000
	100	0.10	2.252	0.11	1.416	0.53	1.072	1.54	1.047	231.06	1.478	300.00	1.158	300.00	1.014	300.00	1.000
	500	0.57	2.210	0.95	1.403	4.83	1.064	23.75	1.040	300.00	1.475	300.00	1.158	300.00	1.012	300.00	1.000
	1000	1.46	2.189	2.70	1.388	14.65	1.053	89.24	1.029	300.00	1.460	300.00	1.146	300.00	1.002	300.00	1.000
	avg.	0.43	2.014	0.77	1.348	4.05	1.066	23.02	1.045	166.95	1.383	199.65	1.126	240.23	1.008	240.71	1.000
overall avg.		0.22	1.836	0.39	1.294	2.22	1.069	12.66	1.051	87.35	1.292	106.13	1.104	173.75	1.022	196.69	1.014

Table 1: Upper bound computations for Non-linear weight function and Similar capacities. Average values over 20 instances.

		(U^{LP})								(U^{LP01})							
		$s=5$		$s=10$		$s=50$		$s=100$		$s=5$		$s=10$		$s=50$		$s=100$	
Class	m	t	U/U^*	t	U/U^*	t	U/U^*	t	U/U^*	t	U/U^*	t	U/U^*	t	U/U^*	t	U/U^*
NS	5	0.07	1.661	0.12	1.247	0.75	1.072	5.06	1.056	3.84	1.217	7.56	1.086	48.70	1.008	111.40	1.000
	10	0.16	1.833	0.29	1.288	1.87	1.069	9.90	1.051	91.27	1.275	111.18	1.098	232.33	1.049	237.97	1.041
	20	0.43	2.014	0.77	1.348	4.05	1.066	23.02	1.045	166.95	1.383	199.65	1.126	240.23	1.008	240.71	1.000
	avg.	0.22	1.836	0.39	1.294	2.22	1.069	12.66	1.051	87.35	1.292	106.13	1.104	173.75	1.022	196.69	1.014
ND	5	0.08	1.661	0.14	1.242	0.71	1.068	4.68	1.052	4.14	1.214	5.71	1.082	41.14	1.008	102.49	1.000
	10	0.18	1.863	0.31	1.300	1.68	1.074	10.01	1.055	82.05	1.281	91.72	1.101	221.78	1.008	236.09	1.194
	20	0.44	2.012	0.77	1.350	3.81	1.071	21.31	1.049	147.49	1.371	195.16	1.123	240.21	1.008	240.99	1.000
	avg.	0.23	1.845	0.40	1.297	2.07	1.071	12.00	1.052	77.90	1.289	97.53	1.102	167.71	1.008	193.19	1.065
LS	5	0.07	1.311	0.16	1.170	1.04	1.071	5.91	1.060	3.68	1.265	7.48	1.111	69.42	1.011	149.72	1.031
	10	0.20	1.389	0.43	1.202	2.73	1.081	14.34	1.067	97.43	1.334	123.23	1.134	236.76	1.011	240.85	1.000
	20	0.47	1.467	0.97	1.216	7.02	1.066	33.71	1.049	204.20	1.421	238.67	1.160	241.11	1.011	244.76	1.002
	avg.	0.25	1.389	0.52	1.196	3.60	1.073	17.99	1.059	101.77	1.340	123.13	1.135	182.43	1.011	211.78	1.011
LD	5	0.07	1.311	0.15	1.168	1.07	1.070	5.98	1.058	2.45	1.260	7.03	1.109	52.34	1.011	123.67	1.053
	10	0.20	1.387	0.39	1.201	2.70	1.081	14.02	1.067	74.71	1.333	106.66	1.132	234.53	1.073	241.88	1.000
	20	0.49	1.466	0.93	1.217	7.63	1.068	32.85	1.052	205.52	1.416	238.18	1.159	241.31	1.011	246.13	1.002
	avg.	0.25	1.388	0.49	1.195	3.80	1.073	17.62	1.059	94.23	1.336	117.29	1.134	176.06	1.032	203.89	1.018

Table 2: Summary of results for the upper bound computations (four classes of instances).

As one could expect, finding the optimal solution of the integer model was much more difficult, and the solver frequently hit the time limit, especially for larger m values: for $s = 100$, this happened 5 times (out of 100) for the instances with $m = 5$, 78 times for those with $m = 10$, and 80 times for those with $m = 20$. On the other hand, the resulting upper bounds are considerably tighter when large s values are used. Observe however that U^{LP01} with $s \leq 10$ is dominated by U^{LP} with $s = 100$ both in terms of upper bound value and CPU time. Overall, the best value U^* was obtained by the integer model for $s = 100$ in 296 cases out of 300.

The results obtained for the other three classes (ND, LS, and LD) were very similar to those reported in Table 1. For this reason we do not give the detailed results, but provide, in Table 2, the average values over each group of 100 instances having the same value of m (and the overall results of each class). It can be observed that only in the simplest ($s = 5$) linear relaxation there is a small difference between the instances with Non-linear and Linear weight functions: in the latter case the linear model provides upper bounds closer to the best computed upper bounds. For larger s values and for the binary linear model (all s values), such behavior does not occur.

The next tables compare the performance of five heuristic approaches to the NLGAP:

- **Baron**: direct application of commercial solver Baron version 18.11.12 on the non-linear formulation (5)-(8) of Section 1;
- **Couenne**: direct application of freeware solver Couenne version 0.5 on the non-linear formulation (5)-(8) of Section 1;
- **H-Agent***: heuristic **H-Agent** of Section 2 improved through procedure **Local Search** of Section 4.1;
- **H-Task***: heuristic **H-Task** of Section 2 improved through procedure **Local Search** of Section 4.1;
- **L01**: 0-1 linear model ($L^{0\text{-}1}$) of Section 3.3, defined by (27)-(31), solved by ILP commercial solver IBM ILOG CPLEX 12.6.0.0. The solver is run as standalone program, without providing an initial solution;
- **Hyb**: hybrid approach of Section 4.2. For given time limit T, the algorithm was executed by assigning $0.1\,T$ to **H-Agent***, $0.1\,T$ to **H-Task***, $0.7\,T$ to **L01**, and the residual time to **Local Search**.

For each instance, the heuristics were run with two different time limits: 10 seconds and 30 seconds. In the former (resp. latter) case the discretization was produced with $s = 50$ (resp. $s = 100$) for **H-Agent*** and **H-Task***, and $s = 5$ (resp. $s = 10$) for **L01**, and **Hyb**. As Baron and Couenne are general purpose solvers, they were instead run with a considerably larger time limit of 600 seconds. All solvers were executed with their default parameter setting. Tables 3-6 report, for the considered time limits, the average percentage gap of each algorithm over the 20 instances of each row. For each instance,

Class		T=10 sec				T=30 sec				T=600 sec	
m	n	H-Task*	H-Agent*	L01	Hyb	H-Task*	H-Agent*	L01	Hyb	Couenne	Baron
5	10	3.989	2.597	3.893	1.537	2.679	2.614	2.107	1.055	5.015	1.067
	50	6.090	5.020	2.631	1.397	6.268	4.916	1.391	1.000	29.532	12.757
	100	5.956	4.743	2.131	1.231	6.110	4.569	1.196	0.907	32.352	14.627
	500	9.854	6.637	1.935	1.711	8.187	6.046	1.138	0.942	35.163	27.887
	1000	19.844	7.892	1.975	1.921	13.248	6.271	1.138	1.054	38.068	44.267
	avg.	9.146	5.378	2.513	1.559	7.298	4.883	1.394	0.992	28.026	20.121
10	10	2.164	1.441	2.408	0.710	1.922	1.439	1.510	0.613	5.781	1.179
	50	8.543	7.660	4.168	2.513	8.949	7.151	2.539	1.884	34.412	12.569
	100	8.498	7.352	3.058	2.079	8.604	7.138	1.919	1.537	35.966	14.254
	500	18.450	10.639	2.549	2.455	11.073	9.207	1.549	1.499	40.349	52.222
	1000	34.860	19.463	2.652	2.622	26.639	12.962	1.680	1.671	100.000	100.000
	avg.	14.503	9.311	2.967	2.076	11.437	7.579	1.839	1.441	43.302	36.045
20	10	1.002	0.621	1.889	0.393	0.957	0.554	1.046	0.423	19.593	0.936
	50	13.116	12.173	7.011	5.310	13.818	11.724	4.728	3.878	39.377	14.072
	100	12.819	13.689	4.954	4.258	12.403	13.266	3.751	3.218	38.800	14.285
	500	33.386	22.309	3.503	3.385	20.054	15.974	2.514	2.479	91.108	100.000
	1000	43.444	35.300	4.401	4.568	41.094	29.046	3.481	3.766	100.000	99.970
	avg.	20.753	16.818	4.352	3.583	17.665	14.113	3.104	2.753	57.776	45.853
overall avg.		14.801	10.503	3.277	2.406	12.134	8.858	2.112	1.728	43.034	34.006

Table 3: Percentage gap of the heuristics on instances with Non-linear weight function and Similar capacities. Average values over 20 instances.

Class		T=10 sec				T=30 sec				T=600 sec	
m	n	H-Task*	H-Agent*	L01	Hyb	H-Task*	H-Agent*	L01	Hyb	Couenne	Baron
5	10	4.459	2.904	3.080	1.220	3.877	2.791	1.790	0.708	1.738	0.631
	50	6.103	5.316	2.569	1.513	5.962	4.998	1.463	1.046	27.260	13.596
	100	5.873	4.624	2.166	1.277	6.201	4.413	1.199	0.904	32.734	15.706
	500	9.463	6.773	1.937	1.693	8.130	6.307	1.126	0.930	87.293	28.262
	1000	20.042	7.631	1.926	1.858	12.799	6.282	1.122	1.038	100.000	100.000
	avg.	9.188	5.450	2.336	1.512	7.394	4.958	1.340	0.925	49.805	31.639
10	10	2.046	1.716	3.092	0.977	2.043	1.715	1.889	0.824	7.861	1.359
	50	9.074	7.106	4.050	2.530	7.970	6.920	2.538	1.903	33.651	12.242
	100	8.133	7.360	3.104	2.135	8.003	7.154	1.992	1.579	35.639	14.730
	500	18.159	10.257	2.746	2.637	10.925	8.866	1.748	1.707	100.000	35.928
	1000	35.713	18.254	2.613	2.571	26.720	11.876	1.650	1.660	100.000	100.000
	avg.	14.625	8.939	3.121	2.170	11.132	7.306	1.963	1.535	55.430	32.852
20	10	0.947	0.791	2.202	0.528	0.966	0.791	1.044	0.428	18.255	1.023
	50	12.847	12.010	6.183	4.520	12.271	11.406	4.288	3.474	37.891	14.070
	100	12.466	13.040	4.866	3.902	11.722	12.476	3.386	2.896	38.522	13.829
	500	33.265	19.758	3.423	3.320	20.140	14.380	2.406	2.395	93.944	72.476
	1000	45.170	33.508	4.550	4.609	42.124	26.264	3.532	3.546	100.000	100.000
	avg.	20.939	15.821	4.245	3.376	17.445	13.063	2.931	2.548	57.722	40.280
overall avg.		14.917	10.070	3.234	2.353	11.990	8.443	2.078	1.669	54.319	34.923

Table 4: Percentage gap of the heuristics on instances with Non-linear weight function and Dissimilar capacities. Average values over 20 instances.

Class		T=10 sec				T=30 sec				T=600 sec	
m	n	H-Task*	H-Agent*	L01	Hyb	H-Task*	H-Agent*	L01	Hyb	Couenne	Baron
5	10	2.932	1.884	5.088	1.561	2.889	1.896	2.738	1.300	9.457	1.824
	50	4.863	4.525	3.110	1.662	4.771	4.593	1.840	1.320	21.594	14.270
	100	4.541	4.043	2.687	1.456	4.533	4.110	1.582	1.171	23.616	16.257
	500	10.229	12.947	2.616	2.315	8.391	11.512	1.549	1.292	100.000	17.823
	1000	22.767	15.014	2.816	2.759	14.493	14.432	1.734	1.658	100.000	94.793
	avg.	9.066	7.683	3.263	1.951	7.015	7.309	1.888	1.348	50.934	28.993
10	10	2.613	2.122	4.867	1.075	2.821	2.273	2.602	1.043	10.282	2.069
	50	6.750	6.123	4.515	2.642	5.776	6.128	2.944	2.210	19.577	14.820
	100	6.318	6.256	3.890	2.685	5.801	5.872	2.414	1.897	80.445	15.210
	500	19.455	22.779	3.931	3.857	12.265	21.570	2.531	2.488	100.000	100.000
	1000	39.021	24.793	4.177	4.171	29.528	21.388	2.810	2.824	100.000	100.000
	avg.	14.832	12.415	4.276	2.886	11.238	11.446	2.660	2.092	62.061	46.420
20	10	2.296	3.618	4.379	1.500	2.092	3.697	2.270	1.039	14.176	3.122
	50	9.989	9.839	7.370	5.112	9.469	8.997	5.068	3.812	42.257	15.959
	100	11.401	14.368	5.659	5.032	9.714	10.110	3.807	3.313	100.000	16.164
	500	34.534	29.110	5.543	5.516	20.911	21.465	3.972	4.027	100.000	100.000
	1000	50.748	42.865	6.890	7.304	43.295	33.544	5.314	5.854	100.000	100.000
	avg.	21.794	19.960	5.968	4.893	17.096	15.563	4.086	3.609	71.287	47.049
overall avg.		15.231	13.352	4.503	3.243	11.783	11.439	2.878	2.350	61.427	40.821

Table 5: Percentage gap of the heuristics on instances with Linear weight function and Similar capacities. Average values over 20 instances.

Class		T=10 sec				T=30 sec				T=600 sec	
m	n	H-Task*	H-Agent*	L01	Hyb	H-Task*	H-Agent*	L01	Hyb	Couenne	Baron
5	10	2.372	2.454	4.747	1.622	2.236	2.507	2.465	1.302	9.782	2.361
	50	4.403	4.778	3.105	1.709	4.344	4.541	1.767	1.261	22.430	15.878
	100	4.218	4.044	2.681	1.417	4.104	3.793	1.532	1.143	27.758	15.869
	500	9.959	12.111	2.623	2.290	8.135	10.783	1.522	1.264	100.000	18.083
	1000	24.870	14.543	2.784	2.718	14.810	13.753	1.718	1.644	100.000	97.894
	avg.	9.164	7.586	3.188	1.951	6.726	7.076	1.801	1.323	51.994	30.017
10	10	2.931	2.744	4.608	1.361	2.468	2.929	2.689	1.071	8.986	2.391
	50	6.433	6.652	4.723	2.661	5.890	6.084	3.021	2.232	22.391	16.975
	100	6.130	6.382	3.814	2.605	5.923	6.323	2.364	1.853	68.134	15.713
	500	19.992	21.576	3.840	3.734	12.043	20.327	2.496	2.448	100.000	96.580
	1000	42.707	24.320	4.143	4.139	31.656	20.467	2.818	2.817	100.000	100.000
	avg.	15.639	12.335	4.226	2.900	11.596	11.226	2.677	2.084	59.902	46.332
20	10	2.807	3.125	4.200	1.202	2.585	3.360	2.443	1.016	12.455	3.858
	50	9.759	9.575	7.087	4.743	9.315	9.183	4.779	3.650	19.261	16.252
	100	11.367	14.013	5.555	4.817	9.250	10.707	3.763	3.234	100.000	16.019
	500	36.750	27.458	5.330	5.272	20.421	25.540	3.826	3.800	100.000	100.000
	1000	51.388	39.924	6.725	6.787	46.732	31.930	5.263	5.734	100.000	100.000
	avg.	22.414	18.819	5.779	4.564	17.660	16.144	4.015	3.487	66.343	47.226
overall avg.		15.739	12.913	4.398	3.138	11.994	11.482	2.831	2.298	59.413	41.192

Table 6: Percentage gap of the heuristics on instances with Linear weight function and Dissimilar capacities. Average values over 20 instances.

the percentage gap was computed as $100\,(U^* - L)/U^*$, where L is the value of the solution found by the heuristic or by the solver. Worth is mentioning that both nonlinear solvers showed numerical instability: in certain cases the returned upper bound value was wrong (worse than the value of a feasible solution found by the heuristics). For this reason, we only considered the value of the best solution returned by each solver.

The tables show that all the proposed heuristics clearly outperform both solvers which, in some cases, are even unable to find a feasible solution. Entries with gap equal (resp. close) to 100.000 indicate that no feasible solution was found for any (resp. most) of the 20 instances. Already with a 10 seconds time limit, all the proposed algorithms exhibit better performance than the solvers, with just two exceptions regarding Baron when $m = 5$, $n = 10$, Classes **NS** and **ND**. Among the heuristics, **H-Task*** is better than **H-Agent***. Both are dominated by **L01** and **Hyb**. The latter turns out to be the best approach for all time limits and benchmarks: It provides, within short time limits, results that are one order of magnitude better than those produced by the solvers in very large CPU times.

The results confirm that, even though all functions are non-decreasing, separable, and twice continuously differentiable, the NLGAP is very hard to solve in practice. Even very small instances with 5 agents and 10 tasks cannot be solved to optimality by state-of-the-art nonlinear solvers within 10 minutes. We also tried the same instances allowing one hour CPU time, and this did not change the picture: both Baron and Couenne hit the time limit and still produced suboptimal solutions, with non negligible gap with respect to U^*.

Between the two non-linear solvers, Baron outperforms Couenne by about 30% on average. The only exception occurs for Non-linear weight functions and Similar capacities (Table 3) for instances with a large number of variables ($n \cdot m \geq 5000$), for which Couenne gets better results by about 10%.

It may be noted that the performance of the lower bounds does not show the same similarity among the four instance classes that was observed for the upper bound computations. In particular, the approximation obtained for instances with Non-linear weight functions appears to be better than for instances with Linear weight functions (both Similar and Dissimilar capacities). Such behavior appears for all algorithms and all time limits. It appears quite surprising that non-linear weight functions are somehow "easier" to solve than linear ones. A possible explanation is that upper bound are less tight for the linear case, due to the fact that our methods, intended to solve a general non-linear problem, do not exploit special properties. Recall indeed that the linearizations we have proposed in Section 3 underestimate the weights on the samples and hence may be not fully appropriate for linear functions.

6 Conclusions

We have studied a continuous, non-linear variant of the well-known generalized assignment problem. The mathematical model of the problem has been used to obtain upper bounds and a heuristic algorithm. We have introduced greedy-type heuristics and a local search optimization. Such approaches and an overall hybrid heuristic have been computationally tested on a large benchmark of random instances, showing that they dominate non-linear solvers in terms of quality and computing time. All the proposed methods can be easily adapted to the case of mixed continuous and integer variables. Although, as mentioned in Section 1, the classical complexity analysis does not apply to general nonlinear problems, a future line of research could be to investigate the complexity of the NLGAP restricted to the rational domain.

Acknowledgments

This research was supported by Air Force Office of Scientific Research. It has also received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 764759.

References

[1] A. Caprara, H. Kellerer, and U. Pferschy. The Multiple Subset Sum Problem. *SIAM Journal on Optimization*, 11(2):308–319, 2000.

[2] Y. Chen, D. Ge, M. Wang, Z. Wang, Y. Ye, and H. Yin. Strong NP-hardness for sparse optimization with concave penalty functions. In D. Precup and Y.W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 740–747, 2017.

[3] Y. Chen, Y. Ye, and M. Wang. Approximation hardness for a class of sparse optimization problems. *Journal of Machine Learning Research*, 20(38):1–27, 2019.

[4] C. D'Ambrosio and S. Martello. Heuristic Algorithms for the General Nonlinear Separable Knapsack Problem. *Computers & Operations Research*, 38(2):505–513, 2011.

[5] C. D'Ambrosio, S. Martello, and L. Mencarelli. Relaxations and heuristics for the multiple non-linear separable knapsack problem. *Computers & Operations Research*, 93:79–89, 2018.

[6] R. Freling, H.E. Romeijn, D. Romero Morales, and A.P.M. Wagelmans. A branch and price algorithm for the multi-period single-sourcing problem. *Operations Research*, 51(6):922–939, 2003.

[7] P.M. Hahn, B.-J. Kim, M. Guignard, J. MacGregor Smith, and Y.-R. Zhu. An algorithm for the generalized quadratic assignment problem. *Computational Optimization and Applications*, 40(3):351–372, 2007.

[8] D.S. Hochbaum. Complexity and algorithms for convex network optimization and other nonlinear problems. *4OR: A Quarterly Journal of Operations Research*, 3:171–216, 2005.

[9] C.G. Lee and Z. Ma. The generalized quadratic assignment problem. Research Report, Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Ontario, M5S 3G8, Canada, 2004.

[10] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Chichester, New York: John Wiley & Sons, 1990.

[11] J.B. Mazzola. Generalized assignment with nonlinear capacity interaction. *Management Science*, 35(8):923–941, 1989.

[12] D.R. Morales and H.E. Romeijn. The generalized assignment problem and extensions. In Ding-Zhu Du and Panos M. Pardalos, editors, *Handbook of Combinatorial Optimization: Supplement Volume B*, pages 259–311. Springer US, Boston, MA, 2005.

[13] T.C. Sharkey and H.E. Romeijn. Greedy approaches for a class of nonlinear Generalized Assignment Problems. *Discrete Applied Mathematics*, 158(5):559–572, 2010.

[14] D.B. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461–474, 1993.

[15] V. Srivastava and F. Bullo. Knapsack problems with sigmoid utilities: Approximation algorithms via hybrid optimization. *European Journal of Operational Research*, 236(2):488–498, 2014.

[16] A.M. Tillmann. On the computational intractability of exact and approximate dictionary learning. *IEEE Signal Processing Letters*, 22(1):45–49, 2015.