An Application of Semi-Lagrangian Relaxation to the Generalised Assignment Problem

Thu H. Dang* Lauren Durrell† Adam N. Letchford*

To appear in *Operations Research Letters*

Abstract

The *Generalised Assignment Problem* is a fundamental and much-studied combinatorial optimisation problem, with many applications. We present a new exact algorithm for the problem, based on *semi-Lagrangian relaxation*. The algorithm managed to solve 3 instances that had previously remained unsolved, and yielded very strong lower bounds for many other instances. Oddly, however, the algorithm performed poorly for some instances.

Keywords: generalised assignment problem; Lagrangian relaxation; combinatorial optimisation

1 Introduction

The *Generalised Assignment Problem* (GAP), first studied in [24], is a fundamental problem in combinatorial optimisation. We have n jobs and m machines, and each job must be assigned to exactly one machine. Machine i has a positive capacity b_i. Processing job j on machine i costs c_{ij} and consumes a_{ij} units of capacity. The objective is to minimise the total cost.

The GAP has received much attention due to its wealth of applications (see [21]). The standard integer programming formulation of the GAP is as follows. For $i = 1, \ldots, m$ and $j = 1, \ldots, n$, let x_{ij} be a binary variable, taking the value 1 if and only if job j is assigned to machine i. We then have:

$$\min \quad \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} x_{ij} = 1 \qquad (j = 1, \ldots, n) \tag{2}$$

$$\sum_{j=1}^{n} a_{ij} x_{ij} \leq b_i \quad (i = 1, \ldots, m) \tag{3}$$

$$x \in \{0,1\}^{m \times n}. \tag{4}$$

*Department of Management Science, Lancaster University, Lancaster LA1 4YX, UK. E-mail: {T.H.Dang,A.N.Letchford}@lancaster.ac.uk

†STOR-i Centre for Doctoral Training, Lancaster University, Lancaster LA1 4YR, UK. E-mail: L.Durrell@lancaster.ac.uk

The constraints (2) ensure that each job is processed, and the constraints (3) ensure that the machine capacities are not exceeded.

It was shown in [26] that the GAP is NP-hard. In fact, it is strongly NP-hard even to check whether a feasible solution exists, as one can show by an easy reduction from the bin packing problem (see [12]). Nevertheless, some effective exact and heuristic algorithms have been proposed for the GAP (see the surveys [9, 19, 21] and Section 2 of this paper).

Many of the existing exact algorithms for the GAP use *Lagrangian relaxation* (LR) to compute lower bounds (e.g., [11, 15, 16, 20, 23, 24]). In [3], a variant of LR, called *semi-Lagrangian* relaxation (SLR), was introduced. For integer programs with a certain special structure, SLR can yield stronger bounds than LR. SLR has been applied with great success to several combinatorial optimisation problems (e.g., [2, 3, 4, 5, 31]).

In this paper, we present a new exact algorithm for the GAP, based on a combination of SLR with an ascent procedure and a primal heuristic. We also show that the algorithm is capable of solving some previously open benchmark instances.

We assume throughout that the reader knows the basics of integer programming, combinatorial optimisation and LR (see [29], [18] and [14], respectively). We also assume that the capacities b_i and consumption values a_{ij} are positive integers and that the costs c_{ij} are non-negative rationals. In addition, we assume for simplicity that any given GAP instance is feasible, i.e., that there exists some x satisfying (2)-(4).

We remark that some authors prefer to work with non-negative profits rather than non-negative costs (e.g., [7, 11, 19, 27]). Using the equations (2), it is easy to convert between the two forms [19].

The paper is structured as follows. In Section 2, we review the relevant literature. In Section 3, we present the new exact algorithm. In Section 4, we present the computational results. Finally, in Section 5, we make some concluding remarks.

2 Literature Review

We now briefly review the relevant literature. Subsection 2.1 covers applications of LR to the GAP, and 2.2 covers exact algorithms for the GAP. Subsection 2.3 is concerned with SLR.

2.1 Lagrangian approaches to the GAP

Consider the integer program (1)-(4). By solving its continuous relaxation, one can obtain a lower bound for the GAP. Unfortunately, this lower bound tends to be rather weak [19]. Several authors (e.g., [11, 15, 16, 19, 20, 23, 24]) have obtained tighter bounds using LR.

Suppose we relax the equations (2) in Lagrangian fashion, using a multiplier vector $\lambda \in \mathbb{Q}^m$. The resulting relaxed problem is to minimise

$$\sum_{j=1}^{n} \lambda_j + \sum_{i=1}^{m} \sum_{j=1}^{n} (c_{ij} - \lambda_j) x_{ij}$$

subject to (3) and (4). This problem decomposes into m independent subproblems, where the subproblem for machine i is:

$$\begin{array}{ll} \min & \sum_{j=1}^{n} (c_{ij} - \lambda_j) x_{ij} \\ & \sum_{j=1}^{n} a_{ij} x_{ij} \leq b_i \\ & x_{ij} \in \{0,1\} \qquad (j = 1, \ldots, m). \end{array}$$

Note that each subproblem is a knapsack problem. Although the knapsack problem is itself NP-hard [17], it can often be solve quickly in practice (e.g., [6]). Moreover, if $c_{ij} \geq \lambda_j$ for any pair (i, j), then the variable x_{ij} can be eliminated from the i-th subproblem.

To obtain a good lower bound, one must compute appropriate values for the multipliers. In [24], it is suggested to set λ_j to the second smallest c_{ij} value for each j. In [11, 15], iterative procedures were presented for adjusting the multipliers in such a way that the upper bound monotonically improves. In [20, 23], the subgradient method was used to compute near-optimal multipliers.

2.2 Exact algorithms for the GAP

In the papers mentioned in the previous subsection, the LR was embedded in a branch-and-bound scheme to solve the GAP exactly. The algorithms in [19, 23] were particularly sophisticated: the one in [19] used a combination of pre-processing, branching, primal heuristics and several upper-bounding procedures, and the one in [23] used information from the solution of the relaxed problem to fix many of the x variables to their optimal values, thereby reducing the size of the knapsack subproblems.

For completeness, we mention that some other methods, besides LR, have been used to solve the GAP exactly. This includes Lagrangian decomposition [16], branch-and-price [25, 27], and cutting planes [1, 7, 28]. Oddly, the authors of [1, 7] did not exploit the known polyhedral results on the GAP, such as those presented in [13].

At the time of writing, the most promising exact algorithms appear to be the ones in [1, 23]. Both algorithms are capable of routinely solving instances with tens of machines and a couple of hundred jobs (or even larger instances if the capacity constraints are loose).

2.3 Semi-Lagrangian relaxation

SLR was first introduced in [3], in the context of the p-median problem. For brevity, we show how it works only for a restricted family of integer programs. Consider an integer program of the form

$$\min\Big\{c^T x:\ Ax \le b,\ Dx = f,\ x \in \mathbb{Z}_+^n\Big\}, \tag{5}$$

where A, b, c, D and f are all non-negative. We split the equation system $Dx = f$ into two subsystems, one of the form $Dx \le f$ and the other of the form $Dx \ge f$. We then relax the second subsystem in Lagrangian fashion, yielding the following relaxed problem:

$$\lambda^T f + \min\Big\{(c^T - \lambda^T D)x:\ Ax \le b,\ Dx \le f,\ x \in \mathbb{Z}_+^n\Big\}.$$

If the multipliers are very large, this relaxed problem is just as hard to solve as the original. On the other hand, if the multipliers are 'reasonably small', the relaxed problem can be solved quite easily. To see why, consider the 'reduced cost' vector $\bar{c} = c - D^T\lambda$. If $\bar{c}_{ij} > 0$ for some pair (i, j), then x_{ij} must take the value 0 in any optimal solution to the relaxed problem. Thus, the relaxed problem can often be simplified substantially.

SLR has proved to be very useful for facility location problems (e.g., [3, 4, 5]) and assignment problems [2, 31]. On the other hand, as far as we know, no-one had applied SLR to the GAP prior to this paper.

3 Applying SLR to the GAP

In this section, we apply SLR to the GAP. In Subsection 3.1, we define our relaxation. In Subsection 3.2, we introduce some useful notation. In Subsection 3.3, we prove some results about the size of the multipliers needed to close the duality gap. Finally, in Subsection 3.4, we present a simple 'ascent' procedure to find optimal (or near-optimal) multipliers.

3.1 The relaxation

Consider once more the integer program (1)-(4). Comparing it with (5), we see that it is in exactly the right form for SLR to be applied. We just split the equations (2) into

$$\sum_{i=1}^{m} x_{ij} \le 1 \qquad (j = 1, \ldots, n)$$

and

$$\sum_{i=1}^{m} x_{ij} \ge 1 \qquad (j = 1, \ldots, n).$$

We then relax the latter inequalities in Lagrangian fashion. The resulting relaxed problem is to minimise

$$\sum_{j=1}^{n} \lambda_j + \sum_{i=1}^{m} \sum_{j=1}^{n} (c_{ij} - \lambda_j) x_{ij}$$

subject to

$$\begin{array}{ll} \sum_{i=1}^{m} x_{ij} \leq 1 & (j = 1, \ldots, n) \\ \sum_{j=1}^{n} a_{ij} x_{ij} \leq b_i & (i = 1, \ldots, m) \\ x \in \{0,1\}^{m \times n}. & \end{array}$$

We will call this relaxed problem $P(\lambda)$, and we will let $L(\lambda)$ denote the corresponding lower bound.

Unlike in the case of classical LR, $P(\lambda)$ is no longer guaranteed to decompose into m smaller subproblems. Nevertheless, it still holds that there exists an optimal solution to $P(\lambda)$ such that $x_{ij} = 0$ for all pairs (i, j) where $c_{ij} \geq \lambda_j$. Thus, when the multipliers are 'reasonably small', the relaxed problem should be relatively easy to solve.

3.2 Some notation

For what follows, we define some more notation. First, we let

$$z_\lambda(x) = \sum_{i=1}^{m} \sum_{j=1}^{n} (c_{ij} - \lambda_j) x_{ij}.$$

Note that, by definition, if x^* is an optimal solution to $P(\lambda)$, then $L(\lambda) = \sum_{j=1}^{n} \lambda_j + z_\lambda(x^*)$.

Next, we construct a certain bipartite graph, which we call G_λ. (A similar graph was used in [4] for a facility location problem.) There is a node in G_λ for each machine and each job. For each machine i and job j, we include the edge $\{i, j\}$ in G_λ if and only if $c_{ij} < \lambda_j$. Clearly, if we wish to solve $P(\lambda)$, we can delete the variable x_{ij} if the edge $\{i, j\}$ is not in G_λ.

Finally, we let c_j^k denote the k-th smallest value of c_{ij} for a given job j. We allow repeated values. For example, if $m = 3$, $c_{1j} = c_{3j} = 80$ and $c_{2j} = 50$, then $c_j^1 = 50$ and $c_j^2 = c_j^3 = 80$. Using this notation, the strategy used in [24] is to set λ_j to c_j^2 for all j.

3.3 Bounding the multipliers

Let us use the term 'duality gap' to denote the difference between $L(\lambda)$ and the optimal cost of the GAP. In this subsection, we examine how large the multipliers λ may need to be to close this gap completely. Our motivation

stems from Section 3 of [4], which shows that for a certain facility location problem, relatively small multipliers suffice to eliminate the duality gap.

In the case of the GAP, however, the situation is less favourable. One might expect that setting λ_j to c_j^m for all j would close the gap, but unfortunately, this is not the case. In fact, even setting all multipliers to much larger values may not suffice. Here is an example.

Example: Let M be a large positive integer. Suppose that $m = 3$, $n = 6$, $b = (100, 101, 102)$,

$$a = \begin{pmatrix} 50 & 50 & 49 & 52 & 48 & 54 \\ 50 & 50 & 49 & 52 & 48 & 54 \\ 50 & 50 & 49 & 52 & 48 & 54 \end{pmatrix} \text{ and } c = \begin{pmatrix} M & M & 1 & 1 & 1 & 1 \\ 1 & 1 & M & M & 1 & 1 \\ 1 & 1 & 1 & 1 & M & M \end{pmatrix}.$$

By construction, there is only one feasible solution to this GAP instance. We must schedule jobs 1 and 2 on machine 1, jobs 3 and 4 on machine 2, and jobs 5 and 6 on machine 3. The cost of this solution is $6M$. Now, $c_j^m = M$ for all j. Consider what happens if we set each multiplier to M. Let x^1 denote the optimal solution to the original GAP instance, and let x^2 denote the solution to $P(\lambda)$ in which jobs 1 and 2 are assigned to machine 2, jobs 3 and 4 are assigned to machine 3, job 5 is assigned to machine 1, and job 6 is not assigned to a machine. One can check that $z_\lambda(x^1) = 0$, whereas $z_\lambda(x^2) = 5 - 5M < 0$. One can also check that x^2 is an optimal solution to $P(\lambda)$. The corresponding lower bound is

$$L(\lambda) = \sum_{j=1}^{n} \lambda_j + z_\lambda(x^2) = 6M + (5 - 5M) = 5 + M.$$

Thus, there remains a significant duality gap even when all multipliers are set to c_j^m. In fact, to close the duality gap, one must set λ_j to $6M - 5$ for all j. We then have $z_\lambda(x^1) = z_\lambda(x^2) = 30 - 30M$ and the lower bound is $6M$ as desired. □

For each job j, let us define

$$\begin{aligned} \delta_j &= c_j^m - c_j^1 \\ U_j &= c_j^m + \sum_{j' \neq j} \delta'_j. \end{aligned}$$

In order to bound the size of the multipliers, we can use the following results.

Proposition 1 *Let ϵ be a small positive quantity. If we set λ_j to $U_j + \epsilon$, then job j will be assigned to a machine in any optimal solution to $P(\lambda)$.*

Proof. Consider a fixed job j and suppose that λ_j has been set to the stated value. Let x^1 be any feasible solution to $P(\lambda)$ that does not schedule

job j. We will construct a new feasible solution to $P(\lambda)$, called x^2, which schedules job j in addition to all of the jobs that were scheduled at x^1. To this end, let x^* be any optimal solution to the original GAP instance. Also define the following set for $k = 0, 1$:

$$S^k = \Big\{ j' \in \{1, \ldots, n\} : \sum_{i=1}^{m} x^1_{ij'} = k \Big\}.$$

(In other words, S^1 is the set of jobs that are scheduled by x^1, and S^0 is the set of jobs that are not.) We now construct x^2 as follows:

- for each machine i, and for each job $j' \in S^1 \cup \{j\}$, we set $x^2_{ij'}$ to x^*_{ij}.
- for each machine i, and for each job $j' \in S^0 \setminus \{j\}$, we set $x^2_{ij'}$ to 0.

One can check that x^2 is a feasible solution to $P(\lambda)$.

Now, for each job $j' \in S^1$, let $i(j')$ be the machine to which job j' was assigned by x^1. Similarly, for each job $j' \in S^1 \cup \{j\}$, let $i'(j')$ be the machine to which job j' was assigned by x^*. We have:

$$\begin{aligned}
z_\lambda(x^2) - z_\lambda(x^1) &= \sum_{i=1}^{m} \sum_{j'=1}^{n} \big(c_{ij'} - \lambda_{j'}\big) x^2_{ij'} - \sum_{i=1}^{m} \sum_{j'=1}^{n} \big(c_{ij'} - \lambda_{j'}\big) x^1_{ij'} \\
&= \sum_{i=1}^{m} \sum_{j' \in S^1 \cup \{j\}} \big(c_{ij'} - \lambda_{j'}\big) x^2_{ij'} - \sum_{i=1}^{m} \sum_{j' \in S^1} \big(c_{ij'} - \lambda_{j'}\big) x^1_{ij'} \\
&= \big(c_{i'(j),j} - \lambda_j\big) + \sum_{i=1}^{m} \sum_{j' \in S^1} c_{ij'} x^2_{ij'} - \sum_{i=1}^{m} \sum_{j' \in S^1}^{n} c_{ij'} x^1_{ij'} \\
&= \big(c_{i'(j),j} - \lambda_j\big) + \sum_{j' \in S^1} c_{i'(j'),j'} - \sum_{j' \in S^1} c_{i(j'),j'} \\
&= \big(c_{i'(j),j} - \lambda_j\big) + \sum_{j' \in S^1} \big(c_{i'(j'),j'} - c_{i(j'),j'}\big) \\
&\leq \big(c^m_j - U_j - \epsilon\big) + \sum_{j' \in S^1} \delta_j \\
&< 0.
\end{aligned}$$

Therefore, any optimal solution to $P(\lambda)$ will schedule job j. □

Proposition 2 *There exists a multiplier vector $\lambda \in \mathbb{Q}^n_+$ that closes the duality gap and satisfies $c^1_j \leq \lambda_j \leq U_j$ for all j.*

Proof. It follows from the proof of the previous proposition that, if we set λ_j to U_j for all j, we will close the duality gap. Suppose that $\lambda_j < c^1_j$ for some job j. Suppose we are given a feasible solution to $P(\lambda)$ such that

$x_{ij} = 1$ for some machine i. If we change x_{ij} from 1 to 0, the cost of the solution decreases by $c_{ij} - \lambda_j \geq c_j^1 - \lambda_j > 0$. Thus, job j will not be scheduled in an optimal solution to $P(\lambda)$, and there will be a positive duality gap. $\square$

We remark that, in the above example, we have $\delta_j = M - 1$ and $U_j = 6M - 5$ for all j. So, the example represents the worst-case scenario, in which we have to set every multiplier to U_j simultaneously, if we wish to close the duality gap. Of course, the example is very artificial.

3.4 An ascent algorithm

We know from Proposition 2 that the dual search for each λ_j can be confined to the interval $[c_j^1, c_j^{m+1}]$, where $c_j^{m+1} = U_j$. We partition this interval into pairwise-disjoint elementary intervals $[c_j^1, c_j^2) \cup [c_j^2, c_j^3) \cup \ldots \cup [c_j^m, c_j^{m+1}) \cup [c_j^{m+1}, c_j^{m+1}]$, where the last interval contains only one value c_j^{m+1}. One can check that the graph G_λ remains unchanged as multipliers vary within any given elementary interval. Therefore, it suffices to restrict the dual search to a single representative point per elementary interval. Hence, for each job j and each $k \in \{1, 2, \ldots, m\}$, we select the representative point $c_j^k + \epsilon$ within the interval $[c_j^k, c_j^{k+1})$ for some fixed $\epsilon > 0$.

Our algorithm proceeds as follows. For each job j, we select an index $\ell(j) \in \{1, 2, \ldots, m\}$. We start the dual search at $\lambda_j = c_j^{\ell(j)} + \epsilon$. We then solve the relaxed problem $P(\lambda)$ and update the lower bound. If the optimal solution of $P(\lambda)$ is infeasible, meaning some jobs are not assigned to any machine, we update the multiplier of each unassigned job to the representative point of the next elementary interval. The process stops once the optimal solution of $P(\lambda)$ is feasible for all λ.

In our preliminary tests, we tried setting the initial $\ell(j)$ values to 2, following the suggestion in [24]. We found, however, that this made our algorithm unnecessarily slow. To improve computational efficiency, we adapt the method in [4]. First, solve the continuous relaxation of the IP (1)-(4), and let v^* be the vector of dual prices corresponding to the constraints (2). For each client j, we then set $\ell(j)$ to the largest integer such that $c_j^{\ell(j)}$ is closest to v_j^*. The scheme is detailed in Algorithm 1.

Note that, in theory, if G_λ is not connected, then the problem $P(\lambda)$ decomposes into smaller subproblems, with one subproblem for each connected component. However, with our choice of initial multipliers, G_λ had a single connected component for all instances that we examined.

4 Computational Experiments

In this section, we present some computational results. Subsection 4.1 describes the instances used, while Subsections 4.2 and 4.3 present the results

Algorithm 1: Lower-Bounding Procedure for the GAP

input : number of machines m, number of jobs n, assignment costs c_{ij}, resource consumptions a_{ij}, machine capacities b_i, positive constant ϵ, time limit T

Solve the continuous relaxation of the IP (1)-(4);
Let v^* be the vector of dual prices for the constraints (2);
for *each job* j **do**
    Compute and store the c_j^k and U_j values;
    Set c_j^{m+1} to U_j;
    Let $\ell(j)$ be the largest index k such that the value c_j^k is closest to v_j^*;
    Let λ_j to $c_j^{\ell(j)} + \epsilon$;
end
Initialise the lower bound to 0;
while *time limit* T *is not exceeded* **do**
    Solve $P(\lambda)$ using any IP solver;
    Let x^* be the solution of the relaxed problem;
    Update the lower bound;
    for *each job* j **do**
        if *job* j *is not currently assigned to a machine* **then**
            Increase $\ell(j)$ to $\min\{\ell(j), m\} + 1$;
            Increase λ_j to $\min\{c_j^{\ell(j)} + \epsilon, U_j\}$;
        end
    end
    if *all jobs are assigned to a machine* **then**
        Stop and output the optimal GAP solution;
    end
end

output : Lower bound or optimal solution

for instances that we call 'easy' and 'large', respectively.

Our algorithm, as outlined above, was implemented in C#, compiled with Visual Studio 2022, and run on a 2.4GHz Intel i5-1135G7 processor with 16GB of RAM under Windows 11. To solve the LPs, we used the simplex solver of CPLEX v. 22.1, with default settings. The time limit T for Algorithm 1 was set to 3 hours. Our detailed results will be made available at the Lancaster University Data Repository, under the heading 'Semi-Lagrangian Relaxation for the Generalised Assignment Problem'[1].

4.1 Test instances

For our experiments, we used an instance set that has been previously used in [22, 8] and can be obtained in the OR-library[2]. We call the instances in this set 'easy'. According to [10], these instances were generated as follows.

[1] `http://www.research.lancs.ac.uk/portal/en/datasets/search.html`
[2] `https://people.brunel.ac.uk/~mastjjb/jeb/orlib/gapinfo.html`

The number of machines m was selected from the set $\{5, 8, 10\}$. The number of jobs n was set as a multiple ρ of the number of machines m, where $\rho \in \{3, 4, 5, 6\}$. For each combination of m and n, five variants were created, resulting in a total of 60 instances. Resource consumptions a_{ij} were random integers between 5 and 25. Costs c_{ij} were random integers between 15 and 25. Capacities b_i were set to $0.8 \sum_{j \in J} r_{ij}/m$.

In addition to the 'easy' instances, our experiments also include the set of 57 'large' instances proposed by Chu and Beasley [10]. These instances are categorised into five types: A, B, C, D, and E. The number of machines m is selected from the set $\{5, 10, 15, 20, 30, 40, 60, 80\}$, and the number of jobs n is selected from $\{100, 200, 400, 900, 1600\}$. The instance names are formed by the instance type followed by the number of machines and then the number of jobs. For example, the instance named 'a05100' represents a type A instance with 5 machines and 100 jobs. As described in [30], each type differs in the calculation of resource consumptions, costs, and capacities, as summarised in Table 1.

Table 1: Parameter settings for 'large' instances

Type	a_{ij}	c_{ij}	b_i
A	$\{5, 6, \ldots, 25\}$	$\{10, 11, \ldots, 50\}$	$9n/m + 0.4 \max_{i \in I} \sum_{j \in J, c_{i'j} = c_j^1} a_{i'j}$
B	$\{5, 6, \ldots, 25\}$	$\{10, 11, \ldots, 50\}$	$6.3n/m + 0.28 \max_{i \in I} \sum_{j \in J, c_{i'j} = c_j^1} a_{i'j}$
C	$\{5, 6, \ldots, 25\}$	$\{10, 11, \ldots, 50\}$	$0.8 \sum_{j \in J} a_{ij}/m$
D	$\{1, 2, \ldots, 100\}$	$111 - a_{ij} + e$, $e \in \{-10, -9, \ldots, 10\}$	$0.8 \sum_{j \in J} a_{ij}/m$
E	$1 - 10 \ln(e)$, $e \sim U(0, 1]$	$1000/a_{ij} - 10e$, $e \sim U[0, 1]$	$0.8 \sum_{j \in J} a_{ij}/m$

4.2 Results for easy instances

Table 2 presents the results for the easy instances. For each combination of m and n, the table shows the instance group name ('Name'), m, n, the mean percentage of remaining edges in G_λ in the last iteration ('%E'), the mean number of iterations of the 'while' loop in Algorithm 1, and the mean running time, measured in seconds.

Our algorithm solves all easy instances to proven optimality. As the optimal solutions are already known for these instances, we omit them here for brevity. We observed that smaller instances generally required fewer iterations to solve, often completing in a single iteration. Overall, the average number of iterations is very low. This suggests that the initialisation of the multipliers is highly effective for these instances.

Additionally, across all easy instances, the percentage of edges remaining ranged from 39% to 97%, with smaller instances generally involving a higher proportion of edges. A closer examination of the results showed that for very small instances ('gap1' to 'gap8'), the multiplier levels $\ell(j)$ in the final iteration were very high, with several set to $m+1$ (i.e., the corresponding multiplier is set to U_j), resulting in a nearly complete graph G_λ. In contrast, for larger instances, the values of $\ell(j)$ in the final iteration were typically smaller, producing a sparser G_λ graph. However, even then, G_λ still contained a significant proportion of the edges.

Name	m	n	%E	Iter	Time(s)
gap1	5	15	89	1.8	0.06
gap2	5	20	88	2.0	0.08
gap3	5	25	87	1.0	0.06
gap4	5	30	92	1.4	0.09
gap5	8	24	60	1.4	0.06
gap6	8	32	60	1.2	0.07
gap7	8	40	66	3.6	0.21
gap8	8	48	76	2.6	0.24
gap9	10	30	56	4.8	0.28
gap10	10	40	53	4.8	0.30
gap11	10	50	44	1.8	0.11
gap12	10	60	50	3.4	0.36

Table 2: Average results on 'easy' instances

4.3 Results for large instances

Table 3 shows the results for the 'large' instances. The first three columns show the instance name, the best-known lower bound from [30, 23, 25] ('L_1') and our computed lower bound ('L_2'), respectively. The remaining columns correspond to those in Table 2. Values of L_1 in bold indicate that the prior lower bound is known to be optimal. Values of L_2 in bold indicate that our lower bound is proven optimal, while italicised L_2 values denote improvements over the prior best-known bounds.

Regarding the quality of the bounds, we observe that for all but two of the instances where the optimal solution was already known, our approach is able to match it (44 out of 57 instances). We also solve to optimality 3 new instances (c401600, d20100 and d20400).

Regarding running time, all type D problems reached the full time limit except for 'd05100', 'd05200', and 'd10100', while all type E instances except 'e201600' were solved within seconds. In some cases, the running time slightly exceeded the specified time limit, likely due to internal CPLEX

Table 3: Performance results on 'large' instances

Name	L_1	L_2	%E	Iter	Time(s)	Name	L_1	L_2	%E	Iter	Time(s)
a05100	**1,698**	**1,698**	63	1	0.04	b05100	**1,843**	**1,843**	93	4	0.44
a05200	**3,235**	**3,235**	62	1	0.04	b05200	**3,552**	**3,552**	85	5	1.90
a10100	**1,360**	**1,360**	32	1	0.04	b10100	**1,407**	**1,407**	33	1	0.09
a10200	**2,623**	**2,623**	30	1	0.04	b10200	**2,827**	**2,827**	46	4	3.46
a20100	**1,158**	**1,158**	15	1	0.04	b20100	**1,166**	**1,166**	18	1	0.11
a20200	**2,339**	**2,339**	15	1	0.06	b20200	**2,339**	**2,339**	17	8	0.97
c05100	**1,931**	**1,931**	88	3	0.4	d05100	**6,353**	**6,353**	100	1	15.8
c05200	**3,456**	**3,456**	85	1	0.4	d05200	**12,742**	**12,742**	100	1	2,195.3
c10100	**1,402**	**1,402**	49	2	0.5	d10100	**6,347**	**6,347**	99	1	2,558.3
c10200	**2,806**	**2,806**	45	6	5.8	d10200	**12,430**	**12,430**	99	1	10,802.8
c10400	**5,597**	**5,597**	47	1	4.1	d10400	**24,961**	24,959	99	1	10,801.1
c15900	**11,340**	**11,340**	29	1	801.2	d15900	**55,404**	55,401	97	1	10,835.3
c20100	**1,243**	**1,243**	24	14	2.0	d20100	6,185	***6,190***	97	1	10,862.4
c20200	**2,391**	**2,391**	22	5	3.5	d20200	12,235	12,227	97	1	10,905.7
c20400	**4,782**	**4,782**	23	9	504.9	d20400	24,563	***24,574***	96	1	10,881.2
c30900	**9,982**	**9,982**	15	2	12,370.8	d30900	54,834	54,831	95	1	10,872.0
c40400	**4,244**	**4,244**	10	5	32.1	d40400	24,350	24,349	94	1	10,862.5
c60900	9,326	9,326	8	1	10,808.0	d60900	54,551	54,551	93	1	10,828.5
c201600	**18,802**	**18,802**	21	1	11,328.8	d201600	97,824	97,822	96	1	10,893.8
c401600	17,145	**17,145**	11	1	10,908.3	d401600	**97,105**	**97,105**	94	1	10,811.4
c801600	16,284	16,283	6	1	10,960.7	d801600	97,034	97,034	93	1	10,836.7
e05100	**12,681**	**12,681**	90	2	1.3	e20400	**44,877**	**44,877**	73	1	5.5
e05200	**24,930**	**24,930**	90	3	1.0	e30900	**100,427**	**100,427**	71	1	9.9
e10100	**11,577**	**11,577**	78	2	3.3	e40400	**44,561**	**44,561**	70	1	187.8
e10200	**23,307**	**23,307**	79	1	1.9	e60900	**100,149**	**100,149**	69	1	710.0
e10400	**45,746**	**45,746**	79	1	2.5	e201600	**180,645**	**180,645**	72	2	10,859.1
e15900	**102,421**	**102,421**	74	1	4.2	e401600	**178,293**	**178,293**	70	1	244.0
e20100	**8,436**	**8,436**	65	1	4.2	e801600	**176,820**	**176,820**	69	1	1,753.2
e20200	**22,379**	**22,379**	73	1	2.7						

processes.

The percentage of edges in the last iteration of our algorithm ranged widely from 6% to 100%, and was extremely high for the type D instances. A closer examination of the output for the type D instances revealed that the multiplier levels $\ell(j)$ were already surprisingly high even in the initial iteration. This indicates that the assignment constraints (2) had relatively large dual prices when the LP relaxation was solved. We do not have a convincing explanation for this phenomenon.

On the other hand, some instances, such as 'c401600' and 'c60900', used a substantially smaller proportion of edges, while either achieving proven optimality or improving the best known lower bounds. This demonstrates that SLR can lead to significant benefits in specific cases.

5 Conclusion

In this study, we proposed an exact method for the GAP based on SLR combined with dual ascent. We also proved that it is sufficient for the multipliers to fall within a specific interval in order to close the duality gap. The computational results show that our algorithm performs quite well, enabling us to improve existing lower bounds and/or prove optimality for some large instances.

The performance is however not consistent across all instances. For some instances, our algorithm significantly reduced the problem size even in the final iteration, highlighting the potential of the method. For other instances, particularly the 'type D' instances, the resulting problem remained nearly as large as the original.

Future work could be aimed at developing more effective reduction techniques, that progressively reduce the problem size over the iterations. In addition, alternative strategies for setting the initial multipliers and/or improving the multiplier update process could help avoid generating overly dense graphs, thereby improving the applicability of the method.

Acknowledgement: The second author gratefully acknowledges financial support from the EPSRC, through the STOR-i Centre for Doctoral Training, under grant "STOR-i 3" (EP/S022252/1).

References

[1] P. Avella, M. Boccia, and I. Vasilyev. A computational study of exact knapsack separation for the generalized assignment problem. *Comput. Optim. Appl.*, 45:543–555, 2010.

[2] I. Belik and K. Jörnsten. A new semi-Lagrangean relaxation for the k-cardinality assignment problem. *J. Infor. Optim. Sci.*, 37:75–100, 2016.

[3] C. Beltrán-Royo, C. Tadonki, and J.-P. Vial. Solving the p-median problem with a semi-Lagrangian relaxation. *Comput. Optim. Appl.*, 35:239–260, 2006.

[4] C. Beltrán-Royo, J.-P. Vial, and A. Alonso-Ayuso. Semi-Lagrangian relaxation applied to the uncapacitated facility location problem. *Comput. Optim. Appl.*, 51:387–409, 2012.

[5] X. Cabezas and S. García. A semi-Lagrangian relaxation heuristic algorithm for the simple plant location problem with order. *J. Oper. Res. Soc.*, 74:2391–2402, 2023.

[6] V. Cacchiani, M. Iori, A. Locatelli, and S. Martello. Knapsack problems—an overview of recent advances. Part I: single knapsack problems. *Comput. Oper. Res.*, 143, 2022. Article 105692.

[7] D.G. Cattrysse, Z. Degraeve, and J. Tistaert. Solving the generalized assignment problem using polyhedral results. *Eur. J. Oper. Res.*, 108:618–628, 1998.

[8] D.G Cattrysse, M. Salomon, and L.N. Van Wassenhove. A set partitioning heuristic for the generalized assignment problem. *Eur. J. Oper. Res.*, 72:167–174, 1994.

[9] D.G. Cattrysse and L.N. Van Wassenhove. A survey of algorithms for the generalized assignment problem. *Eur. J. Oper. Res.*, 60:260–272, 1992.

[10] P.C. Chu and J.E. Beasley. A genetic algorithm for the generalised assignment problem. *Comput. Oper. Res.*, 24:17–23, 1997.

[11] M.L. Fisher, R. Jaikumar, and L.N. Van Wassenhove. A multiplier adjustment method for the generalized assignment problem. *Manag. Sci.*, 32:1095–1103, 1986.

[12] M.R. Garey and D.S. Johnson. *Computers and Intractability.* Freeman, San Francisco, 1979.

[13] E.S. Gottlieb and M.R. Rao. The generalized assignment problem: valid inequalities and facets. *Math. Program.*, 46:31–52, 1990.

[14] M. Guignard. Lagrangian relaxation. *Trabajos de Operativa (TOP)*, 11:151–228, 2003.

[15] M. Guignard and M.B. Rosenwein. An improved dual based algorithm for the generalized assignment problem. *Oper. Res.*, 37:658–663, 1989.

[16] K. Jörnsten and M. Näsberg. A new Lagrangian relaxation approach to the generalized assignment problem. *Eur. J. Oper. Res.*, 27:313–323, 1986.

[17] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum, New York, 1972.

[18] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms.* Springer, Berlin, 6 edition, 2018.

[19] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations.* Wiley, Chichester, 1990.

[20] R.M. Nauss. Solving the generalized assignment problem: an optimizing and heuristic approach. *INFORMS J. Comput.*, 15:249–266, 2003.

[21] T. Öncan. A survey of the generalized assignment problem and its applications. *INFOR*, 45:123–141, 2007.

[22] I.H. Osman. Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches. *Oper. Res. Spektrum*, 17:211–225, 1995.

[23] M. Posta, J.A. Ferland, and P. Michelon. An exact method with variable fixing for solving the generalized assignment problem. *Comput. Optim. Appl.*, 52:629–644, 2012.

[24] G.T. Ross and R.M. Soland. A branch and bound algorithm for the generalized assignment problem. *Math. Program.*, 8:91–103, 1975.

[25] R. Sadykov, F. Vanderbeck, A. Pessoa, I. Tahiri, and E. Uchoa. Primal heuristics for branch and price: the assets of diving methods. *INFORMS J. Comput.*, 31:251–267, 2019.

[26] S. Sahni and T. Gonzalez. P-complete approximation problems. *J. ACM*, 23:555–565, 1976.

[27] M.W.P. Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Oper. Res.*, 45:831–841, 1997.

[28] M.A. Trick. A linear relaxation heuristic for the generalized assignment problem. *Nav. Res. Logist.*, 39:137–151, 1992.

[29] L.A. Wolsey. *Integer Programming.* Wiley, New York, 2nd edition, 2020.

[30] M. Yagiura, T. Ibaraki, and F. Glover. A path relinking approach with ejection chains for the generalized assignment problem. *Eur. J. Oper. Res.*, 169:548–569, 2006.

[31] H. Zhang et al. Solution to the quadratic assignment problem using semi-Lagrangian relaxation. *J. Syst. Eng. Electron.*, 27:1063–1072, 2016.