Formulations and Solution Algorithms for a Dynamic Generalized Assignment Problem

Luigi Moccia
Jean-François Cordeau
Maria Flavia Monaco
Marcello Sammarra

June 2007

CIRRELT-2007-16

Formulations and Solution Algorithms for a Dynamic Generalized Assignment Problem

Luigi Moccia[1,2], Jean-François Cordeau[3,*], Maria Flavia Monaco[1], Marcello Sammarra[1]

1. Dipartimento di Elettronica, Informatica e Sistemistica, Università della Calabria, via P. Bucci 41C, 87036 Rende (CS), Italy
2. ENEA, Ente per le Nuove Tecnologie, l'Energia e l'Ambiente, C.R. Trisaia, s.s. 106 Jonica km 419+500, 75026 Rotondella (MT), Italy
3. Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT), Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7 and Canada Research Chair in Risk Management, HEC Montréal, 3000, chemin de la Côte-Sainte-Catherine Montréal, Canada H3T 2A7

Abstract. This paper studies a Dynamic Generalized Assignment Problem (DGAP) which extends the well-known generalized assignment problem by considering a discretized time horizon and by associating a starting time and a finishing time with each task. Additional constraints related to warehouse and yard management applications are also considered. Two linear integer programming formulations of the problem are introduced. The strongest one models the problem as an origin-destination integer multi-commodity flow problem with side constraints. This model can be solved quickly for instances of small to moderate size. However, because of its computer memory requirements, it becomes impractical for larger instances. Hence, a column generation algorithm is used to compute lower bounds by solving the LP relaxation of the problem. This column generation algorithm is also embedded in a heuristic aimed at finding feasible integer solutions. Computational experiments on large-scale instances show the effectiveness of the proposed approach.

Keywords. Generalized assignment problem, dynamic problem, column generation, warehouse management, yard management.

Acknowledgements. This work was conducted in part while the first author was visiting the Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT). The work was supported by the Italian Ministero dell'Istruzione, dell'Università e della Ricerca Scientifica under the project PROMIS-Logistic Process Management and Intelligence System. We are also grateful to Gilbert Laporte for fruitful discussions.

---

* Corresponding author: jean-françois.cordeau@hec.ca

Dépôt légal – Bibliothèque nationale du Québec,
Bibliothèque nationale du Canada, 2007

1 Introduction

In the *Generalized Assignment Problem* (GAP) weighted tasks (or objects) must be assigned to capacitated agents (or facilities) in such a way that the sum of assignment costs is minimized and the total weight of the tasks assigned to any given facility does not exceed its capacity. The *Dynamic GAP* (DGAP) generalizes the GAP by considering a discretized time horizon and by associating a starting and a finishing time to each task in this horizon. The problem is then to find a minimum cost assignment of tasks to facilities for each period of the planning horizon. The problem is dynamic because the set of tasks changes from period to period and tasks can also be reallocated.

The DGAP has applications in the management of warehouses and storage yards, where a task can be regarded as a group of items to be stored and a facility represents a storage area or position. In these contexts, allowing dynamic reallocations often leads to a significant improvement in space utilization. Reallocations can also be motivated by operational considerations. For example, areas closer to arrival and departure gates are often used for temporary storage to accelerate loading and unloading operations. These areas will therefore face higher storage requests which, due to capacity constraints, will force some groups of items to be reallocated.

Of course, managerial practice tends to limit the number of times that any given item is handled and we will thus consider a task-specific bound on the number of reallocations performed. In fact, fragile items tend not to be reallocated, while others may not be subject to any particular constraint. In addition, we assume here that a group of items belonging to a common task cannot be divided between different storage areas. This assumption, however, makes the DGAP more difficult to solve.

The GAP is a well-known problem whose structure appears as a subproblem in many combinatorial optimization problems. As a consequence, it has been the object of a very large number of studies (for a survey see, e.g., Martello and Toth (1992)).

To the best of our knowledge, the DGAP introduced in this paper has not been studied before. It is, however, closely related to the literature concerning storage assignment for automatic warehouse systems. For example, an implicit assumption in the DGAP is the use of a *shared storage policy* (as opposed to a dedicated storage policy) whose merits were discussed in the seminal paper of Goetschalkx and Ratliff (1990).

Kogan and Shtub (1997) have introduced a related problem, also called a Dynamic Generalized Assignment Problem, in which each task has a due date. An inventory cost and a shortage cost are incurred when a task is finished ahead of or after its due date, respectively. This problem is formulated as a continuous-time optimal control model.

Kim and Park (2003) have addressed the problem of temporarily assigning products to storage areas under a shared storage policy. Each product is characterized by arrival and departure times, a capacity request, and a storage and handling cost function which depends on the assigned storage area. The authors have first analyzed the case where reallocations are disallowed, i.e. a product does not change locations during the time it spends inside the storage system. Through a proper network representation the authors have shown that the problem can be formulated as a minimum cost multi-commodity flow problem with integrality constraints on the flows. A Lagrangean relaxation approach and a subgradient optimization procedure were then presented. The algorithm was also adapted to the more general case where reallocations are allowed.

Our problem is similar to that of Kim and Park (2003) but it also differs from it in a fundamental way: we consider all items in a given group (i.e., a task) to constitute an unbreakable unit of flow. Hence, items that belong to the same group cannot be assigned to different locations. In contrast, the model of Kim and Park (2003) allows splitting the flow of a given commodity in multiple paths as long as the flow on each path is integer. These two problems are equivalent only in the special case where all tasks are unit-sized. Our problem is thus closer to the *Origin-Destination Integer Multi-Commodity Flow Problem* (ODIMCFP) introduced by Barnhart et al. (2000) and in which the flow of a commodity (a group of items in our problem) must use a single path from origin to destination in a network. In fact, the DGAP can thus be seen as an ODIMCFP with side constraints related to storage capacity.

Like the ODIMCFP, the DGAP can be modeled with path variables and solved by column generation (for a survey on column generation see, e.g., Desaulniers et al. (2005)). Because of the special structure of our time-space network, however, the pricing problem becomes a *Weight-Constrained Shortest Path Problem* (WCSPP) (see Dumitrescu and Boland (2001, 2003)). Finally, we should mention that the case of split flows with side constraints can also be tackled effectively by column generation as explained by Holmberg and Yuan (2003).

The rest of the paper is organized as follows. In the next section, we define the DGAP formally and we introduce three mathematical formulations of the problem. We also prove a dominance result between two of these formulations. In Section 3 we describe column generation based algorithms. This is followed by computational experiments in Section 4, and by the conclusion.

2 Mathematical Models of the DGAP

We first introduce some notation in Section 2.1, followed by a mixed-integer formulation of the DGAP in Section 2.2. This formulation is based on assignment variables and it will be denoted by $\mathcal{F}_1$. In Section 2.4, we then introduce a network representation of the problem, leading to a node-arc formulation, $\mathcal{F}_2$, which is described in Section 2.5. In Section 2.6, we prove the dominance of $\mathcal{F}_2$ over $\mathcal{F}_1$ in terms of linear programming lower bounds. However, since $\mathcal{F}_2$ results in very large integer program, we devise a decomposition scheme using a path-based formulation, $\mathcal{F}_3$, which is finally presented in Section 2.7.

2.1 Notation

Let $T = \{1, ..., \overline{t}\}$ represent the discretized planning horizon and let $K = \{1, ..., \overline{k}\}$ be the set of tasks to be assigned during this planning horizon. We also denote by $M = \{1, ..., \overline{m}\}$ the set of storage locations, by O the set of possible arrival positions, and by D the set of possible departure positions for the tasks.

For each task $k \in K$ we then define the following input data:

- $a^k \in T,$ the arrival time;
- $b^k \in T,$ the departure time;
- $q^k,$ the space requirements;
- $o^k \in O,$ the position of the task upon its arrival;
- $d^k \in D,$ the position of the task upon its departure;
- $r^k,$ the maximum number of reallocations allowed;
- $c_a^k,$ the maximum travel time allowed between the arrival position and the first storage position of the task;
- $c_b^k,$ the maximum travel time allowed between the last storage position and the departure position of the task.

The input data pertaining to the spatial aspects of the DGAP are defined as follows:

- $c_{lm}^k,$ the travel time for task $k \in K$ between locations $l, m \in M$;
- $c_{o^k l},$ the travel time for task $k \in K$ between the arrival position, o^k, and $l \in M$;
- $c_{ld^k},$ the travel time for task $k \in K$ between $l \in M$ and the departure position, d^k;
- $Q_l,$ the storage capacity of location $l \in M$.

For notational convenience we also introduce the following sets:

- $T(k) = \{t \in T : a^k \leq t \leq b^k\}, \forall k \in K,$ i.e. the stay of task k;
- $T^-(k) = \{t \in T : a^k \leq t \leq b^k - 1\}, \forall k \in K$;
- $K(t) = \{k \in K : t \in T(k)\}, \forall t \in T,$ i.e. the tasks which are in the system at time t.

2.2 The DGAP model $\mathcal{F}_1$

The model uses the following decision variables:

- $z_{lt}^k \in \{0, 1\}, \forall k \in K, l \in M, t \in T(k)$, where $z_{lt}^k = 1$ if task k is assigned to position l in period t, and $z_{lt}^k = 0$ otherwise;
- $\alpha_{lmt}^k \geq 0, \forall k \in K, l, m \in M, t \in T^-(k)$, where $\alpha_{lmt}^k = 1$ if task k is assigned to position l in period t and to position m in period $t+1$, and $\alpha_{lmt}^k = 0$ otherwise.

With these variables the assignment based formulation $\mathcal{F}_1$ can be written as:

$$\min \sum_{k \in K} \left\{ \sum_{l \in M} (c_{o^k l}^k z_{la^k}^k + c_{ld^k}^k z_{lb^k}^k) + \sum_{l \in M} \sum_{m \in M} \sum_{t \in T^-(k)} c_{lm}^k \alpha_{lmt}^k \right\} \tag{1}$$

subject to

$$\sum_{l \in M} z_{lt}^k = 1 \qquad \forall k \in K, t \in T(k), \tag{2}$$

$$\sum_{k \in K(t)} q^k z_{lt}^k \leq Q_l \qquad \forall l \in M, t \in T, \tag{3}$$

$$\sum_{l \in M} c_{o^k l} z_{la^k}^k \leq c_a^k \qquad \forall k \in K, \tag{4}$$

$$\sum_{l \in M} c_{ld^k} z_{lb^k}^k \leq c_b^k \qquad \forall k \in K, \tag{5}$$

$$z_{lt}^k + z_{m(t+1)}^k - \alpha_{lmt}^k \leq 1 \qquad \forall k \in K, l \in M, m \in M, t \in T^-(k), \tag{6}$$

$$\sum_{l \in M} \sum_{m \in M \setminus \{l\}} \sum_{t \in T^-(k)} \alpha_{lmt}^k \leq r^k \qquad \forall k \in K, \tag{7}$$

$$\alpha_{lmt}^k \geq 0 \qquad \forall k \in K, l \in M, m \in M, t \in T^-(k), \tag{8}$$

$$z_{lt}^k \in \{0,\ 1\} \qquad \forall k \in K, l \in M, t \in T(k). \tag{9}$$

The objective function (1) minimizes the total handling cost which comprises, for each task $k \in K$, the handling from the origin position to the first location at the beginning of the planning horizon, the handling from the last position to the departure position, and the handling due to relocations during the planning horizon.

Constraints (2) ensure that each task is assigned to exactly one position for each time step during its stay. Constraints (3) guarantee that the total space required by the tasks assigned to a location in any time step does not exceed the location's capacity. Constraints on the maximum travel time between the arrival position and the first location, and between the last location and the departure position, are imposed through (4) and (5). Observe that these constraints could be imposed by simply removing from the model the variables $z^k_{la^k}$ for which $c_{o^kl} > c^k_a$ and $z^k_{lb^k}$ for which $c_{ld^k} > c^k_b$. However, we prefer to include them explicitly for the sake of clarity and consistency with the following sections. Finally, constraints (7) reflect the managerial practice of avoiding excessive reallocations and allow task-specific requirements. It is worth pointing out that in the presence of constraints (6) and (8) and nonnegative costs in objective function, variables α^k_{lmt} will assume binary values.

2.3 The DGAP is $\mathcal{NP}$-hard

Consider a particular case of the DGAP where reallocations are not allowed, i.e., set $r^k = 0, \forall k \in K$. The resulting problem is a generalized assignment problem if all tasks arrive and depart at the same time. Since the GAP is known to be $\mathcal{NP}$-hard, the DGAP is thus also $\mathcal{NP}$-hard.

2.4 Network representation

We now introduce a time-space network to represent the DGAP. We construct a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ as follows. For each location $m \in M$ and each time step $t \in T$, we define a position-time node (m, t) and we denote by I the set of all such nodes. We also define $I(t)$ as the set of position-time nodes associated with time step t, and we let $m(i) \in M, \forall i \in I$, denote the location corresponding to node i. A directed arc $(i, j) \in \mathcal{A}$ is then defined between nodes $i, j \in \mathcal{N}$ if and only if $i \in I(t)$ and $j \in I(t + 1)$, i.e., only time consecutive position-time nodes are linked by arcs. For each task we also add to the graph an origin and a destination node. For convenience we maintain the previous notation and the origin node of a task k is denoted by $o^k \in O$ while the destination node is $d^k \in D$. The node set $\mathcal{N}$ is thus the union of the three node subsets I, O and D. An origin node is linked to the position-time nodes that correspond to the the arrival time of the task. These arcs must also be compatible with constraints (4), i.e. an arc exists only if the travel time between the origin and the position satisfies the maximum allowed time. In summary, an arc $(o^k, j) \in \mathcal{A}$ exists if and only if $j \in I(a^k)$ and $c_{o^k m(j)} \leq c_a^k$. Similarly, a destination node d^k is linked with compatible position-time nodes. The task-dependent arc costs c_{ij}^k are equal to the task travel time between the positions $m(i)$ and $m(j)$. Likewise, a position capacity $Q_l, l \in M$ translates into a node capacity $Q_i, i \in I$, where $l = m(i)$.

The resulting graph is illustrated in Figure 1 where the horizontal axis represents the temporal dimension while the vertical axis represents the spatial one.

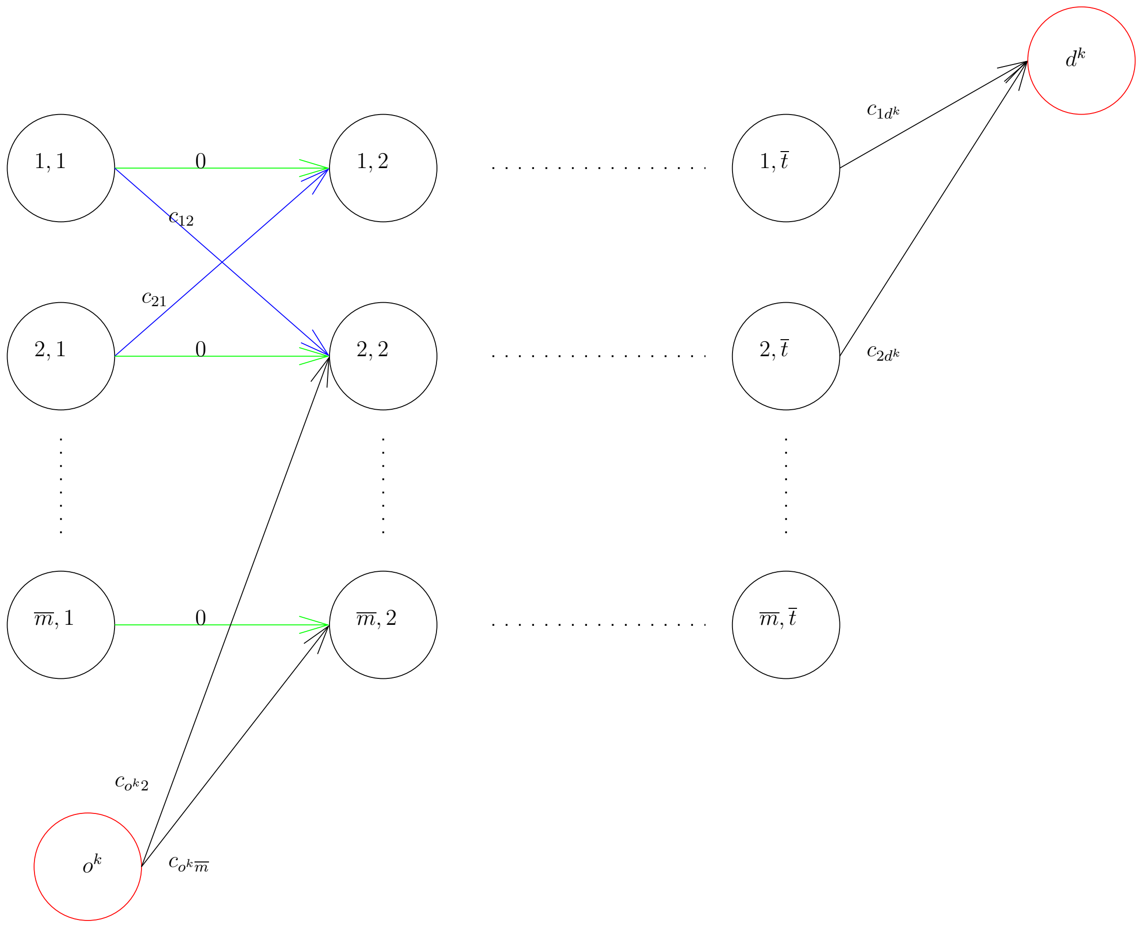

Figure 1: Directed graph to represent the DGAP as an ODIMCFP

This network representation results in an acyclic directed graph. This is a difference with respect to the time-space network representation introduced by Kim and Park (2003) and which results in a graph with cycles. As we noted earlier, our problem also differs from that studied by Kim and Park in the sense that we assume each a task, i.e. a group of items, to be an undivisible unit of flow.

2.5 Node-arc based formulation $\mathcal{F}_2$

We now model the DGAP as an ODIMCFP with side constraints over the graph $\mathcal{G}$ defined in the previous section. We note that constraints (4) and (5) are considered implicitly in the network definition, since no arcs exist between an origin (resp. a destination) node and infeasible positions at the arrival (resp. departure). Therefore, the formulation must handle position-time capacity constraints (3) and upper bounds on relocations (7).

The following additional notation is introduced to state the ODIMCFP formulation. For each node $i \in I$ the set $\delta(i)^+$ represents the nodes $j \in \mathcal{N}$ such that $(i, j) \in \mathcal{A}$. Similarly, $\delta(i)^-$ represents the nodes $j \in \mathcal{N}$ such that $(j, i) \in \mathcal{A}$. The position-time nodes linked with an origin node o^k belong to the set $\Omega(k)$, while the set of nodes linked to a destination node d^k is represented by $\Gamma(k)$. The arcs between position-time nodes related to different positions are grouped in the set $F \subset \mathcal{A}$, i.e., $(i, j) \in F$ if and only if $m(i) \neq m(j)$. This set F is used to compute the number of relocations.

The node-arc based DGAP formulation uses binary variables $x_{ij}^k, \forall (i, j) \in \mathcal{A}, \forall k \in K$, where $x_{ij}^k = 1$ if task k uses arc (i, j), and $x_{ij}^k = 0$ otherwise.

Using this notation, the node-arc formulation $\mathcal{F}_2$ can now be stated as follows:

$$\min \sum_{k \in K} \sum_{(i,j) \in \mathcal{A}} c_{ij}^k x_{ij}^k \tag{10}$$

subject to

$$\sum_{j\in\Omega(k)} x^k_{o^kj} = 1 \qquad \forall k \in K, \tag{11}$$

$$\sum_{i\in\Gamma(k)} x^k_{id^k} = 1 \qquad \forall k \in K, \tag{12}$$

$$\sum_{j\in\delta^+(i)} x^k_{ij} - \sum_{j\in\delta^-(i)} x^k_{ji} = 0 \qquad \forall i \in I, \forall k \in K, \tag{13}$$

$$\sum_{k\in K} q^k \sum_{j\in\delta^+(i)} x^k_{ij} \le Q_i \qquad \forall i \in I, \tag{14}$$

$$\sum_{(i,j)\in F} x^k_{ij} \le r^k \qquad \forall k \in K, \tag{15}$$

$$x^k_{ij} \in \{0,\, 1\} \qquad \forall (i,j) \in \mathcal{A}, k \in K. \tag{16}$$

The objective function (10) minimizes the total handling, as in $\mathcal{F}_1$. Constraints (11) and (12) define the degree of the origin and destination nodes, respectively. Flow conservation for the remaining nodes is ensured by constraints (13). Node capacities are enforced by constraints (14), while task-specific relocation bounds are imposed by constraints (15).

2.6 Dominance of $\mathcal{F}_2$ over $\mathcal{F}_1$

We now compare the two proposed formulations for the DGAP. Even though they are defined on variable spaces of different dimensions, it is possible to show the dominance of $\mathcal{F}_2$ over $\mathcal{F}_1$. Let us denote by P_1 and P_2 the polyhedra associated with $\mathcal{F}_1$ and $\mathcal{F}_2$, respectively:

$$P_1 = \left\{(z,\alpha) \in \Re^{g+h}_+ \mid (2) - (7) \text{ hold}\right\}$$

$$P_2 = \left\{x \in \Re^s_+ \mid (11) - (15) \text{ hold}\right\}.$$

For simplicity of notation, but whithout loss of generality in comparing $\mathcal{F}_1$ and $\mathcal{F}_2$, we assume that $T(k) = T, \forall k \in K$, and that the following conditions hold for every task k: $\max_{l\in M}\{c_{o^kl}\} \le c^k_a$ and $\max_{l\in M}\{c_{ld^k}\} \le c^k_b$. Then, constraints (4) and (5) can be omitted.

The number of variables in formulation $\mathcal{F}_1$ is easy to compute: there are $g = \overline{m} \cdot \overline{k} \cdot \overline{t}$ binary variables z and $h = \overline{m}^2 \cdot \overline{k} \cdot (\overline{t} - 1)$ continuous variables α. To calculate the number of variables in formulation $\mathcal{F}_2$, it is necessary to consider in detail the structure of the underlying graph $\mathcal{G}$. Recalling that $\mathcal{N} = I \cup O \cup D$, the set $\mathcal{A}$ is the union of three disjoint subsets:

$$I \times I = \{(i, j) \in \mathcal{A} \mid i \in I, j \in I\}$$

$$O \times I = \left\{(o^k, j) \in \mathcal{A} \mid o^k \in O, j \in I\right\}$$

$$I \times D = \left\{(i, d^k) \in \mathcal{A} \mid i \in I, d^k \in D\right\}.$$

Since $(i, j) \in I \times I$ if and only if $i \in I(t)$ and $j \in I(t + 1)$, we have:

$$|I \times I| = \overline{m}^2 \cdot (\overline{t} - 1).$$

Furthermore, each task $k \in K$ can use all arcs of this set. Therefore the number of variables corresponding to the subset $I \times I$ is $\overline{k} \cdot \overline{m}^2 \cdot (\overline{t} - 1)$. The arcs of the second and third groups are instead task-dependent; moreover, due to our simplifying assumption on the maximum allowed travel time at the arrival and at the departure of each task k, we have:

$$|O \times I| = |I \times D| = \overline{k} \cdot \overline{m}$$

and, finally, the number of binary variables in $\mathcal{F}_2$ is the following:

$$s = \overline{m}^2 \cdot \overline{k} \cdot (\overline{t} - 1) + 2\overline{k} \cdot \overline{m}.$$

The s variables x_{ij}^k in $\mathcal{F}_2$ are flow variables; when we consider the linear programming relaxation of the model, they keep their meaning, even though non-integer values are obviously allowed. In a similar way, when the integrality constraints are relaxed in $\mathcal{F}_1$, the value of the variable $z_{lt}^k \in [0, 1]$ is the fraction of the task k assigned to position l during the time step t (constraints (2) ensuring that, for each t, each task k is completely assigned), but something particular happens with the α variables. Looking at the objective function (1), α^k_{lmt} should be the fraction of task k assigned to position l in period t and at to position m in period $t+1$, since the associated coefficient in (1) is a relocation cost; but this may not be true. Actually, constraints (6), toghether with (8), correctly define the binary variables α for binary values of the variables z, since they give:

$$\alpha^k_{lmt} = \max\ \{0, z^k_{lt} + z^k_{m,t+1} - 1\} = \begin{cases} 1 & \text{if } z^k_{lt} = z^k_{m,t+1} = 1, \\ 0 & \text{otherwise.} \end{cases}$$

However, they are not able to capture the flow in the space (from l to m) and in the time (between t and $t+1$) that the same variables should represent in the continuous case. This is the reason why $\mathcal{F}_2$ is stronger than $\mathcal{F}_1$.

In order to establish the above claim in a more rigorous way, we introduce a linear transformation

$$L : \Re^s \longrightarrow \Re^{g+h}.$$

For all $x \in P_2$, the vector $L(x) = (z, \alpha)$ is defined as follows:

$$\begin{array}{lll}
z^k_{la^k} = x^k_{o^k j}, & j = (l, a^k) \in I, & \forall l \in M,\ k \in K \\
z^k_{lt} = \sum_{j \in \delta^+(i)} x^k_{ij}, & i = (l,t) \in I, & \forall l \in M,\ k \in K,\ t \in T \setminus \{a^k, b^k\} \\
z^k_{lb^k} = x^k_{id^k}, & i = (l, b^k) \in I, & \forall l \in M,\ k \in K \\
\alpha^k_{lmt} = x^k_{ij}, & i = (l,t),\ j = (m, t+1) \in I, & \forall l \in M,\ k \in K,\ t \in T \setminus \{\bar{t}\}.
\end{array} \tag{17}$$

The transformation L acts as the identity on the s flow variables of the formulation $\mathcal{F}_2$, corresponding to as many variables in $\mathcal{F}_1$, while it computes the remaining $(g+h) - s = (\bar{t}-2)\cdot\overline{k}\cdot\overline{m}$ components of the vector z as their linear combination. Actually, the variables z^k_{lt} for $a^k < t < b^k$ in formulation $\mathcal{F}_1$ can be interpreted as flows at the corresponding nodes $i \in I(t)$ in the graph $\mathcal{G}$, and they are thus redundant (we note, in passing, that they do not appear in the objective function (1)).

By suitably reordering the components of the involved vectors, we can define:

$$\hat{P}_1 = Proj_{\Re^s}(P_1) = \{(\hat{z}, \alpha) \mid (z, \alpha) \in P_1\} \tag{18}$$

to be the polyhedron associated with the formulation $\mathcal{F}_1$ in which the redundant variables z are removed, where $\hat{z} \in \Re^{2\overline{k}\overline{m}}$ is the subvector of z of components $z^k_{la^k}$ and $z^k_{lb^k}$ $\forall l \in M,\ k \in K$.

We are now in a position to state the following result:

Proposition 1 *$\mathcal{F}_2$ is stronger than $\mathcal{F}_1$.*

Proof — Let us denote by $\hat{L}$ the restriction of the linear transformation $L : \Re^s \to \Re^{g+h}$ defined by (17) on the image space $\Re^s \subset \Re^{g+h}$, or, equivalently, the composite function defined by:

$$\hat{L}(x) = Proj_{\Re^s}(L(x)) = (\hat{z}, \alpha) \quad \forall x \in \Re^s.$$

We will show that the image of the polyhedron P_2 under the linear transformation $\hat{L} : \Re^s \to \Re^s$ is strictly included in $\hat{P}_1$:

$$\hat{L}(P_2) \subset \hat{P}_1.$$

Since the formulations $\mathcal{F}_2$ and $\mathcal{F}_1$ share the same objective function, $f_2(x) = f_1(L(x))$, the thesis will follow.

We first prove that $\hat{L}(P_2) \subseteq \hat{P}_1$ by showing that $L(P_2) \subseteq P_1$. For all $x \in P_2$, $L(x) = (z, \alpha) \in P_1$. In fact constraints (2) for $t = a^k$ and $t = b^k$ correspond to constraints (11) and (12), respectively; constraints (2) for $t \in (a^k, b^k)$ follow from (11) and (13); constraints (3) are equivalent to (14); constraints (7) are equivalent to (15), while constraints (6) are satisfied as equalities for $\alpha^k_{lmt} = x^k_{ij},\ i = (l, t),\ j = (m, t+1)$, due again to continuity equations (13), (2) and the layered structure of the graph $\mathcal{G}$.

To prove that the inclusion of the polyhedra is tight, it is sufficient to exhibit a point $(z, \alpha) \in P_1$ whose projection on $\Re^s$, $(\hat{z}, \alpha)$, does not correspond to any point $x \in P_2$. We can construct a feasible solution for $\mathcal{F}_1$ by using the following procedure: first choose a vector $z \in \Re^g$ satisfying constraints (2) and (3); then compute the components of $\alpha \in \Re^h$ imposing constraints (6) and (7), in such a way that $(z, \alpha) \in P_1$. Note that, whenever $z_{lt}^k \in (0,1)$, $z_{m,t+1}^k \in (0,1)$ and $z_{lt}^k + z_{m,t+1}^k < 1$, constraints (6) and (8) will force $\alpha_{lmt}^k = 0$, while no feasible $x \in P_2$ could have a null value for the corresponding variable x_{ij}^k. □

We illustrate this possible situation with the following example reported in Figure 2. Let $\overline{m} = 6$ and, for a given time step t, $z_{lt}^k = \frac{1}{6}, \forall l \in M$. Suppose also that at time step $t+1$ only two nodes collect the flow from the previous layer. It is easy to see that the reallocation variables α_{lmt}^k will have null value for each (l, m) pair.

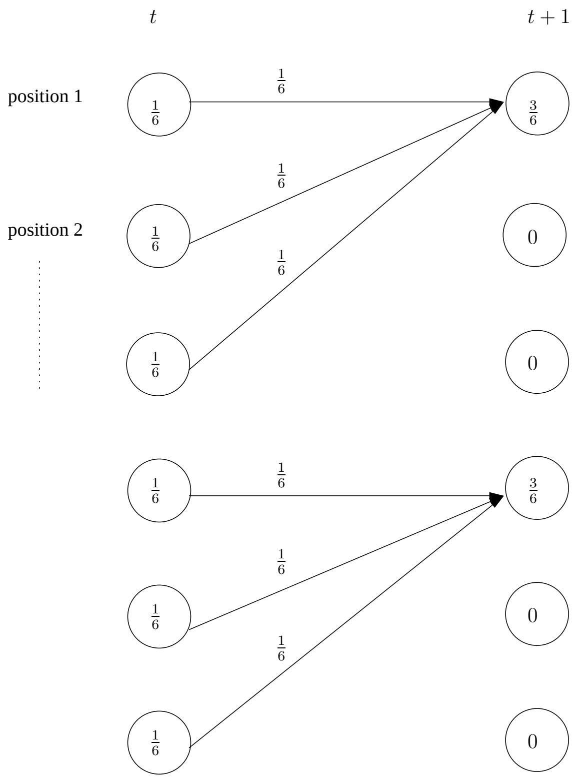

Figure 2: Example of flow with $\alpha_{lmt}^k = 0$

The above example also clearly shows why the bound obtained by solving the linear relaxation of $\mathcal{F}_1$ can be sensibly lower than that computed by using $\mathcal{F}_2$. Whenever the described conditions hold, $f_1(z, \alpha) < f_2(x)$, since in the first model some costs related to effective reallocations are ignored, due to the null value of the corresponding α variables.

2.7 Path based formulation $\mathcal{F}_3$

A path based formulation focuses on paths between the origin and the destination nodes over the graph $\mathcal{G}$. For each task $k \in K$, let $P(k)$ represent the set of all feasible paths on $\mathcal{G}$. Each path must start from the origin node o^k, finish at the destination node d^k, and satisfy constraints (7), i.e., it must use at most r^k relocation arcs. For each feasible path $p \in P(k)$, we now define a binary decision variable y_p^k, where $y_p^k = 1$ if and only if task k is assigned to the path p. Let $P = \bigcup_{k \in K} P(k)$. We also introduce the following notation:

- $\psi_{ij}^p, \forall (i,j) \in \mathcal{A}, \forall p \in P$, ψ_{ij}^p equals to one if the arc (i,j) belongs to the path p, and zero otherwise;
- $\beta_i^p, \forall i \in I, \forall p \in P$, β_i^p equals to one if the path p passes through node i, and zero otherwise.

With the above notation we can introduce specific path-based costs in compact form:

- $c_p^k = \sum_{(i,j) \in \mathcal{A}} c_{ij}^k \psi_{ij}^p, \forall p \in P, k \in K$, thus c_p^k indicates the cost of using the path p for the task k.

Hence, the path based formulation $\mathcal{F}_3$ is the following:

$$\min \sum_{k \in K} \sum_{p \in P(k)} c_p^k y_p^k \tag{19}$$

subject to

$$\sum_{k \in K} \sum_{p \in P(k)} q^k \beta_i^p y_p^k \le Q_i \qquad \forall i \in I, \tag{20}$$

$$\sum_{p \in P(k)} y_p^k = 1 \qquad \forall k \in K, \tag{21}$$

$$y_p^k \in \{0,\, 1\} \qquad \forall k \in K, p \in P(k). \tag{22}$$

Constraints (20) define the node related capacities. One must choose exactly one path for each task, as stated in constraints (21).

3 Column Generation Based Algorithms

The column generation approach looks for an optimal solution of a large Linear Program (LP) without explicitly including all columns, i.e. variables, in the constraint matrix. The full problem is called the *Master Problem* (MP), while the LP with only a subset of the MP columns is called the *Restricted Master Problem* (RMP). In most problems, only a very small subset of all columns will belong to an optimal solution, and all other (non basic) columns can be ignored. In a minimization problem all columns with positive reduced cost can in fact be ignored. The column generation algorithm finds an optimal solution to the MP by solving a series of several smaller RMPs. When the optimal solution of an RMP is found, the algorithm looks for columns not included in the RMP that have negative reduced costs. This is called the pricing problem. If no column of negative reduced cost can be found by the pricing routine then the current optimal solution of the RMP is also optimal for the MP. Otherwise, one or more columns with negative reduced costs are added to the RMP and the algorithm iterates.

We propose to use a column generation algorithm to solve the linear relaxation of the path-based formulation $\mathcal{F}_3$. To this end, we need to devise a pricing routine to find paths (i.e. columns) with negative reduced cost. Let $\pi_i \leq 0, \forall i \in I$, and $\sigma_k, \forall k \in K$, be dual multipliers associated with constraints (20) and (21), respectively.

The reduced cost of the variable y_p^k, denoted as $\bar{c}_p^k$, is:

$$\bar{c}_p^k = \sum_{(i,j)\in\mathcal{A}} c_{ij}^k \psi_{ij}^k - \sum_{i\in I} q^k \beta_i^p \pi_i - \sigma_k. \qquad (23)$$

We introduce modified arc costs, $\tilde{c}_{ij}^k$, over the graph $\mathcal{G}$, as follows:

$$\tilde{c}_{ij}^k = c_{ij}^k - \frac{q^k}{2}(\pi_i + \pi_j), \forall t \in T^-(k), i \in I(t), j \in I(t+1), \qquad (24)$$

$$\tilde{c}_{o^k j}^k = c_{o^k j}^k - \frac{q^k}{2}\pi_j, \forall j \in \Omega(k), \qquad (25)$$

$$\tilde{c}_{i,d^k}^k = c_{i,d^k}^k - \frac{q^k}{2}\pi_i, \forall i \in \Gamma(k). \qquad (26)$$

Given this modified cost structure, every path $p \in P(k)$ will have a cost $\tilde{c}_p^k$ equal to $\bar{c}_p^k + \sigma_k$. Therefore, in order to find a column with a negative reduced cost we look for a path p such that $\tilde{c}_p^k < \sigma_k$. This can be accomplished by solving for each task a *Weight-Constrained Shortest Path Problem* (WCSPP) over the graph $\mathcal{G}$ with the cost structure modified by the current dual variables. This WCSPP presents a special structure: it is defined over a layered graph, and the weights are zero or one. We denote this particular problem as the *r-Constrained Shortest Path Problem on Layered Graph* (r-CSPPLG). The r-CSPPLG is a polynomially solvable problem. The WCSPP in general is $\mathcal{NP}$-hard (Garey and Johnson, 1979). However pseudo-polynomial algorithms exist to solve it. The Label Setting Algorithm (LSA) of Desrochers and Soumis (1988) applied to the WCSPP has a complexity of $\mathcal{O}(|\mathcal{A}|W)$, where W is the right-hand side of the additional constraint (size of the knapsack). This result holds if the weights are larger than zero, while in the r-CSPPLG some of them are equal to zero. Nevertheless, as noted by Dumitrescu and Boland (2003), the LSA, even in the case of zero weights, presents pseudo-polynomial complexity of $\mathcal{O}(|\mathcal{N}|^2 W)$. It is easy to see that the LSA applied to the r-CSPPLG will result in a polynomial algorithm. In fact, only values of W, or r with our notation, less than $\bar{t} - 1$ make sense: for $r \geq \bar{t} - 1$ the knapsack constraint will never be limiting in the layered graph and the r-CSPPLG will correspond to an ordinary shortest path problem. Therefore, the worst case being $r = \bar{t} - 2$, the LSA complexity is $\mathcal{O}((\overline{m} \cdot \bar{t} + 2)^2(\bar{t} - 2)) = \mathcal{O}(\overline{m}^2 \cdot \bar{t}^3)$, proving that r-CSPPLG belongs to $\mathcal{P}$.

Another important aspect of the r-CSPPLG follows by studying the constraint matrix (11)-(13), and (15). Let us denote this matrix by A. It is possible to prove that A is not totally unimodular. In fact, a necessary and sufficient condition for total unimodularity (Nemhauser and Wolsey, 1988) requires that the elements of A belong to $\{-1, 0, +1\}$, and for any subset $\overline{A}$ of the rows of A there exists a partition $(\overline{A}_1, \overline{A}_2)$ of $\overline{A}$ such that each column satisfies $|\sum_{i \in \overline{A}_1} a_{ij} - \sum_{i \in \overline{A}_2} a_{ij}| \leq 1$, where i and j here express the row and column index, respectively, while a_{ij} is an entry of A. Based on that, the proof that this condition does not hold for the r-CSPPLG is straightforward and we omit it. Therefore, the vertices of the r-CSPPLG polytope defined by A are not always integer. An algebraic characterization of the fractional vertices of the r-CSPP polytope (i.e. without the assumption of a layered graph) is given in Monaco et al. (2007). These results are similar to the ones of Dahl and Realfsen (2000) for the Cardinality-Constrained Shortest Path Problem obtained by polyhedral analysis.

To summarize, the pricing problem belongs to $\mathcal{P}$ and some vertices of its polytope do not have integer coordinates. Since the pricing problem does not have the integrality property (Geoffrion, 1974), one can expect that solving the LP relaxation of formulation $\mathcal{F}_3$ will yield tighter lower bounds than solving that of formulation $\mathcal{F}_2$. Unfortunately, our computational experiments show that the linear relaxation of the r-CSPPLG results in integer solutions in the vast majority of the cases. As a result, the lower bounds obtained by solving the column generation formulation are not significantly larger. Since these lower bounds are probably too weak to be used within a branch-and-price algorithm, we have decided to develop heuristic algorithms to obtain feasible integer solutions.

3.1 Column generation scheme - COL0

The algorithm starts with an empty set of columns. To make the RMP feasible, artificial variables v^k with a large cost are thus added to model $\mathcal{F}_3$. Constraints (21) then become: $\sum_{p\in P(k)} y_p^k + v^k = 1, \forall k \in K$. For each task k, the pricing problem, i.e. the r-CSPPLG, is solved with the integer programming formulation implemented in a commercial solver (CPLEX). We use a threshold value of 10^{-3} to accept a column with a negative reduced cost, i.e., it must be $\tilde{c}_p^k \leq \sigma_k - 10^{-3}$. To accelerate the column generation process, we skip the pricing problem for a given task k if it has failed to generate a new column for ς iterations, where ς depends upon the instance size and is computed as $\varsigma = \lceil \lg(\overline{k} \times \overline{m} \times \overline{t}) - 2 \rceil$. This condition is of course removed at the end to prove optimality. This algorithm, which we call COL0, ends with the computation of the LP relaxation lower bound of $\mathcal{F}_3$.

3.2 COL1 heuristic

The COL0 algorithm usually terminates with a fractional solution, meaning that some tasks use more than one path. The COL1 algorithm is a very simple heuristic aimed at obtaining a feasible solution. It consists in performing a branch-and-bound search on the set of columns generated by COL0. Clearly, this set of columns is not always sufficient to produce a feasible integer solution because of the capacity constraints. This heuristic is introduced mostly to benchmark the COL2 heuristic presented in the next section.

3.3 COL2 heuristic

The COL2 heuristic tries to expand the set of columns generated by the COL0 algorithm in the hope of obtaining a feasible integer solution during the branch-and-bound search. To this end, the column generation algorithm is invoked iteratively on perturbed versions of the DGAP instance. The basic idea consists in reducing the capacities of the saturated nodes forcing the generation of new paths that avoid these nodes. These paths are likely to be useful to achieve integer feasibility.

The algorithm consists in five generating phases, the first one being COL0. At the end of the COL0, we look at nodes i with an associated dual value $\pi_i < 0$. We construct a new instance with capacities $Q_i^1, \forall i : \pi_i < 0$, reduced by a factor of $\frac{1}{3}$ with respect to the original instance, i.e. $Q_i^1 = \frac{2}{3}Q_i$. The second phase starts by invoking the column generation algorithm to solve this modified instance. The current set of columns is used to initialize the modified problem, thus achieving optimality with fewer iterations. A third generating phase is carried out by solving the linear relaxation of a new instance with reduced capacities set as follows $Q_i^2 = \frac{1}{3}Q_i$. Again only the nodes with negative dual value π_i are modified. A fourth phase introduces a modified instance where the node capacities Q_i^3 are reduced proportionally to their negative dual value π_i. Let $\hat{\pi}$ denote the smallest π_i, then, if $\pi_i < 0$, $Q_i^3 = (1 - \frac{\pi_i}{\hat{\pi}})Q_i$. Finally, the last generating phase sets to zero the capacities of nodes i such that $\pi_i \leq \overline{\pi}$, where $\overline{\pi}$ is the average of the negative π_i values. A branch-and-bound procedure is then applied over the extended set of columns.

4 Computational Results

In this section, we first describe our test instances. We then present results obtained with the previously described column generation based algorithms, and with a branch-and-bound procedure applied to the mixed integer linear programs $\mathcal{F}_1$ and $\mathcal{F}_2$.

4.1 Test instances

We have created four sets of ten randomly generated instances, denoted by I1, I2, I3 and I4. In the following we describe the procedure used to construct these random instances. In all instances, we have set $r^k = 2, \forall k \in K$. Smaller r^k values would result in too easy instances, while larger values would not make sense in the context of warehouse or yard management problems where excessive reallocations are usually avoided. In order to construct difficult instances we have eliminated constraints (4) and (5) in $\mathcal{F}_1$, making the problem less constrained. A constant value Q was chosen for the space requirement data, i.e. $Q_l = Q, \forall l \in M$. Then, for each instance three parameters were provided as input: $\overline{m}, \overline{t}$ and $f \in \{1, ..., 100\}$, where f determines the saturation target $\overline{Q}$ that the instance should approximate. This saturation target $\overline{Q}$ is computed as $\overline{m} \times Q \times f/100$.

The random generation of the tasks starts with an empty set K. Then a tentative task is generated according to discrete uniform distributions as follows: $q^k \in [1, ..., Q]$, $a^k \in [1, ..., \overline{t}]$, $b^k \in [a^k, ..., \overline{t}]$. The tentative task k is accepted if its space requirement q^k and its duration-of-stay $T(k)$ are such that the space requirements of the already generated tasks $h \in K$ with $T(h) \bigcap T(k) \neq \emptyset$ respect the following condition: $q^k + \sum_{h:t \in T(h)} q^h \leq \overline{Q}, \forall t \in T(k)$. If this condition does not hold the task is discarded and a new task is generated. The instance building procedure ends when the task generation fails 100 times, meaning that the saturation target is binding. The spatial attributes of an instance are constructed by placing the sectors equally spaced on a grid and filling a rectangle, while the arrival and departure positions are located on a line parallel to one side of the rectangle and at a distance from the rectangle of one unity of the chosen grid. As an example Figure 3 illustrates the layout used for the instance set I3. Arrival and departure positions are chosen randomly for each task. Then, the remaining data, c_{lm}^k, the travel time between two locations, are computed as proportional to the product of q^k and of the distances c_{lm} in L_1 norm.

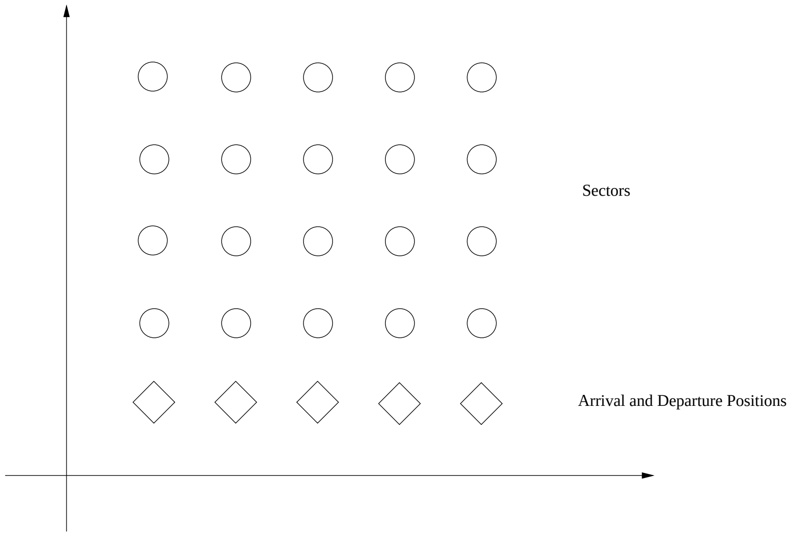

Figure 3: Layout used for the instance set I3

In Table 1 we report for each instance set the three input parameters $\overline{m}, \overline{t}, f$, and the resulting average number of tasks. We provide the number of tasks as an average since the random procedure usually gives different $\overline{k}$ values with the same input parameters. For convenience, the worst case number of variables and constraints of the two formulations $\mathcal{F}_1$ and $\mathcal{F}_2$ are also given in Table 1.

	I1	I2	I3	I4
$\overline{m}$	6	10	20	30
$\overline{t}$	8	10	10	10
f	85	85	75	75
Average $\overline{k}$	47	74	115	152
$\mathcal{F}_1$ variables (worst case)	14100	74000	437000	1276800
$\mathcal{F}_1$ constraints (worst case)	12409	67662	415695	1233476
$\mathcal{F}_2$ variables (worst case)	12408	68080	418600	1240320
$\mathcal{F}_2$ constraints (worst case)	2445	7722	23545	46356

Table 1: Input data for the instance generation procedure and instance dimensions.

4.2 Implementation details and results

The algorithms were implemented in CPLEX 10.0 with the C++ language. Computational experiments, when not reported differently, were run on a 2.5 GHz Pentium IV Linux workstation with 512 MB of memory. For all algorithms we have set a 30 minute time limit, eventually causing a truncated branch-and-bound procedure.

We have first compared the two formulations $\mathcal{F}_1$ and $\mathcal{F}_2$ on the instance set I1 (Table 2). As predicted in Section 2.6, $\mathcal{F}_2$ outperforms $\mathcal{F}_1$: $\mathcal{F}_2$ always obtains the optimal solution in a shorter CPU time. $\mathcal{F}_1$ is able to certify optimality only on one instance, I1-07, and reaches an optimal solution, but without closing the branch-and-bound tree, on instance I1-03. On the other I1 instances the performance of $\mathcal{F}_1$ is even worse. In view of that and considering that the instance set I1 is the easiest of the four sets, we have not considered the assignment based formulation in the remaining experiments.

We have then compared the branch-and-bound of CPLEX applied to formulation $\mathcal{F}_2$ with the COL2 heuristic. This heuristic has been developed having in mind the large DGAP instances with which formulation $\mathcal{F}_2$ is not applicable because of the required computer memory. The large instances are grouped in the set I4. Here $\mathcal{F}_2$ is not able to run on the reference machine with 512 MB of RAM. Therefore we have devised the following computational experiments summarized in Table 6: $\mathcal{F}_2$ is executed on a more powerful machine with 16 GB of RAM and an AMD Opteron 2.4 GHz CPU while COL2 is still performed on the reference machine. The same time limit of 30 minutes has been assigned to both algorithms, hence putting COL2 at a disadvantage. Nevertheless, in nine instances out of ten COL2 obtains better solutions than the truncated branch-and-bound procedure with $\mathcal{F}_2$. Furthermore, $\mathcal{F}_2$ fails at getting a feasible solution on one instance while COL2 always attains feasibility. Of course, the COL2 results are even better on the more powerful machine. We do not report these results but COL2 performs better on every instance. These experiments illustrate the rationale for the column generation heuristic COL2.

The performance of COL2 is further assessed in Tables 3, 4 and 5 which report results on instance sets I1, I2, and I3, respectively. On the instance set I1, as described above, $\mathcal{F}_2$ always obtains optimality. However, COL2 gets near-optimal solutions in a shorter computing time. A similar pattern occurs on the instance set I2, while, on the instance set I3, with comparable computational times, COL2 performs slightly better than $\mathcal{F}_2$ in terms of average solution quality. These results show the reliability of the heuristic. In practice, however, instances of the size of those in sets I1, I2, and I3 are better suited for the exact approach with a truncated branch-and-bound procedure applied to the integer linear programming formulation.

Tables 7 to 10 provide further details regarding the COL2 algorithm. The COL0 data allow us to assess the effectiveness of the column generation scheme. The COL1 results indicate when it was possible to construct a feasible integer solution with the columns (paths) generated by COL0. For COL2 we also provide additional information such as the CPU time required for the five column generation phases (i.e., COL0 plus the four column generation phases on the perturbed problems), and the total number of columns that are fed into the final branch-and-bound procedure. These results highlight the usefulness of the column generation phases embedded in COL2: on the larger instances obtaining feasibility with the simpler COL1 heuristic is improbable, while COL2 substantially expands the set of available paths. Furthermore, the additional computational burden of the COL2 heuristic is relatively small. As mentioned in Section 3, the column generation does not give a better lower bound. Table 11 shows average results upon the four instance sets. The variation between $\mathcal{F}_2$ lower bound and COL0 is negligible, less than 0.1%

5 Conclusions and Future Developments

We have presented a new combinatorial optimization problem, the DGAP, and compared different formulations and algorithms for solving this problem. On small to medium instances a branch-and-bound procedure applied to an origin-destination integer multicommodity flow formulation of the DGAP is highly effective. However, due to memory requirements, this approach becomes impractical for larger instances. To solve large-scale instances we have thus developed a column generation algorithm, COL0, that provides a lower bound. A heuristic algorithm, COL2, iteratively solves perturbed versions of the problem to expand the set of useful columns. Computational experiments show the effectiveness of this column generation based heuristic in terms of achieving feasibility. The development of meta-heuristic algorithms will be the object of a further study.

Acknowledgments

This work was conducted in part while the first author was visiting the Center for Research on Transportation in Montreal. The work was supported by the Italian Ministero dell'Istruzione, dell'Università e della Ricerca Scientifica under the project PROMIS - Logistic Process Management and Intelligence System. We are also grateful to Gilbert Laporte for fruitful discussions.

References

Barnhart, C., Hane, C. A., and Vance, P. H. (2000). Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Operations Research*, 48(2):318–326.

Dahl, G. and Realfsen, B. (2000). The cardinality-constrained shortest path problem in 2-graphs. *Networks*, 36(1):1–8.

Desaulniers, G., Desrosiers, J., and Solomon, M. M., editors (2005). *Column generation*. Springer, Berlin.

Desrochers, M. and Soumis, F. (1988). A generalized permanent labeling algorithm for the shortest path problem with time windows. *INFOR*, 26:191–212.

Dumitrescu, I. and Boland, N. (2001). Algorithms for the weight constrained shortest path problem. *International Transactions in Operational Research*, 8(1):15–29.

Dumitrescu, I. and Boland, N. (2003). Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. *Networks*, 42(3):135–153.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco.

Geoffrion, A. M. (1974). Lagrangean relaxation for integer programming. *Mathematical Programming Study*, 2:82–113.

Goetschalkx, M. and Ratliff, D. H. (1990). Shared storage policies based on the duration stay of unit loads. *Management Science*, 36(9):1120–1132.

Holmberg, K. and Yuan, D. (2003). A multicommodity network-flow problem with side constraints on paths solved by column generation. *INFORMS Journal on Computing*, 15(1):42–57.

Kim, K. H. and Park, K. T. (2003). Dynamic space allocation for temporary storage. *International Journal of Systems Science*, 34(1):11–20.

Kogan, K. and Shtub, A. (1997). DGAP – the dynamic generalized assignment problem. *Annals of Operations Research*, 69:227–239.

Martello, S. and Toth, P. (1992). Generalized assignment problems. In *Algorithms and Computation*. Springer-Verlag.

Monaco, M. F., Sammarra, M., and Moccia, L. (2007). Some observations about the extreme points of the r-constrained shortest path problem polytope. Technical Report 1, Laboratorio di Logistica, Università della Calabria.

Nemhauser, G. L. and Wolsey, L. A. (1988). *Integer and Combinatorial Optimization*. Wiley, New York.

	$\mathcal{F}_1$			$\mathcal{F}_2$		
Instance	Time (min.)	Solution	Gap[a] (%)	Time (min.)	Solution	Gap[a] (%)
I1-01	30.0	11272	5.6	0.1	**11196**	0.0
I1-02	30.0	14476	2.2	0.8	**14464**	0.0
I1-03	30.0	**12796**	0.1	0.1	**12796**	0.0
I1-04	30.0	12320	7.0	3.5	**12198**	0.0
I1-05	30.0	13610	4.9	2.3	**13574**	0.0
I1-06	30.0	12514	4.5	3.7	**12424**	0.0
I1-07	26.8	**14216**	0.0	0.1	**14216**	0.0
I1-08	30.0	11442	5.1	0.5	**11258**	0.0
I1-09	30.0	11430	8.0	19.5	**11256**	0.0
I1-10	30.0	11738	5.2	0.8	**11700**	0.0
Average	*29.7*	*12581*	*4.3*	*3.1*	**12508**	*0.0*

Table 2: Computational Results for CPLEX on Instance Set I1 with $\mathcal{F}_2$ and $\mathcal{F}_1$ formulations. Bold entries correspond to the minimum solution value for each row

a - The gap is computed with respect to the value of the linear relaxation as (upper bound - lower bound)/upper bound.

	$\mathcal{F}_2$			COL2		$\mathcal{F}_2$ vs COL2
Instance	Time (min.)	Solution	Gap[a] (%)	Time (min.)	Solution	Variation[b] (%)
I1-01	0.11	**11196**	0.0	0.04	11311	1.0
I1-02	0.84	**14464**	0.0	0.07	14533	0.5
I1-03	0.08	**12796**	0.0	0.07	12797	0.0
I1-04	3.46	**12198**	0.0	0.13	12331	1.1
I1-05	2.32	**13574**	0.0	0.20	13605	0.2
I1-06	3.70	**12424**	0.0	0.08	12545	1.0
I1-07	0.08	**14216**	0.0	0.06	14393	1.2
I1-08	0.52	**11258**	0.0	0.06	11359	0.9
I1-09	19.50	**11256**	0.0	0.06	11257	0.0
I1-10	0.79	**11700**	0.0	0.07	11799	0.8
Average	*3.14*	**12508**	*0.0*	*0.08*	*12593*	*0.7*

Table 3: Computational Results on Instance Set I1 with $\mathcal{F}_2$ and COL2 algorithms. Bold entries correspond to the minimum solution value for each row

a - The gap is computed with respect to the value of the linear relaxation as (upper bound - lower bound)/upper bound.
b - The variation is computed as 100(COL2-$\mathcal{F}_2$)/$\mathcal{F}_2$.

	$\mathcal{F}_2$			COL2		$\mathcal{F}_2$ vs COL2
Instance	Time (min.)	Solution	Gap[a] (%)	Time (min.)	Solution	Variation[b] (%)
I2-01	2.1	**27744**	0.0	0.5	28025	1.0
I2-02	13.2	**23308**	0.0	0.6	23799	2.1
I2-03	5.9	**24074**	0.0	3.2	24827	3.1
I2-04	4.0	**26139**	0.0	0.5	26804	2.5
I2-05	30.0	**26878**	1.4	30.0	27145	1.0
I2-06	30.0	**24366**	0.1	1.6	24815	1.8
I2-07	30.0	**28185**	0.4	2.2	28816	2.2
I2-08	30.0	**28818**	1.7	2.5	28923	0.4
I2-09	30.0	**24526**	0.7	3.2	25057	2.2
I2-10	12.6	**25652**	0.0	0.6	26377	2.8
Average	*18.8*	**25969**	*0.4*	*4.5*	*26459*	*1.9*

Table 4: Computational Results on Instance Set I2 with $\mathcal{F}_2$ and COL2 algorithms. Bold entries correspond to the minimum solution value for each row

a - The gap is computed with respect to the value of the linear relaxation as (upper bound - lower bound)/upper bound.
b - The variation is computed as 100(COL2-$\mathcal{F}_2$)/$\mathcal{F}_2$.

	$\mathcal{F}_2$			COL2		$\mathcal{F}_2$ vs COL2
Instance	Time (min.)	Solution	Gap[a] (%)	Time (min.)	Solution	Variation[b] (%)
I3-01	30	49829	3.2	29	**49438**	-0.78
I3-02	30	**45832**	3.6	30	46279	0.98
I3-03	30	54291	2.8	30	**54212**	-0.15
I3-04	30	51968	2.4	30	**51965**	-0.01
I3-05	30	50080	4.4	30	**49905**	-0.35
I3-06	30	**46183**	2.1	30	46876	1.50
I3-07	30	**49469**	3.6	30	49754	0.58
I3-08	30	49768	5.5	30	**48519**	-2.51
I3-09	30	**50035**	0.8	6	50348	0.63
I3-10	30	**48985**	3.4	30	48992	0.01
Average	*30*	*49644*	*3.2*	*28*	**49629**	*-0.03*

Table 5: Computational Results on Instance Set I3 with $\mathcal{F}_2$ and COL2 algorithms. Bold entries correspond to the minimum solution value for each row

a - The gap is computed with respect to the value of the linear relaxation as (upper bound - lower bound)/upper bound.
b - The variation is computed as 100(COL2-$\mathcal{F}_2$)/$\mathcal{F}_2$.

	$\mathcal{F}_2$[a]			COL2		$\mathcal{F}_2$ vs COL2
Instance	Time (min.)	Solution	Gap[b] (%)	Time (min.)	Solution	Variation[c] (%)
I4-01	30	82957	8.4	30	**79736**	-3.9
I4-02	30	129147	37.8	30	**84354**	-34.7
I4-03	30	132347	34.3	30	**91630**	-30.8
I4-04	n.a.	n.a.	n.a.	30	**80896**	n.a.
I4-05	30	86725	7.0	30	**82772**	-4.6
I4-06	30	**89026**	3.2	30	89383	0.4
I4-07	30	121265	37.2	30	**78970**	-34.9
I4-08	30	125646	40.6	30	**77741**	-38.1
I4-09	30	128012	35.8	30	**85015**	-33.6
I4-10	30	123226	35.6	30	**83505**	-32.2
Average	*n.a*	*n.a*	*n.a*	*30*	**83400**	*n.a*

Table 6: Computational Results on Instance Set I4 with $\mathcal{F}_2$ and COL2 algorithms. Bold entries correspond to the minimum solution value for each row

a - These $\mathcal{F}_2$ computational experiments were performed on a more powerful machine than the COL2, see Section 4.2 for details.
b - The gap is computed with respect to the value of the linear relaxation as (upper bound - lower bound)/upper bound.
c - The variation is computed as 100(COL2-$\mathcal{F}_2$)/$\mathcal{F}_2$.

	COL0		COL1	COL2	
Instance	Time (min.)	Gen. Columns	Solution	C.G. Time (min.)	Gen. Columns
I1-01	0.02	204	11311	0.03	228
I1-02	0.03	296	15443	0.06	352
I1-03	0.03	260	n.a.	0.06	329
I1-04	0.03	302	12331	0.07	359
I1-05	0.04	344	13647	0.08	417
I1-06	0.03	227	12553	0.06	296
I1-07	0.03	245	14393	0.05	285
I1-08	0.03	241	11389	0.05	276
I1-09	0.02	238	11257	0.04	254
I1-10	0.03	267	11923	0.05	331
Average	*0.03*	*262*	*n.a.*	*0.06*	*313*

Table 7: Computational Results on Instance Set I1 with COL0, COL1 and COL2 algorithms.

	COL0		COL1	COL2	
Instance	Time (min.)	Gen. Columns	Solution	C.G. Time (min.)	Gen. Columns
I2-01	0.15	527	n.a.	0.41	737
I2-02	0.19	482	n.a.	0.40	673
I2-03	0.14	434	n.a.	0.31	634
I2-04	0.11	372	n.a.	0.30	589
I2-05	0.20	518	27259	0.42	717
I2-06	0.17	488	n.a.	0.40	632
I2-07	0.14	505	n.a.	0.33	716
I2-08	0.18	594	29413	0.41	780
I2-09	0.14	483	25275	0.38	659
I2-10	0.12	448	n.a.	0.31	593
Average	*0.15*	*485*	*n.a.*	*0.37*	*673*

Table 8: Computational Results on Instance Set I2 with COL0, COL1 and COL2 algorithms.

	COL0		COL1	COL2	
Instance	Time (min.)	Gen. Columns	Solution	C.G. Time (min.)	Gen. Columns
I3-01	1.2	950	50966	3.5	1725
I3-02	1.2	839	47733	3.5	1601
I3-03	1.1	982	n.a.	3.6	1786
I3-04	1.1	915	n.a.	4.4	1881
I3-05	1.2	936	50327	4.2	1821
I3-06	1.0	794	n.a.	2.3	1307
I3-07	1.1	852	n.a.	3.3	1611
I3-08	1.0	874	n.a.	3.3	1603
I3-09	1.2	836	n.a.	3.6	1573
I3-10	1.3	798	50840	3.8	1588
Average	*1.1*	*878*	*n.a.*	*3.5*	*1650*

Table 9: Computational Results on Instance Set I3 with COL0, COL1 and COL2 algorithms.

	COL0		COL1	COL2	
Instance	Time (min.)	Gen. Columns	Solution	C.G. Time (min.)	Gen. Columns
I4-01	4.5	1525	n.a.	14.4	2848
I4-02	3.1	1197	n.a.	15.2	3174
I4-03	3.9	1479	n.a.	15.1	3082
I4-04	2.9	1216	n.a.	13.3	2769
I4-05	3.6	1179	n.a.	12.6	2175
I4-06	3.6	1387	n.a.	13.6	2773
I4-07	3.6	1410	n.a.	14.7	3165
I4-08	3.4	1195	n.a.	11.7	2377
I4-09	3.3	1266	n.a.	15.8	3077
I4-10	3.0	1284	n.a.	13.2	3025
Average	*3.5*	*1314*	*n.a.*	*14.0*	*2847*

Table 10: Computational Results on Instance Set I4 with COL0, COL1 and COL2 algorithms.

Instance Set	$\mathcal{F}_2$ Lower Bound	COL0 Solution	Variation[a] (%)
I1	11977.4	11956.1	0.18
I2	24949.1	24916.2	0.13
I3	46869.1	46859.4	0.02
I4	78390.1	78375.2	0.02
Average	40546.4	40526.7	0.09

Table 11: Instance set average lower bound with $\mathcal{F}_2$ and COL0

a - The variation is computed as 100($\mathcal{F}_2$-COL0)/COL0.