Bicriteria Multiresource Generalized Assignment Problem

Özlem Karsu,[1] Meral Azizoğlu[2]

[1]*Department of Industrial Engineering, Bilkent University, Ankara 06800, Turkey*

[2]*Department of Industrial Engineering, Middle East Technical University, Ankara 06531, Turkey*

Received 4 October 2012; revised 26 June 2014; accepted 13 October 2014
DOI 10.1002/nav.21607
Published online 7 November 2014 in Wiley Online Library (wileyonlinelibrary.com).

Abstract: In this study, we consider a bicriteria multiresource generalized assignment problem. Our criteria are the total assignment load and maximum assignment load over all agents. We aim to generate all nondominated objective vectors and the corresponding efficient solutions. We propose several lower and upper bounds and use them in our optimization and heuristic algorithms. The computational results have shown the satisfactory behaviors of our approaches. © 2014 Wiley Periodicals, Inc. Naval Research Logistics 61: 621–636, 2014

Keywords: generalized assignment problem; linear programming relaxation; branch and bound; heuristics

1. INTRODUCTION

Assignment problems (APs) are motivated by situations where scarce resources, so-called agents, are to be allocated to tasks. The importance of these assignments stems from their direct applications and their roles as subproblems in several operations research problems. The classical AP assigns the tasks to the agents, each agent handles at most one task and each task is assigned to exactly one agent, and the total cost over all assignments is minimized. The generalized assignment problem (GAP) is an extension of the classical model, which assumes that more than one task can be assigned to one agent, the agents have limited availabilities and the tasks have defined resource consumptions. The multiresource generalized assignment problem (MRGAP) permits the assignment of multiple tasks to an agent subject to the availability of a set of multiple resources consumed by that agent.

The classical GAP minimizes the total cost over all agents; hence there are concerns regarding efficiency. Conversely, in many practical cases, the satisfaction level of each individual agent, which can be represented by a fairness measure, is as important as maximizing profits. Improving the satisfaction levels of all agents is likely to promote their cooperation in similar future projects.

In this study, we consider the MRGAP with two objectives: minimizing the total workload assigned over all agents and minimizing the maximum workload assigned among all agents. The total workload is pertinent to the efficiency concerns of the decision makers. Conversely, the maximum workload objective seeks to generate a balanced workload among the agents, hence, it is related to the fairness concerns of the decision makers.

One real application that we take our motivation from is faced by a well-recognized firm in the heating, ventilating, and air-conditioning sector in Turkey. The problem is to assign agents to the tasks that they call opportunities, such that the agent assigned to an opportunity will follow it for multiple periods until its completion. In each period, the agents have limited time, and the time requirement of an opportunity depends on the experience of its assigned agent. The managers require a single agent to be assigned to an opportunity, as they believe that communication between multiagents would slow the process down.

Our motivating case can be extended to many sectors, including but not limited to, banking (agents replaced by servers of limited capacity and arbitrary pace, opportunities replaced by customers), communications (agents replaced by nodes of the distributed computer systems, opportunities replaced by processors and databases), healthcare (agents replaced by physicians having specialization areas; hospital rooms and emergency facilities with limited capacities, opportunities replaced by patients), transportation (agents replaced by trucks having limited capacities or specified routes, opportunities replaced by transportation items), retail (agents replaced by warehouses with limited capacities,

Correspondence to: Meral Azizoğlu (ma@metu.edu.tr)

opportunities replaced by end-customer items), government (agents replaced by clerks of limited capacity and arbitrary pace, opportunities replaced by customer requests).

In all of the above sectors, the opportunities are the customers, each with a specified service time and a need to be satisfied by a single agent (server). The managers' concern is not only to minimize the total service time spent on all customers (for efficiency), but also to minimize the maximum service time allocated to any one customer (for fairness). For example, for location problems in emergency services, not only the total distance travelled, but also the longest distance between the emergency facility and the patient is an important concern.

In this study, we consider a bicriteria MRGAP. To the best of our knowledge, there is no other reported work on the bicriteria MRGAP.

We generate all efficient solutions with respect to our efficiency measure of total workload over all agents, and fairness measure of maximum workload among all agents.

Our bicriteria MRGAP is NP-Hard in the strong sense as the GAP that minimizes the total workload over all agents is NP-Hard (Non-deterministic Polynomial-time Hard) in the strong sense [17]. The complexity of our problem suggests that any optimization procedure will have computational challenges as the number of agents and tasks increases. Recognizing this fact, we propose optimization algorithms to seek optimal solutions for small- to medium-sized problem instances. For large-sized problem instances, we rely on heuristic procedures that are based on the ideas used in our optimization algorithms.

The article is organized as follows: Section 2 reports on the related literature. In Section 3, we specify our problem and give its mathematical model. In Section 4, we define our classical approach (CA) that uses the successive solutions of the constrained optimization models. In Section 5, we specify our branch and bound (B&B) algorithm together with its reduction and bounding mechanisms. Section 6 presents our heuristic algorithms, and Section 7 reports the results of our computational study. We conclude the article in Section 8.

2. LITERATURE REVIEW

Many practical applications of the AP, GAP, and MRGAP are cited in the literature. Burkard [1] discusses some applications of the AP. The cited applications of the GAP include fixed-charge plant location problems [3], p-median location problems [29], grouping and loading problems in flexible manufacturing systems [20], cell formation problems in cellular manufacturing systems [31], routing problems [7], designing communication networks [12], and allocating cross-trained workers to multiple departments [2]. The examples cited for the MRGAP are allocating processors and databases to the nodes of a distributed computer system, truck routing, cargo loading on ships, warehouse design and workload planning in job shops [9, 10, 24, 26].

Dell'Amico and Martello [5], and Cattrysse and Van Wassenhove [3] give extensive surveys for the solution approaches of the AP and GAP, respectively. The solution approaches for the MRGAP are addressed in [10, 21, 22, 33]. Gavish and Pirkul [10] study different Lagrangean relaxations of the model and develop heuristic procedures and a B&B algorithm based on these relaxations. Mazzola and Wilcox [21] propose a three-phased heuristic procedure that finds an initial feasible solution and then systematically improves the solution. Yagiura et al. [33] propose a very large-scale neighborhood search algorithm based on a Tabu Search (TS) idea. Mitrović-Minić and Punnen [22] develop a very large-scale neighborhood search algorithm for the MRGAP.

The AP, GAP, and MRGAP that consider min–max-type objectives are referred to as the bottleneck AP, bottleneck GAP, and bottleneck MRGAP, respectively. The bottleneck AP, which minimizes the maximum cost (time) over all assignments, has been widely studied in the literature (see Burkard [1]). Punnen and Aneja [28] proposed exact and heuristic methods to solve the general bottleneck (min–max as they refer) combinatorial optimization problems. Seshan [30], and Punnen and Aneja [28] studied a bottleneck AP in which the tasks are divided into categories. Mazzola and Neebe [18, 19], and Martello and Toth [16] considered the bottleneck GAP with the objective of minimizing the maximum cost over all assignments. Mazzola and Neebe [18] proposed mathematical models and a technique for transforming bottleneck GAPs into classical GAPs. Mazzola and Neebe [19] proposed an algorithm that solves a sequence of GAP feasibility problems and uses lower bounds on the general bottleneck linear programming problem. Martello and Toth [16] present a B&B algorithm and several approximation algorithms. Karsu and Azizoglu [15] address the MRGAP that minimizes the maximum cost among all agents. They propose a B&B algorithm that uses the optimal solutions of the linear programming relaxations (LPRs).

There exists limited work on the bicriteria GAP. Zhang and Ong [34] proposed a linear programming-based heuristic procedure and Garrett and Dasgupta [8] presented a search space and fitness landscape analysis. Both studies aimed to generate an approximate set of nondominated solutions.

In this study, we consider a bicriteria MRGAP that considers the maximum cost among all agents and the total cost over all agents, as the two objectives. To the best of our knowledge, there is no other reported work on the bicriteria MRGAP.

3. PROBLEM DEFINITION

We considered n tasks to be assigned to m agents. There are s periods. The available capacity of agent i in period t is

defined as b_{it} and p_{ijt} is the time required by task j in period t if performed by agent i. We assume that all parameters are integers.

We use a binary variable x_{ij} to explain our assignment decisions where:

$$x_{ij}=\begin{Bmatrix}1 \text{ if task } j \text{ is assigned to agent } i\\ 0 \text{ otherwise}\end{Bmatrix}.$$

Our bicriteria MRGAP is as follows:

$$\min\left\{\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{t=1}^{s} p_{ijt}x_{ij},\ \max_i\left\{\sum_{j=1}^{n}\sum_{t=1}^{s} p_{ijt}x_{ij}\right\}\right\}$$

$$\sum_{j=1}^{n} p_{ijt}x_{ij} \le b_{it} \quad \forall i = 1,\ldots,m, \quad \forall t = 1,\ldots,s$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall j = 1,\ldots,n$$

$$x_{ij} = 0 \text{ or } 1 \quad \forall i = 1,\ldots,m, \quad \forall j = 1,\ldots,n$$

The linear programming representation of the problem is as follows:

$$\min\{Z_{\mathrm{T}}, Z_{\mathrm{M}}\}$$

$$\sum_{j=1}^{n} p_{ijt}x_{ij} \le b_{it} \quad \forall i = 1,\ldots,m, \quad \forall t = 1,\ldots,s \qquad (1)$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall j = 1,\ldots,n \qquad (2)$$

$$\sum_{j=1}^{n}\sum_{t=1}^{s} p_{ijt}x_{ij} \le Z_M \quad \forall i = 1,\ldots,m \qquad (3)$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{t=1}^{s} p_{ijt}x_{ij} = Z_{\mathrm{T}} \qquad (4)$$

$$\mathrm{x}_{ij} = 0 \text{ or } \quad \forall i = 1,\ldots,m, \quad \forall j = 1,\ldots,n \qquad (5)$$

The objective functions are to minimize Z_{M} (maximum load among all agents) and minimize Z_{T} (total load over all agents).

Constraint set (1) ensures that the capacities of the agents are not exceeded and Constraint set (2) ensures that each task is assigned to one agent. Constraint sets (3) and (4) define Z_{M} and Z_{T}. The assignment restrictions are given by constraint set (5). We, hereafter, refer to Constraint sets (1) through (5) as $x \in X$.

Our bicriteria MRGAP reduces to the bicriteria GAP when $s = 1$ and to the bicriteria AP when $s = 1$ and all time and capacity values are unity.

A solution r in set X is called efficient if there is no other solution q in set X having $Z_{\mathrm{M}}^q \le Z_{\mathrm{M}}^r$ and $Z_{\mathrm{T}}^q \le Z_{\mathrm{T}}^r$ with strict inequality holding at least once. The resulting objective vector $(Z_{\mathrm{M}}^r, Z_{\mathrm{T}}^r)$ is said to be nondominated. If there exists a solution q such that $(\boldsymbol{Z_{\mathrm{M}}^q}, \boldsymbol{Z_{\mathrm{T}}^q}) \le (\boldsymbol{Z_{\mathrm{M}}^r}, \boldsymbol{Z_{\mathrm{T}}^r})$ (that is $Z_{\mathrm{M}}^q \le Z_{\mathrm{M}}^r$ and $Z_{\mathrm{T}}^q < Z_{\mathrm{T}}^r$ or $Z_{\mathrm{M}}^q < Z_{\mathrm{M}}^r$ and $Z_{\mathrm{T}}^q \le Z_{\mathrm{T}}^r$) then solution q dominates solution r, and we say the objective vector $(\boldsymbol{Z_{\mathrm{M}}^q}, \boldsymbol{Z_{\mathrm{T}}^q})$ dominates the objective vector $(\boldsymbol{Z_{\mathrm{M}}^r}, \boldsymbol{Z_{\mathrm{T}}^r})$.

4. FINDING THE NONDOMINATED OBJECTIVE VECTORS

Consider the min Z_{M} s.t. $\boldsymbol{x} \in X$ problem and let Z_{M}^* be its optimal Z_{M} value. Z_{M}^* is a valid lower bound on the Z_{M} values of all efficient solutions. However, any optimal solution to the problem may not be efficient as there may exist alternate optimal solutions having smaller Z_{T} values.

Among the alternate optimal solutions to the min Z_{M} s.t. $\boldsymbol{x} \in X$ problem, the one having the smallest Z_{T} value requires the optimal solution of the min Z_{T} s.t. $\boldsymbol{x} \in X$ and $Z_{\mathrm{M}} = Z_{\mathrm{M}}^*$ problem. In place of treating $Z_{\mathrm{M}} = Z_{\mathrm{M}}^*$ in the constraint set, we can modify the objective function as $Z_{\mathrm{M}} + \varepsilon_{ZTOT} Z_{\mathrm{T}}$ for a sufficiently small value of $\varepsilon_{ZTOT} > 0$ and get the min $Z_{\mathrm{M}} + \varepsilon_{ZTOT} Z_{\mathrm{T}}$ s.t. $\boldsymbol{x} \in X$ problem.

ε_{ZTOT} should be set small enough that the maximum load value should not increase even for the largest possible value of the total load, $Z_{\mathrm{T}}^{\max}$. That is, $Z_{\mathrm{M}}^* + \varepsilon_{ZTOT} Z_{\mathrm{T}}^{\max} < Z_{\mathrm{M}}^* + Z_{\mathrm{T}}^{\min}$, where $Z_{\mathrm{T}}^{\min}$ is the smallest possible value of Z_{T}. This follows $\varepsilon_{ZTOT} Z_{\mathrm{T}}^{\max} < Z_{\mathrm{T}}^{\min}$, that is, $\varepsilon_{ZTOT} < \frac{Z_{\mathrm{T}}^{\min}}{Z_{\mathrm{T}}^{\max}}$. The maximum value of Z_{T} is not bigger than $\sum_{j=1}^{n} \max_i\{\sum_{t=1}^{s} p_{ijt}\}$ and the minimum value of Z_{T} is not less than 1. Hence, $\varepsilon_{ZTOT} < \frac{1}{\sum_{j=1}^{n} \max_i\{\sum_{t=1}^{s} p_{ijt}\}}$ should hold. In our experiments, we set ε_{ZTOT} to:

$$\frac{1}{\sum_{j=1}^{n} \max\limits_{i}\{\sum_{t=1}^{s} p_{ijt}\} + 1}$$

4.1. Finding the Exact Nondominated Objective Vectors

Procedure 1 (below) generates all nondominated objective vectors by solving the min $Z_{\mathrm{M}} + \varepsilon_{ZTOT} Z_{\mathrm{T}}$ subject to the $\boldsymbol{x} \in X$ and $Z_{\mathrm{T}} \le k$ problem. The procedure is based on the fact that the objective vector values $(Z_{\mathrm{M}}, Z_{\mathrm{T}})$ are integers.

4.1.1. Procedure 1. Finding All Nondominated Objective Vectors

Step 0. Solve min $\mathrm{Z_M} + \varepsilon_{ZTOT}\mathrm{Z_T}$
s.t. $\boldsymbol{x} \in X$
Let the objective vector of the solution be $(\boldsymbol{Z_{\mathrm{M}}^*}, \boldsymbol{Z_{\mathrm{T}}^*})$
$k = Z_{\mathrm{T}}^* - 1$

Step 1. (P1) min $Z_{\mathrm{M}} + \varepsilon_{ZTOT} Z_{\mathrm{T}}$
s.t. $x \in X$ and $\mathrm{Z_T} \le k$

Step 2. If (P1) is infeasible, stop.
Let the solution be $(\boldsymbol{Z}^*_{\mathbf{M}}, \boldsymbol{Z}^*_{\mathbf{T}})$
$k = Z^*_{\mathrm{T}} - 1$
Go to Step 1

An optimal solution to the min $Z_{\mathrm{M}} + \varepsilon_{ZTOT} Z_{\mathrm{T}}$ s.t. $\boldsymbol{x} \in X$ and $Z_{\mathrm{T}} \leq k$ problem is efficient (see Haimes et al. [13] for the general bicriteria problem). Hence each step of Procedure 1 generates an efficient solution and the corresponding nondominated objective vector. The procedure generates all nondominated objective vectors, as it considers all possible values of Z_{T} by varying k systematically between $Z^{\min}_{\mathrm{T}}$ and $Z^{\max}_{\mathrm{T}}$. There can be at most $Z^{\max}_{\mathrm{T}} - Z^{\min}_{\mathrm{T}} + 1$ nondominated objective vectors, hence the procedure iterates pseudo polynomial times.

The MRGAP reduces to GAP when $s = 1$. The feasibility version of the GAP is NP-complete in the strong sense (Martello and Toth [17]) so is the feasibility version of the MRGAP. It follows that a problem of generating even a single efficient solution is strongly NP-hard.

We, hereafter, refer to Procedure 1 as the CA. The CA takes its spirit from the ε-approach that generates single objective subproblems, and transforms all but one of the objectives into constraints (Ehrgott and Gandibleux [6]). The upper bounds of these constraints are given by the ε vector and the exact nondominated objective vector is generated by varying the ε vector. Note that the CA systematically varies the k value and generates the exact nondominated objective vectors.

Alternatively, to generate the set of all nondominated objective vectors, we could solve the min $Z_{\mathrm{T}} + \varepsilon_{ZMAX} Z_{\mathrm{M}}$ s.t. $\boldsymbol{x} \in X$ and $Z_{\mathrm{M}} \leq k$ problem for a properly selected value of ε_{ZMAX} and vary k between $Z^{\min}_{\mathrm{M}}$ and $Z^{\max}_{\mathrm{M}}$.

4.2. Finding the Approximate Nondominated Objective Vectors

Recall that our bicriteria MRGAP is strongly NP-Hard as the feasibility version of the GAP is NP-Complete in the strong sense. It is likely that the CA may fail to solve large-sized problem instances, hence there is a need for heuristic procedures. We implement a variation of the CA so as find an approximate set of the nondominated objective vectors.

We solve the min $Z_{\mathrm{M}} + \varepsilon_{ZTOT} Z_{\mathrm{T}}$ s.t. $\boldsymbol{x} \in X$ and $Z_{\mathrm{T}} \leq k$ problem using a gap, $\alpha\%$. Our integer programming solver CPLEX, sets α to the relative deviation of the best known objective function value (UB) and the best known lower bound on the optimal objective function value (LB). That is, $\alpha = \frac{\mathrm{UB}-\mathrm{LB}}{\mathrm{LB}} \times 100$. Hence, it guarantees that the resulting Z_{M} value deviates from the optimal Z_{M} value by at most $\alpha\%$. Note that the resulting solution has no guarantee of efficiency even when $\alpha = 0$, as there may exist alternative optimal solutions with smaller Z_{T} values.

Below is the stepwise description of the algorithm that generates approximate nondominated objective vectors.

4.2.1. Procedure 2. Finding Approximate Nondominated Objective Vectors

Step 0. Solve min Z_{M}
s.t. $\boldsymbol{x} \in X$
with $\alpha\%$gap
Let the solution be $(\boldsymbol{Z}^{\boldsymbol{\alpha}}_{\mathbf{M}}, \boldsymbol{Z}^{\boldsymbol{\alpha}}_{\mathbf{T}})$
$k_{\mathrm{T}} = Z^{\alpha}_{\mathrm{T}} - 1$
$k_{\mathrm{M}} = Z^{\alpha}_{\mathrm{M}}$

Step 1. (P1) minZ_{T}
s.t. $\boldsymbol{x} \in X$ $Z_{\mathrm{T}} \leq k_{\mathrm{T}}$ and $Z_{\mathrm{M}} \leq k_{\mathrm{M}}$
Solve P1 with $\alpha\%$ gap

Step 2. Case 1. (P1) is infeasible
Solve min Z_{M}
s.t. $\boldsymbol{x} \in X$ and $Z_{\mathrm{T}} \leq k_{\mathrm{T}}$
with $\alpha\%$ gap
If the solution is infeasible, stop
Let the solution be $(\boldsymbol{Z}^{\boldsymbol{\alpha}}_{\mathbf{M}}, \boldsymbol{Z}^{\boldsymbol{\alpha}}_{\mathbf{T}})$
$k_T = Z^{\alpha}_{T} - 1$
$k_M = Z^{\alpha}_{M}$
Go to Step 1
Case 2. (P1) is feasible
Assume $(Z^{\alpha}_{\mathrm{M}}, Z^{\alpha}_{\mathrm{T}})$ solves (P1)
$k_{\mathrm{T}} = Z^{\alpha}_{\mathrm{T}} - 1$
Solve min Z_{M}
s.t. $\boldsymbol{x} \in X$ and $Z_{\mathrm{T}} \leq k_{\mathrm{T}}$
with $\alpha\%$ gap
If the solution is infeasible, stop
Let the solution be $(Z^{\alpha}_{\mathrm{M}}, Z^{\alpha}_{\mathrm{T}})$
$k_{\mathrm{T}} = Z^{\alpha}_{\mathrm{T}} - 1$
$k_{\mathrm{M}} = Z^{\alpha}_{\mathrm{M}}$
Go to Step 1

In Step 0, we solve (P1) with $\alpha\%$ gap, hence an overestimate on the smallest possible value of Z_{M} is found.

In Step 1, we look for a nondominated solution. If the α value is zero, an efficient solution having the same Z_{M} value, but a smaller Z_{T} value, could be found through the optimal solution of the min Z_{T} s.t. $\boldsymbol{x} \in X$ and $Z_{\mathrm{M}} = k_{\mathrm{M}}$ problem. We solved the min Z_{T} s.t. $\boldsymbol{x} \in X$ $Z_{\mathrm{T}} \leq k_{\mathrm{T}}$ and $Z_{\mathrm{M}} \leq k_{\mathrm{M}}$ problem with the hope of getting a nondominated solution. However, the resulting solution may not be nondominated as we are using a gap.

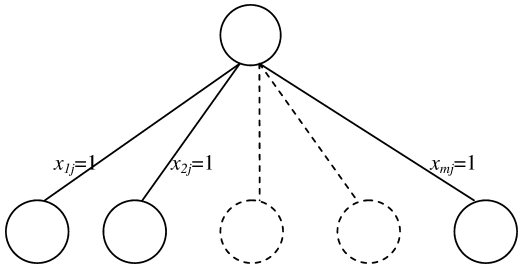

Figure 1. The branching scheme.

Step 2 checks the feasibility of the problem solved in Step 1. If the problem is infeasible then we look for the feasible solution with the smallest Z_M value while satisfying $Z_T \leq k_T$. If the problem is feasible, we updated Z_T^{α} and k_T accordingly then solve the min Z_M s.t. $x \in X$ and $Z_T \leq k_T$ problem. We terminated the procedure in Step 2, after the smallest possible Z_T value is reached.We, hereafter, refer to Procedure 2 as the classical approach heuristic (CAH).

5. BRANCH AND BOUND ALGORITHM

In this section, we present a B&B algorithm that finds the exact set of all nondominated objective vectors together with the corresponding efficient set. The reported B&B algorithms to generate all nondominated objective vectors were provided by Sun [32] for the network flow problems and Ozlen and Azizoglu [25] for a rescheduling problem.

Our B&B algorithm starts with an initial approximate set of nondominated objective vectors and updates the set, whenever a feasible solution that is not dominated by any objective vector in the set is found. We refer to the current nondominated set of objective vectors as an incumbent set (IS). At the termination of the procedure, IS contains the exact set of all nondominated objective vectors and one efficient solution per nondominated objective vector.

At each level of the B&B tree, we selected a task. The task selection was based on the optimal solutions of the LPR of the $\min Z_T + \varepsilon_{ZMAX} Z_M$ s.t. $\boldsymbol{x} \in X$ and $Z_M \leq k$ problem (LPR_T). We selected the task corresponding to the highest fractional variable of the LPR_T. In our preliminary runs, we also tried the highest fractional variable of the LPR_M, however, we could not find better results.

For the selected task, we generate m nodes: each representing the assignment of the task to one of the m agents. Figure 1 illustrates our branching scheme.

A partial solution to the problem, that is, a node of the B&B tree, gives a set of assigned tasks (S) together with their agents. At each node, we removed agent i from all future considerations if it cannot process any unassigned task j in any period t due to its available capacity.

We fathom a node representing the assignment of task j to agent i if one of the following conditions holds:

1. The available capacity of agent i is not sufficient for task j in any period (as the node cannot lead to a feasible solution).
2. $\sum_i n_{i.} < \bar{n}$, where n_i is an upper bound on the number of unassigned tasks that agent i can process by its remaining capacity and $\bar{n}$ is the number of unassigned tasks (as the node cannot lead to a feasible solution). The n_i value that satisfies $\sum_{j=1}^{n_i} \sum_t p_{i\,[j]\,t} \leq \sum_t b_{it}$ and $\sum_{j=1}^{n_i+1} \sum_t p_{i\,[j]\,t} > \sum_t b_{it}$ is a valid upper bound on the number of tasks that agent i can process, once the tasks are ordered such that $\sum_t p_{i\,[j]\,t} \leq \sum_t p_{i\,[j+1]\,t}$ for all j.
3. The maximum and total loads after the assignment are dominated by any solution in IS.

When a new feasible solution is reached, we update IS in two ways:

1. Add the new feasible solution and its objective vector to IS if the new solution is not dominated by any solution in IS.
2. Remove the solutions that are dominated by the new solution, and their objective vectors.

We also update IS when all unassigned tasks at any node can be performed by agent i without violating its capacity, hence a new feasible solution is reached.

5.1. Lower Bounds

For each unfathomed node, we find two lower bounds (LBs) and fathom a node if the lower bound (LB_M, LB_T) is dominated by any solution in IS.

5.1.1. *Lower Bound 1: (LB_{1M}, LB_{1T})*

LB_{1T} assumes that each task is performed by the agent with the least cost. The resulting expression is:

$$LB_{1T} \sum_j \min_i \left\{ \sum_t p_{ijt} \right\}.$$

Such a solution ignores the capacities of the agents, which may lead to infeasible solutions.

LB_{1M} makes the same assumptions as LB_{1T}, and moreover, assumes that the load is equally split between the agents, en route to their perfect load balancing. The resulting expression is:

$$LB_{1M} \frac{\sum_j \min_i \{\sum_t p_{ijt}\}}{m}.$$

5.1.2. Lower Bound 2: (LB_{2M}, LB_{2T})

LB_{2M} and LB_{2T} are calculated by solving the LPR of the min $Z_M + \varepsilon_{ZTOT} Z_T$ s.t. $\boldsymbol{x} \in X$ and min $Z_T + \varepsilon_{ZMAX} Z_M$ s.t. $\boldsymbol{x} \in X$ problems, respectively. For each problem, we simply relax the integrality constraints and strengthen the relaxation by incorporating the following constraints:

$$1. \qquad \sum_j x_{ij} \leq n_i \ \forall i. \tag{6}$$

Constraint set (6) states that the number of tasks assigned to agent i cannot exceed n_i.

2. At the node representing the addition of task j to the set of assigned tasks S, we set $LB_M(S)$ to max LB_{1M}, LB_{2M} at the node S. We define:

$$Z_T \leq Z_T^r - 1, \tag{7}$$

where Z_T^r is the smallest Z_T value in the IS such that $LB_M(S) \geq Z_M^r$. Our aim is to avoid a solution that is dominated by any solution in the IS.

At the node representing the addition of task j to the set of assigned tasks S, we set $LB_T(S)$ to maxLB_{1T}, LB_{2T} at the node S. We define:

$$Z_T \leq Z_M^r - 1, \tag{8}$$

where Z_M^r is the smallest Z_M value in the IS such that $LB_T(S) \geq Z_T^r$. Our aim is to avoid a solution that is dominated by any solution in the IS.

We refer to the optimal solution of the LPR of the min $Z_T + \varepsilon_{ZMAX} Z_M$ s.t. $\boldsymbol{x} \in X$, (6) and (8) problem as LB_{2T} and that of the min $Z_M + \varepsilon_{ZTOT} Z_T$ s.t. $\boldsymbol{x} \in X$, (6) and (7) problem as LB_{2M}. If one of those strengthened LPR models returns all integer decision variables, we obtained a new feasible solution and the IS was updated by adding the new feasible solution and removing all solutions that are dominated by the new solution.

Our preliminary experiments have revealed that LB_{2T} is effective in reducing the tree size, however, an effort spent to compute LB_{2M} cannot be justified by the additional node reductions. For this reason, we do not use LB_{2M}.

For each node, we first calculate LB_{1M} and LB_{1T}. If (LB_{1M}, LB_{1T}) is not dominated by any objective vector in the IS then we calculate LB_{2T} and check for the dominance of (LB_{1M}, LB_{2T}). If (LB_{1M}, LB_{2T}) is dominated by any objective vector in the IS, then the current node is eliminated. If all nodes are eliminated at any level then we backtrack to the previous level.

5.2. Incumbent Set

To find the initial IS to our B&B algorithm, we used the LPRs of the min $Z_T + \varepsilon_{ZMAX} Z_M$ s.t. $\boldsymbol{x} \in X$ and min $Z_M + \varepsilon_{ZTOT} Z_T$ s.t. $\boldsymbol{x} \in X$ problems. For each LPR solution, we assign the fractional tasks to their highest fractional agents, as long as the assignment is feasible and we obtain at most two initial feasible solutions. We improve each feasible solution with respect to Z_M and Z_T in two phases. The first phase helps to reduce the maximum load and it focuses on the bottleneck agent (the agent that has the maximum load, that is, that defines the Z_M value), whereas second phase helps in reducing the total workload.

The first phase assigns each task of the bottleneck agent to all other agents as long as the assignment leads to a feasible solution, that is, the newly assigned agent has available capacity. Then it exchanges all task pairs between bottleneck and other agents, as long as the exchange is feasible. In the second phase, we exchange each task of each agent with each task of all other agents as long as the exchange leads to a feasible solution.

In each phase, the move that leads to the maximum improvement in Z_M or Z_T is performed. We stopped when the number of nonimproving moves reaches 250 or the number of iterations reaches 1000.

At each node, we seek opportunities to update the incumbent efficient set. In doing so, we first modified the LPR solution by assigning the fractional tasks to their highest fractional agents, as long as the assignment is feasible. We then search the neighborhood of the modified solution by reassigning one task from the bottleneck agent to another agent, and interchanging two tasks assigned to different agents, as long as the exchange is feasible. The IS was updated whenever appropriate.

6. HEURISTIC ALGORITHMS

In this section, we present our heuristic algorithms, which have been developed in an attempt to produce high quality solutions in reasonable time.

6.1. Tabu Search Algorithms

We implement two TS algorithms, namely TS_M and TS_T. TS_M is the algorithm proposed by Karsu and Azizoglu [15] for the single objective min–max (min Z_M) problem. It starts with a feasible solution, which is found by modifying the optimal LPR solution of a min–max problem by assigning each fractional task to its highest fractional agent, as long as the assignment is feasible. The neighborhoods, namely N1 and N2 that consider minimizing the maximum load, are similar to the first phase of the initial incumbent algorithm. N1 assigns each task of the bottleneck agent to all other agents as

long as the assignment leads to a feasible solution, that is, the newly assigned agent has available capacity. N2 exchanges all task pairs between the bottleneck agent and other agents, as long as the exchange is feasible.

The TS_T algorithm is similar to the TS_M algorithm, with a modification in N2. We exchanged each task of each agent with each task of all other agents as long as the assignment leads to a feasible solution, as in the second phase of the initial incumbent algorithm.

The TS_M algorithm aims to improve the Z_M value. Meanwhile, we keep track of the corresponding Z_T values and update the IS, whenever we find an eligible solution, that is, a solution which is nondominated by the IS. Similarly, we keep track of the Z_M values in TS_T algorithm and update the IS, whenever possible.

We first use TS_M to obtain an initial set of solutions. Our initial observations indicate that TS_M performs well in terms of generating solutions with low Z_M values. TS_T starts with a feasible solution that is found by TS_M. To ensure a high quality performance, that is, to obtain solutions with low Z_T values, we run the TS_T algorithm twice, each time with a different initial solution. In these two iterations, of the solutions provided by the TS_M algorithm, we start with the solutions with the smallest Z_M and the smallest Z_T values, respectively.

We set the Tabu list size to $O(m \times n)$ and $O(n \times n)$ for N1 and N2, respectively. Based on the results of our initial experiments, we set the Tabu tenure to 50. We use the aspiration criterion as the best solution of Z_M in TS_M (and Z_T in TS_T), that is, Tabu statuses of the moves that improve the best solution are overridden.

The algorithm terminates when the number of nonimproving moves reaches a predetermined limit called "nonimplimit" or the total number of iterations reaches the limit "maxiter." Based on our preliminary experiments, we set the "nonimplimit" value to 250 as it returned high-quality solutions without causing a significant increase in the solution times. We set the "maxiter" value to 1000 iterations.

For detailed information on the TS techniques, one may refer to Glover and Laguna [11].

6.2. Beam Search and Filtered Beam Search Algorithms

The filtered beam search (FBS) is a heuristic application of a B&B algorithm that limits the number of partial solutions evaluated on each level of the search tree. At each level, only the most promising nodes are selected and the rest are permanently discarded. Hence, the search procedure evaluates a polynomial number of nodes, unlike the B&B search that evaluates an exponential number of nodes.

The number of nodes retained is called the "beam width" of the search. There are at most "beam width" nodes, when the B&B tree has been completely explored. As the rest of the nodes are discarded, the solution can be obtained quickly but with no guarantee of optimality. If the evaluation process, called "beam evaluation function," which defines the retained nodes is powerful, then the beam search procedure expectedly returns a good solution. However, such a powerful evaluation function may be time consuming.

Filtering is used to catch the trade-off between the quality of the solutions and the time used to generate them. This process uses a simpler evaluation method and is applied on all nodes of a particular level. This simple evaluation function is called "filter evaluation function." Based on the outcome of the filter evaluation function, a number of nodes are selected for thorough evaluation by "beam evaluation function." This number of nodes is called the "filter width."

The FBS with no filtering step is referred to as the "beam search." A beam search algorithm can be favored if the "beam evaluation function" is computationally efficient and powerful.

In our implementation, we use the following functions for evaluating the partial solutions:

Filter Evaluation Function: As a filter evaluation function, we use our simple lower bounds, LB_{1M} and LB_{1T}. **U**sing simple functions, we favor low-cost evaluations.

Beam Evaluation Function: Our beam evaluation functions are more thorough than the filter evaluation functions. We use the LPR**-**based lower bounds, that is, LB_{2M} and LB_{2T}. **U**sing a thorough function, we favor high**-**quality evaluations.

Increasing the beam and filter widths improves the quality of the solutions, but at the expense of increased computational effort. In the literature, the most suitable values of the beam width and filter width are set by empirical analysis. We perform preliminary experiments to determine the most appropriate values and discuss the results in the next section.

There are several successful applications of the beam search techniques cited in the literature. Two noteworthy applications of the beam search techniques to the multicriteria problems are provided by Honda [14] and Ponte et al. [27]. For more detailed information on the beam and FBS techniques, one may refer to Morton and Pentico [23].

7. COMPUTATIONAL EXPERIMENTS

In this section, we first report our data generation scheme and performance measures, and then discuss the results of our experiment.

7.1. Data Generation

We obtained the processing times and capacities via data generation procedure that has been used previously by [10,

15, 33. The stepwise description of the procedure is outlined below. The constants *a, b*, and *c*, are used as an input.

Step 0. Generate the first period's processing time, p_{ij1}, from a discrete uniform distribution between a and b, that is, U [*a, b*].
Step 1. Set $p_{ijt} = 3p_{ij1}/4 + \gamma_{ijt}\, p_{ijt}/2$ for $t \geq 2$ and $\gamma_{ijt} \sim$ U(0, 1).
Step 2. Set $b_{it} = \ c \sum_{j \in J} p_{ijt}/m$ for all t.

In Step 0, we select $p_{ij1} \sim$U [*a, b*] sets as follows:

Set S1. $p_{ij1} \sim$ U [5,25]
Set S2. $p_{ij1} \sim$ U [10,20]
Set S3. $p_{ij1} \sim$ U [25,35]

Set S1 includes the instances where the distribution range is relatively high. Set S2 is used to see the effect of a decrease in the range of the distribution while maintaining its expected value. Set S3 includes the instances where the range of the distribution is low while the expected value is high.

In Step 2, we set c = 1.2. We try several other values for c, however, similar effects are observed with 1.2 on the solution times and solution quality of the algorithms.

In our experiments, we try different values for s (2, 3, 4, and 5), however, no significant effects of s on the solution times and quality of any algorithm were observed, therefore, s is set to 5.

For each processing time set and *m, n* combination, we generate 10 problem instances.

The algorithms are coded in Visual C++ and solved by a dual core (Intel Core i5 2.27 GHz) computer with 4 GB RAM. All models are solved by CPLEX 12.2. The solution times are expressed in central processing unit seconds.

7.2. Performance Measures

For our optimization algorithms, namely the CA and B&B, we report average and maximum CPU times. For the B&B algorithm, we also give the number of partial solutions evaluated, that is, the number of nodes.

To assess the quality of solutions returned by our heuristic algorithms, we use the following three measures:

- P = Percentage of nondominated objective vectors returned by the heuristic $\mathrm{P} = \frac{|\mathrm{ANS} \cap \mathrm{NS}|}{|NS|}$, where:
  NS represents the exact set of nondominated objective vectors; ANS represents the set of nondominated objective vectors returned by the heuristic procedure.

In our experiments, we use the nondominated set generated by the CA as NS. We set an upper time limit of 1800 s for the execution of the CA.

P fails to measure the performance when the nondominated set is not available. Moreover, P only considers the number of nondominated objective vectors returned, but not the closeness of the approximate solutions to their nondominated counterparts. To overcome these drawbacks, Czyżak and Jaszkiewicz [4] proposed two distance measures. To state the measures, we assume $(Z^r_{\mathrm{M}}, Z^r_{\mathrm{T}})$ is in set NS and $(Z^q_{\mathrm{M}}, Z^q_{\mathrm{T}})$ is in set ANS and calculate the following values:

$$R_{\mathrm{M}} = \max_{(Z^r_{\mathrm{M}}, Z^r_{\mathrm{T}}) \in \mathrm{NS}} Z^r_{\mathrm{M}} - \min_{(Z^r_{\mathrm{M}}, Z^r_{\mathrm{T}}) \in \mathrm{NS}} Z^r_{\mathrm{M}}$$

(range for the Z_{M} values in set NS)

$$R_{\mathrm{T}} = \max_{(Z^r_{\mathrm{M}}, Z^r_{\mathrm{T}}) \in \mathrm{NS}} Z^r_{\mathrm{T}} - \min_{(Z^r_{\mathrm{M}}, Z^r_{\mathrm{T}}) \in \mathrm{NS}} Z^r_{\mathrm{T}}$$

(range for the Z_{T} values in set NS)

$$f((Z^q_{\mathrm{M}}, Z^q_{\mathrm{T}}), (Z^r_{\mathrm{M}}, Z^r_{\mathrm{T}})) = \max\left\{0, \frac{1}{R_{\mathrm{M}}}(Z^q_{\mathrm{M}} - Z^r_{\mathrm{M}}), \frac{1}{R_{\mathrm{T}}}(Z^q_{\mathrm{T}} - Z^r_{\mathrm{T}})\right\}$$

Using these values, the measures are defined as:

- Distance1 (D1): The average distance between the points of set NS and the points in set ANS and is calculated as follows:

$$\mathrm{D1} = \frac{1}{|\mathrm{NS}|} \sum_{(Z^r_{\mathrm{M}}, Z^r_{\mathrm{T}}) \in \mathrm{NS}} \min_{(Z^q_{\mathrm{M}}, Z^q_{\mathrm{T}}) \in \mathrm{ANS}} \{f((Z^q_{\mathrm{M}}, Z^q_{\mathrm{T}}), (Z^r_{\mathrm{M}}, Z^r_{\mathrm{T}}))\}$$

- Distance2 (D2): The maximum distance between the points of set NS and the points in set ANS and is calculated as follows:

$$\mathrm{D2} = \max_{(Z^r_{\mathrm{M}}, Z^r_{\mathrm{T}}) \in \mathrm{NS}} \left\{ \min_{(Z^q_{\mathrm{M}}, Z^q_{\mathrm{T}}) \in \mathrm{ANS}} \{f((Z^q_{\mathrm{M}}, Z^q_{\mathrm{T}}), (Z^r_{\mathrm{M}}, Z^r_{\mathrm{T}}))\} \right\}$$

The smaller are the values of D1 and D2, the better is the performance.

7.3. Preliminary Experiments

The aim of our preliminary experiment is to define the beam and filter evaluation functions and the beam and filter widths to be used in our main experiment. In the preliminary experiments, we use m = 10 and n = 50 and 100.

7.3.1. Beam Evaluation Functions

We try four different beam evaluation functions: the simple lower bounds ($\mathrm{LB}_{1\mathrm{M}}$ and $\mathrm{LB}_{1\mathrm{T}}$) and LPR-based lower bounds

Table 1. Average performances of the beam evaluation functions, $m = 10$, beam width $= 10$

Set	Beam Evaluation Function	n	CPU Time	P	$D1$	$D2$
	LB_{1M}	50	3.78	10.40	0.07	0.16
		100	17.08	8.82	0.08	0.21
	LB_{2M}	50	27.94	15.06	0.06	0.15
S1		100	148.90	11.63	0.07	0.19
	LB_{1T}	50	4.27	28.89	0.05	0.12
		100	10.92	22.21	0.05	0.14
	LB_{2T}	50	33.36	40.36	0.03	0.09
		100	123.07	33.10	0.03	0.09
	LB_{1M}	50	4.83	24.49	0.06	0.19
		100	18.80	6.22	0.10	0.28
	LB_{2M}	50	52.27	26.64	0.05	0.17
S2		100	209.75	8.66	0.09	0.26
	LB_{1T}	50	6.19	32.24	0.03	0.10
		100	31.37	13.34	0.07	0.20
	LB_{2T}	50	40.05	45.95	0.02	0.07
		100	174.39	24.51	0.03	0.10
	LB_{1M}	50	4.54	10.48	0.10	0.27
		100	20.70	3.15	0.10	0.32
	LB_{2M}	50	43.44	13.37	0.08	0.20
S3		100	238.68	5.03	0.07	0.24
	LB_{1T}	50	6.60	15.11	0.07	0.20
		100	24.85	5.33	0.07	0.23
	LB_{2T}	50	36.76	29.46	0.04	0.10
		100	164.89	12.37	0.04	0.14

(LB_{2M} and LB_{2T}). Table 1 reports the average solution times and the average values of the distance measures for the beam search algorithms with beam width of 10.

Note from the above table that, in almost all combinations, LB_{2T} is the best in terms of solution quality, followed by its simple counterpart LB_{1T}. We found that these results hold for other n and m and beam width values. Hence for our main experiment, as beam evaluation functions, we select LB_{2T} and LB_{1T}, owing to their superior performance in terms of solution quality. We refer to the beam search algorithms using LB_{1T} and LB_{2T} as the beam evaluation functions as BS1 and BS2, respectively.

7.3.2. Filter Evaluation Functions

We try the simple lower bounds (LB_{1M} and LB_{1T}) as filter evaluation functions. We use LB_{2T} as the beam evaluation function based on our preliminary results.

Table 2 reports the average solution times and the average values of the distance measures for the two FBS algorithms with a beam width of 10 and a filter width of 25.

Table 2 shows that the performances of the LB_{1T} and LB_{1M} as filter evaluation functions are quite similar. Due to its superior performance as a beam evaluation function, we select LB_{1T} as a filter evaluation function.

Table 2. Average performances of the filter evaluation functions, $m = 10$ and beam width $= 10$, filter width $= 25$

Set	Beam Evaluation Function	n	CPU Time	P	$D1$	$D2$
	LB_{1M}	50	8.10	40.42	0.03	0.09
S1		100	33.33	35.48	0.03	0.09
	LB_{1T}	50	8.98	38.53	0.03	0.09
		100	31.64	35.00	0.03	0.09
	LB_{1M}	50	13.09	43.3	0.02	0.07
S2		100	73.05	25.7	0.04	0.11
	LB_{1T}	50	12.55	42.3	0.02	0.07
		100	41.31	24.9	0.04	0.12
	LB_{1M}	50	12.94	20.64	0.04	0.10
S3		100	52.76	12.35	0.04	0.13
	LB_{1T}	50	9.71	18.71	0.05	0.11
		100	48.72	12.01	0.05	0.15

7.3.3. Beam width and Filter width

Recall that there are at most "beam width" complete solutions, when the B&B tree has been completely explored. In our case, we may have more than "beam width" complete solutions, as we keep the solutions of the discarded nodes, if their LP relaxations give all integer values.

As a beam width value, we try $m/2$, m, $1.5m$ and $2m$ for a given number of agents, that is, m value.

Tables 3 and 4 report the average results for the beam width values for the BS1 and BS2, respectively.

Table 3. Average performances of the beam width values for the BS1, $m = 10$

BS1 Set	Beam width	n	CPU Time	P	D1	D2
	$0.5m$	50	1.25	22.74	0.06	0.14
		100	7.48	17.09	0.06	0.16
	m	50	4.27	28.89	0.05	0.12
S1		100	10.92	22.21	0.05	0.14
	$1.5m$	50	6.26	29.63	0.04	0.12
		100	34.03	25.45	0.05	0.12
	$2m$	50	9.10	31.10	0.04	0.11
		100	37.17	27.15	0.04	0.12
	$0.5m$	50	3.05	27.01	0.04	0.12
		100	11.15	10.43	0.07	0.21
	m	50	6.19	32.24	0.03	0.10
S2		100	31.37	13.34	0.07	0.20
	$1.5m$	50	10.12	36.96	0.03	0.10
		100	33.16	14.88	0.06	0.17
	$2m$	50	13.37	36.96	0.03	0.09
		100	69.10	15.13	0.05	0.17
	$0.5m$	50	2.43	10.48	0.09	0.23
		100	7.94	4.19	0.06	0.21
	m	50	6.60	15.11	0.07	0.20
S3		100	24.85	5.33	0.07	0.23
	$1.5m$	50	7.30	17.22	0.06	0.18
		100	28.78	5.80	0.07	0.21
	$2m$	50	11.08	16.78	0.06	0.18
		100	33.60	6.80	0.07	0.22

Table 4. Average performances of the beam width values for the BS2, $m = 10$

BS2 Set	Beam width	n	CPU Time	P	D1	D2
	0.5m	50	8.17	33.96	0.04	0.11
		100	30.27	28.37	0.04	0.11
	m	50	33.36	40.36	0.03	0.09
S1		100	123.07	33.10	0.03	0.09
	1.5m	50	40.92	42.25	0.03	0.09
		100	167.36	37.99	0.03	0.09
	2m	50	56.66	44.35	0.03	0.08
		100	229.51	38.02	0.03	0.09
	0.5m	50	13.77	39.99	0.03	0.08
		100	47.36	19.83	0.04	0.12
	m	50	40.05	45.95	0.02	0.07
S2		100	174.39	24.51	0.03	0.10
	1.5m	50	79.38	48.16	0.02	0.06
		100	237.93	26.62	0.03	0.11
	2m	50	119.08	48.47	0.02	0.06
		100	345.70	28.84	0.03	0.10
	0.5m	50	13.29	18.70	0.05	0.13
		100	48.40	9.64	0.05	0.15
	m	50	36.76	29.46	0.04	0.10
S3		100	164.89	12.37	0.04	0.14
	1.5m	50	57.98	33.97	0.03	0.10
		100	230.71	13.29	0.04	0.13
	2m	50	87.13	32.70	0.03	0.10
		100	380.68	13.89	0.04	0.14

We try 1.5m, 2m, 2.5m, and 3m for the filter width value after we fix the beam width at m. The results are reported in Table 5.

We use the beam width value of m and filter width value of 2.5m in our main experiments, as we observe that at those values high-quality solutions are obtained in small CPU times.

7.4. Main Experiment

In our main experiment, we test the performances of our optimization algorithms and heuristic algorithms.

Table 5. Average performances of the filter width values, $m = 10$ and beam width $= m$

Set	Filter width	n	CPU Time	P	D1	D2
	1.5m	50	5.02	35.97	0.03	0.09
		100	17.914	29.28	0.04	0.11
	2m	50	6.654	37.19	0.03	0.09
S1		100	26.453	34.92	0.03	0.09
	2.5m	50	8.982	38.53	0.03	0.09
		100	31.638	35.00	0.03	0.09
	3m	50	9.779	38.53	0.03	0.09
		100	42.36	35.91	0.03	0.09
	1.5m	50	6.04	39.91	0.03	0.07
		100	18.98	18.84	0.04	0.13
	2m	50	8.38	43.53	0.02	0.08
S2		100	28.33	25.70	0.04	0.11
	2.5m	50	12.55	42.35	0.02	0.07
		100	41.31	24.89	0.04	0.12
	3m	50	12.86	44.18	0.02	0.07
		100	46.62	23.20	0.04	0.13
	1.5m	50	5.35	20.32	0.05	0.13
		100	19.41	9.32	0.05	0.16
	2m	50	7.32	23.64	0.04	0.11
S3		100	33.97	11.91	0.05	0.16
	2.5m	50	11.44	25.10	0.04	0.10
		100	48.72	12.01	0.05	0.15
	3m	50	11.82	23.69	0.04	0.12
		100	56.68	11.76	0.04	0.12

7.4.1. Optimization Algorithms

We first analyze the performance of our optimization algorithms. Table 6 reports the average and maximum solution times for small-sized problem instances with $m = 5$ agents and $n = 15, 20, 25$ tasks. We use such small instances as the B&B could not return solutions in reasonable time for larger-sized problem instances. The table also includes the number of nodes for B&B.

As can be observed from the table, the CA returns the exact nondominated set in negligible time. However, the solution time and number of nodes evaluated by the B&B algorithm are quite high even for small-sized problem instances.

Table 6. Solution times of the optimization algorithms

			B&B				CA	
			CPU Time		Number of nodes		CPU Time	
Set	m	n	Avg	Max	Avg	Max	Avg	Max
S1	5	15	2.65	7.97	2469	5445	0.51	0.97
		20	17.85	57.25	7793	18925	0.48	0.72
		25	27.95	100.06	9392	25520	0.57	2.26
S2	5	15	109.21	600.00	25030	108091	0.43	1.17
		20	277.44	600.00	45751	98727	0.64	1.05
		25	547.26	600.00	65804	84148	1.11	1.79
S3	5	15	174.99	600.00	30814	90105	0.47	1.28
		20	402.41	600.00	55314	87524	0.42	0.66
		25	562.22	600.00	72067	88453	0.70	1.17

For sets S2 and S3, the B&B algorithm could not return optimal solutions in our time limit of 600 s. Unlike the CA, the B&B algorithm returns the nondominated set in one execution. The nondominated set has many solutions, leading to the return of many partial and complete solutions by our B&B algorithm. Moreover, each partial solution is evaluated for two criteria; hence node eliminations are tougher compared to the single criterion problems.

For the CA, we consider some larger-sized instances, starting with $m = 10$ and $n = 50$ and increasing m and n values in respective increments of 10 and 50. We specify a time limit of 1800 s for each instance. In Table 7, we report on the average and maximum CPU times and the average number of nondominated points for each problem combination.

As can be observed from Table 7, the number of nondominated points increases linearly with increases in m or n. Conversely, as m or n increases, the time to find all nondominated points increases significantly. Hence, this significant increase in the CPU times can be attributed to the increases in the complexity of the integer models that return a single nondominated point. Note that when $n = 50$, the average CPU times are less than 10 s for most combinations. When $n = 150$, the average CPU times are greater than 500 s for most combinations, and the maximum CPU times are greater than 1800 s indicating that there are instances that could not be solved in 1800 s.

Table 7. Performance results of the CA

			CPU Time		|NS|
	m	n	Avg	Max	Avg
S1	10	50	4.04	7.71	16.10
		100	44.39	127.30	18.30
		150	212.94	844.29	37.60
	20	50	9.45	22.11	19.50
		100	69.80	196.07	28.10
S2	10	50	8.30	17.46	17.40
		100	328.84	1800.0(1)[a]	23.00
		150	822.68	1800.0(3)	26.50
	20	50	17. 54	27.66	11.50
		100	30.45	49.65	27.40
S3	10	50	4.41	9.24	16.10
		100	28.60	92.09	36.90
		150	544.79	1800.0(1)	45.67
	20	50	7.28	15.56	9.60
		100	91.51	469.33	27.50

[a]The figures in the parentheses give the number of instances (out of 10) remained unsolved in 1800sec.

7.4.2. *Heuristic Procedures*

We first discuss the performance of our beam search algorithms. Table 8 reports on the average and maximum solution times and the number of LPR problems solved for the two beam search algorithms. Table 9 shows the average solution time per LPR for the beam search algorithms and the average depth of the beam search tree.

Note from Table 9 that for Set S1, when $m = 20$ and $n = 100$, the average tree depth for BS1 is 50 at the termination, that is, 50 out of 100 tasks are searched along each path between the root and any one of the end nodes.

Table 8 reveals that the number of agents, m, and the number of tasks, n have significant effects on the solution times of the BS1 and BS2.

For fixed n, as m increases, the number of branches increases, which triggers an increase in the solution times.

Table 8. CPU times and number of LPs solved for the BS1 and BS2

			BS1				BS2			
			CPU Time		Number of LPs solved		CPU Time		Number of LPs solved	
	m	n	Avg	Max	Avg	Max	Avg	Max	Avg	Max
S1	10	50	4.27	13.66	289	468	33.36	55.24	2093	2392
		100	10.92	33.40	430	662	123.07	240.55	3876	5284
		150	86.95	369.42	587	1186	131.29	240.44	3987	5515
	20	50	10.25	14.20	572	808	104.45	174.05	3805	4645
		100	43.96	96.05	849	1283	758.69	1424.17	14080	18761
S2	10	50	6.19	30.25	358	491	40.05	62.34	2726	3250
		100	31.37	143.44	470	880	174.39	305	5552	7548
		150	116.79	460.31	835	1306	371.80	595.86	7841	10954
	20	50	51.30	108.93	764	937	192.03	398.47	6640	9407
		100	171.76	738.82	1138	1895	583.45	1095.57	12760	25051
S3	10	50	6.60	16.93	391	467	36.76	55.35	2655	3261
		100	24.85	83.35	529	954	164.89	320.67	5450	7309
		150	22.99	39.62	737	1161	446.09	773.81	8914	11625
	20	50	85.64	119.09	843	922	354.88	580.68	9519	10868
		100	275.16	655.77	1456	1801	805.88	1701.35	15894	24784

Table 9. Average solution times per LP and average tree depth for the BS1 and BS2

			Average CPU Time/LP		Average Tree Depth	
	m	n	BS1	BS2	BS1	BS2
S1	10	50	0.007	0.012	32.2	37.5
		100	0.011	0.021	48.8	63.5
		150	0.017	0.025	63.3	59.1
	20	50	0.012	0.019	32.1	37.7
		100	0.021	0.056	50.1	69.2
S2	10	50	0.007	0.013	38.7	40.4
		100	0.012	0.025	52.4	75.1
		150	0.020	0.038	88.0	97.8
	20	50	0.016	0.025	40.6	33.7
		100	0.024	0.038	62.3	52.2
S3	10	50	0.008	0.013	40.9	40.4
		100	0.013	0.025	55.9	76.4
		150	0.019	0.040	78.8	113.0
	20	50	0.023	0.033	44.0	41.0
		100	0.025	0.041	77.3	58.2

This is because our beam width increases as m increases. Similarly, for fixed m, an increase in n makes the tree deeper which in turn increases the solution times. Moreover, as m or n increases, the LPRs become more difficult to solve (Table 9).

Table 8 also reveals that the BS1 runs significantly faster than the BS2. For example, for Set S1, when $m = 10$ and $n = 100$, the average solution times for the BS1 and BS2 are 10.92 and 123.07 s, respectively. This is due to the fact that at each level the BS2 solves the LPR for each of the $m \times$ beam width nodes, whereas the BS1 evaluates $m \times$ beam width nodes by simple lower bounds and solves LPR only for the selected beam width nodes.

For the BS2, we observe that as the range of the processing times decreases, that is, as we move from Set S1 to Set S3, the solution times increase, with a few exceptions. For example, when $m = 10$ and $n = 150$, the average solution times of the BS2 are 131.29, 371.80, and 446.09 s for Set S1, S2, and S3, respectively. This is due to the increase in the number of LPR problems solved as we move from Set S1 to Set S3.

Compared to the BS2, the BS1 uses considerably fewer LPR problems; hence the processing time distributions have less significant effects on the solution times. Similarly, higher solution times are observed for Set S2 and Set S3 where the range of processing times is relatively lower.

Table 10 reports on the average performances of the BS1 and BS2 with respect to our three quality-related measures, namely P, D1, and D2. The table also gives the average number of nondominated solutions returned by the algorithms (|ANS|).

Table 10. Average performances of the BS1 and BS2

			BS1				BS2			
	m	n	P	$D1$	$D2$	|ANS|	P	$D1$	$D2$	|ANS|
S1	10	50	28.89	0.05	0.12	12.8	40.36	0.03	0.09	13.3
		100	22.21	0.05	0.14	12.4	33.10	0.03	0.09	15.0
		150	11.74	0.08	0.21	20.4	23.64	0.04	0.11	24.3
	20	50	26.65	0.05	0.11	14.3	34.95	0.03	0.09	16.1
		100	14.42	0.07	0.16	17.0	23.16	0.04	0.11	19.0
S2	10	50	32.24	0.03	0.10	13.7	45.95	0.02	0.07	14.3
		100	13.34	0.07	0.20	14.4	24.51	0.03	0.10	16.6
		150	11.59	0.07	0.19	15.3	20.21	0.05	0.13	18.5
	20	50	19.59	0.06	0.15	9.7	30.19	0.05	0.13	10.0
		100	11.83	0.09	0.22	13.6	17.62	0.07	0.20	15.5
S3	10	50	15.11	0.07	0.20	10.1	29.46	0.04	0.10	11.8
		100	5.33	0.07	0.23	15.9	12.37	0.04	0.14	20.2
		150	7.80	0.06	0.18	18.7	12.46	0.04	0.12	23.4
	20	50	17.07	0.11	0.25	6.9	33.03	0.09	0.25	8.1
		100	5.65	0.08	0.22	13.5	11.12	0.06	0.17	16.1

Table 11. CPU times and number of LPs solved for the TS, CAH and FBS

			TS		CAH		FBS			
			CPU Time		CPU Time		CPU Time		Number of LPs solved	
	m	n	Avg	Max	Avg	Max	Avg	Max	Avg	Max
S1	10	50	0.61	0.73	3.31	4.79	8.98	11.54	789	895
		100	2.14	2.39	9.74	41.36	31.64	51.85	1467	1969
		150	4.66	4.85	13.70	21.55	51.48	96.46	1431	2738
	20	50	1.05	1.27	6.15	9.99	29.96	46.52	1598	1980
		100	3.55	3.75	26.78	73.80	92.52	203.78	2444	4364
S2	10	50	1.13	1.40	9.51	19.41	12.55	17.88	944	1075
		100	3.69	4.48	22.63	61.45	41.31	69.93	1709	2316
		150	7.07	10.36	37.01	70.08	88.98	140.01	2240	3394
	20	50	1.72	2.28	16.22	24.74	28.86	43.94	1592	2147
		100	5.45	6.80	49.14	76.03	119.41	244.02	2818	4776
S3	10	50	0.58	0.61	2.60	4.93	11.44	17.44	941	1139
		100	2.06	2.20	8.16	17.47	48.72	83.73	1813	2360
		150	4.87	5.15	15.24	38.29	94.93	165.11	2261	3208
	20	50	0.97	1.01	5.36	10.29	48.06	64.90	1988	2194
		100	3.62	3.76	12.58	16.44	146.68	282.46	3153	4555

Table 10 reveals that for both algorithms, with a few exceptions, the solution quality deteriorates as the problem size, that is, m or n, increases and as we go from Set S1 to Set S3.

The BS2 has the advantage of using a thorough function and outperforms the BS1 in all quality-related measures as well as in the number solutions returned. Recall that the high-quality solutions by the BS2 are achieved at the expense of greater computational effort.

We now discuss the performances of the TS, CAH, and FBS. We set $\alpha = 1\%$ in the CAH, that is, solve each problem with $\alpha = 1\%$ gap. Our FBS uses a beam width of m, a filter width of $2.5m$ (see Section 7.3) and starts with the nondominated objective vectors returned by the TS.

Table 11 reports the average and maximum solution times of the TS, CAH, FBS, and the average and maximum number of the LPRs solved in the FBS.

From Tables 7 and 11, we see that the CAH runs significantly faster than the CA with a few exceptions (for Set S2 when $m = 10$, $n = 50$ and $m = 20$, $n = 100$). Such exceptions occur as the CA solves a single model in exponential time and gets an efficient solution whereas the CAH solves two models, each in exponential time and gets an approximate efficient solution, at each iteration.

It can be seen in Table 11 that the TS is the fastest of all three heuristic algorithms, it takes a maximum of 10.36 s over all problem instances. The FBS is the slowest but still provides quick solutions to all large-sized problem instances in reasonable times. Note that maximum CPU times spent by the FBS algorithm are less than 5 min.

The number of agents, m, significantly affects the solution times of all heuristic algorithms. For example, for Set S1 when $n = 100$, as m increases from 10 to 20, the average solution times of the TS, CAH, and FBS increase from 2.14 to 3.55 s, from 9.74 to 26.78 s, and from 31.64 to 92.52 s, respectively. The increase in the solution times for the TS can be attributed to its neighborhood search mechanism where the number of moves increases as m gets larger. The increase in the solution times for the CAH is mainly due to the fact that, as m gets larger the combinatorial subproblems become more difficult to solve. Conversely, the increase in solution time for the FBS can be attributed to the increase in the tree size and deteriorating performance of the LPR bound, with increases in m.

The number of tasks, n, also has a significant effect on the solution times. All solution times increase as n increases for fixed m. For example, for S2 when $m = 10$ as n increases from 100 to 150, the average CPU time for the TS increases from 3.69 to 7.07 s; for the CAH from 22.63 to 37.01 s, and for the FBS from 65.57 to 130.42 s. Similar to the effect of m, an increase in n triggers an increase in the neighborhood search moves for the TS, leading to harder combinatorial subproblems for the CAH, resulting in weaker lower bounds and a deeper tree for the FBS.

We now discuss the quality of the solutions for the TS, CAH, and FBS algorithms. Table 12 reports the average values of P, D1, and D2 and the average number of nondominated objective vectors ($|\text{ANS}|$).

Table 12 demonstrates that the performances of the TS and FBS deteriorate with increases in the problem size parameters m and n.

Note that the CAH performs consistently well over all problem instances by generating at least 39% of the nondominated objective vectors. We also observe that the average

Table 12. Average values of P, D1, D2, and |ANS| for the TS, CAH and FBS

				TS			CAH				FBS			
	m	*n*	*P*	D1	D2	|ANS|	P	D1	D2	|ANS|	P	D1	D2	|ANS|
S1	10	50	3.49	0.11	0.23	9	82.06	0.00	0.02	16.1	38.53	0.03	0.09	13.5
		100	3.78	0.11	0.28	8.5	73.11	0.01	0.03	18.3	35.00	0.03	0.09	15.2
		150	1.05	0.19	0.50	11.6	63.60	0.00	0.02	37.6	23.14	0.04	0.11	24.6
	20	50	0.45	0.15	0.31	8.3	82.29	0.00	0.01	19.5	37.29	0.03	0.10	15.7
		100	0.44	0.17	0.40	8.7	77.85	0.00	0.01	28.1	22.75	0.05	0.13	18.8
S2	10	50	16.79	0.08	0.23	9.6	76.86	0.00	0.02	17.4	42.35	0.02	0.07	14.3
		100	2.78	0.13	0.38	9.3	56.52	0.01	0.02	24.8	24.89	0.04	0.12	15.8
		150	2.26	0.12	0.33	10.5	50.81	0.01	0.03	28.8	21.05	0.05	0.14	18.1
	20	50	0.00	0.21	0.45	5.6	82.54	0.00	0.03	11.5	31.81	0.04	0.12	9.4
		100	2.43	0.22	0.51	7.3	68.51	0.00	0.02	27.4	16.05	0.07	0.18	15.9
S3	10	50	9.48	0.10	0.27	7.9	72.69	0.00	0.02	16.1	25.10	0.04	0.10	11.9
		100	1.89	0.11	0.34	12.5	51.67	0.00	0.02	36.9	12.01	0.05	0.15	19.9
		150	2.67	0.09	0.28	14.1	39.77	0.01	0.02	47.7	11.85	0.04	0.14	22.4
	20	50	2.00	0.24	0.41	6.2	91.52	0.00	0.03	9.6	29.27	0.09	0.25	7.9
		100	1.55	0.16	0.40	9.5	70.04	0.00	0.02	27.5	10.09	0.06	0.16	15.5

number of solutions returned by the CAH is larger than that of the FBS and TS. In almost all combinations, the average *P* value of the TS is below 10%, that is, it generates less than 10% of the nondominated objective vectors. Conversely, the average *P* value of the FBS is above 10%, showing its superior performance over the TS, in terms of reaching exact efficient objective vectors.

For D1 and D2 measures, the TS is the worst and the CAH is the best performing algorithm over all problem instances. For example, for Set S1 when $m = 10$ and $n = 100$, the average (D1, D2) values are (0.11, 0.28), (0.01, 0.03), and (0.03, 0.09) for the TS, CAH, and FBS, respectively.

We perform additional runs to observe the behaviors of FBS, CAH, and TS for large-sized problem instances. To avoid excessive computational time, in the FBS we decrease the beamwidth and filterwidth parameters to 0.2 and 0.5 m, respectively (one-fifth of their previous values) and in the CAH, we increase the α value to α =5% (five times of the previous value). We keep the parameters of the TS at their previous values of nonimplimit = 250, maxiter = 1000, tabu

Table 13. Performances of the CAH, FBS, and TS for large-sized problem instances

			CAH			FBS			TS		
			CPU Time		|ANS|	CPU Time		|ANS|	CPU Time		|ANS|
Set	*m*	*n*	Avg	Max	Avg	Avg	Max	Avg	Avg	Max	Avg
S1	50	200	118.60	156.53	20.8	261.58	407.53	14.4	30.79	36.06	6.4
		400	166.91	204.69	21.2	1183.90	1834.56	14.6	126.44	156.36	8.8
		600	170.66	227.83	15.6	1797.37	3600	11.6	327.86	361.66	8.6
		800	155.63	249.87	11.4	–	–	–	692.99	810.47	9.2
		1000	118.75	159.59	8	–	–	–	1367.23	1633.17	8.2
	100	200	167.37	181.94	15.6	1439.44	2112.37	12.6	82.55	87.26	4.4
		400	609.81	722.28	20.6	2703.80	3600	10.2	303.29	338.71	7.6
		600	863.12	1093.67	20.6	–	–	–	738.53	1035.58	8.8
		800	1110.16	1413.94	19.6	–	–	–	958.06	1092.99	7.2
		1000	1248.15	1248.15	17	–	–	–	2317.74	3600	9
S3	50	200	129.4	144.91	18.8	577.17	1295.45	9.4	30.66	33.71	11
		400	356.73	448.704	29	1658.55	2058.97	14.2	131.24	143.32	13.4
		600	388.89	568.886	25.6	1945.50	2708.73	13	332.38	419.13	17.2
		800	310.5	397.988	18.6	2020.06	2380.85	12.6	688.67	883.00	16.4
		1000	325.56	431.557	17.6	2853.11	3600	14.2	924.21	1328.72	16
	100	200	3600	3600	2	3432.04	3600	1.8	83.03	90.56	5.6
		400	–	–	–	–	–	–	263.52	290.11	10.6
		600	–	–	–	–	–	–	670.17	809.54	11.6
		800	–	–	–	–	–	–	1122.21	1456.30	11.6
		1000	–	–	–	–	–	–	1723.82	2321.73	17.8

tenure = 50. We report the CPU times and the average values of the number of solutions (|ANS|) returned by each algorithm in Table 13 for sets S1 and S3, m = 50, 100 and n = 200–1000, increasing in increments of 200. We generate five problem instances for each setting. We use a time limit of 1 h for each instance and report the solutions returned by the algorithms at the end of 1 h. The empty entries in the table indicate that the algorithms would fail to terminate in 1 h for the majority of the instances.

It is observed from Table 13 that all three algorithms are likely to fall into computational trouble as the problem size increases. The performance of the FBS deteriorates much faster than those of the CAH and TS. The significant increase in the solution times for the FBS with increases in n and m, can be attributed to the increases in the tree size and in the complexity of the solved LPR problems.

The results indicate that the CAH achieves a superior performance in terms of the number of solutions returned. The CAH returns solutions in less than 1 h for all problem instances in Set S1 and for all problem instances with m=50 in Set S3. However, due to its exponential time complexity, the CAH falls into computational trouble as the problem size increases. In S3 and m=100, no instance could be solved by the CAH in 1 h even when n = 200. In S3 and m=100, n = 200 about two solutions have been returned by the CAH at the termination, whereas TS solves this combination at an average solution time of 83.03 s by returning an average of 5.6 solutions. It is also observed that the TS returns heuristic solutions to all large-sized instances in Set 3 in less than half an hour, on average.

To summarize, the CAH should be favored due to its superior performance in terms of the closeness to, and coverage of, the nondominated set. For the large-sized instances that could not be solved by the CAH in reasonable time, TS should be favoured due to its superior solution time performance.

8. CONCLUSIONS

In this study, we consider a bicriteria multiresource generalized AP. Our criteria are the total workload over all agents and maximum workload among all agents. The maximum workload assigned among all agents, serves the fairness concerns of the decision maker, whereas the total workload serves his/her efficiency concerns.

For the small-sized problem instances, we aim to find all nondominated objective vectors and propose optimization algorithms, namely the B&B algorithm and the CA. Our B&B algorithm benefits from the LPRs of the various related problems, whereas the CA uses the successive solutions of the constrained optimization models. Our computational experiment has revealed that the B&B algorithm is likely to fall into computational trouble even for small-sized problem instances and the CA could solve small-sized problem instances with up to five agents and 25 tasks in about 1 s.

For medium- to large-sized problem instances with more than 20 agents, we try to find a set of good approximate nondominated objective vectors, and propose heuristic procedures, namely the beam search and FBS algorithms and CA-based heuristic. The CA-based heuristic follows the ideas used in the exact CA. The beam search and FBS algorithms use the TS algorithm to get an initial set of approximate nondominated objective vectors and the mechanisms derived for the B&B algorithm. Based on the results of our extensive computational study, one can conclude that the CA-based heuristic produces solutions that are very close to the exact nondominated objective vectors.

The results of our experiments on very large size-problems with up to 100 agents and 1000 tasks reveal that the CA heuristic outperforms the FBS and TS algorithms in terms of the number of solutions found. However, CA heuristic fails to solve many problem instances with 100 agents, in 1 h. These unsolved instances, could be solved by the TS algorithm in less than half an hour, but without offering any guarantee for the solution quality.

To the best of our knowledge, our study is the first reported attempt to solve the bicriteria multiresource generalized AP. Future research may consider different fairness measures such as minimizing the deviation between all workloads or minimizing the workload deviations from a specified ideal workload. The efficiency measures might be selected so as to reflect the relative importance of the agents, like total weighted workload.

ACKNOWLEDGMENT

The authors are grateful to the referees and the editor for their helpful and constructive comments.

REFERENCES

[1] R.E. Burkard, Selected topics on assignment problems, Discrete Appl Math 123 (2002), 257–302.
[2] G.M. Campbell and M. Diaby, Development and evaluation of an assignment heuristic for allocating cross-trained workers, Eur J Oper Res 138 (2002), 9–20.
[3] D.G. Cattrysse and L.N. Van Wassenhove, A survey of algorithms for the generalized assignment problem, Eur J Oper Res 60 (1992), 260–272.
[4] P. Czyżak, and A. Jaszkiewicz, Pareto simulated annealing- A metaheuristic technique for multiple-objective combinatorial optimization, J Multi-Criteria Decis Anal 7 (1998), 34–47.
[5] M. Dell'Amico, S. Martello, "Linear assignment," in: M. Dell'Amico, F. Maffioli, S. Martello (Editors), Annotated bibliographies in combinatorial optimization, Wiley, Chichester, England, 1997.
[6] M. Ehrgott and X. Gandibleux, "Multiobjective combinatorial optimization—theory, methodology, and applications," in: M. Ehrgott, X. Gandibleux (Editors), Multiple criteria optimization: State of the art annotated bibliographic surveys,

Kluwer Academic Publishers, Massachusetts, USA (2002), 369–444.
[7] M. L. Fisher, and R. Jaikumar, A generalized assignment heuristic for vehicle routing, Networks 11 (1981), 109–124.
[8] D. Garrett and D. Dasgupta, "Multiobjective landspace analysis and the generalized assignment problem," in: V. Maniezzo, R. Battiti, J. P. Watson (Editors), Learning and intelligent optimization, lecture notes in computer science, Springer, Berlin Heidelberg, 2008, 110–124.
[9] B. Gavish and H. Pirkul, Computer and database location in distributed computer systems, IEEE Trans Comput 35 (1986), 583–590.
[10] B. Gavish and H. Pirkul, Algorithms for the multi-resource generalized assignment problem, Manage Sci 37 (1991), 695–713.
[11] F. Glover and M. Laguna, Tabu search, Kluwer Academic Publishers, Norwell, MA, 1997.
[12] M. D. Grigoriadis, D.J. Tang, and L.S. Woo, "Considerations in the optimal synthesis of some communications networks," in: 45th Joint National Meeting of ORSA/TIMS, Boston, MA, 1974.
[13] Y.Y. Haimes D.A. Wisner, and L.S. Lasdon, On bicriterion formulation of the integrated systems identification and system optimization, IEEE Trans Syst Man Cybern (1971), 296–297.
[14] N. Honda, "Backtrack beam search for multiobjective scheduling problem," in: T. Tanino, T. Tanaka, M. Inuiguchi (Editors), Multi-objective programming and goal programming: Theory and applications. AISC, Springer, Heidelberg, 2003, pp. 147–152.
[15] Ö. Karsu and M. Azizoğlu, The multi-resource agent bottleneck generalized assignment problem, Int J Prod Res 50 (2012), 309–324.
[16] S. Martello and P. Toth, The bottleneck generalized assignment problem, Eur J Oper Res 83 (1995), 621–638.
[17] S. Martello and P. Toth, Knapsack problems: Algorithms and computer implementations, Wiley, Chichester, 1990.
[18] J.B. Mazzola and A.W. Neebe, Bottleneck generalized assignment problems, Eng Cost Prod Econ 14 (1988), 61–65.
[19] J.B. Mazzola and A.W. Neebe, An algorithm for the bottleneck generalized assignment problem, Comput Oper Res 20 (1993), 366–362.
[20] J.B. Mazzola, A.W. Neebe, and C.V.R. Dunn, Production planning of a flexible manufacturing system in material requirements planning environment, Int J Flexible Manuf Syst 1 (1989), 115–142.
[21] J.B. Mazzola and S.P. Wilcox, Heuristics for the multi-resource generalized assignment problem, Naval Res Logistics 48 (2001), 468–483.
[22] S. Mitrović-Minić and A. Punnen, Local search intensified: Very large scale variable neighborhood search for the multi-resource generalized assignment problem, Discrete Optim 6 (2009), 370–377.
[23] T.E. Morton and D.W. Pentico, Heuristic scheduling systems, Wiley, New York, 1993.
[24] R.A. Murphy, "A private fleet model with multi-stop backhaul," in: Working Paper 103 (1986), Optimal Decision Systems, Green Bay, WI.
[25] M. Ozlen and M. Azizoglu, Generating all efficient solutions of a rescheduling problem on unrelated parallel machines, Int J Prod Res 47 (19) (2009), 5245–5270.
[26] H. Pirkul, An integer programming model for the allocation of databases in a distributed computer system, Eur J Oper Res 26 (1986), 401–411.
[27] A. Ponte, P. Luis, and J.R. Figueira, "On beam search for Multicriteria combinatorial optimization problems," in: CPAIOR 2012, Springer, Berlin Heidelberg, 2012, pp. 307–321.
[28] A.P. Punnen and Y.P. Aneja, Minmax combinatorial optimization, Eur J Oper Res 81 (1995) 634–643.
[29] G.T. Ross and R.M. Soland, Modeling facility location problems as generalized assignment problems, Manage Sci 24 (1977), 345–357.
[30] C.R. Seshan, Some generalisations of the time minimising assignment problem, J Oper Res Soc 32 (1981), 489–494.
[31] A. Shtub, Modelling group technology cell formation as a generalized assignment problem, Int J Prod Res 27 (1989), 775–778.
[32] M. Sun, A branch-and-bound algorithm for representative integer efficient solutions in multiple objective network programming problems, Networks 62 (2013), 56–71.
[33] M. Yagiura, S. Iwasaki, T. Ibaraki, and F. Glover, A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem, Discrete Optim 1 (2004), 87–98.
[34] C.W. Zhang and H. L. Ong, An efficient solution to biobjective generalized assignment problem, Adv Eng Softw 38 (2007), 50–58.