Computers and Operations Research 106 (2019) 76–90

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

An exact algorithm for the minimum squared load assignment problem

Özlem Karsu [a,*], Meral Azizoglu [b]

[a] Department of Industrial Engineering, Bilkent University, Ankara 06800, Turkey
[b] Department of Industrial Engineering, Middle East Technical University, Ankara 06800, Turkey

ARTICLE INFO

Article history:
Received 12 May 2018
Revised 21 February 2019
Accepted 22 February 2019
Available online 26 February 2019

Keywords:
Assignment problem
Squared load
Branch
Bound

ABSTRACT

In this study, we consider an assignment problem with the objective to minimize the sum of squared loads over all agents. We provide mixed integer nonlinear and linear programming formulations of the problem and present a branch and bound algorithm for their solution. The results of our computational experiment have shown the satisfactory behavior of our branch and bound algorithm.

© 2019 Elsevier Ltd. All rights reserved.

1. Introduction

Assignment problems are relevant to many practical applications; hence a large body of the operational research literature has been devoted to finding efficient solution algorithms for these problems.

The classical assignment problem (AP) assigns n tasks to m agents, where each task is to be performed by one agent and each agent has to perform one task. The aim is to minimize the total assignment cost (time). Several versions of this classical AP model have been studied, including but not limited to the k-cardinality assignment, the bottleneck assignment (BAP), the balanced assignment, the minimum deviation assignment, the lexicographic bottleneck, the semi-assignment, and the categorized assignment models (see Pentico, 2007, and the related references therein).

The classical AP assumes that a single task will be assigned to an agent, however, in many practical situations an agent can perform multiple tasks as long as its capacity permits. The resulting problem is referred to as generalized assignment problem and a survey by Cattrysse and Van Wassenhove (1992) discusses a variety of its applications. If the agent capacities are determined with respect to multiple resources the problem becomes a multiple resource generalized assignment problem (Karsu and Azizoğlu, 2012, 2014).

When the objective is minimizing the maximum cost (instead of minimizing the total cost), the generalized assignment problem and multiple resource generalized assignment problem are referred to as bottleneck generalized assignment problem and bottleneck multiresource generalized assignment problem, respectively. There are two versions of these bottleneck models: task bottleneck and agent bottleneck, in which the maximum cost over all tasks and the maximum cost over all agents are minimized, respectively (Mazzola and Neebe, 1988; Karsu and Azizoğlu, 2012). A further variation of the BAP is the imbalanced time minimizing assignment problem, where n tasks are to be assigned to m ($m < n$) agents, and some agents will be assigned more than one task. The objective is minimizing the time by which all tasks are completed.

The quadratic assignment problem (QAP), is another important variant of the AP that uses a quadratic objective function of the following form (Lawler, 1963):

$$\sum_{i,j=1}^{n} \sum_{k,p=1}^{n} c_{ijkp} x_{ik} x_{jp}$$

Many real life problems such as facilities location, parallel and distributed computing, and combinatorial data analysis can be modeled as QAPs. Loiola et al. (2007) state that the QAP is one of the hardest NP-hard problems. Integer linear programming, mixed integer linear programming, permutation-based, trace and graph formulations of the QAP have been developed and exact solution methods such as branch and bound, branch and cut, Bender's decomposition, and dynamic programming have been proposed (Loiola et al., 2007; Burkard, 2013; Drezner, 2015). A relaxed version of the QAP is the quadratic semi-assignment

* Corresponding author.
E-mail address: ozlemkarsu@bilkent.edu.tr (Ö. Karsu).

https://doi.org/10.1016/j.cor.2019.02.011
0305-0548/© 2019 Elsevier Ltd. All rights reserved.

problem (QSAP), in which n objects (tasks) are assigned to m locations (agents) so as to minimize the overall distance covered by the flow of materials among different objects. Unlike the QAP, the number of objects is not necessarily equal to the number of locations, i.e., the assignment constraints are replaced by semi-assignment constraints (Pitsoulis, 2009). The QSAP also appears in task allocation to processors so that an overall cost, that includes the assignment cost, the processor usage cost and the communication cost between the tasks of different processors, is minimized. The QSAP is introduced by Greenberg (1969) and shown to be strongly NP-hard by Sahni and Gonzalez (1976). Several studies in the literature have considered the QSAP and the related task allocation problems. The suggested solution approaches include integer programming and column generation methods (Ernst et al., (2006)), and mathematical programming and branch-and-bound algorithms (Billionnet et al., 1992; Magirou and Milis, 1989; Sinclair, 1987; Stone, 1977). Milis and Magirou (1995) define lower bounds for the QAP using the task allocation problem. Malucelli and Pretolani (1995) develop a class of lower bounds for the QSAP and Malucelli (1996) defines some polynomially solvable cases. Saito et al. (2009) discuss the theoretical aspects of the quadratic semi-assignment polytope. Drwal (2014) studies a special case of the QSAP, which assigns n tasks to m server machines while minimizing the sum of worst-case processing times. Several metaheuristic algorithms are proposed for the QSAP (Wang and Punnen, 2017) and for some of its special cases (Punnen and Wang, 2016).

In this paper, we consider an assignment problem with n tasks and m agents, where each task is assigned to one agent. Agents can perform multiple tasks; however there is a concern for distributing the total load (cost) over all agents in a balanced way.

To achieve this, we suggest a fairness-encouraging (or equity-encouraging) objective function: minimizing the sum of squared loads of all agents. This objective helps achieving a balanced distribution without sacrificing much from the total load (cost).

Ensuring a balanced (fair) workload allocation is an important concern in task assignment problems, hence various ways of incorporating fairness into assignment decisions have been discussed in a number of papers (see Karsu and Morton, 2015 and the references therein). One of the methods used to ensure a balanced allocation of a good (bad) is maximizing (minimizing) a specific Schur-concave (Schur-convex) function that aggregates the outcomes (Karsu and Morton, 2015). The sum of squared loads is such a Schur-convex function and we minimize it en route to distributing the workload equitably in a task assignment problem. The function is a symmetric function, hence encourages fairness and is in line with the Pigou-Dalton principle of transfers. Using a symmetric function of the load vector is important since it ensures that the agents are identical. That is, in a setting with two agents, two solutions with load distributions (3, 6) and (6, 3) will have the same total squared load value ($9+36 = 45$). It is also in line with the Pigou Dalton principle of transfers, which states that transferring some load from a relatively worse-off agent to a relatively better-off one should result in a more desirable solution. Consider two load allocations (3,6) and (4,5). The total load is the same in both allocations (i.e. they have the same level of efficiency). However the second allocation is more fair, as it can be obtained from the first one by transferring 1 unit of load from a busy agent (agent 2) to a less busy one (agent 1). This is reflected in the sum of squared loads values of the two distributions: 45 versus 41. One could also use other Schur-convex function forms, but the sum of squares functions (and in general convex functions of the form $\sum_i y_i^\alpha : \alpha \geq 1$) are considered appropriate for ensuring fairness in many applications (see for example Martin et al. (2013) for scheduling and Lulli and Odoni (2007) for air traffic flow management applications).

To the best of our knowledge, this article is the first reported attempt for solving the minimum squared load assignment problem.

The rest of the article is organized as follows: We define the problem and give the associated mathematical models in Section 2. Section 3 presents the branch and bound algorithm. Section 4 discusses the results of our computational experiment to test the performance of the mathematical model and branch and bound algorithm. In Section 5, we give some concluding remarks and suggestions for future work.

2. Problem definition

We consider n tasks to be assigned to m agents. We define a single parameter p_{ij} to denote the time required by task j if performed by agent i. We assume that p_{ij} values are integers. We use a binary variable x_{ij} to explain our assignment decisions where

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

Our objective is to minimize the sum of squared loads over all agents.

In this section we give the nonlinear and linear programming formulations of the sum of squared loads problem.

2.1. The sum of squared loads problem-non-linear model

The nonlinear programming formulation of our problem is as stated below.

The sum of squared loads problem—non linear model (SSL-NL)

$$\begin{aligned} &\textit{Min } ZSQ \\ &\text{s.t.} \\ &\sum_{j=1}^{n} x_{ij} \geq 1 \quad \forall i = 1, ..., m \end{aligned} \tag{1}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall j = 1, ..., n \tag{2}$$

$$\sum_{i=1}^{m} \left(\sum_{j=1}^{n} p_{ij} x_{ij} \right)^2 = ZSQ \tag{3}$$

$$x_{ij} = 0 \text{ or } 1 \quad \forall i = 1, ..., m, \ \forall j = 1, ..., n \tag{4}$$

The objective function is to minimize the total squared load over all agents. Constraint set (1) ensures that at least one task is assigned to each agent and constraint set (2) ensures that each task is assigned to one agent. Constraint set (3) defines the total squared load.

2.2. The sum of squared loads problem-linear model

The sum of squared loads model is a non-linear integer program.

We linearize the model by introducing a continuous variable y_{ijk}, which is defined below:

$$\boldsymbol{y_{ijk}} = \begin{cases} 1 \text{ if tasks } j \text{ and } k \text{ are both assigned to agent } i, \text{ i.e., } & x_{ik} = x_{ij} = 1 \\ 0 & \text{otherwise} \end{cases}$$

Using the new decision variable, our objective function ZSQ can be rewritten in linear terms, as follows:

$$ZSQ = \sum_{i=1}^{m}\left(\sum_{j=1}^{n} p_{ij}x_{ij}\right)^2 = \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k>j}^{n} 2p_{ij}p_{ik}x_{ij}x_{ik} + \sum_{i=1}^{m}\sum_{j=1}^{n} {p_{ij}}^2 x_{ij} = \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k>j}^{n} 2p_{ij}p_{ik}y_{ijk} + \sum_{i=1}^{m}\sum_{j=1}^{n} {p_{ij}}^2 x_{ij}$$

We now state the linear form of our model that uses the linear form of ZSQ.

The sum of squared loads problem-linear model (SSL-L)

Min ZSQ

s.t.

$$\sum_{j=1}^{n} x_{ij} \geq 1 \quad \forall i = 1, ..., m \tag{5}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall j = 1, ..., n \tag{6}$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k>j}^{n} 2p_{ij}p_{ik}y_{ijk} + \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}^2 x_{ij} = ZSQ \tag{7}$$

$$y_{ijk} \geq x_{ij} + x_{ik} - 1 \ \forall i = 1, ..., m \ \forall j = 1, ..., n \ \forall k > j \tag{8}$$

$$y_{ijk} \geq 0 \ \forall i = 1, ..., m \ \forall j = 1, ..., n \ \forall k > j \tag{9}$$

$$x_{ij} = 0 \text{ or } 1 \quad \forall i = 1, ..., m, \ \forall j = 1, ..., n \tag{10}$$

The objective function is to minimize the total squared load over all agents. Constraint set (5) ensures that at least one task is assigned to each agent and constraint set (6) ensures that each task is assigned to one agent. Constraint set (7) defines the total squared load. Constraint sets (8) and (9) support the definition of y_{ijk} variables. The assignment restrictions are given by constraint set (10).

We let ZSQ^* be the optimal objective function value of the above model and $Z_{tot}(P)$ be the total load value, $\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}x_{ij}$ value of its optimal solution.

3. Branch and bound algorithm

In this section, we present a branch and bound (B&B) algorithm that finds an optimal solution to the minimum total squared load over all agents problem (whose nonlinear and linear models (SSL-NL and SSL-L) are presented in Section 2).

At each level of the B&B tree we select a task. At level r, we consider task r and generate m nodes: each representing the assignment of task r to one of the m agents. Fig. 1 illustrates our branching scheme.

A partial solution to the problem, i.e., a node of the B&B tree, gives a set of assigned tasks (S) together with their agents.

For each node, we calculate a lower bound (a set of lower bounds discussed below are considered in a hierarchical way) and eliminate the node if the lower bound is no smaller than the upper bound. If all nodes are eliminated at any level then we backtrack to the previous level. Among the non-eliminated nodes we select the one having the smallest lower bound value for generating new branches. We terminate whenever we reach the root node.

3.1. Lower bounds

In this section, we discuss our procedures to find lower bounds on the minimum total squared load value. Our lower bounds are based on the solutions of the following minimum total cost (load) assignment model.

The sum of loads problem-linear model

$$Min \ Z_{tot} = \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}x_{ij}$$

s.t.

$$\sum_{j=1}^{n} x_{ij} \geq 1 \quad \forall i = 1, ..., m \tag{11}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall j = 1, ..., n \tag{12}$$

$$x_{ij} = 0 \text{ or } 1 \quad \forall i = 1, ..., m, \ \forall j = 1, ..., n \tag{13}$$

We let Z_{tot}^* be the optimal objective function value of the above model. The model is a classical minimum cost network flow formulation for which total unimodularity property holds. This follows that if all p_{ij}values are integer, once the integrality constraints on the x_{ij}values are relaxed, the optimal x_{ij}values are either 0 or 1 (we refer the reader to Ahuja et al. (1993), for network flow models and in-depth treatment of the total unimodularity property). Hence the optimal solution can be found in polynomial time using LP software.

Through the following theorem we show that Z_{tot}^* is a lower bound on the total load of our problem.

Theorem 1. $Z_{tot}^* \leq Z_{tot}(P)$

Proof. : Z_{tot}^* is a lower bound on the total load of all problems that include constraint sets (1), (2) and (4). Hence Z_{tot}^* is a lower bound on the total load of our problem that minimizes the total squared load. #

Consider the following dual model of the minimum total load assignment problem.

The dual model for the sum of loads problem-linear model

$$Max \ Z_{tot} = \sum_{i=1}^{m} u_i + \sum_{j=1}^{n} v_j$$

s.t.

$$u_i + v_j \leq p_{ij} \quad \forall i = 1, ..., m, \ \forall j = 1, ..., n \tag{14}$$

$$u_i \geq 0 \quad \forall i = 1, ..., m \tag{15}$$

$$v_j \text{ unrestricted in sign } \forall j = 1, ..., n \tag{16}$$

We denote the optimal dual solution as (u_i^*, v_j^*). Z_{tot}^* is the optimal objective function value, i.e., $Z_{tot}^* = \sum_{i=1}^{m} u_i^* + \sum_{j=1}^{n} v_j^*$. To find Z_{tot}^* one may solve the dual program.

We now discuss the extension of the total load problem to any partial solution (node) S, where Set A is the set of agents having no task assignment and Set B is the set of not-yet-assigned tasks. The resulting model is as stated below:

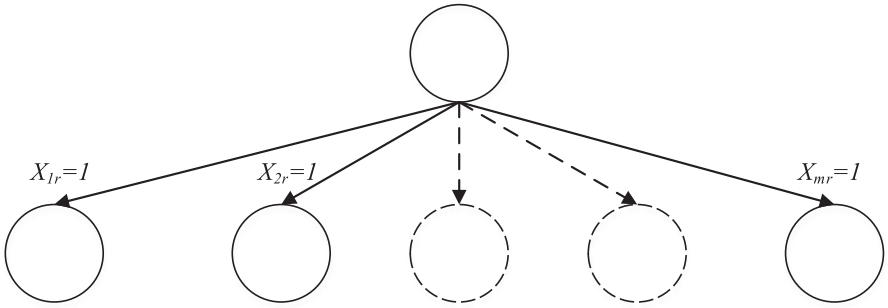

Fig. 1. The branching scheme.

The sum of loads problem for partial solution *S*-linear model

$$Min\ Z_{tot}(S) = \sum_{i=1}^{m} \sum_{j \in B} p_{ij} x_{ij}$$

s.t.

$$\sum_{j \in B} x_{ij} \geq 1 \quad i \in A \tag{17}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad j \in B \tag{18}$$

$$x_{ij} = 0 \text{ or } 1 \quad \forall i = 1, ..., m,\ j \in B \tag{19}$$

We let $Z_{tot}^*(S)$ be the optimal objective function value of the above model.

Now consider the following dual model of the assignment problem for node *S*.

The dual model for the sum of loads problem for partial solution *S*

$$Max\ Z_{tot}(S) = \sum_{i \in A} u_i + \sum_{j \in B} v_j$$

s.t.

$$u_i + v_j \leq p_{ij} \quad i \in A,\ j \in B \tag{20}$$

$$u_i \geq 0 \quad i \in A \tag{21}$$

$$v_j \text{ unrestricted in sign} \quad j \in B \tag{22}$$

We denote the optimal dual solution as $(u_i^*(S), v_j^*(S))$. The optimal objective function value is $Z_{tot}^*(S) = \sum_{i \in A} u_i^*(S) + \sum_{j \in B} v_j^*(S)$

Through the following theorem we show that one may also benefit from the optimal dual values at the root node, i.e., (u_i^*, v_j^*) values, while finding lower bounds on the total load value at partial solution *S*.

Theorem 2. $\sum_{i \in A} u_i^*(S) + \sum_{j \in B} v_j^*(S) \geq \sum_{i \in A} u_i^* + \sum_{j \in B} v_j^*$

Proof. : An optimal solution at the root node, i.e., (u_i^*, v_j^*) values, is feasible for the dual problem solved at node *S*. The feasibility is ensured as: $u_i^* + v_j^* \leq p_{ij}\ \forall i, \forall j$ at the root node, therefore $u_i^* + v_j^* \leq p_{ij}\ i \in A\ j \in B$ for any subsets of agents (A) and tasks (B).

Note that the objective function value of any feasible solution provides a lower bound on the optimal solution value $\sum_{i \in A} u_i^*(S) + \sum_{j \in B} v_j^*(S)$ as the dual objective is of maximization type. #

Table 1
Data of the example problem, total load minimizing solution and optimal values of the dual variables.

Agent/Task	1	2	3	4	5	6	7	8	9	10	u_i^*
1	18	13	13	12	16	18	16	13	10	10	2
2	10	18	15	10	12	17	15	11	7	7	0
3	7	8	12	11	15	15	14	14	8	19	0
v_j^*	7	8	11	10	12	15	14	11	7	7	104

Table 2
Data of the reduced problem, total load minimizing solution and optimal values of the dual variables.

Agent/Task	3	4	5	6	7	8	9	10	$u_i^*(S)$
1	13	12	16	18	16	13	10	10	–
2	15	10	12	17	15	11	7	7	–
3	12	11	15	15	14	14	8	19	0
$v_j^*(S)$	12	10	12	15	14	11	7	7	

Theorem 2 follows that the optimal dual variables at a particular node may be used to find lower bounds for all nodes emanating from that node.

We illustrate the lower bounds on the total load values through the following 10-task and 3-agent assignment problem instance.

The optimal solution of the minimum total load problem assigns Task 3 to Agent 1, Tasks 4, 5, 8, 9, and 10 to Agent 2 and all other tasks to Agent 3. The objective function value of the solution, $Z_{tot}^* = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij} = 104$.

Table 1 includes the optimal values of the dual variables.

$$Z_{tot}^* = \sum_{i=1}^{m} u_i^* + \sum_{j=1}^{n} v_j^* = 2 + 0 + 0 + 7 + 8 + 11 + 10 + 12 + 15 + 14 + 11 + 7 + 7 = 104$$

The optimal solution of the minimum total squared load problem assigns Tasks 3 and 7 to Agent 1, Tasks 4, 5, 8, 10 to Agent 2 and all other tasks to Agent 3.

The objective function value of this solution is $ZSQ = \sum_{i=1}^{m} (\sum_{j=1}^{n} p_{ij} x_{ij})^2 = 3885$ and the resulting total load $Z_{tot}^*(P) = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij} = 107$. Note that $Z_{tot}^* = 104 \leq Z_{tot}\ (P) = 107$.

Now consider a partial schedule *S* where Task 1 is assigned to Agent 2 and Task 2 is assigned to Agent 1. The reduced problem has 8 tasks and 3 agents as seen in Table 2.

The optimal solution of the minimum total load problem assigns Tasks 4, 5, 8, 9, 10 to Agent 2 and all other tasks to Agent 3. The optimal dual solution at that node is also given in Table 2.

The objective function value of the solution, $Z_{tot}^*(S) = 88$. The total load of the already-fixed assignments (Task 1 to Agent 2 and Task 2 to Agent 1) is $13 + 10 = 23$. Hence a lower bound on the total load is $18 + 13 + 88 = 111$.

Using the dual solution at node S, a lower bound on the total load of the remaining tasks is

$$\sum_{i\in A} u_i^*(S) + \sum_{j\in B} v_j^*(S) = 1 + 0 + 12 + 10 + 12 + 15 + 14 + 11 + 7 + 7 = 88.$$

Hence a lower bound on the total load of the partial schedule with assigned tasks 1 and 2 is $13 + 10 + 88 = 111$.

The lower bound found by using the dual variables of the root node is

$$\sum_{i\in A} u_i^* + \sum_{j\in B} v_j^* = 0 + 11 + 10 + 12 + 15 + 14 + 11 + 7 + 7 = 87.$$

Note that $\sum_{i\in A} u_i^* + \sum_{j\in B} v_j^* = 87 \le 88 = \sum_{i\in A} u_i^*(S) + \sum_{j\in B} v_j^*(S)$

One may solve the assignment problem at any node and benefit from the dual variables to eliminate the nonpromising descendant nodes without solving the corresponding dual problem (see Rinnooy Kan et al., 1975, for the minimum total cost single machine scheduling problem). An alternative option is to solve the assignment problem only at the root node and use the dual variables of the root node solution to find lower bounds at all other nodes (see Azizoglu and Kirca, 1999, for the minimum total cost unrelated parallel machine scheduling problem). In this study, we use a different approach by solving the assignment problem at some defined levels and use the dual variables for a number of descendant levels.

Finding lower bounds on the optimal total squared load value

We let LB_{ZTOT} be any valid lower bound on the total load. One valid lower bound on the total load value is the minimum possible total load value, i.e., objective function value of the minimum total load problem. Theorem 3 below defines a lower bound on the total squared load that is based on the equal distribution of LB_{ZTOT} to the agents.

Theorem 3. LB_{ZTOT}^2/m is a valid lower bound on the optimal total squared load value.

Proof. : An optimal distribution of LB_{ZTOT} over all agents gives a lower bound on the optimal total squared load value as LB_{ZTOT} is a lower bound on the total load of any feasible solution.

We now show that en route to minimizing total squared load, LB_{ZTOT} is evenly distributed among the agents, i.e., the load of each agent is LB_{ZTOT}/m.

When the load is evenly distributed among the agents, the total squared load of any agent is $(LB_{ZTOT}/m)^2$. Now assume e units of load are removed from one agent and put over the load of another agent.

Such a move will change the objective function value by

$$\begin{aligned}&(LB_{ZTOT}/m - e)^2 + (LB_{ZTOT}/m + e)^2 - 2*(LB_{ZTOT}/m)^2\\ &= (LB_{ZTOT}/m)^2 - 2eLB_{ZTOT}/m + e^2 + (LB_{ZTOT}/m)^2\\ &\quad + 2eLB_{ZTOT}/m + e^2 - 2*(LB_{ZTOT}/m)^2\\ &= 2e^2 > 0 \text{ as } e > 0.\end{aligned}$$

Note that any movement towards an uneven distribution of the load will increase the total squared load value. Hence the total squared load over all agents, i.e.,$m(LB_{ZTOT}/m)^2 = LB_{ZTOT}^2/m$ is a lower bound on the optimal total squared load value. #

We now discuss the extension of LB_{ZTOT} to a partial schedule S where the load of agent i is $L_i(S)$ ($L_i(S)$ is the load due to the already-fixed assignments of the partial solution). We let $LB_{ZTOT}(S)$ be a valid lower bound on the total load of the remaining assignments for node S. $LB_{ZTOT}(S)$ may be found either using the optimal solution of the total load problem at node S or the optimal dual variables of the total load problem found at any parent node (Recall Theorem 2).

We define two lower bounds on the optimal squared load value. We use the following notation in bound calculations:

$L_{TOT}(S)$ = the realized total load over all agents at node S

$$L_{TOT}(S) = \left(\sum_{i=1}^{m} L_i(S)\right)$$

$L_{MAX}(S)$ = the realized maximum load over all agents at node S

$$L_{MAX}(S) = Max_i L_i(S)$$

Average load based bounds: These lower bounds aggregate the current loads of the agents, hence use total load $L_{TOT}(S)$ while making the distribution.

Theorem 4 below defines a lower bound on the total squared load using aggregate realized load and a lower bound on the remaining total load.

Theorem 4. $(L_{TOT}(S) + LB_{ZTOT}(S))^2/m$ is a valid lower bound on the total squared load of node S.

Proof. : Using the result of Theorem 3, a lower bound on the total squared load, can be stated as follows.

$$m((L_{TOT}(S) + LB_{ZTOT}(S))/m)^2 = (L_{TOT}(S) + LB_{ZTOT}(S))^2/m.\#$$

Note that one can calculate $LB_{ZTOT}(S)$ by solving the (reduced) minimum total load problem at node S or using the dual variables of one its parent nodes based on Theorem 2. We call the average load based bound that uses the $LB_{ZTOT}(S)$ from the minimum total load problem LB1 and the average load based bound that calculates $LB_{ZTOT}(S)$ using the dual variables LB2.

Fill up strategy based bounds: These lower bounds use the current load of each agent, i.e., individual L_i values, while making the distribution.

The remaining load $LB_{ZTOT}(S)$ is distributed to the agents using Procedure 1.

Procedure 1. Two cases arise:

Case 1. $mL_{MAX}(S) - L_{TOT}(S) \le LB_{ZTOT}(S)$

$LB_{ZTOT}(S) - (mL_{MAX}(S) - L_{TOT}(S))$ units of load are distributed evenly over $L_{MAX}(S)$

The resulting load is

$$\begin{aligned}&LB_{ZTOT}(S) - (mL_{MAX}(S) - L_{TOT}(S)) + mL_{MAX}(S)\\ &= LB_{ZTOT}(S) + L_{TOT}(S)\end{aligned}$$

Hence a lower bound on the optimal squared load is $m((LB_{ZTOT}(S) + L_{TOT}(S))/m)^2 = (LB_{ZTOT}(S) + L_{TOT}(S))^2/m$

Note that in this case the fill up strategy based bound is the same as the average load based bound.

Case 2. $mL_{MAX}(S) - L_{TOT}(S) > LB_{ZTOT}(S)$

In this case we distribute $LB_{ZTOT}(S)$to the agents starting from the one having the minimum load, i.e., L_i value. The agent with the minimum load is loaded till its load reaches the second minimum L_i value. We then update the remaining load and distribute it evenly between two agents till their loads approach to the

third minimum load. We again update the remaining load and distribute it evenly among three agents till their loads become equal the fourth minimum load. We continue in this manner till all $LB_{ZTOT}(S)$units of load are distributed.

We let LL_i (S) be load of agent i returned by Procedure 1.

Theorem 5. Procedure 1 returns a valid lower bound on the optimal total squared load value for node S.

Proof. : Procedure 1 distributes $LB_{ZTOT}(S)$ such that the load of any agent cannot be moved to any other agent without increasing the total squared load.

Let LL_i (S) be load of agent i returned by Procedure 1. Two cases arise for the movement of load from agent i to another agent j

Case 1. $LL_i(S) > LL_j(S)$

In this case $LL_i(S) = L_i(S)$ and any movement from agent i is not possible.

Case 2. $LL_i(S) \leq LL_j(S)$

Assume e units of load are removed from agent i and assigned to agent j. Such a move will change the objective function value by

$$\begin{aligned}
&(LL_i(S)-e)^2 + \left(LL_j(S)+e\right)^2 - LL_i(S)^2 - LL_j(S)^2 \\
&= LL_i(S)^2 - 2eLL_i(S) + e^2 + LL_j(S)^2 + 2eLL_j(S) \\
&\quad + e^2 - LL_i(S)^2 - LL_j(S)^2 \\
&= -2eLL_i(S) + 2eLL_j(S) + 2e^2 \\
&= 2e\left(LL_j(S) - LL_i(S)\right) + 2e^2 > 0 \text{ as } LL_i(S) \leq LL_j(S) \text{ and } e > 0.
\end{aligned}$$

Note that any movement between two agents cannot be done without increasing the total squared load value. Hence Procedure 1 guarantees to distribute $LB_{ZTOT}(S)$ among the agents such that the resulting total squared load is a lower bound on the optimal total squared load. #

Similar to the case in the average load based bounds, one can calculate LB_{ZTOT} (S) by solving the (reduced) minimum total load problem at node S or using the dual variables of one its parent nodes based on the result of Theorem 2. We call the fill up strategy based bound that uses the LB_{ZTOT} (S) from the minimum total load problem LB3 and the fill up strategy based bound that calculates LB_{ZTOT} (S) using the dual variables LB4.

At a partial solution S, given the same $LB_{ZTOT}(S)$, the fill-up strategy based bound dominates the average load based bound. Through the following theorem we show that $LB3$(S) $(LB4(S))$ dominates $LB1(S)$ $(LB2(S))$.

Theorem 6. $LB1$(S) $\leq$ $LB3$(S)

Proof. : $LB3$(S) and $LB1$(S) both distribute $LB_{ZTOT}(S)$ + $L_{TOT}(S)$units of load among the agents. The distribution of $LB1$(S) is even among all agents whereas $LB3$(S) may make uneven distribution due to uneven current loads. In the proof Theorem 2 we show that a perfectly even distribution, as in $LB1$(S), provides the smallest total squared load value. Hence $LB1$(S) $\leq$ $LB3$(S).

Using the result of Theorem 6, we first evaluate node S using $LB1$(S). If the node cannot be fathomed then we use $LB3$(S).

Corollary 1. $LB2$(S) $\leq$ $LB4$(S)

Proof. : The only difference between LB1(S) (LB3(S)) and LB2(S) (LB4(S)) is the way that LB_{ZTOT} is calculated. The proof of Theorem 6 is valid for any LB_{ZTOT}, therefore the result is immediate.

We illustrate the lower bounds on the total load values through the following 10-task and 3-agent assignment problem instance. We consider the partial solution seen in Fig. 2.

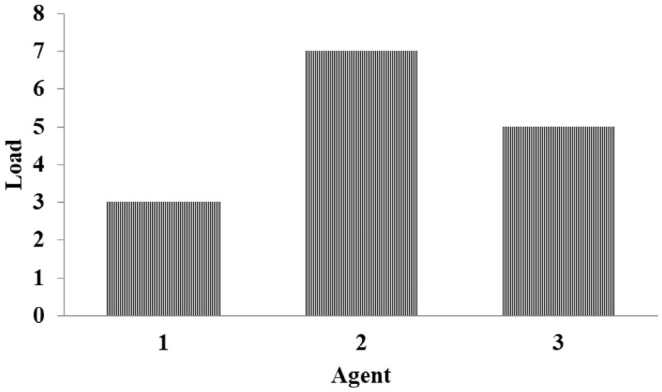

Fig. 2. Initial load allocation in the example problem.

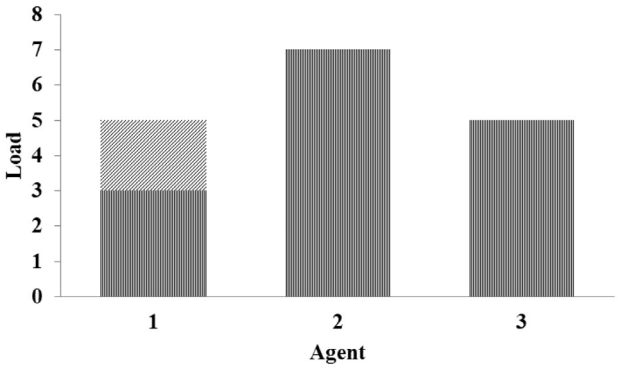

Fig. 3. Load allocation in the example problem - agent 1 filled first.

$L_{MAX}(S) = Max_i L_i = 7.$

Assume $LB_{ZTOT}(S) = 8.$

$$\text{Average load based bound} = (L_{TOT}(S) + LB_{ZTOT}(S))^2/m$$
$$= (15+8)^2/3 = 176.3$$

Fill-up strategy based bound:

Remaining Load = 8

$3 * 7 - 3 - 5 - 7 = 6$

$(8 \geq 6)$ then

$3 * (7 + (8-6)/3)^2 = 176.3$

Now assume $LB_{ZTOT}(S) = 3.$

Average load based bound $= (15+3)^2/3 = 108.$

Fill-up strategy based bound:

First fill agent 1 up to the load of agent 3 (See Fig. 3). Remaining Load becomes 3–2 = 1.

Distribute the remaining load equally among agents 1 and 3 till the remaining load is fully distributed or till the load of agent 2 is reached. 1 unit of remaining load is depleted when the loads of two agents reach to 5.5 units as seen in Fig. 4.

The loads of the resulting distribution are 5.5, 7 and 5.5 units for agents 1, 2 and 3 respectively. The resulting total squared load = 30.25 + 49 + 30.25 = 109.5 is a lower bound on the optimal total squared load.

Note that *average load based bound* = 108 $\leq$ 109.5 = *Fill-up strategy based bound.*

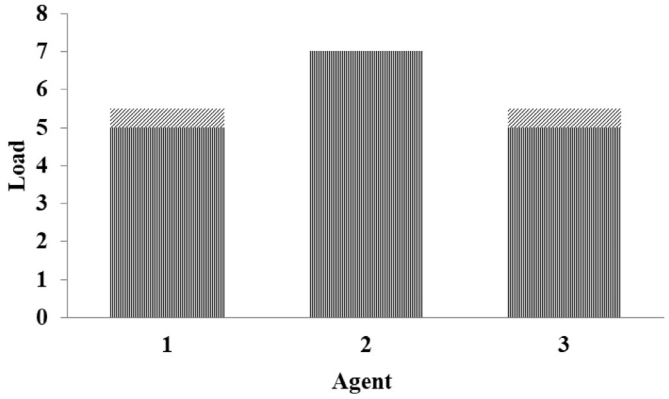

Fig. 4. Load allocation in the example problem - agents 1 and 3 filled next.

3.2. Upper bounds

Our B&B algorithm starts with an initial upper bound, UB, and updates the upper bound whenever a feasible solution with smaller total squared load is reached.

We propose four simple procedures to calculate an initial upper bound at the root node. We extend one of the upper bounds to the partial solutions where we solve assignment problem to find a lower bound.

UB1: Solve the linear programming relaxation of the linear total squared load model.

The optimal relaxed solution may be infeasible as some tasks may be assigned to more than one agent. We reassign each of those tasks to an agent that has the highest partial assignment value. After the reassignment, there may be some agents with no assigned tasks. In such a case we randomly select an agent with at least two assigned tasks and transfer one of its tasks to one of the empty agents. We continue till no empty agent remains. We let UB1 be the objective function of the resulting feasible solution with no partial assignments and no empty agents.

UB2: We assign each task to an agent that handles the task with minimum load. If all agents were assigned to at least one task then the resulting solution would be feasible to our problem. Otherwise there is at least one agent with no task assignments. In such a case we randomly select an agent(s) with at least two assigned tasks and transfer one of its tasks to the empty agent(s). We let UB2 be the objective function of the resulting feasible solution with no partial assignments and no empty agents.

UB3: We assign a single task to each agent, starting from the first agent. The selected task is the one having the minimum load for that agent. We assign the remaining tasks to their minimum load agents as in UB2. The assignment is feasible as it returns all agents with at least one assignment and one agent assignment to each task. We let UB3 be the objective function of the resulting feasible solution with no partial assignments and no empty agents.

UB4: Any feasible solution to the minimum total load problem is feasible for the minimum total squared load problem as the problems have the same constraint set. This follows an optimal solution to the minimum total load problem is feasible for the minimum total squared load problem; hence its total squared load value, UB4, is an upper bound on the minimum total squared load value.

We extend our idea to find UB4 to the nodes where we solve the total load problem to obtain lower bounds. At those nodes, we evaluate the total squared load value of the feasible solution that minimizes the total load of the unassigned tasks with the hope of improving UB.

At the root node of the branch and bound algorithm, we calculate UB1, UB2, UB3 and UB4 and take their minimum as an initial upper bound, UB. That is, we start with UB = min (UB1, UB2, UB3 and UB4). We update UB whenever we find a complete solution with smaller objective function value. At termination, UB gives the optimal total squared load.

4. Computational experiments

In this section, we demonstrate the computational efficiency of our branch and bound algorithm by conducting experiments on a set of test instances. We discuss the data generation scheme and results of our experiment.

We generate three sets of problem instances (named sets S1, S2 and S3). The time requirements, i.e., p_{ij} values, for sets S1, S2 and S3 are generated from discrete uniform distributions U[5, 25], U[10, 20], and U[25, 35], respectively.

Set S1 includes problem instances, where the range of the processing times 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 (note that the expected value is the same as in S1 while the range is lower). Set S3 is used to see the effect of a higher expected value while maintaining the same range with S2.

We create problem instances with $m = 5$ and $m = 10$ agents. For each value of m, we select n values starting from 25, in increments of 5. We generate 10 problem instances for each m, n and processing time set combination. In total, 360 instances are considered in the main experiment.[1]

The algorithm is coded in Visual C++ and solved by a quad-core (Intel Core i5 3.30 GhZ) computer with 8 GB RAM. All models are solved by CPLEX 12.5. The solution times are expressed in Central Processing Unit (CPU) seconds.

We first conduct an experiment to see the effects of using four lower bounds altogether versus using only dual and minimum total load based bounds at different levels. We let k be the number of consecutive levels for which dual based bounds (LB2, LB4) are used in the branch and bound algorithm. After a series of such k consecutive levels, we use minimum total load based bounds (LB1, LB3) for one level. We conduct preliminary experiments on relatively small size instances of Set S1 (for $m = 5$, $n = 25$, 30, 35, 40 and for $m = 10$, $n = 25$, 30) to analyse the effects of using different k values. Note that when $k = 0$, four lower bounds are used altogether at each level of the branch and bound tree.

Fig. 5a and b show the average solution times for different k, for instances where $m = 5$, $n = 30$ and $m = 10$, $n = 30$, respectively. It is observed that the solution times decrease as k increases up to a certain level (15) and after that level it starts to increase. We have similar results for the other problem sets. Hence, in our experiments we set k to 15.[2]

We also tried using average load based bounds in the first levels of the branch and bound tree to avoid the computational effort of the fill up strategy. However this strategy does not provide satisfactory results, hence it is not used in the experiment.

[1] The test instances are available at: http://staff.bilkent.edu.tr/ozlemkarsu/.

[2] We conducted Wilcoxon signed rank test to verify our observation that the solution time decreases as k moves towards 15 and increases afterwards. For example, for m=5, n=30 and k=1 and k=5 settings, we tested the null hypothesis that the median solution time difference is 0 against the alternate that it is more than 0. The p value is found as 0.002, which indicates that the test rejects our null hypothesis at 5% significance level. There is enough statistical evidence to conclude that the median solution time for k=1 is more than the median solution time when k=5. Similar conclusions are made with p values of 0.0352 (for k=5 and k=10) and 0.004 (for k=10 and k=15). We then tested the null hypothesis that for k=15 and k=20 settings the median solution time difference is 0 against the alternate that it is *less* than 0. The p value of 0.002 indicates statistical evidence to conclude that the median solution time for k=15 is less than that of k=20. We observed similar results for the other settings.

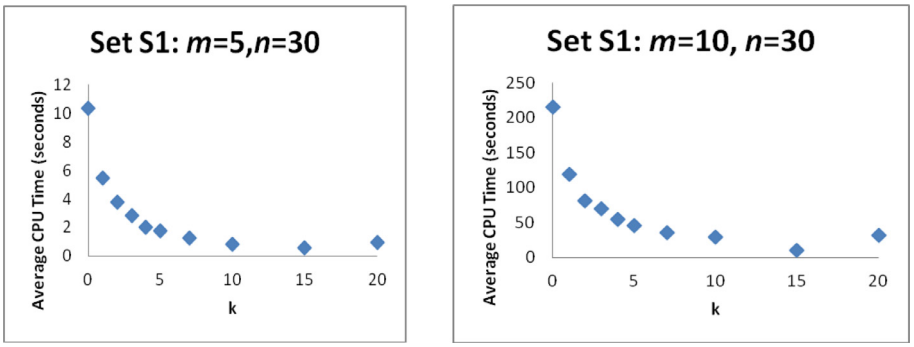

Fig. 5. (a) Effect of k on the solution times. (b). Effect of k on the solution times.

Table 3
Results of the branch-and-bound algorithm.

			CPU time			*% of nodes at which a model is solved (Avg.)*	*Avg.% of nodes fathomed*			
Set	*m*	*n*	*Avg*	*Max*	*Avg # of nodes (*10^5)*		*LB1*	*LB2*	*LB3*	*LB4*
S1	5	25	0.09	0.28	0.29	1.65	0.00	62.68	0.00	8.10
		30	0.65	5.73	3.31	0.99	0.00	65.36	0.00	6.95
		35	0.22	1.19	0.78	1.27	0.00	70.51	0.00	4.96
		40	6.10	57.97	7.86	1.78	0.00	71.77	0.00	6.00
		45	19.20	179.39	57.89	1.17	0.00	72.06	0.00	6.41
		50	92.74	849.49	167.19	1.00	0.00	73.86	0.00	5.05
		55	23.98	135.95	41.97	1.34	0.00	73.76	0.00	5.10
		60	36.01	297.19	51.33	1.41	0.00	73.99	0.00	4.20
		65	72.05	428.45	90.78	1.47	0.00	74.71	0.00	4.24
		70	350.31	2432.33	561.26	1.39	0.00	75.72	0.00	3.00
	10	25	6.47	50.50	21.16	1.18	0.26	59.62	0.20	10.33
		30	8.73	37.37	61.88	0.68	0.12	72.02	0.04	9.40
		35	87.36	427.64	912.38	0.29	0.01	76.88	0.01	7.90
		40	275.05	1394.28	1719.72	0.88	0.05	77.82	0.02	7.59
		45	1125.05	3600(2)*	1612.47	1.06	0.06	80.69	0.01	7.33
S2	5	25	0.75	4.82	2.40	1.92	0.00	68.43	0.00	9.55
		30	4.65	41.47	36.31	0.81	0.00	69.71	0.00	8.54
		35	0.69	4.34	6.10	1.54	0.00	69.92	0.00	6.29
		40	22.88	126.47	45.86	1.67	0.00	71.30	0.00	8.16
		45	19.13	121.83	51.71	1.11	0.00	71.30	0.00	8.36
		50	373.11	3600(1)	969.76	0.83	0.00	72.13	0.00	7.21
		55	336.37	1746.93	519.47	1.49	0.00	73.63	0.00	6.15
		60	98.19	307.59	135.31	1.59	0.00	73.77	0.00	5.50
		65	318.54	1243.40	366.78	1.49	0.00	74.99	0.00	4.87
		70	824.09	3600(2)	1919.90	1.26	0.00	75.75	0.00	4.00
	10	25	103.96	346.19	392.10	1.52	0.29	72.50	0.26	15.81
		30	206.92	1403.79	1686.50	0.54	0.13	78.49	0.01	10.36
		35	740.14	3600(1)	6924.40	0.21	0.00	77.69	0.01	12.03
S3	5	25	1.64	9.72	5.14	1.98	0.00	69.10	0.00	9.63
		30	16.80	155.30	149.01	0.79	0.00	69.36	0.00	9.65
		35	5.55	45.63	71.96	1.26	0.00	70.28	0.00	7.75
		40	68.75	390.17	140.02	1.62	0.00	71.23	0.00	8.55
		45	91.44	459.60	203.26	1.30	0.00	70.65	0.00	9.22
		50	411.84	3600(1)	1508.09	0.77	0.00	71.77	0.00	7.78
		55	747.54	3600(2)	1153.72	1.44	0.00	72.23	0.00	7.62
	10	25	3050.34	3600(8)	4139.55	1.25	0.17	34.54	0.28	12.87

* The numbers in the parentheses show the number of instances that could not be solved in one hour.

In Tables 3 and 4, the results of our experiment are reported. Table 3, we give the results of the B&B algorithm and in Table 4, we compare the performance of the CPLEX MIP solver with the B&B algorithm. We set a time limit of 1 h for both algorithms. We stop the experiments for a given set when at least 2 out of 10 instances cannot be solved in one hour.

Table 3 shows the average and maximum CPU times and the average number of nodes in the B&B tree. The percentage of nodes at which the total load minimizing LP models are solved (so as to find lower bounds), the average percentage of nodes eliminated by dual based bounds (*LB2, LB4*) and optimal Z_{TOT} based bounds (*LB1, LB3*) are also reported. The percentage value is calculated separately for each instance and the average value across 10 instances is found. We also state the number of instances for which optimality could not be verified within an hour.

It is observed that Set S1 instances, where the range of the processing times is relatively large, constitute the easiest to solve instances. In this set the B&B algorithm finds the optimal solutions within an hour for problems with sizes up to 70 tasks when $m=5$ and up to 40 tasks when $m=10$.

Set S3 instances, with relatively higher average processing times and lower range, are the hardest-to-solve instances. In this set, the time limit is reached when $n=50$ and $n=25$, for $m=5$ and $m=10$ cases, respectively.

As expected, for fixed m (n), the solution times and the tree size increase as n (m) increases, with a few exceptions. One of these

Table 4
CPLEX MIP solver versus B&B algorithm results.

			CPU timeCPLEX		*CPU timeB&B*		*CPU timeB&B without LB3 and LB4*		*Deviation of MIP solver solution from the B&B solution*	
Set	*m*	*n*	*Avg*	*Max*	*Avg*	*Max*	*Avg*	*Max*	*Avg*	*Max*
	5	25	41.85	93.56	0.09	0.28	0.16	0.58	0.00	0.00
		30	800.43	2571.15	0.65	5.73	2.13	20.29	0.00	0.00
S1		35	3264.57 (6*)	3600.00	0.22	1.19	1.40	7.94	0.03	0.24
	10	25	6.39	26.09	6.47	50.50	52.07	387.56	0.00	0.00
		30	962.23 (1)	3600.32	8.73	37.37	96.50	777.38	0.00	0.00
		35	3441.83 (9)	3600.57	87.36	427.64	1746.15	3600 (1)	0.05	0.26

* The numbers in the parentheses show the number of instances that could not be solved in one hour.

Table 5
Objective function values of the CPLEX MIP Solver and optimal objective function values by the B&B (I.:Instance number).

S1					S2					S3				
m	*n*	*I.*	*CPLEX*	*Optimal*	*m*	*n*	*I.*	*CPLEX*	*Optimal*	*m*	*n*	*I.*	*CPLEX*	*Optimal*
5	70	1	65152	62696	5	65	1	112359	109273	5	45	1	287265	278490
		2	57076	56529			2	118405	109782			2	305580	291387
		3	62389	61012			3	116256	109814			3	297655	289180
		4	62727	61316			4	113283	106698			4	282809	278486
		5	71660	62599			5	115377	110431			5	288517	284679
		6	61695	60574			6	114959	110182			6	287571	277077
		7	71738	69228			7	111570	106389			7	294607	290437
		8	68345	63421			8	111044	106620			8	295565	287555
		9	66,890	63053			9	117373	107330			9	291943	281789
		10	61006	60287			10	113752	109120			10	281358	277165
10	40	1	6669	6669	10	30	1	11054	10904					
		2	6980	6980			2	11275	11131					
		3	6728	6728			3	10857	10845					
		4	6868	6868			4	10006	9941					
		5	6139	6139			5	11127	10979					
		6	5616	5616			6	10126	9937					
		7	7616	7601			7	10384	10384					
		8	6499	6499			8	10989	10900					
		9	6262	6256			9	10717	10650					
		10	7612	7584			10	10473	10398					

exceptions occurs when $m=5$, $n=50$, which is due to a single instance with a relatively high average processing time, which makes it hard to solve. We observe that the dual based lower bound(s) (*LB2, LB4*) are effective in eliminating unpromising nodes over all instances while the optimal Z_{TOT} based bound(s) (*LB1, LB3*) become useful in instances where $m=10$.

In Table 4 we give the average and maximum solution times for the sum of squared loads problem-linear model (SSL-L) returned by the CPLEX MIP solver and the solution times of the B&B algorithm for comparison. We also report the CPU times of a simpler version of the proposed B&B algorithm in which the elaborate lower bounds (LB3 and LB4) are not used. The table also includes the deviation of the MIP solver solution from the optimal solution.

As seen in the table, the size of the problems that CPLEX MIP solver can solve within an hour is significantly smaller compared to the ones by the B&B algorithm. The B&B algorithm without the elaborate lower bounds performs worse than the version using these bounds, yet its performance is still much better than that of the CPLEX MIP solver.

We could find the optimal solutions by the B&B algorithm for all these instances, hence we report the average and maximum percentage deviation of the CPLEX solution from the optimal solution returned by the B&B algorithm (100*(CPLEX solution – OPT)/OPT). We also report the number of instances for which CPLEX terminated due to the time limit. CPLEX could only solve Set S1 instances with $m=5$ and $n=25$, 30 and $m=10$ and $n=25$ optimally within one hour. For Set S2 and S3, we try instances with $m=5$, $n=25$ and $m=5$, $n=30$, however none of them could be solved within an hour. In Set 2 instances, the average and maximum% deviations from the B&B solutions are 0.088 and 0.60, respectively when $m=5$, $n=25$ and 0.63 and 2.12, respectively when $m=5$, and $n=30$. We observe similar results for Set S3 instances with average (maximum)% deviations of 0.35 (1.23) when $m=5$, $n=25$ and 1.33 (2.33) when $m=5$, $n=30$. The deviations increase as the problem size increases, for example the maximum% deviation can go up to 5.71 for Set 3 instances when $m=5$, $n=45$. The results show that the B&B algorithm outperforms the model solutions returned by the CPLEX MIP solver. The CPLEX MIP solver could not guarantee optimality within an hour and the returned solution is not optimal in most cases. B&B algorithm terminates much quicker and guarantees optimality.

To observe the quality of solutions returned by CPLEX MIP Solver in more detail, in Table 5 we provide the objective function values (ZSQ values) of the solutions returned by CPLEX within an hour and the optimal solutions returned by the B&B algorithm, for the largest problem instances in our data set that could be solved to optimality. For all of these largest instances CPLEX stopped due to time limit without verifying optimality. We provide the detailed results of all instances in Appendix A.

Table 5 shows that in all combinations with one exception the objective function values of the CPLEX MIP Solver after one hour of execution are much worse than those easily returned by the B&B Algorithm. The exception is $m=10$ and $n=40$ for Set 1 combination, where 7 out 10 solutions by the CPLEX MIP Solver turned out to be optimal.

Recall that the upper bound mechanisms that we discuss in Section 2 are simple heuristic solutions. We also compare the

Table 6
Performances of the simple heuristic solutions.

			Avg. % deviation from the optimal				*# of instances for which it returns the best solution*		
Set	*m*	*n*	*UB1*	*UB2*	*UB3*	*UB4*	*UB2*	*UB3*	*UB4*
S1	5	25	125.88	11.50	19.74	12.55	7	1	9
		30	111.44	10.52	12.99	9.97	3	4	6
		35	111.91	8.03	11.17	8.86	7	2	9
		40	115.70	7.82	10.83	7.09	3	1	9
		45	134.68	8.23	9.43	8.21	5	2	8
		50	139.67	8.13	11.04	8.63	6	4	7
		55	119.36	4.61	7.18	4.70	7	3	8
		60	138.83	5.23	8.28	5.38	7	2	7
		65	126.66	4.33	6.25	4.15	4	2	8
		70	134.74	4.31	5.74	4.61	2	4	6
	10	25	112.78	24.68	30.18	18.18	2	1	9
		30	103.14	18.28	25.46	16.54	3	3	6
		35	109.02	16.00	20.54	13.08	1	2	8
		40	96.54	13.92	19.42	9.49	1	2	8
		45	102.70	14.48	18.49	14.91	2	2	6
S2	5	25	77.21	13.30	15.97	11.72	3	2	8
		30	85.05	13.98	14.27	13.43	1	6	4
		35	80.76	10.42	11.97	9.62	2	5	6
		40	80.49	9.77	11.60	8.94	4	1	8
		45	87.08	9.69	9.82	10.19	2	6	3
		50	95.43	9.89	11.68	10.45	4	4	7
		55	80.49	6.76	7.03	6.88	5	6	8
		60	91.20	6.43	7.63	6.80	5	3	7
		65	83.21	4.71	5.39	4.83	4	4	7
		70	91.68	6.36	6.88	6.07	2	3	7
	10	25	72.54	24.14	28.33	19.65	1	1	9
		30	61.63	21.42	24.02	14.90	2	1	9
		35	78.13	21.87	23.27	23.32	2	5	3
S3	5	25	61.55	14.03	15.63	12.97	2	5	5
		30	71.40	14.55	14.15	14.09	1	6	4
		35	64.13	10.74	12.06	10.00	2	5	6
		40	64.20	10.06	11.51	9.30	4	1	8
		45	69.84	9.60	9.30	10.08	2	6	3
		50	75.74	9.71	11.07	10.29	4	4	7
		55	65.30	6.99	6.82	7.17	5	6	7
	10	25	57.71	23.31	27.12	19.75	1	1	9

solution of the B&B algorithm with these heuristic solutions to provide better insight on how well the algorithm works. Table 6 shows the average deviation of the upper bounds (UB1, UB2, UB3, UB4) from the optimal solution calculated as follows (100*(UB – OPT)/OPT). We also report the number of instances at which the corresponding upper bound gives the best solution over the four upper bounds (out of 10 instances), including the ties.

We observe that UB1 performs poorest and it never returns the best value, therefore it is not reported in the table. The other upper bounds have comparable quality, with UB4 performing consistently well. Yet, even when the best heuristic value is used, the minimum (average) deviation from the optimal solution is always above 4%, and can be as high as 20% as seen in the table.

Finally, we generate instances with approximately same number of tasks and agents to see how the algorithm performs for such cases and report the associated results in Table 7. We did not report the results for the sets in which more than 3 instances could not be solved within an hour (indicated with – sign). The k parameter is set to approximately $1.5m$ (15 for instances with $m = 10$ and $m = 12$, 20 for instances with $m = 15$). The detailed results are given in Appendix B.

We observe that as the number of tasks and number of agents get closer, most of the instances are solved quicker by the CPLEX MIP solver. For example, for Set 2 instances with $n = 20$, the CPLEX solution times decrease as m increases from 10 to 15. However we do not observe any decrease in the solution times of the B&B algorithm, when m and n values get closer, except for the case when they are equal. Still, the B&B algorithm outperforms the CPLEX MIP solver in almost all instances.

Table 7
Results for the instances with close m and n values.

			B&B algorithmCPU Time		*CPLEX MIP solverCPU time*	
Set	*m*	*n*	*Avg*	*Max*	*Avg*	*Max*
1	10	15	0.025	0.04	0.24	0.31
		20	0.766	4.85	0.83	2.00
	12	15	0.039	0.08	0.22	0.27
		20	3.268	18	0.69	0.92
	15	15	0.019	0.02	0.12	0.13
		20	0.363	1.03	0.89	2.86
2	10	15	0.311	1.06	3.17	5.36
		20	8.508	66.1	1454.86 (3)	3600
	12	15	0.543	1.13	4.88	12.46
		20	66.435	186.03	157.07	433.85
	15	15	0.02	0.02	0.13	0.14
		20	380.666	1575.97	83.60	480.28
3	10	15	30.315	74.13	1910.33 (2)	3600
		20	7.535	61.06	–	–
	12	15	112.546	246.45	–	–
		20	–	–	–	–
	15	15	0.023	0.05	0.1467	0.153
		20	–	–	–	–

5. Conclusions

In this study, we consider an assignment problem with the objective to minimize the sum of squared loads of all agents. We propose mixed integer nonlinear and linear programming formulations of the problem and present a branch and bound algorithm for

their solution. The results of our computational experiment have shown the satisfactory behavior of our branch and bound algorithm. The algorithm is capable of solving the instances with up to 70 tasks when there are 5 agents and up to 45 tasks when there are 10 agents.

To the best of our knowledge, this study is the first reported attempt to solve the assignment problem with the objective to minimize the total squared agent loads. Future research may consider the development of heuristic approaches like meta-heuristic approaches that produce high quality solutions in reasonable times. Another important direction for future research might be studying load balancing problems with different measures that reflect the relative importance of the agents, like total weighted squared workload. Future research can also be performed on extending the models to solve other practical problems such as the ergonomic job rotation problem (Otto and Battaïa, 2017), which aims at ensuring a balanced distribution of risks among individual work assignments.

Acknowledgments

We thank the anonymous reviewers for their constructive comments.

Appendix A

Table A.1
Set1, $m = 5$ instances.

n	*I.*	*CPLEX Time*	*B&B Time*	*CPLEX Soln.*	*B&B Soln.*	*n*	*I.*	*B&B Time*	*CPLEX Soln.*	*B&B Soln.*	*n*	*I.*	*B&B Time*	*CPLEX Soln.*	*B&B Soln.*
25	1	84.0	0.07	7665	7665	45	1	0.19	23887	23704	60	1	17.41	57173	54430
	2	44.3	0.05	8654	8654		2	0.35	32372	32217		2	0.43	52299	50528
	3	7.7	0.02	8131	8131		3	188.25	30745	30646		3	6.82	55069	52074
	4	11.2	0.07	8991	8991		4	0.2	25660	25243		4	4.67	48534	48041
	5	8.4	0.01	9098	9098		5	3.91	29763	29538		5	29.58	52068	50623
	6	57.0	0.18	8873	8873		6	0.19	23675	23675		6	0.08	48496	48229
	7	29.4	0.02	9340	9340		7	7.08	30578	30557		7	0.2	45291	45110
	8	8.0	0.27	7215	7215		8	0.65	29773	29633		8	1.9	43783	43087
	9	74.9	0.02	10159	10159		9	1.21	25885	25836		9	302.91	47691	45553
	10	93.6	0.16	8111	8111		10	0.17	22735	22735		10	2.07	61721	60124
30	1	2571.2	0.05	16954	16954	50	1	0.17	33085	32929	65	1	17.58	56087	55251
	2	417.4	0.13	12107	12107		2	27.12	36637	36065		2	18.66	56741	56353
	3	419.1	0.4	11788	11788		3	0.91	30693	30371		3	0.19	55719	54628
	4	1097.9	0.02	15679	15679		4	1.6	29210	29191		4	16.22	52068	51034
	5	1587.0	5.48	13235	13235		5	2.04	32531	32398		5	408.95	56902	56047
	6	163.8	0.02	11228	11228		6	837.5	35766	35598		6	90.14	56335	55669
	7	552.0	0.02	13102	13102		7	46.49	38178	36085		7	0.86	51772	50866
	8	363.1	0.05	12580	12580		8	0.3	33667	33019		8	4.14	52924	52462
	9	625.6	0.09	14648	14648		9	0.48	34671	34027		9	7.81	52220	51718
	10	207.2	0.02	13453	13453		10	0.61	36140	35187		10	127.91	59005	54523
35	1	3600	0.13	16610	16606	55	1	0.97	37615	36556	70	1	13	65152	62696
	2	2082.4	0.05	13885	13885		2	1.26	46634	45591		2	0.27	57076	56529
	3	2395.7	0.02	15623	15623		3	81.45	43138	43131		3	0.29	62389	61012
	4	3126.2	0.2	17823	17823		4	15.57	44770	43625		4	885.83	62727	61316
	5	3600	0.27	15617	15617		5	136.9	39300	38670		5	137.13	71660	62599
	6	3600	0.05	16704	16664		6	0.34	41779	40900		6	24.64	61695	60574
	7	3440.0	1.17	16099	16099		7	0.16	47217	46393		7	2400.72	71738	69228
	8	3600	0.03	20862	20862		8	0.12	36075	34820		8	2.54	68345	63421
	9	3600	0.05	15667	15667		9	0.91	34819	34799		9	0.12	66890	63053
	10	3600	0.12	16154	16154		10	3.64	32196	31655		10	0.58	61006	60287
40	1	3600	0.19	22626	22479										
	2	3600	60.06	19925	19812										
	3	3600	0.29	21605	21605										
	4	3600	0.18	19480	19480										
	5	3600	0.25	23482	23482										
	6	3600	0.03	24067	24067										
	7	3600	0.35	19050	18968										
	8	3600	0.05	19551	19551										
	9	3600	1.04	18131	18023										
	10	3600	0.78	22351	22255										

CPLEX solution time is reported as 3600 s (1 h) when the time limit of 1 h is reached. For $n \geq 45$ instances, the CPLEX solution times are not reported as CPLEX reaches the time limit. B&B soln. is the optimal solution.

Table A.2
Set1, $m = 10$ instances.

n	*I.*	*CPLEX Time*	*B&B Time*	*CPLEX Soln.*	*B&B Soln.*	n	*I.*	*B&B Time*	*CPLEX Soln.*	*B&B Soln.*
25	1	5.30	0.4	2315	2315	35	1	0.18	5657	5657
	2	26.09	4.61	2675	2675		2	258.59	5333	5333
	3	3.29	4.34	2738	2738		3	11.91	4906	4906
	4	3.18	0.05	2723	2723		4	63.78	4878	4878
	5	3.18	0.19	3028	3028		5	439	5470	5470
	6	3.94	0.36	2470	2470		6	3.47	5179	5168
	7	2.76	0.95	2369	2369		7	60.38	6360	6360
	8	5.01	0.68	2472	2472		8	0.34	5057	5057
	9	8.58	50.55	2593	2593		9	16.17	5473	5473
	10	2.57	2.67	2734	2734		10	42.02	5132	5132
30	1	2766.42	15.41	4527	4527	40	1	5.64	6669	6669
	2	269.36	2.62	4850	4850		2	2.43	6980	6980
	3	3600	37.76	4595	4595		3	2.93	6728	6728
	4	41.1	2.4	3579	3579		4	2.6	6868	6868
	5	753.33	20.27	4762	4762		5	217.91	6139	6139
	6	72.04	3.31	3559	3559		6	135.75	5616	5616
	7	26.21	1.34	3809	3809		7	1422.36	7616	7601
	8	595.68	0.12	4715	4715		8	886.56	6499	6499
	9	1432.61	5.36	4489	4489		9	39.41	6262	6256
	10	65.276	0.62	4251	4251		10	74.03	7612	7584

CPLEX solution time is reported as 3600 s (1 h) when the time limit of 1 h is reached. For $n \geq 35$ instances, the CPLEX solution times are not reported as CPLEX reaches the time limit. B&B soln. is the optimal solution.

Table A.3
Set 2, $m = 5$ instances.

n	*I.*	*B&B Time*	*CPLEX Soln.*	*B&B Soln.*	n	*I.*	*B&B Time*	*CPLEX Soln.*	*B&B Soln.*	n	*I.*	*B&B Time*	*CPLEX Soln.*	*B&B Soln.*
25	1	0.24	16125	16029	40	1	1.87	43130	42351	55	1	1.15	79928	76197
	2	0.15	16491	16491		2	125.59	41182	40610		2	9.47	86563	82722
	3	0.05	16263	16263		3	5.8	43174	42234		3	1587.45	84045	81283
	4	0.52	16956	16949		4	0.12	40763	40218		4	29.54	84491	81827
	5	0.02	16507	16507		5	0.24	44356	43576		5	1743.24	83342	78285
	6	0.97	17242	17242		6	0.12	44580	44377		6	5.93	83182	80559
	7	0.03	16871	16871		7	29.89	41708	41340		7	1.8	87464	82747
	8	4.91	16075	16075		8	1.28	41370	41038		8	1.38	78202	75910
	9	0.07	17799	17799		9	62.79	40776	40128		9	7.14	76128	75418
	10	0.61	16461	16422		10	2.23	42343	41992		10	21.14	74494	73209
30	1	0.12	27243	26945	45	1	1.02	52007	50968	60	1	72.68	103258	98654
	2	0.49	23773	23701		2	0.81	57222	56503		2	0.44	99250	95950
	3	4.11	23780	23766		3	131.14	57021	55559		3	25.08	104210	97323
	4	0.12	26140	26010		4	1.11	51664	51011		4	61.51	101255	95131
	5	40.89	24953	24681		5	3.17	56110	53694		5	301.4	101741	95968
	6	0.03	23418	23418		6	0.37	51038	50291		6	3.79	97226	94382
	7	0.06	25021	24501		7	59.64	56403	55910		7	0.25	95607	91400
	8	0.04	23921	23921		8	4.71	55175	54754		8	217.94	95324	91408
	9	0.12	25754	25726		9	1.37	53870	52124		9	310.25	95957	91572
	10	0.02	24512	24266		10	0.87	50799	50146		10	3.52	106583	102784
35	1	1.1	32402	32098	50	1	1.51	66863	65044	65	1	74.87	112359	109273
	2	0.09	30922	30596		2	97.89	69979	67158		2	9.63	118405	109782
	3	0.03	32293	32154		3	1.48	64235	63109		3	32.85	116256	109814
	4	0.2	33551	33551		4	0.17	63874	62284		4	103.64	113283	106698
	5	0.43	31458	31458		5	4.81	65678	63867		5	1010.9	115377	110431
	6	0.04	32126	31969		6	3600	69035	68152		6	583.03	114959	110182
	7	0.43	32497	32241		7	28.54	69579	67773		7	5.89	111570	106389
	8	0.08	35695	35551		8	0.8	65980	65524		8	0.59	111044	106620
	9	0.13	32227	32227		9	0.62	66889	65970		9	1267.99	117373	107330
	10	4.46	32810	32434		10	0.15	69135	66697		10	143.53	113752	109120

CPLEX reaches the time limit of 1 h for all instances; hence CPLEX solution times are not reported. B&B soln. is the optimal solution.

Table A.4
Set 2, $m = 10$ instances.

n	*I.*	*B&B Time*	*CPLEX Soln.*	*B&B Soln.*	*n*	*I.*	*B&B Time*	*CPLEX Soln.*	*B&B Soln.*
25	1	263.21	6912	6892	30	1	35.56	11054	10904
	2	25.92	7090	7090		2	341.03	11275	11131
	3	50.93	7231	7231		3	256.55	10857	10845
	4	3.72	7221	7203		4	21.73	10006	9941
	5	1.17	7492	7492		5	1527.21	11127	10979
	6	50.24	6978	6978		6	4.42	10126	9937
	7	150.79	7082	7082		7	8.13	10384	10384
	8	59.14	7075	7072		8	6.09	10989	10900
	9	357.61	7274	7274		9	3.77	10717	10650
	10	126.42	7195	7195		10	1.88	10473	10398

CPLEX reaches the time limit of 1 h for all instances; hence CPLEX solution times are not reported. B&B soln. is the optimal solution.

Table A.5
Set 3, $m = 5$ instances.

n	*I.*	*B&B Time*	*CPLEX Soln.*	*B&B Soln.*	*n*	*I.*	*B&B Time*	*CPLEX Soln.*	*B&B Soln.*	*n*	*I.*	*B&B Time*	*CPLEX Soln.*	*B&B Soln.*
25	1	0.24	86610	86604	35	1	2.64	184221	171510	45	1	2.44	287265	278490
	2	0.16	87939	87666		2	0.19	168959	168245		2	2.72	305580	291387
	3	0.06	87919	87138		3	0.05	173754	171516		3	407.15	297655	289180
	4	1.43	89031	89031		4	0.59	176162	175286		4	4.02	282809	278486
	5	0.02	88762	87682		5	1.17	172627	169811		5	6.33	288517	284679
	6	4.1	90102	89904		6	0.04	177574	170884		6	1.76	287571	277077
	7	0.03	88758	88535		7	5.05	175420	171891		7	457.67	294607	290437
	8	9.56	87158	87158		8	0.19	181733	179383		8	19.51	295565	287555
	9	0.05	91257	90777		9	0.09	172290	171572		9	4.24	291943	281789
	10	1.08	87672	87664		10	49.24	175035	172715		10	11.17	281358	277165
30	1	0.22	135563	133505	40	1	16.17	229354	224800					
	2	2.48	127747	126459		2	389.89	225718	220991					
	3	9.34	129673	126724		3	21.46	231905	224738					
	4	0.41	132604	131607		4	0.65	229513	219738					
	5	157.41	129700	129027		5	1.25	230858	227727					
	6	0.03	127731	125478		6	0.13	239903	229417					
	7	0.31	130682	128750		7	92.68	230810	223172					
	8	0.14	129386	126778		8	5.68	230291	221803					
	9	0.29	132562	130905		9	153.38	224577	220371					
	10	0.02	128070	127406		10	4.99	228607	223912					

CPLEX reaches the time limit of 1 h for all instances; hence CPLEX solution times are not reported. B&B soln. is the optimal solution.

Appendix B

Table B.1
Set 1 instances.

m	*n*	*I.*	*CPLEX Time*	*B&B Time*	*Opt. Soln.*	*m*	*n*	*I.*	*CPLEX Time*	*B&B Time*	*Opt.Soln.*	*m*	*n*	*I.*	*CPLEX Time*	*B&B Time*	*Opt.Soln.*
10	15	1	0.31	0.03	967	12	15	1	0.27	0.04	772	15	15	1	0.12	0.02	543
		2	0.28	0.03	1348			2	0.27	0.08	1053			2	0.12	0.02	770
		3	0.23	0.02	1155			3	0.23	0.03	1003			3	0.11	0.02	850
		4	0.23	0.01	1225			4	0.22	0.02	1025			4	0.13	0.01	692
		5	0.23	0.02	1413			5	0.18	0.02	939			5	0.12	0.02	596
		6	0.24	0.02	981			6	0.18	0.02	791			6	0.12	0.02	775
		7	0.21	0.03	1015			7	0.21	0.04	901			7	0.12	0.02	772
		8	0.24	0.03	981			8	0.17	0.08	978			8	0.12	0.02	686
		9	0.23	0.04	1324			9	0.21	0.04	954			9	0.11	0.02	901
		10	0.23	0.02	1122			10	0.24	0.02	1001			10	0.11	0.02	722
	20	1	0.94	0.24	1755		20	1	0.90	1.02	1469		20	1	2.86	0.4	1170
		2	2.00	0.33	2327			2	0.77	2	1797			2	0.61	0.41	1286
		3	0.51	0.05	2315			3	0.66	1.49	1626			3	0.62	0.04	1151
		4	0.42	0.04	1889			4	0.68	0.38	1593			4	0.80	0.34	1038
		5	0.94	0.06	1865			5	0.54	0.08	1364			5	0.49	0.05	1122
		6	0.44	0.08	1755			6	0.53	0.23	1588			6	1.28	1.03	1157
		7	1.08	0.23	2385			7	0.62	0.41	1339			7	0.51	0.64	1175
		8	0.56	0.61	1647			8	0.68	7.91	2032			8	0.50	0.46	1197
		9	0.75	4.85	2142			9	0.92	18	1415			9	0.55	0.12	1215
		10	0.66	1.17	2089			10	0.55	1.16	1616			10	0.72	0.14	1004

Since both CPLEX and B&B provide the optimal solution, we only report the optimal solution.

Table B.2
Set 2 instances.

m	*n*	*I.*	*CPLEX Time*	*B&B Time*	*CPLEX Soln.*	*B&B Soln.*	*m*	*n*	*I.*	*CPLEX Time*	*B&B Time*	*Opt. Soln.*
10	15	1	3.29	1.06	2753	2753	12	20	1	323.18	52.2	3955
		2	4.81	0.21	2938	2938			2	426.09	41.01	4158
		3	5.07	0.17	2801	2801			3	41.61	51.71	4086
		4	2.19	0.08	2890	2890			4	62.66	47.6	4078
		5	5.36	0.4	3120	3120			5	13.42	11.48	3852
		6	2.10	0.02	2653	2653			6	19.72	4.48	3968
		7	1.71	0.07	2735	2735			7	139.60	60.38	3849
		8	2.84	0.41	2714	2714			8	49.81	61.53	4338
		9	2.38	0.48	2956	2956			9	433.85	147.93	3853
		10	1.97	0.21	2807	2807			10	60.77	186.03	4173
	20	1	349.93	0.73	4584	4584	15	15	1	0.12	0.02	1563
		2	3600	0.56	4985	4985			2	0.14	0.02	1741
		3	244.25	0.15	4966	4966			3	0.13	0.02	1829
		4	793.76	2.74	4903	4903			4	0.14	0.02	1699
		5	787.87	0.11	4631	4631			5	0.14	0.02	1651
		6	53.31	0.17	4590	4590			6	0.14	0.02	1787
		7	3600	0.37	4954	4948			7	0.13	0.02	1720
		8	66.78	1.52	4422	4422			8	0.13	0.02	1697
		9	1452.26	12.63	4864	4864			9	0.13	0.02	1848
		10	3600	66.1	5060	5060			10	0.13	0.02	1739
12	15	1	12.46	0.91	2249	2249		20	1	480.28	470.79	3232
		2	4.49	1.01	2384	2384			2	19.52	58.05	3268
		3	5.31	0.2	2400	2400			3	94.37	25.93	3236
		4	2.28	0.23	2426	2426			4	12.75	346.95	3152
		5	1.83	0.07	2318	2318			5	11.85	53.53	3170
		6	2.26	0.24	2251	2251			6	32.11	1575.97	3213
		7	5.63	0.74	2385	2385			7	84.69	252.7	3239
		8	4.07	1.13	2349	2349			8	16.89	199.32	3304
		9	4.41	0.68	2339	2339			9	11.65	26.78	3280
		10	6.11	0.22	2393	2393			10	71.91	796.64	3166

CPLEX solution time is reported as 3600 s (1 h) when the time limit of 1 h is reached. Since both CPLEX and B&B provide the optimal solution for $m = 12$ and $m = 15$ instances, we only report the optimal solution for these instances. For the other instances, B&B soln. is the optimal solution.

Table B.3
Set 3 instances.

m	*n*	*I.*	*CPLEX Time*	*B&B Time*	*CPLEX Soln.*	*B&B Soln.*	*m*	*n*	*I.*	*CPLEX Time*	*B&B Time*	*CPLEX Soln.*	*B&B Soln.*
10	15	1	1840.20	74.13	16238	16238	12	15	1	3600	142.17	13484	13484
		2	2896.99	39.82	16663	16663			2	3600	182.48	13799	13799
		3	733.74	20.78	16346	16346			3	3600	43.63	13845	13845
		4	928.09	7.69	16555	16555			4	3600	98.45	13901	13901
		5	3600	54.25	17085	17085			5	3600	15.98	13643	13643
		6	1738.99	1.28	15988	15988			6	3600	37.13	13486	13486
		7	1346.82	5.56	16190	16190			7	3600	131.3	13800	13800
		8	921.50	34.86	16139	16139			8	3600	168.6	13704	13704
		9	3600	47.56	16711	16711			9	3600	246.5	13694	13694
		10	1496.86	17.22	16352	16352			10	3600	59.26	13808	13808
	20	1	3600	0.67	26424	26424	15	15	1	0.14	0.02	9528	9528
		2	3600	0.55	27365	27365			2	0.15	0.02	9946	9946
		3	3600	0.18	27286	27286			3	0.15	0.02	10154	10154
		4	3600	2.1	27163	27163			4	0.15	0.02	9844	9844
		5	3600	0.12	26531	26531			5	0.15	0.02	9736	9736
		6	3600	0.17	26430	26430			6	0.15	0.05	10052	10052
		7	3600	0.38	27274	27268			7	0.15	0.02	9895	9895
		8	3600	1.06	26022	26022			8	0.15	0.02	9842	9842
		9	3600	9.06	27064	27064			9	0.15	0.02	10203	10203
		10	3600	61.06	27514	27514			10	0.14	0.02	9944	9944

CPLEX solution time is reported as 3600 s (1 h) when the time limit of 1 h is reached. B&B soln. is the optimal solution.

References

Ahuja, R.K., Magnanti, T.L., Orlin, J.B., 1993. Network Flows: Theory, Algorithms, and Applications. Prentice Hall, New Jersey.

Azizoğlu, M., Kırca, Ö, 1999. Scheduling jobs on unrelated parallel machines. *IIE Trans.* 31 (2), 153–161.

Billionnet, A., Costa, M.C.Costa, Sutter, A., 1992. An efficient algorithm for a task allocation problem. *J. Assoc. Comput. Mach.* 39, 502–518.

Burkard, R.E., 2013. Quadratic assignment problems. In: Pardalos, P, Du, Dd-Z (Eds.), Handbook of Combinatorial Optimization. Springer, New York, pp. 2741–2814.

Cattrysse, D.G., Van Wassenhove, L.N., 1992. A survey of algorithms for the generalized assignment problem. *Eur. J. Oper. Res.* 60 (3), 260–272.

Drezner, Z., 2015. Quadratic assignment problem. In: Laporte, G, Nickel, S., Saldanha da Gama, F. (Eds.), Location Science. Springer, pp. 345–363.

Drwal, M, 2014. Algorithm for quadratic semi-assignment problem with partition size coefficients. *Optim. Lett.* 8, 1183.

Ernst, A., Jiang, H., Krishnamoorthy, M, 2006. Exact solutions to task allocation problems. *Manag. Sci.* 52.10, 1634–1646.

Greenberg, D.E., 1969. A quadratic assignment problem without column constraints. *Nav. Res. Logist. Q.* 16, 417–422.

Karsu, Ö., Azizoğlu, M., 2012. The multi-resource agent bottleneck generalised assignment problem. *Int. J. Prod. Res.* 50 (2), 309–324.

Karsu, Ö., Azizoğlu, M., 2014. Bicriteria multiresource generalized assignment problem. *Nav. Res. Logist.* 61 (8), 621–636.

Karsu, Ö., Morton, A., 2015. Inequity averse optimization in operational research. *Eur. J. Oper. Res.* 245 (2), 343–359.

Lawler, E.L., 1963. The quadratic assignment problem. *Manag. Sci.* 9 (4), 586–599.

Loiola, E.M., de Abreu, N.M.M., Boaventura-Netto, P.O., Hahn, P., Querido, T., 2007. A survey for the quadratic assignment problem. *Eur. J.* Oper. Res. 176 (2), 657–690.
Lulli, G., Odoni, A., 2007. The European air traffic flow management problem. Transp. Sci. 41 (4), 431–443.
Magirou, V.F., Milis, J.Z., 1989. An algorithm for the multiprocessor assignment problem. *Oper. Res. Lett.* 8, 351–356.
Malucelli, F., 1996. A polynomially solvable class of quadratic semi-assignment problems. *Eur. J. Oper. Res.* 91, 619–622.
Malucelli, F., Pretolani, D., 1995. Lower bounds for the quadratic semi-assignment problem. *Eur. J. Oper. Res.* 83.2, 365–375.
Martin, S., Ouelhadj, D., Smet, P., Vanden Berghe, G., Özcan, E., 2013. Cooperative search for fair nurse rosters. Expert Syst. Appl. 40 (16), 6674–6683.
Mazzola, J.B., Neebe, A.W., 1988. Bottleneck generalized assignment problems. *Eng. Costs Prod. Econ.* 14 (1), 61–65.
Milis, I.Z., Magirou, V.F., 1995. A Lagrangian relaxation algorithm for sparse quadratic assignment problems. *Oper. Res. Lett.* 17.2, 69–76.
Otto, A., Battaïa, O., 2017. Reducing physical ergonomic risks at assembly lines by line balancing and job rotation: a survey. *Comput. Ind. Eng.* 111, 467–480.
Pentico, D.W., 2007. Assignment problems: a golden anniversary survey. *Eur. J. Oper. Res.* 176 (2), 774–793.
Pitsoulis, L., 2009. Quadratic Semi-Assignment Problem, Encyclopedia of Optimization. Springer, pp. 3170–3171.
Punnen, A.P., Wang, Y., 2016. The bipartite quadratic assignment problem and extensions. *Eur. J. Oper. Res.* 250.3, 715–725.
Rinnooy Kan, A.H.G., Lageway, B.J., Lenstra, J.K., 1975. Minimizing total costs in one–machine scheduling. *Oper. Res.* 23, 908–927.
Sahni, S., Gonzalez, T., 1976. P-complete approximation problems. *ACM J.* 23, 555–565.
Saito, H., Fujie, T., Matsui, T., Matuura, S., 2009. A study of the quadratic semi-assignment polytope. *Discret. Optim.* 6, 37–50.
Sinclair, J.B., 1987. Efficient computation of optimal assignments for distributed tasks. *J. Parallel Distrib. Comput.* 4, 342–362.
Stone, H.S., 1977. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Trans. Softw. Eng.* SE-3, 85–93.
Wang, Y., Punnen, A.P., 2017. The boolean quadratic programming problem with generalized upper bound constraints. *Comput. Oper. Res.* 77, 1–10.