Task assignment in tree-like hierarchical structures

Cem Evrendilek1 · Ismail Hakki Toroslu2 · Seyedsasan Hashemikhabir2

Published online: 18 November 2016
© Springer Science+Business Media New York 2016

Abstract Many large organizations, such as corporations, are hierarchical by nature. In hierarchical organizations, each entity, except the root, is a sub-part of another entity. In this paper, we study the task assignment problem to the entities of a tree-like hierarchical organization. The inherent tree structure introduces an interesting and challenging constraint to the standard assignment problem. Given a tree rooted at a designated node, a set of tasks, and a real-valued function denoting the weight of assigning a node to a task, the Maximum Weight Tree Matching (MWTM) problem aims at finding a maximum weight matching in such a way that no tasks are left unassigned, and none of the ancestors of an already assigned node is allowed to engage in an assignment. When a task is assigned to an entity in a hierarchical organization, the whole entity including its children becomes responsible for the execution of that particular task. In other words, if an entity has been assigned to a task, neither its descendants nor its ancestors can be assigned to any task. In the paper, we formally introduce MWTM, and prove its NP-hardness. We also propose and experimentally validate an effective heuristic solution based on iterative rounding of a linear programming relaxation for MWTM.

Cem Evrendilek
cem.evrendilek@ieu.edu.tr
Ismail Hakki Toroslu
toroslu@ceng.metu.edu.tr
Seyedsasan Hashemikhabir
sasan@ceng.metu.edu.tr

1 Computer Engineering Department, Izmir University of Economics, 35330 İzmir, Turkey
2 Computer Engineering Department, Middle East Technical University, 06531 Ankara, Turkey
Keywords Task assignment · Integer programming · Linear programming relaxation · Heuristics · NP-hardness

1 Introduction

In the standard assignment problem (or sometimes referred to as linear assignment problem) (Burkard et al. 2009), the number of tasks and the number of agents are equal, and a scalar value is used to represent the cost/performance of assigning a task to an agent. The objective of the assignment problem is to determine such an assignment that assigns each task to a different agent and the total cost/profit is minimized/maximized. Many different variations of this problem (Geetha and Nair 1993; Armstrong and Jin 1995; Contreras et al. 2011; Li et al. 2013; Rainwater et al. 2014; de Klerk et al. 2015) have already been studied, including generalized assignment problem (Cohen et al. 2006; Fleischer et al. 2006) and Nonlinear Assignment Problems (Pardalos and Pitsoulis 2000). In this work, we investigate a new version of the standard assignment problem, which also appears in real-life applications.

In real life, many large organizations such as corporations, governments, and military have hierarchical structures. Hierarchical organizations can be viewed as trees where each node corresponds to an entity in the organization, and entity/sub-entity relationships are represented as parent–child relationships. In military task assignment, a task is assigned to a military unit as a whole including all of its sub-units, and each assignment has a cost associated with it. For example, the engagement rules for artillery target assignment directly correspond to this kind of a problem. For an artillery battery, which is composed of several artillery units, targets may be assigned to the whole artillery or to its sub-units, and each assignment has a cost/reward associated with it related to how many artillery units are used in the assignment.

Given a rooted tree, a set of tasks, and a real-valued function denoting the weight of assigning a node to a task, the MWTM problem seeks a matching with the maximum total weight such that all the tasks are assigned, and no node already assigned is allowed to have any ancestors that have also been assigned. In the standard assignment problem, agents are flat with no structure imposed on them, and one task is assigned to one agent. However, in the MWTM problem, since agents are organized as a tree, and sub-entities in the tree represent sub-parts of the agents, an additional constraint, named hereafter as the hierarchy constraint, is introduced to the assignment problem: When a task is assigned to an agent, no other assignment can be made to its descendants as they are assumed to be a part of an agent already assigned. Equivalently, if an agent is already assigned to a task, none of its ancestors in the tree can be assigned. In more general terms, on every path in a tree from the root to a leaf, there can be at most one assignment. This, in turn, leads to the observation that the number of leaves in the tree should be at least equal to the number of tasks to be executed. Otherwise, no feasible assignment exists. This result is proved later in the paper in Sect. 2.

A simpler version of the MWTM problem, in which each node has the same assignment weight for all the tasks to be performed (i.e., \( w_{i,j} = w_{i,k}, \forall j, k \in J \), where \( w_{i,j} \) corresponds to the weight of assigning node \( i \) to task \( j \) and \( J \) denotes the set of tasks), was introduced in Gulek and Toroslu (2010a). It is called tree-like weighted set
packing in Gulek and Toroslu (2010a) since the set/subset relationships form a tree, and the weight assigned to each set (or each node in the tree) can be interpreted as the weight of assigning a task to that node. The same hierarchy (independence) constraint has been enforced in order to prevent the selection of two sets having set/subset relationships (either directly or indirectly), and finally, the number of sets to be packed (selected) is specified to maximize the total weight. Tree-like weighted set packing problem is effectively a simpler version of the MWTM problem studied in this paper, and an effective dynamic programming solution to it has been developed in Gulek and Toroslu (2010a).

Although many different versions of assignment problems have been defined and explored, the literature contains very few problems remotely related to the MWTM problem. Two examples are Wang et al. (2013), and Shima et al. (2006). Similar to MWTM, both of these problems introduce different kinds of set constraints on the vertices of a bipartite graph, and they have both been shown to be NP-hard. Therefore, heuristic solutions have been proposed, namely a greedy heuristic in Wang et al. (2013), and a genetic algorithm based solution in Shima et al. (2006). Both these solutions have been shown to be quite effective. One more example of an interesting assignment problem interpretation with star constraints can be found in the work of Walteros et al. (2014). Walteros et al. (2014) attack this problem arising in the context of multi-sensor multi-target tracking by studying two different formulations. First, a continuous nonlinear program and its linearization are introduced. Second, the standard multidimensional assignment problem formulation is stated as a set partitioning problem to be solved via branch and price. These approaches are both shown to be computationally viable in Walteros et al. (2014).

The MWTM problem, already introduced in Gulek and Toroslu (2010b), has been attacked by using a genetic algorithm (GA) based heuristic. In Gulek and Toroslu (2010b), it has been shown that GA works quite effectively in terms of solution quality for randomly generated inputs. Although the number of iterations was not very large, due to the computational cost of each genetic operator among the chromosome populations, each iteration takes a considerable amount of time to complete, and therefore we have observed that the execution times needed to achieve results of high quality comparable to those obtained in this paper are much longer.

Since the GA approach uses a generic heuristic (slightly customized for the problem), it cannot be considered directly comparable to the problem-specific heuristic employed in this study, which is much more effective. Moreover, although the GA approach has been applied to different sized inputs, some significant input parameters such as the shape of the tree and the distributions of the weights have not been explored in its evaluation in Gulek and Toroslu (2010b). We have hence compared the quality of the solutions found by our heuristic proposed in this paper with the optimal solutions obtained from the integer linear programming (ILP) formulation developed. This paper has the following additional contributions compared to Gulek and Toroslu (2010b):

- The MWTM problem is shown to be NP-hard,
- An ILP formulation for the problem is presented,
A Heuristic approach based on iterative rounding of an LP relaxation for \textit{MWTM} is developed,

The effectiveness of the proposed heuristic is verified through an experimental study.

Iterative LP relaxation or rounding algorithms have previously been used (Vazirani 2002). A 2-approximation algorithm is presented in Jain (2001) to find a minimum-cost subgraph having at least a specified number of edges in each cut. The approach taken in Jain (2001) has been generalized and formalized in Jain (2000). In order to exploit the full power of LP, a new technique called iterative rounding was introduced in Jain (2000). Iterative rounding is used in Jain (2000) to iteratively recompute the best fractional solution while maintaining the rounding of the previous phases. Although an iterative rounding based heuristic solution is developed in this paper for \textit{MWTM}, the presence of the hierarchy constraint makes it difficult to consider fractional values from the highest to the lowest in an effort to come up with an approximation bound.

The rest of the paper has been organized as follows. The next section formally introduces the problem, and proves its NP-hardness. Sect. 3 develops an ILP formulation for \textit{MWTM}, and Sect. 4 presents how its LP relaxation can be iteratively used to develop an effective heuristic. Sect. 5 describes the experiments and their results. Finally, the last section presents concluding remarks and open problems.

2 Computational complexity of MWTM

We will now formally define the \textit{MWTM} problem.

\textbf{Definition 1} A tree \(T(V, E)\) rooted at a node \(r \in V\) where \(V = \{1, 2, \ldots, n\}\), and a separate set \(J = \{1, 2, \ldots, m\}\) of tasks are given. Associated with each node \(i \in V\) is a real-valued function \(w_{i,j}\) denoting the weight of assigning node \(i\) to task \(j\) for all \(i \in V\) and \(j \in J\). \textit{MWTM} is then the problem of finding an assignment of the nodes to all the tasks with the maximum total weight in such a way that the assignment between the nodes and the tasks forms a matching, and no node assigned to a task is allowed to have any ancestors (or descendants) that have also been assigned to a task.

It should be noted that the requirement for the weight function to be defined for all combinations of nodes and tasks in \textit{MWTM} stems from a deliberate decision. \textit{MWTM} might have some possible variants where assignments between some combinations of nodes and tasks can be disallowed. As \textit{MWTM} can be directly reduced to these forms, their NP-hardness would easily follow once \textit{MWTM} is shown to be NP-hard.

The constraint associated with the hierarchical structure of the tree dictates that no two nodes on the same path from the root \(r\) to a leaf node in \(T\) can ever be simultaneously assigned in any feasible solution to an instance of \textit{MWTM}.

\textbf{Definition 2} Two paths in a tree from the root to two distinct nodes \(i\) and \(j\) are said to be \textit{independent paths} if \(i\) is neither an ancestor nor a descendant of \(j\).

In light of this definition, the hierarchy constraint can simply be restated as the requirement that the paths from the assigned nodes to the root are all pairwise independent.
A trivial observation to be made at this point is that it is both necessary and sufficient to ensure that the number of leaf nodes is greater than or equal to the number of tasks in a given instance of MWTM for the existence of a solution.

**Lemma 1** A given instance of MWTM represented by \((T(V, E), r, w, J)\) where \(T\) is a tree rooted at \(r\), and \(w(i, j)\) is the weight of assigning node \(i\) to task \(j\) for all combinations of \(i \in V\) and \(j \in J\) has a solution if and only if \(|\lambda| \geq |J|\) where \(\lambda\) denotes the set of leaf nodes in \(T\).

**Proof** Let us prove the sufficiency part first. If a given MWTM instance has a solution, then there exists an assignment of \(|J|\) nodes in \(T\) to \(|J|\) tasks. The hierarchy constraint by definition requires that no two among these \(|J|\) nodes has an ancestor-descendant relationship, and they are, hence, on \(|J|\) mutually independent paths (see Definition 2). Therefore, the number of leaves in \(T\) denoted by \(|\lambda|\) cannot be less than the number of available independent paths from these \(|J|\) nodes to the root.

In order to prove the necessity part, we proceed as follows: As there are as many as \(|\lambda| \geq |J|\) leaf nodes, any subset of \(|J|\) leaves out of \(\lambda\) can be freely selected, and assigned to available tasks in a random order. Since each selected node is on an independent path ensuring that the hierarchy constraint is not violated, a feasible solution can be obtained. \(\square\)

MWTM can be shown to be NP-hard by proving first that the decision version of it is NP-complete. The decision version of the MWTM problem is stated as follows: Given a tree \((T(V, E))\) rooted at a node \(r \in V\), a set \(J\) of tasks, a function \(w(\cdot, \cdot)\) denoting the weights of the assignments of nodes to tasks, and a real \(\kappa\), does there exist an assignment of the nodes to all the tasks with total weight larger than or equal to \(\kappa\) such that every task is assigned to a node, a node is assigned to at most one task, and no node assigned to a task has any ancestors assigned? We now show that the decision version of MWTM is NP-complete by a polynomial time reduction from 3-SAT, which is one of Karp’s 21 NP-complete problems (Karp 1972). 3-SAT is defined as the problem of deciding whether a satisfying truth assignment is possible for the variables of a given Boolean formula in conjunctive normal form (CNF), where each clause is a disjunction of exactly three literals and each literal is either a variable or its negation.

A given instance of the 3-SAT problem is transformed to a corresponding instance of MWTM in time polynomial in the size of the input. Let a given instance of 3-SAT have \(n\) variables denoted by \(x_i\) where \(1 \leq i \leq n\) and a 3-CNF formula \(C_1 \land C_2 \land \ldots \land C_m\), where each \(C_i = C_{i,1} \lor C_{i,2} \lor C_{i,3}\) is a disjunction of three literals. The transformation starts by introducing the root node designated by \(r\) to the initially empty tree \(T\) of the corresponding MWTM instance at level 0. The root node \(r\) is numbered as 1. For each variable \(x_i\), two nodes numbered \(2i\) and \(2i + 1\) corresponding to assigning \(true\) to \(x_i\) and \(false\) to \(\neg x_i\) respectively are then created. The parents of all such nodes are set to point to node \(r\). As there are \(n\) distinct variables in the given 3-SAT instance, the root \(r\) of \(T\) in the corresponding MWTM instance becomes populated with a total of \(2n\) children at level 1 of \(T\) after this step. These are called variable nodes (see Fig. 1).

In the final step of the construction of \(T\), for each literal \(C_{i,j}\) where \(1 \leq i \leq m\), and \(1 \leq j \leq 3\), a node numbered \(1 + 2n + 3(i - 1) + j\) is created. The parent of such a node is set to \(2k\) if \(C_{i,j} = x_k\), and to \(2k + 1\) otherwise if \(C_{i,j} = \neg x_k\) where...
1 \leq k \leq n$. The tree $T$ constructed is shown in Fig. 1. While parent–child relationships are indicated by solid lines in this figure, dashed lines depict the weight function $w_{i,j}$.

It should be noted that the variable nodes at level 1 will have as many children as there are occurrences of the corresponding literal at level 2, while the nodes corresponding to literals in clauses at level 2 will have a single edge to their parent as shown in the figure. The nodes at level 2 are accordingly called literal nodes.

Once we obtain the tree $T$ in the MWTM instance corresponding to the given instance of 3-SAT, we also set the number of tasks to $m + n$. Each task $t_i$ for $1 \leq i \leq m$ corresponds to satisfying the respective clause $C_i$. We call these clausal tasks. Each task $t_i$ for $m + 1 \leq i \leq m + n$, however, are used to enforce the condition that the corresponding variable $x_{i-m}$ is set to one of the values true or false consistently over all clauses. We call such tasks enforcement tasks.

Apparently, the total number of nodes in $T$ in the corresponding instance of MWTM is given by $1 + 2n + 3m$, where $n$ and $m$ are the number of variables and clauses respectively specified in the given 3-SAT instance. The number of tasks, on the other hand, is $n + m$. After the total weight in the corresponding instance of the decision version of MWTM is initialized as $\kappa = n + m$, the concluding step of the transformation is to appropriately set the corresponding values $w_{i,j}$ for all nodes $1 \leq i \leq 1 + 2n + 3m$ and all tasks $1 \leq j \leq m + n$ as shown in Eq. (1) below:

$$w_{i,j} = \begin{cases} 
0, & \text{if } i = 1 \land 1 \leq j \leq m + n \\
1, & \text{if } 2 \leq i \leq 2n + 1 \land j = m + \left\lfloor \frac{i}{2} \right\rfloor \\
1, & \text{if } 2n + 2 \leq i \leq 2n + 3m + 1 \land j = \left\lfloor \frac{i - 2n - 2}{3} \right\rfloor + 1 \\
0, & \text{otherwise}
\end{cases}$$

(1)
The weights to carry out any one task by the root are all initialized to zero. For a variable node $2 \leq i \leq 2n + 1$ at level 1 corresponding to $x_{\lfloor \frac{i}{2} \rfloor}$ or $\neg x_{\lfloor \frac{i}{2} \rfloor}$ depending on whether $i$ is even or odd respectively, however, the weights of executing tasks are set in such a way that a consistent assignment of truth values to individual variables can be enforced. The only task whose execution by the variable node $i$ can make a positive contribution to the solution is therefore the corresponding enforcement task $tm_{\lfloor \frac{i}{2} \rfloor}$. A level 2, are the literal nodes ranging from $2n + 2$ to $1 + 2n + 3m$, corresponding to the literals in the clauses of the given 3-SAT instance. Each literal can accordingly be set to satisfy a clause in which it occurs. Therefore, the weight $w_{i,j}$ of assigning node $i$ at level 2 to a clausal task $t_j$ is appropriately defined as 1 to reflect a feasible assignment. In this case, if node $i$ is assumed to correspond to a literal $C_{p,q}$, then the clausal task $t_j$ corresponds to the clause $C_p$. Therefore, the equalities $i = 1 + 2n + 3(p - 1) + q$, and $j = p$ must both hold. Noting that $q$ can only take on the values 1 through 3 inclusive readily gives $p = \lfloor \frac{i - 2n - 2}{3} \rfloor + 1$, and $q = (i - 2n - 2) \mod 3 + 1$. All other combinations of nodes and tasks are associated with weight 0.

It should be pointed out that an MWTM instance so constructed from a given instance of 3-SAT would always lend itself to a feasible solution by Lemma 1 since the number of leaf nodes in $T$ is greater than or equal to the number of tasks. This is easily seen by assuming without loss of generality that at least one of a variable or its negation for each of the $n$ variables is used as a literal in one of the $m$ clauses in the given 3-SAT instance. Provided that $t$ denotes the number of variable nodes without any children in $T$ in the corresponding instance of MWTM, $m \geq (2n - t)/3$ always holds, where $0 \leq t \leq n$. As $m$, $n$, and $t$ are all non-negative, $m \geq (2n - t)/3 = \frac{2}{3}n - \frac{1}{3}t \geq \frac{1}{3}n - \frac{1}{3}t \geq \frac{1}{3}n - \frac{1}{3}t$ is easily obtained. Multiplying both sides in $m \geq \frac{1}{3}n - \frac{1}{3}t$ by two, and then adding $m$ to both sides, we obtain $3m \geq m + n - t$, and then $3m + t \geq m + n$ by rearranging. The feasibility of the corresponding MWTM instances obtained through the transformation described is hence confirmed.

Given the transformation described, we make the following straightforward observation to be used in a lemma to follow.

**Observation 1** In any solution with a total weight of $\kappa = n + m$ to the corresponding MWTM instance obtained from a given 3-SAT instance through the transformation described, a literal node at level 2 can be assigned to a related clausal task if and only if no other literal node corresponding to its negation has any assignments.

*Proof* In any solution with a total weight of $n + m$ to the corresponding MWTM instance, the variable nodes at level 1 are assigned to all the enforcement tasks such that a consistent assignment of truth values to variables is ensured. Then, the children at level 2 of only the unassigned variable nodes can be used to fulfill the clausal tasks.

We can easily prove the following lemma now.

**Lemma 2** A given 3-SAT instance with $n$ variables, and $m$ clauses is satisfiable if and only if the corresponding MWTM instance obtained through the transformation described above has a solution with total weight $\kappa = n + m$. 

\[\square\]
Proof Let us first prove the sufficiency part: If a given 3-SAT instance is satisfiable, then there exists an assignment of truth values to all \( n \) variables, which makes all \( m \) clauses evaluate to true. This, in turn, implies that at least one literal in every clause can be made true. The corresponding \( MWTM \) instance is therefore seen to have an optimal assignment with weight \( n + m \): Each clausal task in this scheme is assigned to one literal node corresponding to a literal satisfying this clause. Each enforcement task is assigned to the variable node representing the negation of the literal that evaluates to true in the satisfying truth assignment to the given 3-SAT instance. This is indeed a matching, since each task is matched to a different node and no node which is a parent of an already assigned literal node is assigned to a task. The latter condition is guaranteed by the fact that if a literal node is assigned to a clausal task, then the node corresponding to the negation of this literal at a higher level can only be used to accomplish the respective enforcement task. The total weight is also the maximum possible.

In order to prove the necessity part, let us assume that there exists a solution with total weight \( n + m \) to the corresponding \( MWTM \) instance. A truth assignment for the given 3-SAT instance can be obtained by setting each variable \( x_i \) to true if the corresponding enforcement task \( m + i \) is assigned to node \( 2i + 1 \), and to false if the assignment is to node \( 2i \). This truth assignment definitely satisfies the 3-CNF expression of the given 3-SAT instance by Observation 1.

To illustrate the idea in the reduction process, let us consider the following example.

Example 1 A 3-CNF formula \((p \lor \neg q \lor \neg p) \land (p \lor r \lor \neg s) \land (q \lor r \lor s)\) with four variables, and three clauses is given. While the variables are named \( p, q, r, \) and \( s \), the clauses are denoted by \( C_1 = (p \lor \neg q \lor \neg p) \), \( C_2 = (p \lor r \lor \neg s) \), and \( C_3 = (q \lor r \lor s) \). The corresponding tree structure obtained through the transformation just described is given in Fig. 2, while the accompanying weights of assigning nodes to tasks are shown in Fig. 3. While the nodes are numbered from 1 through 18, tasks are called \( t_{C_1}, t_{C_2}, \) and \( t_{C_3} \) corresponding to the clausal tasks, and \( t_p, t_q, t_r, \) and \( t_s \) corresponding to the enforcement tasks.

The given 3-CNF Boolean expression is satisfiable if and only if the \( MWTM \) instance identified with the corresponding tree structure given in Fig. 2, and accom-
panying weight function depicted in Fig. 3, has an assignment with a total weight of $3 + 4 = 7$.

It should be noted that the construction constrains the weight values for variable nodes 2 through 9 in the table to the left of Fig. 3 through enforcement tasks in such a way that a node corresponding to a variable and another node corresponding to its negation cannot at the same time contribute to a solution. An inspection of the weight values for literal nodes 10 through 18 in the table on the right in Fig. 3 similarly reveals that only one of three such nodes can contribute to a solution through a corresponding clausal task.

If a given 3-CNF formula is satisfiable, any satisfying truth assignment induces a straightforward matching of the nodes to all the available tasks. In this matching, the variable nodes at level 1 evaluating to false with respect to the given truth assignment are all assigned to their corresponding enforcement tasks, leaving only their negations for a consistent instantiation over the entire set of clauses. There are several ways to satisfy the above example formula. Let us assume that we pick an assignment as follows: Variables $p$ and $q$ are both assigned to true while $r$ and $s$ can be assigned randomly. If we reflect these choices on the instance of $MWTM$ obtained, clausal tasks $t_{C1}$ and $t_{C2}$ are assigned respectively to nodes 10 and 13, both corresponding to $p$, while task $t_{C3}$ is assigned to node 16 corresponding to $q$. Then, we can assign enforcement task $t_p$ to node 3 corresponding to $\neg p$, and task $t_q$ to node 5 corresponding to $\neg q$. Finally, we assign task $t_r$ to either one of the nodes 6 or 7, and task $t_s$ to either one of the nodes 8 or 9. For the last two, $r$ and $s$, the choice is not really important, since neither one of these variables has been used in satisfying the clauses. \hfill \□

The transformation described in this section is certainly polynomial in the size of the given 3-SAT instance. This is easily observed by noting that the number of nodes created to form a tree in the corresponding instance of $MWTM$ is $2n + 3m + 1$. For each of these nodes, $O(n + m)$ additional processing is needed to initialize its weight for a possible assignment to a total of $n + m$ tasks in the worst case. The total time is hence proportional to $O((n + m)^2)$. 

Fig. 3 $w_{i,j}$ values for the corresponding $MWTM$ instance obtained from the example 3-SAT instance

| $t_{c1}$ | $t_{c2}$ | $t_{c3}$ | $t_p$ | $t_q$ | $t_r$ | $t_s$ |
|---------|---------|---------|------|------|------|------|
| 1       | 0       | 0       | 0    | 0    | 0    | 0    |
| 2       | 0       | 0       | 1    | 0    | 0    | 0    |
| 3       | 0       | 0       | 0    | 1    | 0    | 0    |
| 4       | 0       | 0       | 0    | 0    | 1    | 0    |
| 5       | 0       | 0       | 0    | 0    | 0    | 1    |
| 6       | 0       | 0       | 0    | 0    | 1    | 0    |
| 7       | 0       | 0       | 0    | 0    | 0    | 1    |
| 8       | 0       | 0       | 0    | 0    | 0    | 0    |
| 9       | 0       | 0       | 0    | 0    | 0    | 0    |
Lemma 3 The decision version of the MWTM problem is NP-complete.

Proof A given solution to an instance of the decision version of MWTM can be easily checked in polynomial-time as to its feasibility. We simply verify that all the tasks are assigned, no node is assigned to more than one task, the hierarchy constraint is not violated, and the total weight of the solution is greater than or equal to the specified \( \kappa \) value. Therefore, the decision version of the problem belongs to the complexity class NP. This fact coupled with Lemma 2 and the fact that the transformation is polynomial in the size of the given 3-SAT instance establish the desired result. \( \square \)

Lemma 3 simply leads to the NP-hardness of the optimization version of MWTM as recapped in the following theorem.

Theorem 1 The MWTM problem is NP-hard.

3 An ILP formulation for the MWTM problem

In an instance of MWTM, the number of nodes organized as a tree, \( T \), and the number of tasks are given by \( n \) and \( m \) respectively. The weight of executing each task \( j \) by a node \( i \) is also denoted by \( w_{i,j} \), where \( i \in \{1 \ldots n\} \) and \( j \in \{1 \ldots m\} \). Let \( r \) designate the root of this tree, \( T \). Let us denote by \( \lambda \subseteq \{1 \ldots n\} \) the set of leaf nodes of \( T \). Each unique path from the root \( r \) to a leaf node \( k \in \lambda \) is represented by a set of nodes on this path, which is denoted by \( \Pi_k \). An integer linear programming (ILP) formulation of the MWTM problem can thus be given as:

\[
\begin{align*}
\text{maximize} & \quad \sum_{i=1}^{n} \sum_{j=1}^{m} w_{i,j} \times x_{i,j} \\
\text{subject to} & \quad \sum_{j=1}^{m} x_{i,j} \leq 1, \forall i \in \{1 \ldots n\} \\
& \quad \sum_{i=1}^{n} x_{i,j} = 1, \forall j \in \{1 \ldots m\} \\
& \quad \sum_{i \in \Pi_k} \sum_{j=1}^{m} x_{i,j} \leq 1, \forall k \in \lambda \\
& \quad x_{i,j} \in \{0, 1\}, \forall i \in \{1 \ldots n\} \text{ and } \forall j \in \{1 \ldots m\}
\end{align*}
\]

The objective function (2a) employs the binary decision variable \( x_{i,j} \), which is set to 1 if a node \( i \) is assigned to a task \( j \), and to 0 otherwise. The constraints (2b) simply mean that a node can be assigned to at most one task. The constraint class (2c) is used to enforce the condition that every task is executed by a single node. In order to ensure that at most one node can be assigned to a task on any path from the root to a leaf node, (2d) is used. Finally, (2e) is there to make sure that decision variables \( x_{i,j} \) can only take on the integer values 0 and 1.
4 The bottom-up assignment heuristic

In this section, a heuristic solution is developed for MWTM. The given ILP formulation can readily be relaxed to an LP by substituting the constraint \(0 \leq x_{i,j} \leq 1\) in place of (2e). As \(x_{i,j}\) can take on fractional values in the range [0–1], the BOTTOM-UP-ASSIGNMENT (BOA) procedure given in Algorithm 1 is used to obtain a feasible integer solution.

Before giving a detailed explanation of BOA, a high level description of the heuristic can be presented as follows: First, a call is made to obtain a solution to the LP relaxation of the ILP formulation of MWTM. Then, this possibly fractional solution is converted to a feasible, partial 0–1 solution where leaf nodes with greater fractional assignments are favored. The remaining nodes and tasks that are still not allocated at this current iteration, if any, form a smaller instance of MWTM, which is simply handed over to a subsequent iteration. At this successive iteration, a new call to the LP relaxation for the smaller instance is issued. This process is repeated as long as there are tasks not assigned yet. The entire heuristic hence operates via making leaf-assignments between successive calls to the LP relaxation.

BOA in Algorithm 1 assumes that the number of tasks and the nodes are \(m\) and \(n\) respectively. The number of tasks, \(m\), is greater than 1 to address only the non-trivial

---

**Algorithm 1 BOTTOM-UP-ASSIGNMENT(** \(T', r, w, J\) **)**

**Input:** \(T(V,E)\) is a tree with \(V = \{1,2,\ldots,n\}\) rooted at node \(r \in V\); \(J = \{1,2,\ldots,m\}\) is the set of tasks; \(w\) is a 2-dimensional array where \(w_{i,j}\) denotes the weight of assigning node \(i\) to task \(j\) for all \(i \in V,\) and \(j \in J\).

**Output:** A feasible assignment \(\alpha\).

1: \(\alpha \leftarrow \emptyset\);
2: \(\text{tasksLeft} \leftarrow [1\ldots m];\ \text{nodesLeft} \leftarrow [1\ldots n]\);
   // a call to LP with substitutions \(x_{i,j} \leftarrow 1\ \forall (i, j) \in \alpha\)
3: \(x_{i,j} \leftarrow \text{LP}(T,r,w,J,\alpha)\ \forall i \in [1\ldots n] \) and \(j \in [1\ldots m]\);
4: \(T' \leftarrow \text{deleteNodes}(T,\{i|\ (i, j) \in \alpha\})\); // delete all nodes assigned
5: \(\lambda \leftarrow \text{leaves}(T')\); // \(\lambda\) leaves with a non-zero assignment are examined in decreasing order of \(x_{i,j}\) values
6: \(\lambda\ \text{while (} \max x_{i,j} \neq 0\ \text{where } (i \in \lambda \cap \text{nodesLeft}) \text{and } (j \in \text{tasksLeft})\) can be found do
7: \(\alpha \leftarrow \alpha \cup \{(i,j)\}\); // record this assignment
8: \(\lambda \leftarrow \lambda - \{i\}\);
9: \(\lambda \leftarrow \lambda - \{j\};\ \\text{tasksLeft} \leftarrow \text{tasksLeft} - \{j\};\ \\text{nodesLeft} \leftarrow \text{nodesLeft} - \{i\};\)
10: \(\Pi_i \leftarrow \text{set of nodes on the path from } i \text{ to } r \text{ in } T'\);
11: \(\\text{for each } k \in (\Pi_j - \{i\})\) do // remove all nodes from \(i\) up to \(r\) in \(T'\) from consideration
12: \(\text{nodesLeft} \leftarrow \text{nodesLeft} - \{k\};\)
13: \(T' \leftarrow \text{deleteNodes}(T',\{i\})\); // delete \(i\) in \(T'\)
14: \(\text{if (} \text{tasksLeft} \neq \emptyset\) then
15: \(\ T' \leftarrow \text{deleteNodes}(T',\lambda);\ \\text{to give ancestors a chance}
16: \(\text{leavesLeft} \leftarrow \text{nodesLeft} \cap \text{leaves}(T');\ \\text{nodesLeftInT'} \leftarrow \text{nodesLeft} \cap \text{nodes}(T');\)
17: \(\text{if (} \text{nodesLeftInT'} \text{ has nodes with } x_{i,j} \neq 0\) and \(|\text{leavesLeft}| \geq |\text{tasksLeft}|\) then
18: \(\text{go to step 5;\)}
19: \(\text{else\)}
20: \(\text{go to step 3;\)}
21: \(\text{return assignment } \alpha;\)
instances of MWTM. It is also assumed that the root of the tree is dummy, i.e., it cannot be assigned to a task, as the other nodes would be rendered useless otherwise.

BOA starts by initializing the set $\alpha$ of assignments at line 1 to be empty. At line 2, both sets tasksLeft and nodesLeft are initialized to keep track of the remaining tasks and the remaining nodes respectively. The call to LP , next at line 3, takes as parameters the original MWTM instance along with the assignments made so far. The effect of the variable $\alpha$ is to set all $x_{i,j}$ values to 1 for all node-task pairs $(i, j) \in \alpha$ at subsequent calls to the corresponding LP formulation. Line 4 deletes all the nodes in $T$ assigned so far by BOA to obtain a new tree $T'$. This is achieved simply by pruning the assigned nodes, and hence their descendants from $T$ to obtain $T'$. It should be observed at this point that the most recent LP relaxation formulation at line 3 corresponds exactly to this residual MWTM instance as represented by the current values of the variables $T'$, nodesLeft, tasksLeft, and $w$ held at the time line 4 is executed.

After the set $\lambda$ is populated with a copy of the leaf nodes in $T'$ at line 5, the leaves with a non-zero assignment in it are examined in the order of non-increasing $x_{i,j}$ values in the while-loop between lines 6 through 14. The loop iterates as long as it is possible to find a non-zero assignment between the leaf nodes and the tasks not yet assigned. Among the leaves in $\lambda$ which have not yet been assigned to a task, only those not removed due to the hierarchy constraint are considered eligible. This is reflected by the expression $(i \in \lambda \cap nodesLeft)$ where nodesLeft keeps track of the remaining nodes in the original tree $T$ which have yet been neither assigned nor left in a non-assignable state as a result of the hierarchy constraint.

The thread of control is transferred to line 15 to test whether any tasks have been left unassigned as soon as it breaks out of the loop. If all tasks have already been assigned, the set of assignments constructed so far is returned at line 22 as the solution. Otherwise, the remaining leaf nodes are first deleted from $T'$ at line 16 to give their ancestors a chance before a new call to LP is made. Then at line 17, the following are computed: the set of the leaves in $T'$ that are also in nodesLeft denoted by leavesLeft, and the set of all the nodes in $T'$ that are also in nodesLeft represented by nodesLeftInT'. Finally, at line 18, a conditional check consisting of the conjunction of two expressions is performed. The former expression evaluates to true if there are nodes in $T'$ that can still be used for further assignments. The latter expression called the feasibility invariant is maintained throughout the entire execution of the algorithm. This basically ensures that the number of the leaf nodes still assignable are always greater than the number of the remaining tasks. If both expressions evaluate to true, execution continues by setting $\lambda$ safely to the leaf nodes in $T'$ at line 5 to prepare for the subsequent execution of the while-loop once more. Otherwise a jump to line 3 occurs, performing a new invocation to LP. In this case, all the deletions performed at line 16 in $T'$ are effectively rolled back at line 4 by taking into account only the assignments previously made.

4.1 The implementation issues and correctness of BOA

Both deleteNodes() and leaves(), which are based on post-order traversal, run in time proportional to the number of nodes in the tree on which they operate. An implementation making possible an efficient evaluation at the start of every iteration of
the while-loop employs max-heaps, one for every unassigned task. The roots of the max-heaps are also organized as a max-heap. Overall running time complexity of the heuristic is, however, dominated by the calls to LP at line 3As after each call, if a feasible solution exists, BOA assigns at least one task before the next call to LP, the total number of LP calls made is equal to the number of tasks, \( m \), in the worst case. Since LP lends itself to polynomial solutions \( (\text{Khachiyan} 1979; \text{Karmarkar} 1984) \), the worst case running time of the BOA heuristic is also polynomial. The overhead originating from the repetitive nature of the heuristic is discussed in the next section. It is shown through experiments that the actual observed average value for the number of times the call at line 3 to LP gets executed is almost a constant.

If there is a feasible solution, in the form of 0–1 assignments, to the ILP formulation of a given MWTM instance, its LP relaxation clearly has a fractional assignment with a total weight at least that of ILP. In such a case, this fractional assignment can always be converted to a feasible 0–1 assignment by BOA in Algorithm 1. In an effort to prove this, a series of lemmas will be presented and some observations regarding the algorithm will be made.

**Definition 3** In a feasible solution to the LP relaxation of a given MWTM instance, a node \( i \) in the tree \( T \) associated with at least one non-zero \( x_{i,j} \), but without having any such descendants in \( T \) is defined as an effective leaf with respect to the corresponding LP relaxation solution. The set of all such nodes is termed effective leaves.

In light of this definition, the following lemma can now be stated regarding the LP relaxation formulation corresponding to a given MWTM instance.

**Lemma 4** If the LP relaxation to a given MWTM instance has a solution, then the number of effective leaf nodes in the corresponding LP relaxation is greater than or equal to the number of tasks in the given problem instance.

**Proof** If a given MWTM instance’s LP relaxation has a solution, then the constraints (2b) through (2d) must hold. Therefore, we obtain by summing constraints (2c) over all possible \( j \) values:

\[
\sum_{j=1}^{m} \sum_{i=1}^{n} x_{i,j} = m
\]  
(3)

Let \( \lambda_e \) denote the set of effective leaves in \( T \) with respect to the particular LP relaxation solution. Since no nodes other than those in \( \lambda_e \) and their ancestors can have a non-zero \( x_{i,j} \) value associated with them, we next sum constraints (2d) over all the effective leaf nodes to obtain:

\[
\sum_{k \in \lambda_e} \sum_{i \in \Pi_k} \sum_{j=1}^{m} x_{i,j} \leq |\lambda_e|
\]  
(4)

As the sum of individual \( x_{i,j} \) values in (3) is less than or equal to the sum in (4) over all paths leading to effective leaf nodes, we conclude:

\[
m \leq |\lambda_e|
\]  
(5)
We can now establish the following lemma by noting that the number of effective leaves with respect to the corresponding LP relaxation solution of a given $MWTM$ instance actually forms a lower bound for the number of leaf nodes in $T$.

**Lemma 5** The corresponding LP relaxation of a given $MWTM$ instance, denoted by $(T(V, E), r, w, J)$, has a solution if and only if $|\lambda| \geq |J|$ where $\lambda$ denotes the set of leaf nodes in $T$.

**Proof** As for sufficiency; if the LP relaxation has a solution, then, by Lemma 4, $|\lambda_e| \geq |J|$ where $\lambda_e$ is the set of effective leaves. As $|\lambda| \geq |\lambda_e|$, $|\lambda| \geq |J|$ follows easily.

In order to prove necessity, on the other hand, we observe by Lemma 1 that if $|\lambda| \geq |J|$, then the given $MWTM$ instance has a solution. This latter result definitely implies the existence of a solution to the corresponding LP relaxation formulation. $\square$

**Definition 4** An execution of BOA between successive calls to LP at line 3 is called an iteration.

**Theorem 2** The BOA heuristic in Algorithm 1 returns a feasible solution whenever there exists one.

**Proof** The algorithm will keep performing iterations to obtain an assignment until we run out of the tasks in $tasksLeft$. Every single iteration of BOA is launched at line 3 to discover an assignment for a smaller residual $MWTM$ instance. This smaller instance is identified by the values that the tree $T'$, $nodesLeft$, and $tasksLeft$ have right after the statement at line 4 is executed.

It is known by Lemma 5 that when the number of leaf nodes in $nodesLeft$ (given by $leaves(T') \cap nodesLeft$ as would be computed at line 5) is greater than or equal to the number of remaining tasks in $tasksLeft$ at the start of an iteration $i$ before a call to LP, there must exist a feasible solution to the corresponding LP relaxation formulation at line 3. This, in turn, implies through Lemma 4 that the number of effective leaf nodes in $T'$ computed at line 4 is greater than or equal to the number of tasks in $tasksLeft$. Therefore, at least one assignment between an effective leaf and an available task will be performed in the while-loop between lines 6 through 14 in every iteration of the algorithm, and BOA will eventually terminate.

An additional observation can be made by noting that the feasibility variant cannot be violated so long as the while-loop iterates. The reason is that it holds at the start, and the only type of modification allowed in the body of the loop is the assignment of an effective leaf to an available task. Such an assignment, however, removes exactly one leaf node and one task from consideration, ensuring that the feasibility invariant is still maintained.

Once the control breaks out of the while-loop, either a feasible solution by BOA is returned if there are no more tasks left, or otherwise all the useless leaf nodes which survived the previous while-loop are deleted at line 16. These leaf node deletions are the only deletions that can possibly violate the feasibility invariant. Hence, once such a violation is detected at line 18, a jump at line 21 initiates the next iteration where all such deletions are effectively rolled back by reconstructing $T'$ from scratch. In case
there are no such violations, control goes once more to the while-loop. Consequently, the feasibility invariant is maintained from one iteration to the next throughout the entire execution of the algorithm.

BOA will then always find a feasible solution as long as the feasibility constraint holds at the start of the first iteration. This, however, is already guaranteed by Lemma 1, hence completing the proof.

It is clearly not easy, if not impossible, to generate a feasible 0–1 assignment at once using only possibly fractional non-zero assignments obtained from the corresponding LP relaxation solution for all the \( MWTM \) instances. The difficulty stems from the fact that the distribution of fractional assignment values returned by the corresponding LP relaxation solution may not easily lend itself to an integer valued assignment for all tasks without violating the hierarchy constraint. Therefore, as in BOA, LP might need to be called iteratively in order to cover the tasks not already assigned in the previous iterations. In an attempt to reduce the number of iterations, however, at each iteration in BOA, as many fractional assignment values as allowed by the feasibility invariant are checked as to their eligibility to contribute to a feasible solution. Moreover, some zero valued assignments in previous iterations may change into non-zero in subsequent iterations, leading to solutions with smaller total weights. Thus, we prefer to use the earliest LP results with non-zero assignments.

An example is provided below for a better understanding of how BOA operates.

**Example 2** An \( MWTM \) instance is depicted in Fig. 4. The corresponding tree representing the hierarchical structure of an organization has 6 nodes numbered in level order as shown in the figure where the root node is denoted by 1. It is assumed in this particular example that the organization has 3 tasks to be executed not explicitly shown in the figure. The weights of executing these tasks, namely \( t_1 \), \( t_2 \), and \( t_3 \) are given in this order as a triple inside each node. In this example, the corresponding ILP formulation will produce the optimal solution with the following assignments highlighted with the corresponding weight values in red in Fig. 4:

- Task \( t_1 \) is assigned to node 4 with weight 6,
- Task \( t_2 \) is assigned to node 5 with weight 4,
- Task \( t_3 \) is assigned to node 3 with weight 8.

![Fig. 4 A sample tree structure with six nodes, and three tasks not explicitly shown. Weights of executing each task \( t_1 \), \( t_2 \), and \( t_3 \) are given in this order inside each node as triples. Red values (also underlined) correspond to node-task assignments obtained from ILP solution (Color figure online)](image-url)
Fig. 5  Green values (also underlined) correspond to fractional node-task assignments obtained from the corresponding LP relaxation solution where task $t_1$ is assigned to both nodes 3 and 4, task $t_2$ is assigned to nodes 5 and 6, and finally task $t_3$ is assigned to nodes 2 and 3 all with the same value $\frac{1}{2}$ (Color figure online)

The next figure, Fig. 5, presents a solution obtained by the corresponding LP relaxation on the same problem instance (assignments are shown in green). The solution is as follows:

- Task $t_1$ is assigned to nodes 3 and 4 both with the same fractional value 0.5, contributing to the total weight by $7 (= 8/2 + 6/2)$,
- Task $t_2$ is assigned to nodes 5 and 6 both with the same value 0.5 again, contributing to the total weight by $4 (= 4/2 + 4/2)$,
- Finally, Task $t_3$ is assigned to nodes 2 and 3 both with the same value 0.5, causing this time an increase of $8 (= 8/2 + 8/2)$ in the total weight.

Since LP is allowed to make fractional assignments, the weight 19 of the solution achieved by LP is even higher than the optimal 18 found by ILP. The direct application of LP unfortunately cannot produce an integer assignment for the given $MWTM$ instance. BOA in Algorithm 1, however, will work its way to a feasible solution as follows on this example:

- After a call to LP is made at line 3 in the first iteration, potentially fractional assignment values with non-zero $x_{i,j}$ will be processed from the largest to the smallest for the leaf nodes of the tree. In this example, all the assignment values happen to be the same, namely 0.5. Such leaves may, therefore, be processed in any order. Although different heuristics may also be developed for breaking ties such as considering the depths of nodes or favoring nodes with higher $w_{i,j}$ values, we assume for the sake of this example that the assignments with the same value are processed in increasing order of node identifiers and then in increasing order of task numbers. As a result, first, task $t_1$ is assigned to node 3 with weight 8. Then, task $t_2$ gets assigned to node 5 with weight 4. These assignments in the first iteration are shown in green as depicted in Fig. 6. At this point, there is obviously no leaf node left with a non-zero assignment that can be used to make any further assignments, leaving task $t_3$ hence unassigned. It should noted that while these assignments are made, all the nodes violating the hierarchy constraint are also removed from consideration. This is evidently reflected by leaving only the nodes 4 and 6 in $nodesLeft$.
- Once it is realized that no more assignments are possible, the remaining leaves, namely 4 and 6, are deleted at line 16 from the tree $T'$ leaving only the nodes 1
and 2 in it. As there are no nodes in the tree that are also in nodesLeft, a jump to line 3 initiates the second iteration of the algorithm. 

With tasksLeft = \{t_3\} and nodesLeft = \{4, 6\} at the start of the second iteration, the only remaining task is t_3, and the remaining nodes that are eligible for assignments are 4 and 6. Now a call is made to LP formulated with the assignments made in the first iteration in mind. This formulation corresponds exactly to an MWTM instance where the tree denoted by T’ is obtained at line 4 by pruning nodes 3 and 5 from the original tree denoted by T, and the set of eligible nodes and target tasks to be matched are as dictated by the values of nodesLeft and tasksLeft at the moment. The algorithm, hence, terminates by assigning the only remaining task t_3 to either node 4 or node 6 with weight 4. Fig. 6 shows this assignment in the second iteration in orange. The total weight achieved by BOA is hence 8 + 4 + 4 = 16, which is slightly less than the optimal ILP solution. □

5 Experiments

In order to measure the performance of the BOA heuristic in Algorithm 1, several experiments were performed for varying problem parameters.

All the parameters employed are presented in Table 1, and their explanations are as follows:

1. #Nodes it represents the number of nodes in the tree in a given MWTM instance. In order to generate a variety of tree sizes, the following values are employed in the experiments: 16 (small tree), 32, 64, and 128 (large tree).

Table 1 The MWTM parameters used in the experiments

| Parameter         | Values                      |
|-------------------|-----------------------------|
| #Nodes            | 16, 32, 64, 128             |
| Branching Factor  | 1.5, 2.0, 2.5               |
| #Tasks/#Nodes     | 0.125, 0.25, 0.5            |
| Weight Distribution| Increasing, decreasing, random |
2. Branching Factor this parameter is defined to be the average branching factor of a node in the tree in a given instance of MWTM. It is tuned throughout the experiments to control the type of the trees generated in a scale ranging from deep to shallow for fixed values of the #Nodes parameter. The values used in the experiments are 1.5 (deep tree), 2.0, and 2.5 (shallow tree). For a given value of #nodes, a tree with a particular branching factor $bf \in \{1.5, 2.0, 2.5\}$ is obtained by generating its nodes in level-order. Starting from the root, each node is allocated as many children as specified by a random integer in the interval $[bf - \epsilon, bf + \epsilon]$, where $\epsilon = 1.25$ has been adopted for randomization in the experiments. The process of creating new child nodes terminates as soon as the number of nodes in the tree reaches a value equal to #Nodes.

3. #Tasks/#Nodes this is defined to be the ratio of the number of the tasks to the number of the nodes in the tree associated with a given MWTM instance. This parameter is used to generate a range of MWTM instances ranging from those with a very few tasks, called sparse to those with a large number of tasks, called dense in proportion to the tree size. The values used are 0.125 (sparse), 0.25, and 0.5 (dense). As this ratio increases, the flexibility to use non-leaf nodes for assignments decreases.

4. Weight Distribution the weight $w_{i,j}$ of assigning a node $i$ to a task $j$ has a value chosen from the range $[1, \frac{#Nodes}{2}]$. The following three weight distributions are used: (i) the weights are increasing from the root to the leaves, (ii) the weights are decreasing from the root to the leaves, and (iii) the weights are assigned randomly without regard to the respective depths of the nodes. In order to generate trees with a distribution of weights increasing from the root to the leaves, the weights associated with a node are scaled accordingly by a factor directly proportional to the depth of that node. Therefore, $w_{i,j}$ is set to a random integer in the interval $[(d - 1) \times \frac{#Nodes}{2} + 1, d \times \frac{#Nodes}{2}]$ where $d > 0$ is the depth of node $i$. It should be noted that the weights regarding the root are actually not needed, as such cases would correspond to trivial instances of MWTM with only one task. Finally, in order to ensure that the weights are decreasing, a random integer computed exactly as in the former case is simply subtracted from a large constant, selected as 900 throughout the experiments.

Combining the values of these four parameters gives rise to $4 \times 3 \times 3 \times 3 = 108$ different test cases. For each test case, 20 instances of the problem are then randomly generated, and their averages are taken in the experiments. All the data set generated has been made available at the URL¹ provided. We record the total number of LP calls made at line 3 in BOA for every instance. Corresponding to each instance, both the execution time in milliseconds, and the solution obtained are also recorded once for the corresponding ILP formulation, which gives the optimal solution, and once for BOA expected to return a suboptimal solution.

The experiments aim at assessing the effectiveness of BOA against the ILP solvers. This requires that the execution times to get possibly sub-optimal solutions by BOA are compared to the times to find the optimal by ILP solvers. Since BOA calls also the same solver one or more times for the LP relaxation of the same MTWM instance,

¹ https://dl.dropboxusercontent.com/u/23067097/models.rar.
the choice of the LP/ILP solver or the computational environment will not have a significant impact on the relative performance. The observed execution times might however be scaled to be smaller or larger. The quality of the solutions obtained by BOA will not be affected by these considerations either, and the ILP solver will always produce optimal solutions. Yet, determining the optimal solutions with ILP could be infeasible due to the very long execution times in some instances. All the tests performed were run on a machine with a 4 GB of RAM and an Intel Core 2 Duo T9550 2.66 Ghz mobile processor. Microsoft Solver Foundation 3.0 was employed as the LP/ILP solver library, and the code was developed in C#5.0.

In this section, the results of the experiments are presented through a series of seven tables with a common structure. As the topmost two rows are used to set the values for the parameters `Branching Factor` and `#Tasks/#Nodes`, the leftmost two columns display the values for the parameters `Weight Distribution` and `#Nodes`. The final six tables, i.e., all except the first, can be divided into three categories, each with two tables. While the first table in each category presents a comparison between the execution times of ILP and BOA, the second evaluates the quality of the solutions by BOA against the optimal. These three groups correspond to the three distinct values that the `Branching Factor` parameter can take on, namely 2.5, 2.0, 1.5, and are presented in this order. Of the four parameters only one, namely the `Branching Factor`, is fixed, and the average results are given for all combinations of the other three parameters in these groups of tables. Finally, an additional row labeled `Method` is inserted as the third from the top to allow us to specify either ILP or BOA in these tables. It should be noted that the cells at the same position in both tables in the same group correspond to the exact same combination of parameter values.

The colors yellow and green are used consistently to highlight the cells containing `NaN` and `∞` respectively in all the tables. The cells in yellow marked with `NaN` in a table mean that no feasible solution exists. For some combinations of parameters, no feasible solution was possible. Especially when the instances become dense, and the associated trees become deep, as would be expected, it becomes more difficult to find a feasible solution satisfying the hierarchy constraint. Such configurations are characterized with high `#Tasks/#Nodes` values, and with the low values of the `Branching Factor` parameter. The results in the tables following confirm this expectation. All such cases leading to infeasibility are shown in yellow. Moreover, when the weight distribution is such that it is decreasing from the root to the leaves, finding an optimal solution becomes even more difficult using ILP. Under these circumstances, the execution time for ILP grows very quickly after the number of nodes become larger than 16. We do not include these extremely large execution times in the tables, and indeed we have canceled those resolution processes without finding the optimal values. All such cells displayed in green are marked with an `∞` symbol. The existence of feasible solutions by BOA in the corresponding cells, on the other hand, is an evidence for the existence of the optimal solutions for those cases as well. In order to verify, therefore, the quality of a solution by BOA in these situations, we make use of the corresponding possibly fractional LP relaxation solution as a potential upper bound. A quick inspection of the relevant cells reveals that the difference is very small even in these cases which definitely guarantees an even smaller distance to the actual optimal. It is hence suspected that BOA might even have achieved it.
Table 1

| Weight Distribution | Branching Factor | 1.5 | 2 | 2.5 |
|---------------------|------------------|-----|---|-----|
|                     | #Tasks/#Nodes    | 1/8 | 1/4 | 1/2 | 1/8 | 1/4 | 1/2 | 1/8 | 1/4 | 1/2 |
| 16                  | 1                | 1   | 1  | 1   | 1   | 1   | 1   | 1   | 1   | 1   |
| 32                  | 1                | 1   | 1  | 1   | 1   | 1   | 1   | 1   | 1   | 1   |
| 64                  | 1                | 1   | NaN| 1   | 1   | 1   | 1   | 1   | 1   | 1   |
| 128                 | 1                | NaN| 1  | 1   | 1   | 1   | 1   | 1   | 1   | 1   |

- **Fig. 7** The average number of times LP is called at line 3 in BOA in Algorithm 1 for all test cases. The cells in yellow are marked with the symbol NaN to mean that there exists no feasible solution (Color figure online)

| Weight Distribution | Branching Factor | 2.5 |
|---------------------|------------------|-----|
|                     | #Tasks/#Nodes    | 1/8 | 1/4 | 1/2 |
| 16                  | 5.46875          | 0.78125 | 7.03125 | 0.78125 | 5.46875 | 2.34375 |
| 32                  | 8.59375          | 5.46875 | 12.5 | 7.03125 | 22.65625 | 17.96875 |
| 64                  | 27.34375         | 18.75 | 49.21875 | 50.78125 | 126.5625 | 111.71875 |
| 128                 | 135.9375         | 118.70118 | 275.78125 | 246.875 | 695.3125 | 624.4629 |

- **Fig. 8** The execution times when the average branching factor parameter is set to 2.5 corresponding to shallow trees. The symbol ∞ in a green cell indicates a very large value (Color figure online)

The table in Fig. 7 displays the average number of LP invocations performed at line 3 in BOA in Algorithm 1 for each of 108 different test cases. As the table clearly reflects, the number of times the call to the corresponding LP relaxation is executed is very close to 1. The cells marked with NaN all correspond to the test cases for which no feasible solutions exist, as explained above.

The two tables in Figs. 8 and 9 display the execution times, and the solutions respectively when the parameter representing the average branching factor of a node in the tree is set to 2.5, which corresponds to shallow trees. There are only 3 out of 36 test cases where BOA is slightly slower in Fig. 8. These correspond to the test cases where: (i) \#Tasks / \#Nodes = 1/8, Weight Distribution = random, \#Nodes = 128, (ii)
Fig. 9 The solutions obtained when the average branching factor parameter is set to 2.5. The values in the green cells are the estimated upper bounds obtained by the corresponding possibly fractional LP relaxation solutions (Color figure online)

#Tasks/#Nodes = 1/4, Weight Distribution = increasing, #Nodes = 64, and (iii) #Tasks/#Nodes = 1/4, Weight Distribution = random, #Nodes = 64. BOA, on the other hand, achieves optimal or almost optimal solutions for these test cases as seen in Fig. 9. Also an examination of the cells corresponding to these test cases in the table in Fig. 7 reveals that they all have the value one.

These execution time anomalies were observed to occur when BOA finds an almost optimal solution in only one iteration, and can therefore be explained by the overhead introduced by BOA. When BOA obtains an almost optimal solution with a single LP call, it would be natural to expect ILP itself to discover the optimal integer assignments equally quickly. As BOA has some additional computations, its running time for such cases would be slightly longer than that of ILP.

Even when it takes intolerably long to compute the optimal by ILP, the values in the corresponding green cells in Fig. 8 are all available for BOA as an indication of its running time performance. In terms of solution quality, BOA always achieves optimal solutions when the Weight Distribution is such that it is increasing from the root to the leaves. Otherwise, the solutions obtained as shown in Fig. 9 are so close to the corresponding optimal values that it is easily seen to perform within 1% of even the upper bounds obtained via the corresponding LP relaxation solution.

The tables in Figs. 10 and 11 display the execution times, and the solutions respectively when the Branching Factor parameter is set to 2.0. This time, there are 4 out of 36 test cases in which BOA turns out to be slower than the ILP solver library, and these correspond to the cells in Fig. 10 characterized by: (i) #Tasks/#Nodes = 1/8, Weight Distribution = random, #Nodes = 64, (ii) #Tasks/#Nodes = 1/8, Weight Distribution = random, #Nodes = 128, (iii) #Tasks/#Nodes = 1/4, Weight Distribution = random, #Nodes = 64, and (iv) #Tasks/#Nodes = 1/2, Weight Distribution = increasing, #Nodes = 16. An inspection of the respective cells corresponding to these test cases in both Figs. 7 and 11 confirms once more that BOA finds solutions with optimal or almost optimal values in exactly one
| Weight Distribution | Branching Factor | Method | 1/8 | 1/4 | 1/2 |
|---------------------|-----------------|--------|-----|-----|-----|
|                     | #Tasks/#Nodes    | ILP    | BOA | ILP | BOA |
| Weights are increasing from root to leaves | 16 | 4.6875 | 1.5625 | 6.25 | 2.34375 | 4.3402778 | 5.2083333 |
|                     | 32 | 8.59375 | 3.90625 | 10.15625 | 8.59375 | 20.833333 | 15.625 |
|                     | 64 | 25 | 18.75 | 53.90625 | 42.96875 | 136.36364 | 113.63636 |
|                     | 128 | 144.38477 | 114.0625 | 250.625 | 255.46875 | 652.64423 | 621.39423 |
| Weights are decreasing from root to leaves | 16 | 8.59375 | 0.78125 | 16.40625 | 3.90625 | 126.1161 | 5.5803571 |
|                     | 32 | 25.78125 | 8.59375 | 66.4063 | 17.96875 | 1184.659 | 24.147727 |
|                     | 64 | ∞ | 70.3125 | ∞ | 186.71875 | ∞ | 294.03409 |
|                     | 128 | ∞ | 675 | ∞ | 1981.25 | ∞ | 4321.0227 |
| Weights are random from root to leaves | 16 | 10.50115 | 2.85 | 12.17652 | 4.22551 | 19.302387 | 7.3009267 |
|                     | 32 | 18.877 | 8.601 | 41.45527 | 21.577755 | 32.276818 | 26.730682 |
|                     | 64 | 58.682465 | 66.808455 | 142.84314 | 155.81968 | 1610.3582 | 540.5999 |
|                     | 128 | 272.5094 | 399.55046 | 2003.8215 | 1224.0517 | 24866.999 | 8480.4036 |

**Fig. 10** The execution times when the average branching factor parameter is set to 2.0. The symbol ∞ in a green cell indicates a very large value (Color figure online)

| Weight Distribution | Branching Factor | Method | 1/8 | 1/4 | 1/2 |
|---------------------|-----------------|--------|-----|-----|-----|
|                     | #Tasks/#Nodes    | ILP    | BOA | ILP | BOA |
| Weights are increasing from root to leaves | 16 | 55.15 | 55.15 | 119.15 | 119.15 | 209.83333 | 209.83333 |
|                     | 32 | 292.8 | 292.8 | 575.65 | 575.65 | 1053.75 | 1053.75 |
|                     | 64 | 1439.5 | 1439.5 | 2796.5 | 2796.5 | 5004.1818 | 5004.1818 |
|                     | 128 | 6768.4 | 6768.4 | 13728 | 13728 | 23750.462 | 23750.462 |
| Weights are decreasing from root to leaves | 16 | 1786.65 | 1785.6 | 3554.5 | 3551.9 | 7082.7857 | 7079.0714 |
|                     | 32 | 3510.05 | 3504.85 | 6895.35 | 6885.9 | 13679.364 | 13678.909 |
|                     | 64 | 6594.1538 | 6572.65 | 12796.208 | 12748.6 | 24993.333 | 24975.909 |
|                     | 128 | 11110.642 | 11051.7 | 20462.263 | 20355.1 | 36935.409 | 36903.909 |
| Weights are random from root to leaves | 16 | 15.55 | 15.55 | 31.35 | 31.3 | 59.533333 | 59.533333 |
|                     | 32 | 62.85 | 62.85 | 125.125 | 125.05 | 246.09091 | 246.09091 |
|                     | 64 | 254 | 253.9 | 506.05 | 505.75 | 998.0625 | 997.125 |
|                     | 128 | 1020.2 | 1020.15 | 2037.3 | 2036.85 | 4047.9167 | 4036.5 |

**Fig. 11** The solutions obtained when the average branching factor parameter is set to 2.0. The values in the green cells are the estimated upper bounds obtained by the corresponding possibly fractional LP relaxation solutions (Color figure online)

iteration, making a single LP call. This confirms the validity of the previous analysis, stating that ILP performs very fast for the instances whose LP formulations also return integer assignments.

BOA always achieves optimal or very close to optimal solutions as shown in Fig. 11. For example, when #Tasks/#Nodes = 1/2 for a 128-node tree, and the weights are randomly distributed among all nodes, the ILP produces the optimal goal value as 4047.9167 and BOA heuristic generates 4036.5. This is a case with one of the largest difference between the optimal solution and our heuristic solution. Even in this case, the difference between the two solutions is much less than 1%. For some cases where, instead of ILP, LP relaxation solutions were used as upper bounds, the differences are
Fig. 12 The execution times when the average branching factor parameter is set to 1.5 corresponding to deep trees. While the cells in yellow marked with the symbol NaN represent the parameter combinations for which there are no feasible solutions, the symbol ∞ in a green cell indicates a very large value (Color figure online)

| Weight Distribution | Branching Factor | #Tasks/#Nodes | 1/8 | 1/4 | 1/2 |
|---------------------|------------------|---------------|-----|-----|-----|
|                     | Method           | ILP           | BOA | ILP | BOA | ILP | BOA |
|                     |                  |               |     |     |     |     |     |
| Weights are increasing from root to leaves |                  |               |     |     |     |     |     |
|                     | 16               | 3.90625       | 0.78125 | 3.125 | 4.6875 | 21.484375 | 5.859375 |
|                     | 32               | 4.6875        | 3.90625 | 11.71875 | 8.59375 | 46.875 | 15.625 |
|                     | 64               | 24.21875      | 16.40625 | 43.75 | 45.3125 | NaN | NaN |
|                     | 128              | 150.78125     | 120.3125 | 298.4375 | 245.3125 | NaN | NaN |
| Weights are decreasing from root to leaves |                  |               |     |     |     |     |     |
|                     | 16               | 9.375         | 3.125 | 14.80263 | 5.7565789 | 9.375 | 3.125 |
|                     | 32               | 46.835        | 7.03125 | 179.6875 | 17.96875 | NaN | NaN |
|                     | 64               | 87.93         | 89.761395 | 619.223 | 152.217 | NaN | NaN |
|                     | 128              | 467.584       | 527.416 | 2981.5173 | 1508.0479 | NaN | NaN |

Fig. 13 The solutions obtained when the average branching factor parameter is set to 1.5. While the cells in yellow marked with the symbol NaN represent the parameter combinations for which there are no feasible solutions, the values in the green cells are the estimated upper bounds obtained by the corresponding possibly fractional LP relaxation solutions (Color figure online)

| Weight Distribution | Branching Factor | #Tasks/#Nodes | 1/8 | 1/4 | 1/2 |
|---------------------|------------------|---------------|-----|-----|-----|
|                     | Method           | ILP           | BOA | ILP | BOA | ILP | BOA |
|                     |                  |               |     |     |     |     |     |
| Weights are increasing from root to leaves |                  |               |     |     |     |     |     |
|                     | 16               | 71.5          | 71.5 | 154.2 | 154.2 | 212.875 | 212.875 |
|                     | 32               | 415.3         | 415.3 | 698.95 | 698.95 | 510 | 510 |
|                     | 64               | 2035.35       | 2035.35 | 3861.45 | 3861.45 | NaN | NaN |
|                     | 128              | 9762.8        | 9762.8 | 19354.3 | 19354.3 | NaN | NaN |
| Weights are decreasing from root to leaves |                  |               |     |     |     |     |     |
|                     | 16               | 1786.3        | 1784.4 | 3545.4211 | 3542.6316 | 7071.8 | 7071.8 |
|                     | 32               | 3444.8        | 3431.6 | 6847.4 | 6833.4 | NaN | NaN |
|                     | 64               | 6297.7083     | 6273.2 | 11957.885 | 11934.85 | 25421.6 | 25403 |
|                     | 128              | 9810.1167     | 9734.15 | 15903.15 | 15859.6 | NaN | NaN |
| Weights are random from root to leaves |                  |               |     |     |     |     |     |
|                     | 16               | 15.55         | 15.55 | 30.3 | 30.2 | 58.5 | 58.5 |
|                     | 32               | 63.25         | 63.2 | 125.1 | 124.6 | 242 | 242 |
|                     | 64               | 253.7         | 253.65 | 503.45 | 502.95 | NaN | NaN |
|                     | 128              | 1019.6        | 1019.35 | 2032.4 | 2030.85 | NaN | NaN |

slightly higher. For example, when #Tasks/ #Nodes = 1/4 for a 128-node tree, and the weights are decreasing from the root to the leaves, the upper bound to the optimal is 20462.263, and BOA achieves 20355.1. Even for this upper bound, the difference is slight. Potentially, BOA might even have the same solution as the actual optimal, or would at least have achieved a closer value to the actual optimal.

Figures 12 and 13 display the execution times, and the solutions respectively when the parameter representing the average branching factor is set to 1.5 which corresponds to deep trees. In 4 out of the 36 test cases presented in Fig. 12, BOA takes
longer to reach a solution. The first two of these correspond to the cases where the parameter \#Nodes is set to either 64 or 128 when \#Tasks/\#Nodes = 1/8 and Weight Distribution is random. The cells corresponding to these two test cases in Fig. 7 both have the value 1.1. Furthermore, it is seen from the corresponding cells in Fig. 13 that BOA finds solutions very close to optimal. The last two test cases correspond, however, to the combinations of parameters when \#Nodes is set to either 16 or 64 when \#Tasks/\#Nodes = 1/4 and Weight Distribution is such that it is increasing from the root to the leaves. A quick inspection of the corresponding cells for the last two test cases in the corresponding tables reveals that BOA found the optimal solutions after a single LP invocation, confirming the validity of the prior justification.

The results of the experiments show that for all cases BOA generates goal values very close to the optimal obtained by ILP. The results are either identical or very slightly different. In addition, in the latter case, the distance to the optimal is always much less than 1%. Moreover, with Weight Distribution increasing from the root to the leaves, BOA always finds optimal solutions.

When the parameter Weight Distribution is such that it is decreasing from the root to the leaves, it takes forever to compute the optimal by ILP, as shown by the corresponding cells marked \(\infty\) throughout the tables. Under the same setting, on the other hand, BOA returns in polynomial time almost optimal solutions that are within 1\% of even the upper bounds obtained via the corresponding LP relaxation solution. ILP runs faster than BOA in only 11 out of a total of 108 different test cases. All 11 of these execution time anomalies are seen to occur when BOA discovers an almost optimal solution after at most 1 or 1.1 LP calls on average. These test cases are therefore thought likely to correspond to the instances that can be solved efficiently by ILP. In such a case, ILP can essentially find a solution by making only a very few LP relaxation calls via a branch and bound algorithm. It is then easily anticipated that the additional overhead posed by BOA makes it less efficient than ILP.

## 6 Conclusion

In this paper we introduced a new version of the assignment problem, called the \(MWTM\) problem. In \(MWTM\), as is the case with the standard assignment problem, a one-to-one assignment is sought between a set of tasks and a set of agents (nodes) to maximize the total profit (weight) value. Moreover, there is an additional constraint in \(MWTM\) preventing some combinations of the assignments. Since agents are organized in a tree structure representing hierarchical (agent–sub-agent) relationships, when an agent is assigned to a task, none of its sub-agents or super-agents can be assigned to any other task. This problem is shown to be NP-hard. Therefore, we proposed an iterative LP-relaxation based solution. Through experiments we have shown that our heuristic solution, BOA, is very effective, and produces a solution that is optimal, or very close to optimal in a very reasonable time, performing only a small number of iterations. In most cases, the solution is achieved within a single iteration.

It is left as an open question to characterize those instances of \(MWTM\) that lend themselves to an optimal solution in only a single iteration so that they can be identified from the parameters of the problem. It would also be interesting as a non-trivial open
problem to explore the existence of approximation algorithms with guaranteed bounds for MWTM.

Acknowledgements The authors are thankful to the editor and the anonymous referees for their insightful comments that contributed to improving the quality of the paper.

References

Armstrong RD, Jin Z (1995) On solving a variation of the assignment problem. Eur J Oper Res 87(1):142–147
Burkard RE, Dell’Amico M, Martello S (2009) Assignment problems. SIAM, Philadelphia
Cohen R, Katzir L, Raz D (2006) An efficient approximation for the generalized assignment problem. Inf Process Lett 100(4):162–166
Contreras I, Díaz JA, Fernández E (2011) Branch and price for large-scale capacitated hub location problems with single assignment. INFORMS J Comput 23(1):41–55
de Klerk E, Sotirov R, Truetsch U (2015) A new semidefinite programming relaxation for the quadratic assignment problem and its computational perspectives. INFORMS J Comput 27(2):378–391
Fleischer L, Goemans MX, Mirrokni VS, Sviridenko M (2006) Tight approximation algorithms for maximum general assignment problems. In: SODA’06, ACM, New York, pp 611–620
Geetha S, Nair KPK (1993) A variation of the assignment problem. Eur J Oper Res 68(3):422–426
Gulek M, Toroslu IH (2010a) A dynamic programming algorithm for tree-like weighted set packing problem. Inf Sci 180(20):3974–3979
Gulek M, Toroslu IH (2010b) A genetic algorithm for maximum-weighted tree matching problem. Appl Soft Comput 10(4):1127–1131
Jain, K.: (2000) Enhancing techniques in LP based approximation algorithms. PhD thesis, Georgia Institute of Technology
Jain K (2001) A factor 2 approximation algorithm for the generalized steiner network problem. Combinatorica 21(1):39–60
Karmarkar N (1984) A new polynomial-time algorithm for linear programming. Combinatorica 4(4):373–395
Karp RM (1972) Reducibility among combinatorial problems. In: Miller RE, Thatcher JW (eds) Complexity of computer computations,. The IBM research symposia seriesPlenum Press, New York, pp 85–103
Khachiyan L (1979) A polynomial algorithm in linear programming. Sov Math Dokl 20(1):191–194
Li HL, Huang YH, Fang SC (2013) A logarithmic method for reducing binary variables and inequality constraints in solving task assignment problems. INFORMS J Comput 25(4):643–653
Pardalos PM, Pitsoulis L (2000) Nonlinear assignment problems: algorithms and applications. Combinatorial Optimization. Springer, New York
Rainwater C, Geunes J, Romeijn HE (2014) Resource-constrained assignment problems with shared resource consumption and flexible demand. INFORMS J Comput 26(2):290–302
Shima T, Rasmussen SJ, Sparks AG, Passino KM (2006) Multiple task assignments for cooperating uninhabited aerial vehicles using genetic algorithms. Comput Oper Res 33(11):3252–3269
Vazirani VV (2002) Approximation algorithms. Springer, Berlin
Walteros JL, Vogiatzis C, Pasiliao EL, Pardalos PM (2014) Integer programming models for the multidimensional assignment problem with star costs. Eur J of Oper Res 235(3):553–568
Wang F, Zhou S, Shi N (2013) Group-to-group reviewer assignment problem. Comput Oper Res 40(5):1351–1362