List of Selected Number of Optimal Solutions of the Assignment Problem by Time Criterion

Lasko M. Laskov
Department of Informatics
New Bulgarian University
Sofia, Bulgaria
llaskov@nbu.bg

Marin L. Marinov
Department of Informatics
New Bulgarian University
Sofia, Bulgaria
mlmarinov@nbu.bg

***Abstract*—In this paper we present a solution of the *assignment problem* with an algorithm with complexity $O(n^{\frac{9}{2}})$. The discussed algorithm allows an effective approach for generation of a list of selected number of optimal solutions of this problem. If it is predefined that the list does not contain more than n_0 number of optimal solutions, then the proposed algorithm has complexity $\widetilde{n}O(n^{\frac{9}{2}})$, where $\widetilde{n} = \min\{n_0, n_1\}$ with n_1 being the number of perfect matchings in the graph. The method is based on the Hopcroft-Karp algorithm for maximum matching in a bipartite graphs [16].**

***Index Terms*—combinatorial optimization algorithms, assignment problem, time criterion, matchings in graph**

I. Introduction

The assignment problem (AP) is a fundamental research topic in the field of combinatorial optimization [2]. It deals with the problem how to assign n number of tasks (problems, projects, computational problems) to n number of agents (workers, companies, machines). The AP can be found in various scientific fields, including economics, machine learning [17], and distributed computer systems [20], [19].

While the naive solution of the AP leads to the unfeasible factorial complexity, it has been proved that the problem is NP-hard, and the first known work to propose a polynomial solution [11] introduces the well-known *Hungarian algorithm*. Since then many improvements and versions [12], [19] and applications (see for example [7]) of the algorithm has been proposed. Some of these works even lead to discovery of important data structures, for example the work [5] introduces the Fibonacci heap.

An important category of methods is based on the graph theory [10]. They represent the AP in the terms of weighted bipartite graphs, and look for solutions by searching for matchings (see [16]). More recent works investigate the AP in the terms of network flows [1], by even solving the unbalanced version of the problem in which the number of agents and tasks are not equal.

Another type of AP occur when algorithms also aim to minimize the latest competition time [6]. In literature it is sometimes referred to as *linear bottleneck assignment problem* [3], [18], and the proposed solutions again are based on weighted bipartite graphs.

978-1-6654-7625-6/22/$31.00 ©2022 IEEE

In this paper we will present two versions of the assignment problem minimizing the time criterion. In Sect. II we will formulate the time criterion assignment problem (TCAP), and we will show an algorithm that finds one optimal solution. In Sec. III we will show how a list of optimal solutions of the TCAP can be generated.

II. Time criterion assignment problem

In this section we will present the time criterion assignment problem (TCAP). First we will give definition of the problem, and then we will discuss the calculation of one optimal solution.

A. Problem statement

A given project is being defined as the execution of n independent tasks. For the execution of the tasks, n agents apply. The agent i can execute the task j for time t_{ij}, where $i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, n\}$. We assume that the *time matrix* $T(t_{ij})$ is preliminary defined. Every task must be executed by a single agent, and every agent can execute a single task. We will represent the admissible plans for assignment of the tasks to different agents with the permutations of the first n natural numbers.

The permutation

$$w = \begin{pmatrix} 1 & 2 & \cdots & n \\ j_1 & j_2 & \cdots & j_n \end{pmatrix}$$

will be denoted with the shorter $w = (j_1, j_2, \ldots, j_n)$. Also, we denote $w(i) = j_i$, $\forall i \in \{1, 2, \ldots, n\}$.

The choice of the plan w shows that the agent i has been assigned the task $w(i)$. The time for the execution of the plan w is:

$$G_T(w) = \max\{T(1, w(1)), \ldots, T(n, w(n))\},$$

where $T(i, j)$ is the element of the matrix T, located on the i-th row and the j-th column.

Definition 1 (Optimal plan): The plan $\widetilde{w}$ is called an *optimal plan*, when for each plan w holds the inequality $G_T(\widetilde{w}) \leq G_T(w)$.

Then the TCAP can be defined:

Problem 1 (TCAP): For an arbitrary time matrix T, find an optimal solution w.

A special feature of the Problem 1 is that the function $G_T(w)$ is not linear.

It is obvious that in the case in which n is a relatively small number, the problem can be solved using a *brute-force* method. In this case all possible plans w are examined, resulting in $n!$ number of comparisons, and making the approach unfeasible for a standard computer system in the cases in which $n > 13$.

A *branch-and-bound* method (see [14], [9] and more recent [13], [15], [4]) is another possible approach. Since an algorithm that incorporates the branch-and-bound technique will have an exponential complexity $O(2^n)$, it is a possible solution in the cases is which $n \leq 25$.

In this work we use the algorithm of *Hopcroft and Krap* [16] for maximum matchings in bipartite graphs, which allows us to present a more effective solution of the TCAP.

For our purposes it will be more convenient to formulate the Problem 1 with a full weighted bipartite graph $G(X, Y, E, t)$, where $X = \{1, 2, \ldots, n\}$, $Y = \{1, 2, \ldots, n\}$, $E = \{\{i, j\} : i \in X, j \in Y\}$, and the function t that maps to each edge $\{i, j\}$ the number t_{ij}, i.e. $t(\{i, j\}) = T(i, j)$. Then each plan can be represented as a perfect matching and vice versa.

We will denote a perfect matching $\{\{1, j_1\}, \{2, j_2\}, \ldots, \{n, j_n\}\}$ using the permutation $w = (j_1, j_2, \ldots, j_n)$. Then we will call a perfect matching $\widetilde{w}$ *an optimal perfect matching*, when for each perfect matching it holds $G_T(\widetilde{w}) \leq G_T(w)$. Now the TCAP can be defined in the terms of perfect matching in $G(X, Y, E, t)$:

Problem 2 (TCAP): For an arbitrary full weighted bipartite graph $G(X, Y, E, t)$, find an optimal perfect matching.

When solving the Problem 2, we will assume that the graph $G(X, Y, E, t)$ is defined with the time matrix $T(t_{ij})$.

We will note that our goal is the problem for the composition of a list of selected number of optimal solutions (Sec. III). However, for its representation we need the solution of Problem 2.

B. Find one optimal solution

For the purposes of the solution of the Problem 2 we will use a special finite unweighted bipartite graphs sequence (UBGS) $G_k(X, Y, E_k)$, where $X = \{1, 2, \ldots, n\}$, $Y = \{1, 2, \ldots, n\}$, $E_0 = \varnothing$ and $E_k \subset E_{k+1}$. The sets E_k are defined by induction using the following procedure, which we will call under the sequence.

Procedure 1 (UBGS): Using mathematical induction we define the following procedure for generation of UBGS. The input of the procedure is the weighted bipartite graph $G(X, Y, E, t)$ with an arbitrary adjacency matrix of weights $T(t_{ij})$ of order n. Initially, we will denote with V all weights of the edges contained in the matrix T, in other words $V = \{t_{ij} : i \in X, j \in Y\}$. We shall say that an edge $\{i, j\}$ is *blocked* when $t_{ij} \in V$. When an edge $t_{ij} \notin V$, we shall call $\{i, j\}$ a *free* edge. On the k-the step of the procedure all free edges are stored in E_k and they are the edges of the unweighted graph G_k.

a) Step 0: All edges are blocked, and $E_0 = \varnothing$, because there are no free edges. The adjacency matrix of the unweighed bipartite graph $G_0(X, Y, E_0)$ is the zero matrix of order n.

b) Step 1: We calculate the minimal weight in V, $m_1 = \min\{V\}$, and define as blocked edges all edges $\{i, j\}$ for which $t_{ij} > m_1$. All the other edges are defined as free edges and are included in the adjacency matrix of the unweighted bipartite graph $G_1(X, Y, E_1)$, A_1 such that:

$$a_{ij}^{(1)} = \begin{cases} 0, & \text{if} \quad t_{ij} > m_1 \\ 1, & \text{otherwise.} \end{cases}$$

We update the set V by removing all edges $\{i, j\} \in E_1$. Then $V = \{t_{ij} : t_{ij} > m_1\}$.

c) Step 2: We calculate the minimal weight in the updated V, $m_2 = \min\{V\}$, and then define all free edges $\{i, j\}$, for which $t_{ij} \leq m_2$, and denote the set of the free edges with E_2. The adjacency matrix of the unweighted bipartite graph $G_2(X, Y, E_2)$ is A_2, and it has elements:

$$a_{ij}^{(2)} = \begin{cases} 0, & \text{if} \quad t_{ij} > m_2 \\ 1, & \text{otherwise.} \end{cases}$$

Again, we update the set V by removing all free edges from it, and we get $V = \{t_{ij} : t_{ij} > m_2\}$.

d) Induction step: After the step k of the inductive procedure, let us define the number m_k, the unweighted bipartite graph $G_k(X, Y, E_k)$, the adjacency matrix A_k, and the set V of those elements t_{ij} for which $t_{ij} > m_k$. The following two cases are possible:

1) $V = \varnothing$. In this case all edges are free and the adjacency matrix A_k has elements $a_{ij}^{(k)} = 1$ for all i and j. We set $k_0 = k$ and the procedure stops.
2) $V \neq \varnothing$. In this case the inductive procedure continues. We calculate $m_{k+1} = \min\{V\}$, we define the free edges $\{i, j\}$ for which $t_{ij} \leq m_{k+1}$, and the set $E_{k+1} = \{\{i, j\} : t_{ij} \leq m_{k+1}\}$. The adjacency matrix A_{k+1} of the unweighed bipartite graph $G_{k+1}(X, Y, E_{k+1})$ has elements:

$$a_{ij}^{(k+1)} = \begin{cases} 0, & \text{if} \quad t_{ij} > m_{k+1} \\ 1, & \text{otherwise.} \end{cases}$$

The set V is updated $V = \{t_{ij} : t_{ij} > m_{k+1}\}$.

□

Proposition 1: If we apply the UBGS procedure to an arbitrary square matrix $T(t_{ij})$ of order n, the following properties hold.

1) For all numbers m_k and m_{k+1} that are calculated with UBGS, $m_k < m_{k+1}$.
2) UBGS stops after the step k_0, where $k_0 \leq n^2$.
3) $E_{k+1} = E_k \cup \{\{i, j\} : t_{ij} = m_{k+1}\}$ for each $k \in \{0, 1, \ldots, k_0 - 1\}$.
4) For each element t_{ij} of the matrix T exists an index $k \in \{1, 2, \ldots, k_0\}$ for which $m_k = t_{ij}$.

The proof of Proposition 1 follows directly from the definitions of m_k and E_k. Hence, for an arbitrary matrix $T(t_{ij})$ the UBGS Procedure 1 defines the finite sequences:

$$m_1 < m_2 < \cdots < m_{k_0} \tag{1}$$

$$A_1, A_2, \ldots, A_{k_0} \tag{2}$$

$$G_1(X, Y, E_1), G_2(X, Y, E_2), \ldots, G_{k_0}(X, Y, E_{k_0}), \tag{3}$$

where A_k has the following elements:

$$a_{ij}^k = \begin{cases} 1, & \text{if} \quad t_{ij} \leq m_k \\ 0, & \text{if} \quad t_{ij} > m_k \end{cases}$$

and is the adjacency matrix of the graph $G_k(X, Y, E_k)$.

Now we will introduce a procedure that verifies whether an unweighted bipartite graph has a perfect matching (PM).

Procedure 2 (PM): For an arbitrary square matrix Z with elements equal to 0 or 1, the procedure verifies if the unweighted bipartite graph G_Z with adjacency matrix Z has a perfect matching. The procedure calculates the pair $\{ind,\ w_0\}$, where

$$ind = \begin{cases} 0, & \text{if} \quad G_Z \quad \text{does not have a perfect matching} \\ 1, & \text{if} \quad G_Z \quad \text{has a perfect matching.} \end{cases}$$

If $ind = 1$, then w_0 is a perfect matching of G_Z.

The procedure PM is implemented using the Hopcroft-Krap algorithm [16]. In this case the complexity of PM is evaluated $O(n^{\frac{5}{2}})$. Of course, for the implementation other algorithms can be used as well: for example the algorithm of Kuhn [11], however in this case the complexity will be $O(n^3)$. The improvement of the complexity of the procedure PM, improves the complexity of the presented algorithm in which PM is used, as it is shown in Proposition 3.

We will solve the Problem 2 with the Algorithm 1. It takes as an input the adjacency matrix $T(t_{ij})$ of order n with the weights of the graph $G(X, Y, E, t)$. The output of the algorithm is one optimal perfect matching w_0. The set of blocked edges is stored in an array v that has n^2 elements, where each element is composed by the pair of weight t_{ij} and its corresponding indexes (i, j) in the matrix $T(t_{ij})$. Since v is one-dimensional, the correspondence between its index and the indexes of $T(t_{ij})$ is given by $s = (i-1)n + j$, and thus $v(s) = (v(s, 1), v(s, 2))$, where $v(s,\ 1) = t_{ij}$, $v(s, 2) = (i, j)$. We denote the zero square matrix of order n by $0_{n\times n}$, and by $v = [v]_{q_0}$ we denote the removal of the elements from v which are placed on positions $q_0(i)$ for each i.

Additionally, we will use the following variables to store the current record of the algorithm:

- r_0 stores the minimum weight in the array v;
- the list v_0 stores on which locations of T there are elements that are equal to r_0;
- the list q_0 stores on which locations of v there are elements that are equal to r_0.

Proposition 2: Algorithm 1 is correct.

Proof. It is clear that each iteration of the loop with a counter k_0 implements one step from the inductive procedure. In this manner, the loop consecutively calculates the elements of the sequences (1), (2) and (3). The loop stops when a graph G_{A_0} is found that belongs to the sequence (3) and contains a perfect matching. Apparently, if $G_{A_0} = G_{\widetilde{k}}$, the following holds.

- If $i < \widetilde{k}$, then the graph G_i does not have perfect matchings.
- If $i \geq \widetilde{k}$, then the graph G_i has perfect matchings.

Also, the following statements hold.

1) Every perfect matching of G_{A_0} is an optimal perfect matching in the graph $G(X, Y, E, t)$.
2) Every optimal perfect matching of $G(X, Y, E, t)$ is a perfect matching in the graph G_{A_0}.

Indeed, let us suppose that $\widetilde{w}$ is a perfect matching of $G_{A_0} = G_{\widetilde{k}}$. Then:

$$G_T(\widetilde{w}) = m_{\widetilde{k}}. \tag{4}$$

Now, let w be a perfect matching of the graph $G(X, Y, E, t)$ with adjacency matrix $T(t_{ij})$. With i_0 we denote the index for which

$$t_{i_0 w(i_0)} = \max\left\{t_{iw(i)} : i \in \{1, 2, \ldots, n\}\right\} = G_T(w) \tag{5}$$

Algorithm 1 Time criterion assignment problem (TCAP).

Input: $T(t_{ij})$
Output: an optimal perfect matching w_0

```
k_0 ← 1
while k_0 > 0 do
  vr ← v
  r_0 ← v(1,1)
  v_0 ← {}, q_0 ← {}
  q ← 0, k ← |vr|
  A_0 ← 0_{n×n}
  while k > 0 do
    w ← vr(1)
    vr ← vr \ {vr(1)}
    k ← k − 1, q ← q + 1
    if w(1) < r_0 then
      r_0 ← w(1), v_0 ← {w(2)}, q_0 ← {q}
    else if w(1) = r_0 then
      v_0 ← v_0 ∪ {w(2)}, q_0 ← q_0 ∪ {q}
    end if
  end while
  for k_1 ← |v_0| to 1 do
    v_1 ← v_0(1), v_0 ← v_0 \ {v_0(1)}
    A_0(v_1(1), v_1(2)) ← 1
  end for
  {ind, w} ← PM(A_0) {using Procedure 2}
  if ind = 0 then
    v ← [v]_{q_0} {remove elements on locations q_0(i)}
    k_0 ← |v|
  else
    w_0 ← w
    break{solution found, terminate the main loop}
  end if
end while
```

By its construction, the graph $G_{\widetilde{k}-1}(X,Y,E_{\widetilde{k}-1})$ has no perfect matching. Then $t_{i_0w(i_0)} > m_{\widetilde{k}-1}$, and hence,

$$t_{i_0w(i_0)} \geq \min\{V\} = m_{\widetilde{k}}, \tag{6}$$

where $V = \{t_{ij} : t_{ij} > m_{\widetilde{k}-1}\}$. From the inequalities (5), (6) and (4) it follows that

$$G_T(w) = t_{i_0w(i_0)} \geq m_{\widetilde{k}} = G_T(\widetilde{w}).$$

Let $\widehat{w}$ is a perfect optimal matching in the graph $G(X,Y,E,t)$. Then $G_T(\widehat{w}) \leq \widetilde{k}$ because there is a perfect matching w_0, for which $G_T(w_0) = \widetilde{k}$.

If we assume that $G_T(\widehat{w}) < m_{\widetilde{k}}$, then $\widehat{w}$ must be a perfect matching of $G_{\widetilde{k}-1}(X,Y,E_{\widetilde{k}-1})$. However, this is a contradiction with the definition of $\widetilde{k}$. Hence we have proven that $G_T(\widehat{w}) = m_{\widetilde{k}}$.

□

Proposition 3: The complexity of Algorithm 1 is $O(n^{\frac{9}{2}})$.

Proof. From Proposition 2 we know that the Algorithm 1 finds the solution of Problem 2 after no more than k_0 number of iterations, where $k_0 \leq n^2$. During each iteration, there is a single call to the PM procedure, and additional calculations with complexity $O(n^2)$. This shows that each iteration of the algorithm performs $O(n^{\frac{5}{2}})$ number of calculations. From here it follows that the entire complexity of the algorithm is $n^2O(n^{\frac{5}{2}}) = O(n^{\frac{9}{2}})$.

III. List of optimal solutions

In this section we will focus on the generation of a list of optimal solutions (LOS) of TCAP. For this purpose we formulate:

Problem 3 (LOS): For an arbitrary weighted bipartite graph $G(X,Y,E,t)$ and an arbitrary natural number n_0, generate a list L of optimal perfect matchings with the following properties:

1) If the TCAP Problem 2 has more than $n_0 - 1$ optimal matchings, then L contains n_0 elements.
2) If the TCAP Problem 2 has less than n_0 optimal matchings, then L contains all optimal matchings.

We will note that the purpose of this work is to find the solution of Problem 3. However, to formulate the corresponding algorithm, we use the solution of Problem 2 in combination with the solution of the helper Problem 4 (given below) for finding of a list of perfect matchings in an unweighted bipartite graph (LPMUBG).

Problem 4 (LPMUBG): Let the unweighted bipartite graph $G(X,Y,E)$ has an adjacency matrix A with elements $A(i,j)$ that are equal either to 0 or 1. For an arbitrary natural number n_0 we will generate a list L of perfect matchings in the graph G that has the following properties:

1) If G has more than $n_0 - 1$ perfect matchings, then L contains n_0 elements.
2) If G has less than n_0 perfect matchings, then L contains all perfect matchings.

For the solution of the Problem 4 we will use again the Procedure 2 (PM). Using it we will check whether the graph G has perfect matchings. If G has no perfect matchings, we will set $L = \varnothing$, and the problem is solved. That is why we will assume that G has at least one perfect matching. In this case, using PM we calculate one perfect matching v.

Let us formulate the Problem 4 using the array $X_0 = \{w, v, B, n\}$, where:

- $w = ()$ is a sequence;
- v is one perfect matching in the graph $G(X,Y,E)$;
- B is a matrix of the type $(n+1) \times n$ with elements

$$B(i,\, j) = \begin{cases} A(i,j), & \text{if } i \in X \text{ and } j \in Y \\ j, & \text{if } i = n+1 \text{ and } j \in Y. \end{cases} \tag{7}$$

We will reduce the solution of Problem 4 to the solution of a number of subtasks with reduced size, in which we will look for a single solution. After that, we will apply the procedure PM to each of theses subtasks, and the problem is solved. The set of subtasks will be generated by *branching* of the original problem.

Apart from the previously introduced notations, we will use also the following additional terms. If $X = \{1, 2, \ldots, n\} = Y$, then $[X]_1 = \{2, 3, \ldots, n\}$, $[Y]_j = \{1, 2, \ldots, (j-1), (j+1), \ldots, n\}$. For an arbitrary matrix B, we denote with

$$[B]^{i_1,i_2,\ldots,i_k}_{j_1,j_2,\ldots,j_k}$$

the sub-matrix that is received from B by removing of the rows with indexes $i_1, i_2, \ldots, i_k$ and columns with indexes $j_1, j_2, \ldots, j_k$.

It is clear that if A is a square matrix of order n, then $[A]^1_j(p,q) = A(1+p, f_j(q))$, where

$$f_j(q) = \begin{cases} q, & \text{if} \quad q < j \\ q+1, & \text{if} \quad q \geq j \end{cases} \tag{8}$$

and both p and $q \in \{1, 2, \ldots, (n-1)\}$.

The following procedure describes one branching step (BS).

Procedure 3 (BS): Suppose that the graph $G(X,Y,E)$ contains at least one perfect matching, and Problem 4 is formulated using the array $X_0 = \{w, v, B, n\}$. Then we apply branching on X_0 in the following way.

1) For each $j \in Y$ that satisfies the condition $A(1,j) = 1$, we define:
   a) $A = [B]^{n+1}$ and $A_j = [A]^1_j$;
   b) the graph $G_j(X_1, Y_1, E_j)$, where $X_1 = \{1, 2, \ldots, n-1\} = Y_1$ and $E_j = \{\{p,q\} : A_j(p,q) = 1, p \in X_1, q \in Y_1\}$.
2) For each $j \in Y$, that satisfies the conditions $A(1,j) = 1$, and the graph G_j has at least one perfect matching, we define the problem $X_j = \{w_j, v_j, B_j, n-1\}$, where $w_j = (j)$, $B_j = [B]^1_j$ and v_j is calculated in the following way:
   a) using the procedure PM for $Z = A_j$ we find perfect matching $\widetilde{w} = (\widetilde{j}_1, \widetilde{j}_2, \ldots, \widetilde{j}_{n-1})$ in the graph $G_j(X_1, Y_1, E_j)$;
   b) we define $v_j = (j, f_j(\widetilde{j}_1), f_j(\widetilde{j}_2), \ldots, f_j(\widetilde{j}_{n-1}))$, where $f_j(q)$ is defined with the equation (8).

After the branching step of X_0 is completed, we define r number of problems:

$$X_{j_s} = \{w_{j_s}, v_{j_s}, B_{j_s}, n-1\},\ s \in \{1, 2, \ldots, r\}.$$

It is clear that $0 < r \leq n$. Without loss of generality we can consider that $0 < j_1 < j_2 < \cdots < j_r$.

Algorithm 2 List of perfect matchings in an unweighted bipartite graph (LPMUBG).

Input: adjacency matrix A, and a natural number n_0
Output: list of perfect matchings L

```
L is a list, Q is a queue
{ind, w_0} ← PM(A) {using Procedure 2}
if ind = 0 then
  return ∅ {returns an empty list}
else
  X_0 = {( ), w_0, B, n} {see equation (7)}
  enqueue(Q, {X_0})
  while |Q| > 0 and |L| + |Q| < n_0 do
    {w, v, B, k} ← front(Q), dequeue(Q)
    if k > 2 then
      for j ← 1 to k do
        if B(1, j) = 1 then
          w_1 ← w_1 ∪ {B(k + 1, j)}
          B_1 ← [B]_j^1
          {a, b} ← PM([B_1]^k)
          if a = 1 then
            b(s) ← B_1(k, b(s)), ∀s ∈ {1, 2, ..., k − 1}
            v_1 ← w_1 ∪ b
            enqueue(Q, {w, v_1, B_1, k − 1})
          end if
        end if
        if |L| + |Q| ≥ n_0 then
          j ← k + 1
        end if
      end for
    else
      push(L, {v})
      if A(n − 1, v(n))A(n, v(n − 1)) = 1 then
        v_1 ← v, v_1(n − 1) ← v(n), v_1(n) ← v(n − 1)
        push(L, {v_1})
      end if
    end if
  end while
  if |Q| > 0 then
    for i ← 1 to n_0 − |L| do
      push(L, {Q(i, 2)})
    end for
  end if
end if
```

Based on the Procedure 3 (BS) we can now formulate Algorithm 2 (given below) for solving the LPMUBG problem. We denote with Q the queue of problems to be branched. With $|Q|$ we denote the number of elements in Q. During the iterations we will fill the list L, while its current number of elements is denoted with $|L|$. With *enqueue()*, *dequeue()*, *front()* and *push()* we denote the standard operations of the data structures *queue* and *list*. The general steps of the algorithm are illustrated in the flowchart in Fig. 1.

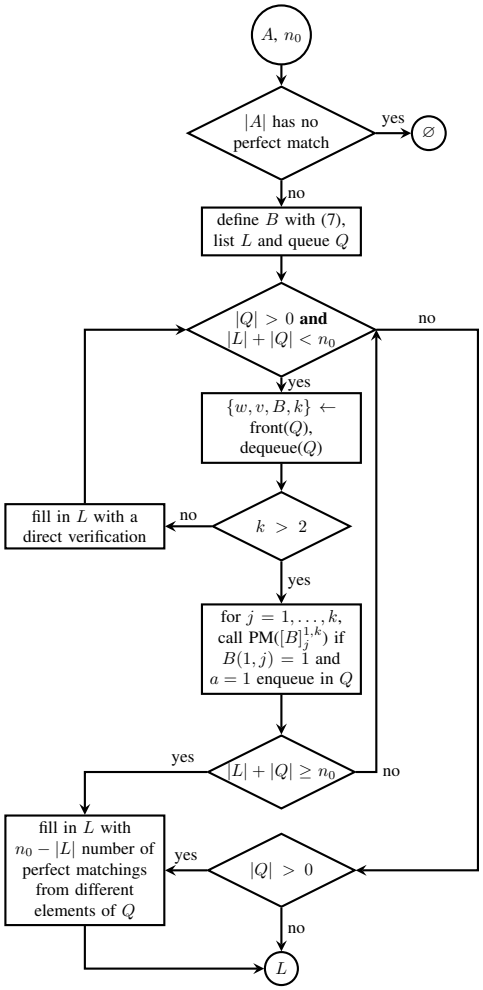

Fig. 1. Flowchart of the LPMUBG algorithm.

Theorem 1: The Algorithm 2 is correct and has complexity $\widetilde{n}O(n^{\frac{9}{2}})$, where $\widetilde{n} = \min\{n_0, n_1\}$, and n_1 is the number of perfect matchings in the graph $G(X, Y, E)$.

Since we now dispose of an effective solution of Problem 4, the solution of Problem 3 is obtained using the following Algorithm 3.

Algorithm 3 List of optimal solutions (LOS) of TCAP.

Input: adjacency weights matrix $T(t_{ij})$ of $G(X, Y, E, t)$, and a natural number n_0
Output: list of optimal solutions L of TCAP

```
{w_0, A_0} ← TCAP(T(t_ij)) {Algorithm 1}
L ← LPMUBG(A_0, n_0) {Algorithm 2}
```

The correctness of the Algorithm 3 follows from the correctness of Algorithm 1 and Algorithm 2. Besides that, its complexity is $\widetilde{n}O(n^{\frac{9}{2}})$, where $\widetilde{n} = \min\{n_0, n_1\}$ and n_1 is the

number of perfect matchings in the graph $G(X, Y, E)$. The latter follows from the fact that the complexity of Algorithm 1 is evaluated $O(n^{\frac{9}{2}})$, and the complexity of Algorithm 2 is evaluated $\widetilde{n}O(n^{\frac{9}{2}})$.

We will illustrate the described LOS algorithm with the following examples, based on the weight matrix T of order 15 given in equation (9) below.

$$\begin{pmatrix} 24 & 26 & 42 & 15 & 29 & 25 & 35 & 23 & 19 & 25 & 25 & 25 & 15 & 18 & 25 \\ 7 & 14 & 16 & 1 & 30 & 25 & 7 & 11 & 21 & 20 & 12 & 11 & 10 & 11 & 20 \\ 20 & 13 & 15 & 35 & 5 & 1 & 26 & 6 & 16 & 15 & 15 & 8 & 13 & 22 & 15 \\ 21 & 16 & 25 & 20 & 18 & 18 & 6 & 46 & 25 & 23 & 26 & 5 & 31 & 9 & 23 \\ 12 & 46 & 27 & 48 & 28 & 5 & 67 & 13 & 23 & 27 & 14 & 35 & 21 & 32 & 27 \\ 23 & 5 & 5 & 9 & 5 & 19 & 32 & 42 & 32 & 19 & 22 & 23 & 19 & 18 & 19 \\ 35 & 7 & 26 & 6 & 67 & 32 & 21 & 11 & 15 & 21 & 21 & 12 & 16 & 17 & 21 \\ 5 & 23 & 65 & 10 & 16 & 67 & 57 & 28 & 32 & 18 & 18 & 21 & 27 & 23 & 18 \\ 8 & 11 & 25 & 35 & 11 & 21 & 17 & 21 & 21 & 24 & 22 & 20 & 23 & 35 & 24 \\ 11 & 25 & 11 & 21 & 17 & 21 & 24 & 22 & 24 & 20 & 21 & 17 & 19 & 30 & 19 \\ 25 & 9 & 12 & 15 & 26 & 14 & 22 & 21 & 18 & 21 & 19 & 26 & 21 & 16 & 21 \\ 26 & 42 & 15 & 29 & 25 & 35 & 23 & 19 & 25 & 25 & 25 & 24 & 31 & 13 & 16 \\ 26 & 6 & 16 & 15 & 15 & 8 & 13 & 22 & 15 & 25 & 25 & 22 & 18 & 26 & 25 \\ 6 & 46 & 25 & 23 & 26 & 5 & 31 & 9 & 23 & 20 & 12 & 32 & 32 & 22 & 23 \\ 67 & 13 & 23 & 27 & 14 & 35 & 21 & 32 & 27 & 15 & 15 & 25 & 25 & 16 & 25 \end{pmatrix} \quad (9)$$

Example 1: Let us solve Problem 2 with the input weight matrix T, given in equation (9). Applying Algorithm 1 we get one perfect matching:

$$w_0 = (13, 8, 15, 12, 11, 5, 9, 4, 2, 3, 6, 14, 7, 1, 10),$$

and minimal time $r_0 = 15$. It can be directly verified that $G_T(w_0) = 15$.

Example 2: Let us solve Problem 3 with the input weight matrix T, given in equation (9). We will solve the problem for two different values of n_0: $n_0 = 3$ and $n_0 = 270$.

1) In the case in which $n_0 = 3$ we get the list

$$L = \{(4, 13, 15, 12, 11, 2, 9, 1, 5, 3, 6, 14, 7, 8, 10), \\ (13, 1, 15, 12, 11, 2, 9, 4, 5, 3, 6, 14, 7, 8, 10), \\ (13, 2, 15, 12, 11, 3, 9, 4, 5, 1, 6, 14, 7, 8, 10)\}.$$

2) In the case in which $n_0 = 270$ we get a list with 240 elements. This means that the problem has exactly 240 optimal solutions for this particular example.

With a direct verification we can assert that $G_T(w) = 15$ for each $w \in L$ that corresponds to the solution of Example 1.

IV. Conclusion

The implementation of Algorithm 1 is verified using example matrices of order $n < 8$, for which Problem 2 is already solved using the existing methods. For the validation of Algorithm 2, we solve Problem 3 using a greedy method. Most of our experiments are with matrices of order $n \in \{30, 31, \ldots, 50\}$, in which the implementations of both Algorithm 1 and Algorithm 2 proof their effectiveness. On the other hand, in the cases in which $n = 50$ and n_0 is big enough, the time consumed by the solution of Problem 3 grows significantly. In these cases we plan to incorporate parallel computing techniques [8] in our implementation. Also, the presented approach for solving Problem 3 allows us to solve the problem for generation of a list of optimal solutions of the problem for minimum time with maximum profit.

References

[1] R. K. Ahuja, J. B. B. Orlin, C. Stein, R. E. Tarjan, "Improved algorithms for bipartite network flow," SIAM Journal on Computing, vol. 23, no. 5, 1994, pp. 906–933.

[2] R. E. Burkard, "Selected topics on assignment problems," Discrete Applied Mathematics, vol. 123, iss. 1–3, 2009, pp. 257–302.

[3] R. E. Burkard, M. Dell'Amico, S. Martello, "Assignment problems," Society for Industrial and Applied Mathematics, 2002.

[4] K. Flezar, "A branch-and-bound algorithm for the quadratic multiple knapsack problem," European Journal of Operational Research, vol. 298, iss. 1, 2021, pp. 89–98.

[5] M. L. Fredman, R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," Journal of the ACM, vol. 34, no. 3, 1987, pp. 596–615.

[6] D. R. Fulkerson, I. Glicksberg, and O. Gross, "A production line assignment problem," Tech. Rep. RM-1102, The Rand Corporation, Santa Monica, CA, 1953.

[7] B. Gabrovšek, T. Novak, J. Povh, D. R. Poklukarm J. Žerovnik, "Multiple Hungarian Method for k-Assignment Problem," Mathematics, vol. 8, no. 11, 2020, pp. 1–18.

[8] W. Gropp, E. Lusk, and A. Skjellum, "Using MPI: portable parallel programming with the message-passing interface," The MIT Press, 2014.

[9] R. Horst, "Global optimization: deterministic approaches," Springer-Verlag, Berlin Heidelberg, 1996, pp. 115–178.

[10] J. B. Jensen, G. Z. Gutin, "Digraphs theory algorithms and applications," Springer Publishing Company, 2008.

[11] H.W. Kuhn, "The Hungarian method for the assignment problem," Naval Research Logistics Quarterly, vol 2, 1955, pp. 83–97.

[12] H.W. Kuhn, "Variants of the Hungarian method for assignment problems," Naval Research Logistics Quarterly, vol 3, no. 4, 1956, pp. 253–258.

[13] Ö. Karsu, M. Azizoglu, "An exact algorithm for the minimum squared load assignment problem," Computers & Operations Research, vol 106, 2019, pp. 76–90.

[14] A. H. Land, and A. G. Doig, "An automatic method of solving discrete programming problems," Econometrica, vol. 28, no. 3, 1960, pp. 497–520.

[15] J. Niebling, E. Gabriele, "Branch–and–Bound-based algorithm for non-convex multiobjective optimization," SIAM Journal on Optimization, vol 29, 2019, pp. 794–821.

[16] J. E. Hopcroft, and R. M. Krap, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," SIAM Journal on Computing, vol. 2, no. 4, 1973, pp. 225–231.

[17] P.-E. Sarlin, D. DeTone, T. Malisiewicz, A. Rabinovich, "SuperGlue: Learning Feature Matching with Graph Neural Networks," 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 4937–4946.

[18] M. Z. Spivey, "Asymptotic moments of the bottleneck assignment problem," Mathematics of Operations Research, vol. 36, no. 2 , 2011, pp. 205–226.

[19] K. Shah, P. Reddy, S. Vairamuthu, “Improvement in Hungarian algorithm for assignment problem,” In: Suresh, L., Dash, S., Panigrahi, B. (eds) Artificial Intelligence and Evolutionary Algorithms in Engineering Systems. Advances in Intelligent Systems and Computing, Springer, vol. 324, 2015, pp. 1–8.

[20] C. C. Shen and W. H. Tsai, “A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion,” IEEE Transactions on Computers, vol. C-34, no. 3, 1985, pp. 197–203.