List of Optimal Solutions of the Minimum Cost for Minimum Time Assignment Problem

Lasko M. Laskov(✉) and Marin L. Marinov

Informatics Department, New Bulgarian University, Sofia, Bulgaria
{llaskov,mlmainov}@nbu.bg

Abstract. One of the fundamental problems in the disciplines of combinatorial optimization and operation research is the *assignment problem* (AP). Even though the AP and its variants has been explored extensively in the specialized literature, most of the resources are focused on the calculation of a single optimal solution. In this paper we propose an approach that generates a list of optimal solutions.

We focus on the bi-criteria variant of the AP in which two objectives are minimized: *cost* and *time*. Our approach finds the maximal subset of assignments that are Pareto optimal, and have minimal cost. In particular, the method finds a list of all assignments that are optimal with respect to the cost criterion.

Keywords: Pareto optimal · Assignment problem · Branch and bound

1 Introduction

The *assignment problem* (AP) [5] is a fundamental problem in the fields of combinatorial optimization and operation research [11,22,24]. It has been explored since the middle of the 20th century [9,15,16], and continued to be in the focus of investigations [1,20,27]) with the progress of information technologies and computer science until now-a-days (see for example [4,8,14]). The applications of AP are numerous, and can be found in many scientific disciplines such as economics [10], data science, machine learning and pattern recognition [23], distributed computer systems [25,26], and many others.

In its basic form, the AP can be formulated as the problem of finding an assignment of n *tasks* (jobs, projects, processes) to n other *agents* (workers, companies, processors) that is efficient according to a certain formal criterion. Most commonly the criterion is to minimize the total cost of the proposed plan. In the above informal definition the number of tasks and agents is equal, and in this case the AP called *balanced*. In the case in which the number of tasks and agents differ, the AP is said to be *unbalanced* [5].

T. Zlateva and G. Tuparov (Eds.): CSECS 2023, LNICST 514, pp. 87–101, 2023.
https://doi.org/10.1007/978-3-031-44668-9_7

The naive solution of AP results in a full exhaustion algorithm that leads to unfeasible factorial complexity $O(n!)$, which makes the problem hard to solve even for relatively small values of n. However, different algorithms exist that solve the AP in polynomial time, with the first known such method called *Hungarian algorithm* published in 1956 in the famous work of Kuhn [15], and an year latter extended to its variants in [16]. The computational complexity of the initial version of the Hungarian algorithm was $O(n^4)$ which latter is shown that can be improved to $O(n^3)$ [7,28]. Different versions and improvements of the Hungarian algorithm continue to be a subject of research in more recent works (see for example [8,25]).

A whole section of methods treat AP as a network flow problem [1,7] and adopt the tools of graph theory [2,6] to find an efficient solution. Most of the methods in this category bring the AP to the problem of searching for matchings in bipartite graphs [12,26].

At the same time, a global optimization technique known as *branch & bound* (see [13,17,19]) has been shown to lead to efficient solutions to other combinatorial optimization problems such as the *Travelling Salesman Problem* (TSP) (see [18,21]), and some versions of the *knapsack problem* [8].

Most of the existing methods and algorithms in the literature aim to find an efficient approach to find a single optimal solution of the AP. The main goal of this paper is to present an effective approach for composition of a list of all assignments that have minimal cost and satisfy an additional condition. This approach is presented with a solution of the AP in the case in which two independent optimization criteria are defined: time and cost. The main problem is brought to finding of the maximal subset of Pareto optimal solutions that have minimal cost.

The paper is organized as follows. In Sect. 2 we introduce the main notations and definitions. In Sect. 3 we present the solution of the main problem. In Sect. 4 we solve the problem for description of all solutions of the AP. The solution follows from the presented approach in Sect. 3. Finally, Sect. 5 contains discussion of the presented methods and conclusions.

2 Main Notations and Definitions

In this section we will introduce the basic notations.

With $N(n) = \{1, 2, \ldots, n\}$ we denote the set of the first n natural numbers, for each natural number n. With P^n we denote the permutations of the first n natural numbers. We denote the permutation

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

with the shorter $w = (j_1, j_2, \ldots, j_k)$. Also, $w(s) = j_s$, $\forall s \in N(n)$.

Let B is an arbitrary matrix. With $B(i, j)$ we denote the element of B on the ith row and jth column. We denote with

$$[B]^{i_1,i_2,\ldots,i_k}_{j_1,j_2,\ldots,j_s} \tag{2}$$

the sub-matrix that results from B after we delete the rows with indexes $i_1, i_2, \ldots, i_k$, and columns with indexes $j_1, j_2, \ldots, j_s$.

For an arbitrary square *cost matrix* A of order n and an arbitrary permutation $w \in P^n$, we define the *cost function*:

$$F_A(w) = \sum_{i=1}^{n} A(i, w(i)). \tag{3}$$

Similarly, for an arbitrary square *time matrix* T of order n and an arbitrary permutation $w \in P^n$, we define the *time maximization function*:

$$G_T(w) = \max_{i \in N(n)} \{T(i, w(i))\}. \tag{4}$$

The function t assigns to each variation $w = (j_1, j_2, \ldots, j_k)$, $k \in N(n)$ of the first n natural numbers the number

$$t(w) = \max\{T(1, j_1), T(2, j_2), \ldots, T(k, j_k)\}.$$

It is clear that when $k = n$, $t(w) = G_T(w)$.

2.1 Main Problem

Let n independent tasks must be distributed among n agents. The agent i can execute the task j for time t_{ij} that results in costs a_{ij}, $i, j \in N(n)$. The objective is to find the maximal set of plans that are executed for minimal cost, and for shortest possible time.

Let the cost matrix $A(a_{ij})$ and the time matrix $T(t_{ij})$ are predefined square matrices of order n.

We will call each permutation $w \in P^n$ a *plan.*

Definition 1. *We will call an* ***optimal plan*** *each plan $\widetilde{w}$, for which the following equalities hold:*

$$F_A(\widetilde{w}) = r_0 \text{ and } G_T(\widetilde{w}) = t_0$$

where:
$r_0 = \min\{F_A(w) : w \in P^n\}$, $t_0 = \min\{G_T(w) : w \in P^n \text{ and } F_A(w) = r_0\}$.

Now let us define the two sets:
$W = \{w \in P^n : F_A(w) = r_0\}$ and $V = \{w \in W : G_T(w) = t_0\}$.

Definition 2. *The plan $\widehat{w}$ is called* ***Pareto optimal*** *when there does not exist a plan w, for which one of the following holds:*

- $F_A(w) < F_A(\widehat{w})$ *and* $G_T(w) \leq G_T(\widehat{w})$, *or*
- $F_A(w) \leq F_A(\widehat{w})$ *and* $G_T(w) < G_T(\widehat{w})$.

Remark 1. The set V of all optimal plans is the set of all Pareto optimal solutions that have minimal cost r_0.

Now the main problem that we will call *Minimum Cost for Minimum Time* (MCMT) is defined in the following way.

Problem 1 (MCMT). For arbitrary square matrices A and T of the same order, and an arbitrary chosen natural number n_0, compile a list S_0 of optimal plans with the following properties.

1. If more than $n_0 - 1$ optimal plans exist, then S_0 contains n_0 elements.
2. If the optimal plans are less than n_0, then S_0 contains all the optimal plans.

To solve the MCMT problem, we will use the procedure $\mathcal{H}$ and the function $\mathcal{T}$.

With $\mathcal{H}$ we denote the procedure that for an arbitrary square matrix M of order k calculates the pair $\{r_0, w_0\}$, where $r_0 = \min\limits_{w \in P^k} \{F_M(w)\}$, $w_0 \in P^k$ and $F_M(w_0) = r_0$. In our implementations of $\mathcal{H}$, we use the Hungarian algorithm [7,15,28] to solve the classical AP with computational complexity $O(n^3)$.

With $\mathcal{T}$ we denote the function

$$\mathcal{T}(M) = \max\{\max\{M_r\},\ \max\{M_c\}\}, \tag{5}$$

where M_r is the set of row minima of M and M_c is the set of column minima of M.

3 Solution of MCMT Problem

The challenging feature of the MCMT problem is that the optimal plan is determined by two criteria:

- the *cost* criterion, for which the function F_A is minimized;
- the *time* criterion, for which the function G_T is minimized.

Moreover, the function G_T is not a linear function, and for that reason the whole problem is nonlinear.

3.1 General Structure of the Solution

Step 1. Calculate a plan w_0 that is executed with a minimal cost. Define:

- minimal cost $r_0 = F_A(w_0)$;
- the current record $S_0 = \{\}$ and $t_0 = G_T(w_0)$.

Step 2. Define the stack $S = \{X_0\}$, where X_0 is storage of the initial problem.

Step 3. While $S \neq \varnothing$, update t_0 and S_0.

When the loop from *Step 3* completes, S_0 stores the solution of the MCMT problem, and t_0 stores the minimal time.

In *Step 1* of the solution, the procedure $\mathcal{H}$ applied on $Z = A$ calculates the minimal possible cost r_0 and a plan w_0, for which $F_A(w_0) = r_0$. Also, the initial record is defined $t_0 = G_T(w_0)$.

We store the sub-problems that result from the branching process in *Step 3* in a list of the form $\{w, B, Bt, k, dt\}$, where w is a vector; B and Bt are matrices, and k, dt are special numbers.

The *initial problem* is stored with $X_0 = \{w, B, Bt, k, dt\}$ where: $w = (\,) = \varnothing$; $Bt = T$; $dt = \mathcal{T}(Bt)$; $k = n$ and B is a $(n+1) \times n$ matrix with elements

$$B(i,j) = \begin{cases} A(i,j), & \text{if } i \in N(n) \text{ and } j \in N(n) \\ j, & \text{if } i = n+1 \text{ and } j \in N(n). \end{cases} \tag{6}$$

In *Step 2* the set $S = \{X_0\}$ is defined.

The update of t_0 and S_0 in *Step 3* combines the branching process with the bound by the cost and time criteria.

3.2 Branching Procedure

If $n > 2$, the branch of X_0 replaces X_0 in S with a finite number of sub-problems, resulting from the following inductive procedure.

The **base case** of the induction is composed by the following two stages.

1. For each $j \in \{1, 2, \ldots, n\}$ we define $w_j = (j)$, $B_j = [B]^1_j$, $Bt_j = [Bt]^1_j$ and $dt_j = \max\{Bt(1,j), \mathcal{T}(Bt_j)\}$.
2. For each $j \in \{1, 2, \ldots, n\}$:
   - using the procedure $\mathcal{H}$ for $Z = [B_j]^n$ we calculate $\{\widetilde{r}, \widetilde{w}\}$;
   - if $B(1,j) + \widetilde{r} = r_0$, we push the sub-problem $X_{w_j} = \{w_j, B_j, Bt_j, n-1, dt_j\}$ into the stack S.

The first stage of the base case implements branching, while the second stage implements bound by the cost criterion.

Inductive Step. Let $X_{w'} = \{w', B', Bt', k, dt'\}$ is a sub-problem that is a result of the previous branching and $w' = (j_1, j_2, \ldots, j_{n-k})$, where j_s are different $n-k$ natural numbers, for which $j_s \leq n$. If $k > 2$, then the branching procedure replaces $X_{w'}$ in S with the finite number of its sub-problems, that are a result from the following two stages.

1. For each $i \in \{1, 2, \ldots, k\}$ we define $w_i = (j_1, j_2, \ldots, j_{n-k}, B'(k+1, i))$; $B'_i = [B']^1_i$; $Bt'_i = [Bt']^1_i$ and $dt'_i = \max\{t(w_i), \mathcal{T}(Bt'_i)\}$.
2. For each $i \in \{1, 2, \ldots, k\}$:
   - using the procedure $\mathcal{H}$ for $Z = [B'_i]^k$ we calculate $\{\widetilde{r}, \widetilde{w}\}$;
   - if $\sum_{s=1}^{n-k} B(s, j_s) + B(n-k+1, B'(k+1, i)) + \widetilde{r} = r_0$, we push the sub-problem $X_{w_i} = \{w_i, B'_i, Bt'_i, k-1, dt'_i\}$ into the stack S.

Again, the first stage of the inductive step implements branching, while the second stage implements bound by the cost criterion.

3.3 Update of t_0 and S_0

The current values of t_0 and S_0 will be updated only after the extraction of a sub-problem $X_{w'} = \{w', B', Bt', 2, dt'\}$ from the stack S. In this case $w' = (j_1, j_2, \ldots, j_{n-2})$ is a variation of the first n natural numbers from $(n-2)$ class, and there exists exactly two permutations

$$w_1 = (j_1, j_2, \ldots, j_{n-2}, x, y) \quad \text{and} \quad w_2 = (j_1, j_2, \ldots, j_{n-2}, y, x)$$

from P^n.

We will suppose that a function *edn* is defined, such that for an arbitrary variation w' of the first n natural numbers from class $(n-2)$, calculates the pair of permutations $end(w') = \{w_1, w_2\}$.

If $X_{w'} = \{w', B', Bt', 2, dt'\}$ is at the top of the stack S, then we define $\{w_1, w_2\} = end(w')$ and pop $X_{w'}$ out from S.

For each $i \in \{1, 2\}$, if $F_A(w_i) = r_0$ and $G_T(w_i) < t_0$, then we define $t_0 = G_T(w_i)$ and $S_0 = \{w_i\}$. Otherwise, if $F_A(w_i) = r_0$ and $G_T(w_i) = t_0$, then we push w_i into S_0, if $|S_0| < n_0$.

3.4 Bound Procedure

The current record t_0 allows us to implement a bound procedure based on time criterion.

Let for the sub-problem $X_{w'} = \{w', B', Bt', k, dt'\}$ the inequality is fulfilled

$$t_0 < dt' \tag{7}$$

From the definition of dt' it follows that if $w \in P^n$ and $w(i) = w'(i), \forall i \in \{1, \ldots, n-k\}$, then it is fulfilled that

$$G_T(w) \geq dt' > t_0.$$

Therefore, w is not an optimal assignment. This allows each sub-problem $X_{w'}$, for which (7) is fulfilled, to be ignored and to be removed from the stack S.

3.5 Algorithm for Solving MCMT Problem

The following Algorithm 1 describes in details the proposed solution of the MCMT problem.

The **while** loop of Algorithm 1 implements *Step 3* of the general structure of the solution. If S has at least one element, then store the top of the stack S in $\{w', B', Bt', k, dt'\}$, and we remove it from the stack with $pop(S)$. Then the body of the loop contains three possibilities.

1. If $dt' > t_0$, then bound is performed, and the algorithm moves to the next iteration of the loop.

Algorithm 1. Solution of the MCMT problem.

function MinCostMaxTime(A, T, n_0)
  $\{r_0, w_0\} \leftarrow \mathcal{H}(A)$
  $t_0 \leftarrow G_T(w_0)$, $S_0 \leftarrow \varnothing$ ▷ S_0 is a list
  $w \leftarrow \varnothing$, B is a matrix defined with (6)
  $Bt \leftarrow T$, $dt \leftarrow \mathcal{T}(Bt)$
  $X_0 \leftarrow \{w, B, Bt, n, dt\}$, push($S$, X_0) ▷ S is a stack
  while $S \neq \varnothing$ **do**
    $\{w, B, Bt, k, dt\} \leftarrow top(S)$, $pop(S)$
    if $dt \leq t_0$ **then**
      if $k > 2$ **then**
        for $j \leftarrow 1$ to k **do**
          $w_1 \leftarrow w \cup \{B(k+1, j)\}$
          $B_1 \leftarrow [B]_j^1$
          $\{\overline{r}, \widetilde{w}\} \leftarrow \mathcal{H}([B_1]^k)$
          $\overline{w} \leftarrow \{B_1(k, \widetilde{w}(1)), B_1(k, \widetilde{w}(2)), \ldots, B_1(k, \widetilde{w}(k-1))\}$
          $v_1 \leftarrow w_1 \cup \overline{w}$, $r_1 \leftarrow F_A(v_1)$
          if $r_1 = r_0$ **then**
            $t_1 \leftarrow G_T(v_1)$
            if $t_1 < t_0$ **then**
              $t_0 \leftarrow t_1$
              $S_0 \leftarrow \varnothing$
              $Bt_1 \leftarrow [Bt]_j^1$
              $dt_1 \leftarrow \max\{t(w_1), \mathcal{T}(Bt_1)\}$
              $push(S, \{\{w_1, B_1, Bt_1, k-1, dt_1\}\})$
            end if
          end if
        end for
      else
        $V \leftarrow end(w)$
        for $i \leftarrow 1$ to 2 **do**
          $r_1 \leftarrow F_A(V(i))$, $t_1 \leftarrow G_T(V(i))$
          if $r_0 = r_1$ **and** $t_1 < t_0$ **then**
            $t_0 \leftarrow t_1$, $w_0 \leftarrow V(i)$, $S_0 \leftarrow \{w_0\}$
          else if $r_0 = r_1$ **and** $t_1 = t_0$ **and** $|S_0| < n_0$ **then**
            $S_0 \leftarrow S_0 \cup \{V(i)\}$
          end if
        end for
      end if
    end if
  end while
end function

2. If $dt' \leq t_0$ and $k > 2$ the algorithm performs branching of (w', B', Bt', k, dt'), and the resulting sub-problems are pushed into the stack S. If the conditions are met, the current state of t_0 and S_0 are updated.
3. If $dt' \leq t_0$ and $k = 2$, t_0 and S_0 are updated.

After the completion of the **while** loop, the solution of the MCMT problem is the list S_0, with $|S_0|$ being the number of optimal assignments, r_0 stores the minimal cost and t_0 is the minimal possible time.

The correctness of Algorithm 1 follows from the fact that the **while** loop stops after a finite number of iterations. This is so, because the number of sub-problems that can be stored in S is restricted up to a given constant. Besides that, any update of S_0 and t_0 decrements the number of elements in S by 1. The correctness of the branching procedure can be proved by induction using the construction of the sub-problems. The computational complexity of the algorithm is $n_1 O\left(n^4\right)$, where $n_1 = |W|$ is the number of optimal solutions with respect to the cost criterion.

Example 1. Let the matrix A be defined by (12) and the matrix T be defined with (13) given in the appendix of the paper.

We solve Problem 1 for $n_0 = 4$. We calculate that the minimum cost is $r_0 = 218$, the minimum time is $t_0 = 85$ and

$$\begin{aligned} S_0 = \{&(18, 12, 13, 17, 16, 8, 9, 20, 7, 3, 19, 4, 14, 1, 15, 11, 2, 6, 10, 5),\\ &(18, 12, 13, 17, 16, 8, 9, 20, 7, 5, 19, 4, 14, 1, 15, 11, 2, 6, 10, 3),\\ &(18, 12, 13, 17, 16, 8, 9, 20, 10, 3, 19, 4, 14, 1, 15, 11, 2, 6, 7, 5),\\ &(18, 12, 13, 17, 16, 8, 9, 20, 10, 5, 19, 4, 14, 1, 15, 11, 2, 6, 7, 3)\}. \end{aligned} \tag{8}$$

Now we solve Problem 1 with the same matrices A and T but for $n = 1025$. As expected, we get a minimum cost of $r_0 = 218$ and a minimum time of $t_0 = 85$. However, the list S_0 has 256 elements. This shows that the number of all optimal plans of Problem 1 is equal to 256.

The analysis of the above solution shows that only 226 bounds on time criterion are performed. At the same time, 2707 iterations of the branching loop are performed. From this observation we can conclude that the time estimates that are used are not always efficient enough. The reason is that the estimate dt does not fully account for the influence of the cost matrix A – a fact that is confirmed by the conducted experiments.

4 List of Solutions of the AP

We select an arbitrary square matrix $A(a_{ij})$ of order n with non-negative elements.

Definition 3. *We shall call the plan $\widetilde{w}$ an **optimal solution of the AP**, when*

$$F_A(\widetilde{w}) \leq F_A(w),$$

for each $w \in P^n$.

Now we can formulate the AP in the following formal way: for an arbitrary square cost matrix A, find an optimal solution $\widetilde{w}$. Based on this formulation, the problem for generation of *Minimum Cost Optimal Solutions List* (MCOSL) is given in Problem 2.

Problem 2 (MCOSL). For an arbitrary square cost matrix A and an arbitrary natural number n_0, generate a list S_0 of optimal solutions of the AP with the following properties.

1. If the AP has more than $n_0 - 1$ optimal solutions, then S_0 contains n_0 elements.
2. If the AP has less than n_0 solutions, then S_0 contains all optimal solutions.

We will solve MCOSL problem following the approach in Sect. 3. In this case, the general form of the approach is as follows.

1. We calculate the elements of the pair $\{r_0, w_0\}$, where $r_0 = \min\limits_{w \in P^n} \{F_A(w)\}$, $w_0 \in P^n$ and $F_A(w_0) = r_0$.
2. We define a list S_0 and a queue S. In the list S_0 we collect the calculated optimal solutions, while in the queue S we place the non-intersecting subtasks, each of them resulting in at least one optimal solution of the initial AP.
3. The list S_0 and the queue S are updated until at least one of the following two conditions are violated:
$$S \neq \varnothing, \tag{9}$$
$$|S_0| + |S| < n_0, \tag{10}$$
where $|\cdot|$ denotes the number of elements in the sequence (list or queue).
4. When the update process stops, there are two possible cases.
   - *Case 1.* $S = \varnothing$. In this case S_0 contains all optimal solutions.
   - *Case 2.* $|S_0| + |S| \geq n_0$. In this case we insert into S_0 the optimal solutions of the sub-problems that are contained in S. The result is a list S_0 of n_0 optimal solutions.

The implementation of the above scheme is given in Algorithm 2.

The correctness of Algorithm 2 can be proved in analogy of the proof of Algorithm 1.

Example 2. We will solve Problem 2 for $n_0 = 4$ and the matrix A, given in (12) given in the paper appendix.

Our implementation of Algorithm 2 calculates that the minimal cost is $r_0 = 218$ and the list of optimal solutions is:

$$\begin{aligned} S_0 = \{&(3, 9, 11, 17, 1, 8, 14, 6, 12, 18, 4, 19, 7, 16, 15, 13, 2, 20, 10, 5),\\ &(3, 12, 11, 17, 1, 8, 9, 20, 7, 5, 4, 19, 14, 16, 15, 13, 2, 6, 10, 18),\\ &(18, 9, 11, 17, 1, 8, 14, 6, 12, 5, 4, 19, 7, 16, 15, 13, 2, 20, 10, 3),\\ &(18, 12, 11, 17, 1, 8, 9, 20, 7, 5, 4, 19, 14, 16, 15, 13, 2, 6, 10, 3)\}. \end{aligned} \tag{11}$$

Algorithm 2. Solution of the MCOSL problem.

function MinCostList(A, n_0)
  $\{r_0, w_0\} \leftarrow \mathcal{H}(A)$
  $w \leftarrow \varnothing$
  B is a matrix defined with (6)
  $X_0 \leftarrow \{w, w_0, B, n\}$,
  enqueue(S, X_0), $S_0 \leftarrow \varnothing$ ▷ S is a queue, S_0 is a list
  while $S \neq \varnothing$ **and** $|S_0| + |S| < n_0$ **do**
    $\{w, v, B, k\} \leftarrow$ front(S), dequeue(S)
    if $k > 2$ **then**
      for $j \leftarrow 1$ to k **do**
        $w_1 \leftarrow w \cup \{B(k+1, j)\}$
        $B_1 \leftarrow [B]_j^1$
        $\{\overline{r}, \overline{w}\} \leftarrow \mathcal{H}([B_1]^k)$
        $\overline{w} \leftarrow \{B_1(k, \overline{w}(1)), \ldots, B_1(k, \overline{w}(k-1))\}$ ▷ original B indexing
        $v_1 \leftarrow w_1 \cup \overline{w}$
        $r_1 \leftarrow F_A(v_1)$
        if $r_1 = r_0$ **then**
          enqueue(S, $\{w_1, v_1, B_1, k-1\}$)
        end if
      end for
    else
      pushBack(S_0, $\{v\}$)
      if $|S_0| + |S| < n_0$ **then**
        $v_1 \leftarrow v$
        $v_1(n-1) \leftarrow v(n)$
        $v_1(n) \leftarrow v(n-1)$
        if $F_A(v_1) = r_0$ **then**
          pushBack(S_0, $\{v_1\}$)
        end if
      end if
    end if
  end while
  if $S \neq \varnothing$ **then**
    for $s \leftarrow 1$ to $n_0 - |S_0|$ **do**
      pushBack(S_0, $\{S(s, 2)\}$)
    end for
  end if
  return S_0
end function

Now we solve Problem 2 with the same input matrix A, but for $n_0 = 10^4$. As we can expect, the resulting minimum cost is $r_0 = 218$. However, the list S_0 has 1024 elements. This shows that the number of all optimal solutions of the AP for the matrix A is equal to 1024.

If the matrix A is of 30-th order, and its elements are natural numbers whose values do not exceed 70, then we can expect that the number of optimal solutions is comparable to 10^4.

5 Conclusion

AP is a fundamental problem in the field of operation research. The first known polynomial-time algorithm that solves it is the Hungarian algorithm [15], which is considered as one of the methods that initiate the foundations of the discipline combinatorial optimization [5]. Nowadays, the AP together with its variants like for example quadratic AP [14], and k-assignment problem [8], is a subject of contemporary research with huge significance for many different practical tasks in economics, computer science, machine learning, and many more. Hungarian algorithm itself is investigated and its improvements are looked for in contemporary works (see for example [25]).

Despite its popularity, the vast portion of the existing algorithms are focused on the calculation of a single optimal solution of the AP, and usually the problem of discovering of more than one optimal solution is not considered. We find that methods for generation of a list of optimal solutions for the AP are an important extension of the methods that solve this fundamental task, which is significant for its practical applications.

In this paper we propose a method for composition of a list of selected assignments, that satisfy an additional condition. An important part in our method is given to the procedure $\mathcal{H}$ that implements the Hungarian algorithm. The approach is demonstrated with the solutions of the two problems: MCMT and MCOSL. In the problem MCMT we compose the list of those solutions of AP that are Pareto optimal. In the MCOSL problem an additional condition is the predefined number of elements in the list.

The correctness of the computer program implementation of Algorithm 2 is verified experimentally using matrices of order $n < 8$ for which Problem 2 is feasible to be calculated using other methods. Similarly, the implementation of the solution of Problem 1 is verified using matrices A and T of order $n < 8$, for which brute force techniques can be applied. At the same time, the computational complexity of our implementations is experimentally tested with matrices up to 30-th order (see (12) and (13) in the appendix below).

We have used two different approaches in the implementation of the described algorithms: the symbolic computational system Mathematica [29], and the relatively new programming language for scientific computing Julia [3]. The Mathematica system provides a large set of tools, that are extremely helpful in both implementation and validation steps, while the Julia programming language combines the abilities of dynamic languages together with efficiency, and capability to handle big data inputs. Both implementations are used to verify our experiments and to prove the efficiency of the proposed algorithms.

A Appendix: Input matrices

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

$$\begin{pmatrix}
57 & 56 & 65 & 63 & 65 & 53 & 58 & 44 & 56 & 71 & 70 & 42 & 58 & 72 & 65 & 57 & 56 & 65 & 63 & 65 \\
69 & 69 & 71 & 79 & 57 & 77 & 72 & 55 & 99 & 65 & 52 & 74 & 80 & 69 & 62 & 69 & 69 & 71 & 79 & 62 \\
65 & 73 & 77 & 55 & 69 & 66 & 58 & 85 & 60 & 72 & 80 & 65 & 84 & 66 & 66 & 65 & 73 & 77 & 60 & 66 \\
55 & 85 & 46 & 75 & 63 & 50 & 84 & 70 & 57 & 63 & 73 & 84 & 36 & 56 & 57 & 55 & 85 & 51 & 72 & 57 \\
76 & 42 & 63 & 54 & 44 & 88 & 49 & 50 & 33 & 63 & 85 & 15 & 68 & 57 & 54 & 76 & 47 & 60 & 48 & 54 \\
55 & 61 & 67 & 53 & 81 & 72 & 72 & 76 & 82 & 85 & 63 & 49 & 38 & 49 & 71 & 60 & 58 & 61 & 63 & 71 \\
63 & 74 & 55 & 83 & 74 & 42 & 74 & 65 & 84 & 15 & 49 & 59 & 70 & 75 & 61 & 60 & 68 & 65 & 73 & 56 \\
68 & 50 & 73 & 72 & 59 & 76 & 68 & 25 & 72 & 65 & 13 & 24 & 62 & 50 & 63 & 62 & 60 & 63 & 54 & 66 \\
49 & 80 & 72 & 42 & 57 & 83 & 79 & 57 & 46 & 69 & 60 & 73 & 61 & 60 & 56 & 59 & 70 & 54 & 49 & 62 \\
79 & 78 & 58 & 51 & 72 & 79 & 57 & 70 & 59 & 64 & 69 & 58 & 59 & 56 & 61 & 69 & 60 & 65 & 56 & 52 \\
76 & 51 & 60 & 75 & 69 & 57 & 72 & 68 & 66 & 64 & 68 & 59 & 59 & 63 & 69 & 58 & 58 & 65 & 55 & 79 \\
52 & 57 & 60 & 77 & 66 & 55 & 38 & 66 & 61 & 57 & 46 & 57 & 62 & 65 & 52 & 59 & 62 & 40 & 87 & 79 \\
56 & 69 & 72 & 56 & 56 & 54 & 75 & 74 & 67 & 66 & 72 & 68 & 68 & 62 & 59 & 61 & 49 & 82 & 69 & 52 \\
79 & 58 & 50 & 59 & 57 & 75 & 44 & 57 & 58 & 54 & 76 & 59 & 68 & 61 & 66 & 59 & 68 & 63 & 55 & 58 \\
75 & 57 & 56 & 64 & 56 & 23 & 69 & 58 & 53 & 67 & 55 & 56 & 52 & 59 & 56 & 85 & 70 & 52 & 65 & 66 \\
67 & 73 & 67 & 59 & 75 & 62 & 68 & 65 & 46 & 85 & 76 & 58 & 80 & 55 & 85 & 80 & 69 & 68 & 69 & 75 \\
55 & 75 & 50 & 81 & 59 & 60 & 64 & 56 & 70 & 59 & 66 & 80 & 25 & 75 & 72 & 51 & 76 & 60 & 81 & 59 \\
62 & 60 & 63 & 59 & 63 & 75 & 58 & 25 & 67 & 68 & 19 & 14 & 72 & 63 & 59 & 63 & 70 & 63 & 59 & 63 \\
59 & 70 & 59 & 46 & 56 & 73 & 79 & 52 & 49 & 75 & 50 & 83 & 74 & 56 & 57 & 69 & 70 & 59 & 46 & 56 \\
69 & 65 & 62 & 50 & 62 & 79 & 52 & 73 & 65 & 54 & 79 & 71 & 55 & 57 & 71 & 69 & 65 & 62 & 50 & 62
\end{pmatrix} \tag{13}$$

References

1. Ahuja, R.K., Orlin, J.B., Stein, C., Tarjan, R.E.: Improved algorithms for bipartite network flow. SIAM J. Comput. **23**(5), 906–933 (1994). https://doi.org/10.1137/S0097539791199334
2. Bang-Jensen, J., Gutin, G.Z.: Digraphs Theory Algorithms and Applications. Springer, Berlin (2008). https://doi.org/10.5555/1523254
3. Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: a fresh approach to numerical computing. SIAM Rev. **59**(1), 65–98 (2017). https://doi.org/10.1137/141000671
4. Burkard, R.: Selected topics on assignment problems. Discret. Appl. Math. **123**, 257–302 (2009). https://doi.org/10.1016/S0166-218X(01)00343-2
5. Burkard, R., Dell'Amico, M., Martello, S.: Assignment problems. SIAM, Society for Industrial and Applied Mathematics, University City, Philadelphia (2009). https://doi.org/10.1137/1.9781611972238
6. Diestel, R.: Graph Theory, 5th edn. Springer, Berlin (2017). https://doi.org/10.1007/978-3-662-53622-3
7. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. J. ACM **19**(2), 248–264 (1972). https://doi.org/10.1145/321694.321699
8. Flezar, K.: A branch-and-bound algorithm for the quadratic multiple knapsack problem. Eur. J. Oper. Res. **298**, 89–98 (2021). https://doi.org/10.1016/j.ejor.2021.06.018
9. Fulkerson, D.R., Glicksberg, I.L., Gross, O.A.: A production line assignment problem. Technical report, RM-1102, The Rand Corporation, Santa Monica, CA (1953)

10. Graham, B.S.: Econometric methods for the analysis of assignment problems in the presence of complementarity and social spillovers. In: Handbook of Social Economics, vol. 1, pp. 965–1052. North-Holland (2011). https://doi.org/10.1016/B978-0-444-53707-2.00002-5
11. Hall, M.: Combinatorial Theory, 2nd edn. Wiley, New York (1998). https://doi.org/10.5555/286075
12. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM J. Comput. **2**(4), 225–231 (1973). https://doi.org/10.1137/0202019
13. Horst, R.: Global Optimization: Deterministic Approaches, 2nd edn., pp. 115–178. Springer, Heidelberg (1996). https://doi.org/10.1057/jors.1994.88
14. Karsu, Ö., Azizoglu, M.: An exact algorithm for the minimum squared load assignment problem. Comput. Oper. Res. **106**, 76–90 (2019). https://doi.org/10.1016/j.cor.2019.02.011
15. Kuhn, H.W.: The Hungarian method for the assignment problem. Naval Res. Logistics Q. **2**(1–2), 83–97 (1955). https://doi.org/10.1002/nav.3800020109
16. Kuhn, H.W.: Variants of the Hungarian method for assignment problems. Naval Res. Logistics Q. **3**(4), 253–258 (1956). https://doi.org/10.1002/nav.3800030404
17. Land, A.H., Doig, A.G.: An automatic method of solving discrete programming problems. Econometrica **28**(3), 497–520 (1960). https://doi.org/10.2307/1910129
18. Marinov, M.L., Laskov, L.M.: The travelling salesman problem with the symbolic computational system mathematica. In: Proceedings of the Forty-seven Spring Conference of the Union of Bulgarian Mathematicians, pp. 147–157 (2018)
19. Niebling, J., Eichfelder, G.: Branch-and-bound-based algorithm for nonconvex multiobjective optimization. SIAM J. Optim. **29**(1), 794–821 (2019). https://doi.org/10.1137/18M1169680
20. Orlin, J.B., Ahuja, R.K.: New scaling algorithms for the assignment and minimum mean cycle problems. Math. Program. **54**, 41–56 (1992). https://doi.org/10.1007/BF01586040
21. Padberg, M., Rinaldi, G.: A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. IAM Review **33**, 60–100 (1991). https://doi.org/10.1137/1033004
22. Papadimitriou, C.H., Steiglitz, K.: Combinatorial optimization: algorithms and complexity. Dover Books on Computer Science, Prentice-Hall Inc, Hoboken, New Jersey (1982). https://doi.org/10.1109/TASSP.1984.1164450
23. Sarlin, P.E., DeTone, D., Malisiewicz, T., Rabinovich, A.: Superglue: learning feature matching with graph neural networks. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4937–4946 (2020). https://doi.org/10.48550/arXiv.1911.11763
24. Schrijver, A.: A Course in combinatorial optimization. copyright A. Schrijver, CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands and Department of Mathematics, University of Amsterdam, Plantage Muidergracht 24, 1018 TV Amsterdam, The Netherlands (2017). https://homepages.cwi.nl/ lex/files/dict.pdf
25. Shah, K., Reddy, P., Vairamuthu, S.: Improvement in Hungarian algorithm for assignment problem. Adv. Intell. Syst. Comput. **324**, 1–8 (2015). https://doi.org/10.1007/978-81-322-2126-5_1
26. Shen, C.C., Tsai, W.H.: A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion. IEEE Trans. Comput. **C-34**(3), 197–203 (1985). https://doi.org/10.1109/TC.1985.1676563
27. Shmoys, D.B.: Éva Tardos: an approximation algorithm for the generalized assignment problem. Math. Program. **62**, 461–474 (1993). https://doi.org/10.1007/BF01585178

28. Tomizawa, N.: On some techniques useful for solution of transportation network problems. Networks **1**(2), 173–194 (1971). https://doi.org/10.1002/net.3230010206
29. Wolfram, S.: An Elementary Introduction to the Wolfram Language, 2nd edn. Wolfram Media, Incorporated (2017)