Pareto Optimal Solutions of the Minimal Cost Minimal Time Assignment Problem

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

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

***Abstract*—We propose an exact algorithm for calculation of a list of all Pareto optimal solutions of a biobjective assignment problem (AP) with a linear objective function (representing a cost criterion) and a non-linear bottleneck objective function (representing a time criterion). The method, along with all needed helper functions for its implementation, is described in details.**

We provide comprehensive examples that illustrate the results, and a graphical illustration of the presented algorithm that also gives a geometrical proof of the method.

The correctness of the algorithms is proved, and their computational complexity is shown. The presented algorithms can be adopted to solve the other standard biobjective AP with the first function being a linear one, and the second function being from the bottleneck type.

***Index Terms*—combinatorial optimization, assignment problem, Pareto optimality, biobjective optimization**

I. Introduction

The assignment problem (AP) [1] is a fundamental subject in combinatorial optimization and operations research. It has been investigated since the 1950s [2] until nowadays [3]–[5].

Many practical problems are modeled based on two objective functions to form a *biobjective* (or *bicriteria*) AP (see for example [6], [7]). If the two functions are conflicting [3], this means that it cannot be found a solution that improves either one of the criteria, without making worse the other criterion. In this case we are interested to find the set of non-dominant solutions that are also called *Pareto optimal* [6], [8], [9].

[8] is a key publication that investigates the topic of Pareto optimal solutions in networks that have two objective functions defined: a linear, and a bottleneck objective function. The authors present their methods as more or less universal, by pointing out that they solve the shortest path problem, the AP, and the minimum spanning tree problem. The authors propose three algorithms in which an important stage is the sorting of the edges with respect to the bottleneck criterion.

The problem of biobjective and multiobjective optimization is a current topic of research in the last two decades. Particularly the problem of Pareto optimal solutions, performance of the algorithms that search for them, and the methods to evaluate their effectiveness, gain special attention [9]. In this work it is shown how the choice of test examples affect methods performance evaluation, which reinforces our belief that algorithms' complexity must be proven also analytically.

A whole category of methods that solve the basic AP adopts methods from the graph theory and is based on finding matchings in bipartite graphs. In the recent work [10] the authors give detailed description of the AP based on bipartite graphs. Something more, the authors focus particularly on the bottleneck AP, focusing on bottleneck assignment sensitivity.

A work that investigates multiobjective optimization problems is [11] which considers single linear objective function and l number of bottleneck objective functions. Even though the AP is not particularly examined, the authors claim that the AP can be represented as equivalent to a transportation problem with a single linear and two bottleneck criteria.

A biobjective AP is examined in [6] where both objective functions are from the bottleneck type. The presented approach looks for Pareto optimal solutions of the bicriteria AP of type MAXMIN-MINMAX. The method has computational complexity $O(n^4 \log n)$. The result of the algorithm is a subset of the non-dominating solutions.

Many of the existing contemporary efforts to solve effectively biobjective and multiobjective versions of the AP are adopting *heuristic approaches*. For example, in [7] are described four variants of the resource dependent AP in which the cost of a given assignment is the product of task cost parameter by a convex cost function of the agent. While the proposed solution in [7] is an approximation method, our goal is to find an exact algorithm that solves the biobjective AP.

This paper is a part of our effort to examine extensively the problem of biobjective AP in the case in which at least one of the two objective functions is from the bottleneck type. Here we focus on the version of the problem in which the first objective function is a linear one (MINSUM), while the second is a bottleneck function (MINMAX).

Following [12], we denote the examined biobjective AP with *MINSUM-MINMAX problem*. We propose an algorithm that solves the MINSUM-MINMAX problem and calculates the list of all Pareto optimal solutions that can be represented:

$$P = P_1 \cup P_2 \cup \ldots \cup P_m, \tag{1}$$

where each P_j, $j = 1, 2, \ldots, m$ is a subset of equivalent Pareto optimal solutions, and m is the number of such sets.

979-8-3503-1291-1/23/$31.00 ©2023 IEEE

This paper is organized as follows. In Sec. II we introduce the basic mathematical notations and definitions and we formulate the MINSUM-MINMAX problem. The algorithms are given in Sec. III, and also their correctness and computational complexity are proved. Sec. IV contains the conclusions.

II. Basic Notations and Problems Formulation

In this section we give the basic mathematical notations and we formulate the MINSUM-MINMAX problem.

A. Basic Notations

With $\mathbb{N}(n) = \{1, 2, \ldots, n\}$ we denote the set of first n natural numbers. For each vector w with $w(j)$ we denote its element on position j, and for each matrix A with $A(i, j)$ we denote its element located on row i and column j.

$[A]^{i_1,i_2,\ldots,i_k}_{j_1,j_2,\ldots,j_s}$ denotes the matrix that results from A after we delete the rows with indexes $i_1, i_2, \ldots, i_k$, and columns with indexes $j_1, j_2, \ldots, j_s$.

With $a^{(i)}$ we denote the value of a variable a, and $A^{(i)}$ to denote the value of a matrix A, that is calculated on the i-th iteration of a loop.

For each square matrix A of order n, and each vector (assignment) $w \in \mathbb{N}^k(n)$, $k \in \mathbb{N}(n)$, we define the two functions $F_A(w)$ and $G_A(w)$.

The first function $F_A(w)$ represents a linear function that is defined by the sum of the weighs as elements of A defined by the assignment w.

$$F_A(w) = \sum_{j=1}^{k} A(j, w(j)) \tag{2}$$

We assume that the function $F_A(w)$ represents a *cost criterion*.

The second (bottleneck) function $G_A(w)$ is defined as the maximum of the weight for a given assignment.

$$G_A(w) = \max_{j \in \mathbb{N}(k)} \{A(j, w(j))\} \tag{3}$$

The function $G_A(w)$ represents a *time criterion*.

One of the standard approaches to represent an AP is as a problem defied on a bipartite graph. With $G = (U, V, E)$ we denote a bipartite graph with vertices $U \cup V$ and a set of edges E. Since the objective of our research are the balanced APs, without any loss of generality we assume that $U = \mathbb{N}(n)$ and $V = \mathbb{N}(n)$. Then, the edge $\{i, j\}$ from E denotes that the i-th agent from U can be assigned on the j-th position from V. Each assignment is represented with a perfect matching in the bipartite graph G, and vice versa. We denote each perfect matching M with a special vector $w = (j_1, j_2, \ldots, j_n)$, where $\{i, j_i\} \in M$, for each $i \in \mathbb{N}(n)$. Therefore, the components of w are different natural numbers that are less than $n + 1$, in other words $w = (j_1, j_2, \ldots, j_n)$ *is an assignment exactly when it is a permutation of* $\mathbb{N}(n)$.

Definition 1: We will say that the 0-1 matrix A defines the set W of perfect matchings of the bipartite graph G with $w \in W$, if and only if w is a perfect matching of the bipartite graph with an adjacency matrix A.

With W_G we denote the set of all perfect matchings in the bipartite graph G. We assume that $G(U, V, E)$ is a complete bipartite graph that is given along with the two weights matrices A and T, which are defined by the objective functions.

B. MINSUM-MINMAX Problem

In the case of MINSUM-MINMAX problem, for each each $\{i, j\} \in E$ of the complete bipartite graph $G(U, V, E)$, $A(i, j) > 0$ is the *cost*, and $T(i, j) > 0$ is the *time* criterion. Then, the number $F_A(w)$ is the *cost*, and the number $G_T(w)$ is the *time* of the assignment w.

Definition 2: We will say that the assignment $\widehat{w}$ is Pareto optimal with respect to $\{F_A, G_T\}$, when there *does not exist* an assignment w, for which at least one of the following statements holds:

- $F_A(w) < F_A(\widehat{w})$ and $G_T(w) \leq G_T(\widehat{w})$;
- $F_A(w) \leq F_A(\widehat{w})$ and $G_T(w) < G_T(\widehat{w})$.

Definition 3: We will call two assignments w' and w'' equivalent, when $F_A(w') = F_A(w'')$ and $G_T(w') = G_T(w'')$.

The assignment w'' is dominated by the assignment w' when $F_A(w') < F_A(w'')$ and $G_T(w') \leq G_T(w'')$ or $F_A(w') \leq F_A(w'')$ and $G_T(w') < G_T(w'')$.

With P (given in (1)) we denote the set of all Pareto optimal assignments.

Definition 4: We will call the subset $P_0 \subseteq P$ a *minimal complete set (MCS)* of Pareto optimal solutions, when $P_0 = \{w_1, w_2, \ldots, w_m\}$, and $w_j \in P_j$ for each $j \in \mathbb{N}(m)$.

Problem 1 (MINSUM-MINMAX): Let p is an arbitrary natural number. Calculate a list

$$L = \{(Q_1, i_1, a_1, t_1), \ldots, (Q_m, i_m a_m, t_m)\}, \tag{4}$$

where for each $j \in \{1, 2, \ldots, m\}$ the following hold:

1) $Q_j \subseteq P_j$ and $|P_j| = i_j$.
2) If $|P_j| > p - 1$, then $|Q_j| = p$.
3) If $|P_j| < p + 1$, then $Q_j = P_j$.
4) For each $w \in P_j$ it is fulfilled $F_A(w) = a_j$ and $G_T(w) = t_j$.

Without loss of generality, we assume that the sets P_j are enumerated in such way, that $a_j \leq a_{j+1}$.

Corollary 1: Let us suppose that the list (4) is composed for $p = 1$. Then $P_0 = \bigcup_{j=1}^{m} Q_j$ is a (MCS).

III. MINSUM-MINMAX Algorithm

In this section we described the proposed algorithm that solves the MINSUM-MINMAX problem (Prob. 1). The algorithm 4) is based on a number of helper functions, which we describe preliminary.

We will use a function $DesSort(T)$ that calculates two vectors (lookup tables) u and q, that contain the information of the unique sorted elements of the matrix T. The table u contains the unique elements in decreasing order, and the table q contains the lists of the coordinates of the corresponding values from u in the matrix T.

Example 1: Given the input matrix T that is defined in (11), the function $DesSort(T)$ outputs the lookup tables u and q:

$$u = (11, 10, 9, 8, 7, 6, 5), q = (\{(3,2)\}, \{(1,3),(2,3),(3,4),(4,3)\}, \{(1,4),(2,2),(4,1)\}, \{(1,1),(2,4),(4,2),(4,4)\}, \{(1,2),(3,3)\}, \{(2,1)\}, \{(3,1)\}). \quad (5)$$

We define a function $Refresh(A, u, q, t)$ (Alg. 1) that takes as an input a matrix A, the two lookup tables u and q, and an arbitrary number t. The function modifies the matrix A:

$$A(i,j) = b, \text{ if } T(i,j) \geq t, i \in \mathbb{N}(n), j \in \mathbb{N}(n), \quad (6)$$

where b denotes a barrier large number. The two lookup tables u and q are also modified, so that the elements of A that have been marked with b in the previous calls of the function, are not processed again in the next calls.

Algorithm 1 $Refresh(A, u, q, t)$

Input: matrix A, tables u and q, number t
Output: modifies A, u and q

```
while u ≠ ∅ do
  if u(1) ≥ t then
    u ← u \ {u(1)}, v ← q(1), q ← q \ {q(1)}
    while v ≠ ∅ do
      w ← v(1), v ← v \ {v(1)}, A(w(1), w(2)) ← b
    end while
  else
    break
  end if
end while
```

Example 2: Let the weight matrices A and T are defined with (11). Then, the two lookup tables u and q, produced by the function $DesSort(T)$, are given in (5). Let $b = 34$ and $t = 10$, and we will trace the calculations of Alg. 1.

Since $u \neq \varnothing$, the first iteration of the outer **while** loop starts. The current value $u(1) = 11 \geq t = 10$, then from the tables u and q are removed the first elements, and because v stores the coordinates $(3,2)$, the algorithm sets $A(3,2) = 34$.

Again, $u \neq \varnothing$ and $u(1) \geq 10$. The second iteration of the outer loops starts, and it removes the first elements from both lookup tables. Since in this case v stores the list of coordinates $(1,3),(2,3),(3,4),(4,3)$, the inner loop executes four iterations which set $A(1,3) = A(2,3) = A(3,4) = A(4,3) = 34$.

Once again, $u \neq \varnothing$, and the outer loop enters the third iteration. Now $u(1) = 9 < t = 10$, the algorithm stops and:

$$A = \begin{pmatrix} 5 & 7 & 34 & 6 \\ 4 & 7 & 34 & 11 \\ 8 & 34 & 9 & 34 \\ 7 & 9 & 34 & 4 \end{pmatrix}, \; u = (9,8,7,6,5), \text{ and}$$

$$q = (((1,4),(2,2),(4,1)), ((1,1),(2,4),(4,2),(4,4)), ((1,2),(3,3)), ((2,1)), ((3,1))).$$

The helper function $TimeMin(T)$ calculates (w_0, t_0), where w_0 is an assignment that has the minimal time t_0. The algorithm that implements $TimeMin(T)$ is given in [13].

We denote with $Hungarian(Z)$ the function that implements the Hungarian method [2] with computational complexity $O(k^3)$. The function takes as an input an arbitrary square matrix Z of order k, and calculates the pair (a, w), where $a = \min_{w \in P^k} \{F_Z(w)\}$, $w \in P^k$, $F_Z(w) = a$, and P^k is the set of all permutations of $\mathbb{N}(n)$.

The Hungarian method is used as a stage in the following branching procedure, defined as a function $Branch(X, a, t)$ (see Alg. 2). The branching procedure is applied on the initial version of the Prob. 1, and after that on its sub-problems that are calculated by the branching. The initial problem is stored as a tuple of six elements

$$X = \{v, w, \bar{A}, T, n, \tau\}, \quad (7)$$

- $v = \varnothing$ is an empty vector;
- w is defined in the pair (a, w) that is calculated by the function $Hungarian(A)$;
- $\bar{A}$ is a matrix with $(n+1)$ rows and n columns, where

$$\overline{A}(i,j) = \begin{cases} A(i,j), & \text{if } i \in \mathbb{N}(n) \text{ and } j \in \mathbb{N}(n), \\ j, & \text{if } i = n+1 \text{ and } j \in \mathbb{N}(n); \end{cases} \quad (8)$$

- $\tau = \tau(v, T)$ is an evaluation, such that for each assignment $\widetilde{w}$ it holds that $G_T(\widetilde{w}) \geq \tau$.

The function $Branch(X, a, t)$ modifies the current record t and composes a list of selected sub-problems $X^{(s)}$, which are stored in the form $\{v^{(s)}, w^{(s)}, A^{(s)}, T^{(s)}, n-1, \tau^{(s)}\}$. In the general case, for a given sub-problem $X = \{v, w, \widetilde{A}, \widetilde{T}, k, \tau\}$ that is a result of a call to the branching procedure, the following hold:

- $v \in \mathbb{N}^{n-k}(n)$ is a variation without repetition of class $(n-k)$ of n elements;
- $w \in \mathbb{N}^n(n)$ is a permutation of $\mathbb{N}(n)$ that contains such an assignment, that $F_A(w) = a$, and $w(i) = v(i), \forall i \in \mathbb{N}(n-k)$;
- $\widetilde{A}$ is a $(k+1) \times k, \forall k \in \{2, \ldots, n\}$ matrix;
- $\widetilde{T}$ is a $k \times k, \forall k \in \{2, \ldots, n\}$ matrix;
- $\tau = \tau(v, \widetilde{T})$, where

$$\tau(v, \widetilde{T}) = \max\Big\{T(1, v(1)), \ldots, T(n-k, v(n-k)), \max_{i \in \mathbb{N}(k)} \{\min_{j \in \mathbb{N}(k)} \{\widetilde{T}(i,j)\}\}, \max_{j \in \mathbb{N}(k)} \{\min_{i \in \mathbb{N}(k)} \{\widetilde{T}(i,j)\}\}\Big\}.$$

The computational complexity of the Alg. 2 is determined by the computational complexity of the function $Hungarian(Z)$. During the branching process of a problem $X = \{v, w, \widetilde{A}, \widetilde{T}, k, \tau\}$ the function $Hungarian(Z)$ is applied k number of times with input matrices of order $(k-1)$. Therefore, the complexity of $Branch(X, a, t)$ is $O(k(k-1)^3)$.

We assume that the graph G is given with the two weight matrices A and T (see Sec. II-B). We define the following two subsets of the set of all assignments W_G in the graph G.

- The set of all assignments with minimal cost:

$$\overline{W} = \Big\{\overline{w} \in W_G : F_A(\overline{w}) = \min_{w \in W_G} \{F_A(w)\}\Big\}. \quad (9)$$

Algorithm 2 The branching procedure $Branch(X, a, t)$

Input: $X = \{v, w, \widetilde{A}, \widetilde{T}, k, \tau\}$, a and t
Output: The list S and updated record t

```
S ← ∅
for i ← 1 to k do
  v_i ← v ∪ (Ã(k+1, i)), A_i ← [Ã]_i^1
  (ā, w̄) ← Hungarian([A_i]^k)
  if a = F_A(v_i) + ā then
    w_i ← v_i ∪ (A_i(k, w̄(1)), ..., A_i(k, w̄(k−1)))
    T_i ← [T̃]_i^1, τ ← τ(v_i, T_i)
    S ← S ∪ {v_i, w_i, A_i, T_i, k−1, τ}
    if t > G_T(w_i) then
      t ← G_T(w_i)
    end if
  end if
end for
```

- The set of all assignments with minimal cost, that have the minimal possible time:

$$\widehat{W} = \left\{ \widehat{w} \in \overline{W} : G_T(\widehat{w}) = \min_{\overline{w} \in \overline{W}} \{G_T(\overline{w})\} \right\}. \quad (10)$$

We define a function $MinCostMinTime(A, T, p)$, which takes as parameters the matrices A and T, and an arbitrary natural number p. The function calculates the tuple $\{Q, i, a, t\}$:

- $Q \subseteq \widehat{W}$;
- $i = |\widehat{W}|$, $a = \min_{w \in W_G} \{F_A(w)\}$ and $t = \min_{\overline{w} \in \overline{W}} \{G_T(\overline{w})\}$;
- if $|\widehat{W}| > p - 1$, then $|Q| = p$;
- if $|\widehat{W}| < p + 1$, then $Q = \widehat{W}$.

It is directly verified that $\widehat{W}$ is a set of equivalent Pareto optimal solutions.

The implementation of $MinCostMinTime(A, T, p)$ (Alg. 3) traverses only the assignments with minimal cost, and separates those of them that have the minimal possible time. The traversal is performed by the function $Branch(X, a, t)$.

Alg. 3 is illustrated by the following Examp. 3.

Example 3: Let the weight matrices A and T be defined with (11) below. The function $MinCostMinTime(A, T, p)$ performs the following calculations.

$$A = \begin{pmatrix} 5 & 7 & 2 & 6 \\ 4 & 7 & 13 & 11 \\ 8 & 11 & 9 & 6 \\ 7 & 9 & 8 & 4 \end{pmatrix} T = \begin{pmatrix} 8 & 7 & 10 & 9 \\ 6 & 9 & 10 & 8 \\ 5 & 11 & 7 & 10 \\ 9 & 8 & 10 & 8 \end{pmatrix} \quad (11)$$

Before the **while** loop, the algorithm evaluates that $a = 21$, $t = 10$, and the tuple that stores the problem X is initialized with $X^{(0)}$, given on the first row of Tab. I.

On the *first iteration* of the **while** loop the algorithm performs the branching $Branch(X, 21, 10)$ which results in $(S_1^{(1)}, t^{(1)})$ with $t^{(1)} = 10$, and $S_1^{(1)} = (X^{(1)})$ (see second row of Tab. I).

On the *second iteration* of the **while** loop the branching procedure returns $(S_1^{(2)}, t^{(2)})$ with $t^{(2)} = 10$, $S_1^{(2)} = (X^{(21)}, X^{(22)})$ (see the third and fourth row of Tab. I). Therefore, after the second iteration S contains two sub-problems. As a result, the next two iterations of the **while** loop will execute the inner **for** loop. After the fourth iteration of the **while** loop, the algorithm stops and the result is:

$$Q = ((3, 2, 1, 4), (3, 1, 4, 2)), i = 2, a = 21, t = 10. \quad (12)$$

The correctness of Alg. 3 follows from the correctness of the functions $Hungarian(Z)$ and $Branch(X, a, t)$, and also from the fact that the main loop of the algorithm stops after a finite number of iterations. It is enough to note that the problems, that are included in S for further processing, define different assignments with minimal cost. Also, each iteration of the loop substitutes a problem from S with a finite number of sub-problems that have less number of free variables. Therefore, it exists such a constant $C^\star$, that is determined by the number of assignments with minimal cost, such that the complexity of the $MinCostMinTime(A, T, p)$ is evaluated to $C^\star O(n^5)$.

Based on the above helper functions, we define the Alg. 4 that solves the MINSUM-MINMAX problem (Prob. 1).

We will illustrate the Alg. 4 with the following Examp. 4.

Example 4: We will solve Prob. 1 using Alg. 4 when the weight matrices A and T are defined with (11) and $p = 4$.

Algorithm 3 $MinCostMinTime(A, T, p)$

Input: Weight matrices A and T, and $p \in \mathbb{N}$
Output: $\{Q, i, a, t\}$

```
(a, w) ← Hungarian(A), t ← G_T(w), i ← 0
Q = ∅ {Q is a queue}
X ← {∅, w, Ā, T, n, τ} {initial problem for branching}
S = ∅, push(S, X) {S is a stack}
while S ≠ ∅ do
  X ← top(S), pop(S) {X = (v, w, Ā, T, k, τ)}
  if τ ≤ t then
    if k > 2 then
      (S_1, t) ← Branch(X, a, t), S ← S ∪ S_1
    else
      for j ← 1 to 2 do
        a_1 ← F_A(w), t_1 ← G_T(w)
        if a = a_1 and t_1 < t then
          t ← t_1, Q ← {w}, i ← 1
        else
          if a = a_1 and t_1 = t then
            if i < p then
              enqueue(Q, w)
            end if
            i ← i + 1
          end if
        end if
        x ← w(n−1), w(n−1) ← w(n), w(n) ← x
      end for
    end if
  end if
end while
```

TABLE I
VALUES OF THE SUB-PROBLEMS X THAT RESULT FROM BRANCHING PROCEDURE IN ALG. 3

X	v	w	$\overline{A}$	T	k	τ
$X^{(0)}$	$\varnothing$	$(3,2,1,4)$	$\begin{pmatrix} 5 & 7 & 2 & 6 \\ 4 & 7 & 13 & 11 \\ 8 & 11 & 9 & 6 \\ 7 & 9 & 8 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}$	$\begin{pmatrix} 8 & 7 & 10 & 9 \\ 6 & 9 & 10 & 8 \\ 5 & 11 & 7 & 10 \\ 9 & 8 & 10 & 8 \end{pmatrix}$	4	8
$X^{(1)}$	(3)	$(3,2,1,4)$	$\begin{pmatrix} 4 & 7 & 11 \\ 8 & 11 & 6 \\ 7 & 9 & 4 \\ 1 & 2 & 4 \end{pmatrix}$	$\begin{pmatrix} 6 & 9 & 8 \\ 5 & 11 & 10 \\ 9 & 8 & 8 \end{pmatrix}$	3	10
$X^{(21)}$	$(3,2)$	$(3,2,1,4)$	$\begin{pmatrix} 8 & 6 \\ 7 & 4 \\ 1 & 4 \end{pmatrix}$	$\begin{pmatrix} 5 & 10 \\ 9 & 8 \end{pmatrix}$	2	10
$X^{(22)}$	$(3,1)$	$(3,1,4,2)$	$\begin{pmatrix} 11 & 6 \\ 9 & 4 \\ 2 & 4 \end{pmatrix}$	$\begin{pmatrix} 11 & 10 \\ 8 & 8 \end{pmatrix}$	2	10

Algorithm 4 Solution of the MINSUM-MINMAX problem

Input: Weight matrices A and T, and $p \in \mathbb{N}$
Output: The list L defined in (4)

```
t ← ∞, a ← 0, L ← ∅
(w_0, t_0) ← TimeMin(T)
b ← F_A(w_0) {b is barrier large number}
(u, q) ← DesSort(T)
while t_0 < t do
  (Q, i, a, t) ← MinCostMinTime(A, T, p)
  L ← L ∪ {(Q, i, a, t)}
  if t_0 < t then
    (A, m, q) ← Refresh(A, m, q, t)
  end if
end while
```

After the initialization of the variables, the algorithm calculates (w_0, t_0), where $t_0 = 8$ and $w_0 = (1, 4, 3, 2)$.

The next step of the algorithm is to select a barrier large number b. If $W' = \{w' \in W_G : G_T(w') = t_0\}$, then for

$$a' = \min_{w \in W'} \{F_A(w)\}, \tag{13}$$

it holds that $F_G(w_0) \geq a'$. This allows to select $b = F_G(w_0)$ as a barrier large number. In particular, $b = F_G(w_0) = 34$.

The function $DesSort(T)$ calculates the lookup tables of the unique sorted values u and the corresponding coordinates in the matrix T (see Examp. 1).

Since $t = \infty$, the algorithm enters the first iteration of the **while** loop. The calculations of $MinCostMinTime(A, T, 4)$ are given in details in Examp. 3 with result (Q, i, a, t) given in (12). (Q, i, a, t) is included as the first element in the list L. By the definition of Alg. 3 it follows that $Q \subseteq \widehat{W}$ and hence, Q is a set of Pareto equivalent optimal solutions.

The function $Refresh(A, u, q, t)$ edits A, u and q, because $t_0 = 8 < 10 = t$. The calculations are performed with Alg. 1, and are given in details in Examp. 2.

Since the current value of t is 10, the algorithm will start the second iteration of the **while** loop. For the current value of the cost matrix A the function $MinCostMinTime(A, T, 4)$ calculates (Q, i, a, t), where $Q = \{(2, 1, 3, 4)\}$, $i = 1$, $a = 24$ and $t = 8$. Since $t = 8$, then $t = t_0$, and the following holds.

1) Q is the set of equivalent Pareto optimal solutions

$$Q = \left\{ w : G_T(w) = t_0, F_A(w) = \min_{w \in W'} \{F_A(w)\} \right\},$$

where $W' = \{w' \in W_G : G_T(w') = t_0\}$. Particularly, for a' from (13) it is fulfilled $a' = 24$.

2) This is the last iteration, and the algorithms stops.

The result is the following list:

$$L = ((Q_1, 2, 21, 10), (Q_2, 1, 24, 8)), \tag{14}$$

where $Q_1 = \{w_1, w_2\}$ with $w_1 = (3, 2, 1, 4)$, $w_2 = (3, 1, 4, 2)$, and $Q_2 = \{w_3\}$ with $w_3 = (2, 1, 3, 4)$.

The list L gives full information about the Pareto optimal solutions of the Prob. 1. We have verified that $|P_1| = i_1 = 2$ and $|P_2| = i_2 = 1$ (see (14)). Since the calculations are performed for $p = 4$, this means that $P_1 = Q_1 = \{w_1, w_2\}$ and $P_2 = Q_2 = \{w_3\}$. Therefore, the equation (1) has the form $P = Q_1 \cup Q_2 = \{w_1, w_2, w_3\}$.

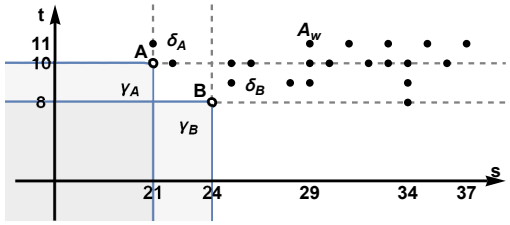

Fig. 1. Plot of the solution of Examp. 4 with cost criterion s as abscissa, and time criterion t as ordinate. Black points denote assignments, while white points denote Pareto optimal solutions

Fig. 1 illustrates the result in the Examp. 4. This time the abscissa represents the cost criteria, and is denoted by s, while the ordinate represents the time criteria t. Each assignment is represented with a point A_w with Cartesian coordinates $(F_A(w), G_T(w))$. If the assignment $w \neq w_j$, $j \in \{1, 2, 3\}$, then the point is depicted as a solid black dot. The points $A(21, 10)$ and $B(24, 8)$ are plotted as white

dots. The point $A(21, 10)$ corresponds to the equivalent Pareto optimal solutions w_1 and w_2. The point $B(24, 8)$ corresponds to the single assignment from the class Q_2.

Both angles $\gamma_A = \{(s,t) : s \leq 21 \text{ and } t \leq 10\}$, and $\gamma_B = \{(s,t) : s \leq 24 \text{ and } t \leq 8\}$ are colored in gray. On Fig. 1 it is clear that there is not such point A_w colored in black, that is contained in the set $F = \gamma_A \cup \gamma_B$. Besides that, $A \notin \gamma_B$ and $B \notin \gamma_A$, which shows that for the assignments w_1, w_2 and w_3 is fulfilled Def. 2.

The rays of the angles $\delta_A = \{(s,t) : s \geq 21 \text{ and } t \geq 10\}$, and $\delta_B = \{(s,t) : s \geq 24 \text{ and } t \geq 8\}$ are plotted in dashed lines. Each assignment w for which A_w falls within δ_A is dominated by w_1, and each assignment w for which A_w falls within δ_B is dominated by w_3. Fig. 1 clearly shows that each assignment $w \notin Q_1 \cup Q_2$ is dominated either by w_1, or by w_3. Hence, all Pareto optimal solutions are contained in $Q_1 \cup Q_2$.

Theorem 1: Alg. 4 is correct.

Proof: The proof uses mathematical induction and the fact that the functions $TimeMin(A, T, a)$, $DesSort(T)$, $Refresh(A, u, q, t)$, and $MinCostMinTime(A, T, p)$ are proved to be correct. We prove that each iteration of the **while** loop correctly calculates each element of the list L.

Before the loop, the algorithm defines $t = \infty$, $a = 0$, $L = \varnothing$. It evaluates (T_0, w_0, t_0) with $TimeMin(A, T, a)$, b with $F_A(w_0)$, and (u, q) with $DesSort(T)$.

Base case. We prove the correctness of the first iteration of the loop. The correctness of $MinCostMinTime(A, T, p)$ proves that the element $(Q^{(1)}, i^{(1)}, a^{(1)}, t^{(1)})$ is the member of the list L that we look for. Also, we prove that $P_1 = \left\{w \in W_G : F_A(w) = a^{(1)} \text{ and } G_T(w) = t^{(1)}\right\}$ is the first set of Pareto optimal solutions in (1) and $|P_1| = i^{(1)}$.

If the condition $t_0 < t^{(1)}$ is fulfilled, $(A^{(1)}, u^{(1)}, q^{(1)})$ is evaluated using the call to $Refresh(A, u, q, t^{(1)})$, and the **while** loop proceeds to its second iteration.

Inductive step. We assume that the first k iterations have correctly calculated the first k elements of the list L, $t_0 < t^{(k)}$, and as a result the algorithm has evaluated $(A^{(k)}, u^{(k)}, q^{(k)})$.

During the $(k+1)$-st iteration, the algorithm calculates $(Q^{(k+1)}, i^{(k+1)}, a^{(k+1)}, t^{(k+1)})$, and defines

$$P_{k+1} = \left\{w \in W_G : F_A(w) = a^{(k+1)} \text{ and } G_T(w) = t^{(k+1)}\right\}.$$

We prove that the above P_{k+1} is the $(k+1)$-st Pareto optimal assignment that also takes place in (1), and that $(Q^{(k+1)}, i^{(k+1)}, a^{(k+1)}, t^{(k+1)})$ is the correctly calculated $(k+1)$-st element of the list L.

The **while** loop stops after no more than n^2 number of iterations, because for each $t^{(k)}$ there exists an element $T(i, j)$ in the matrix T, for which $t^{(k)} = T(i, j)$. ∎

Proposition 1: The computational complexity of the Alg. 4 is $C^\star O(n^7)$, where the constant $C^\star$ is determined by the computational complexity of $MinCostMinTime(A, T, p)$.

Proof: Indeed, the complexity of each of the functions $TimeMin(A, T, a)$, $DesSort(T)$ and $Refresh(A, u, q, t)$ do not exceed the complexity of $MinCostMinTime(A, T, p)$. Therefore, the k-th iteration of the **while** loop has complexity $C_k^\star O(n^5)$, where $C_k^\star$ is determined by the complexity of $MinCostMinTime(A, T, p)$, and depends on the number of assignments with minimal cost, with the cost matrix $A^{(k)}$. We denote with $C^\star = \max\{C_1^\star, C_2^\star, \ldots, C_m^\star\}$, and the resulting complexity of Alg. 4 is $C^\star O(n^7)$. ∎

IV. Conclusion

The algorithm that we present in this paper finds the complete description of the set P of all Pareto optimal solutions of the biobjective AP with one linear and one bottleneck objective functions. For each set P_i of equivalent Pareto optimal solutions the algorithm finds the exact number of elements, and separates a subset Q_i with a predefined bound of the number of elements. The selection of a single element from each P_i will result in the MSC of Pareto optimal solutions. The correctness of Alg. 4 is proved, and its computational complexity is evaluated. Besides the provided analytical proof, the graphical representation in Fig. 1 also gives a geometrical proof that the problem is solved correctly.

The function $MinCostMinTime(A, T, p)$ (Alg. 3) solves the problem for finding of the set of all assignments with minimal cost, that are realized for minimal possible time. It is clear that a particular case of the latter problem, is the problem for finding all assignments with minimal cost.

References

[1] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment problems*, ser. SIAM. University City, Philadelphia: Society for Industrial and Applied Mathematics, 2009.

[2] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1–2, pp. 83–97, March 1955.

[3] T. Oncan, Z. Şuvak, M. H. Akyüz, and I. K. Altınel, "Assignment problem with conflicts," *Computers & Operations Research*, vol. 111, pp. 214–229, 2019.

[4] S. Dhouib, "An intelligent assignment problem using novel heuristic: The dhouib-matrix-ap1 (dm-ap1): Novel method for assignment problem," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 10, no. 1, p. 135–141, 2022.

[5] K. Morita, S. Shiroshita, Y. Yamaguchi, and Y. Yokoi, "Fast primal-dual update against local weight update in linear assignment problem and its application," *Information Processing Letters*, vol. 183, p. 106432, 2024.

[6] Y. Ge, M. Chen, and H. Ishii, "Bi-criteria bottleneck assignment problem," in *2012 Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS)*, 2012, pp. 1–5.

[7] L. Yedidsion and D. Shabtay, "The resource dependent assignment problem with a convex agent cost function," *European Journal of Operational Research*, vol. 261, no. 2, pp. 486–502, 2019.

[8] O. Berman, D. Einav, and G. Handler, "The constrained bottleneck problem in networks," *Operations Research*, vol. 38, no. 1, pp. 178–181, 1990.

[9] H. Ishibuchi, L. He, and K. Shang, "Regular Pareto front shape is not realistic," in *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2019, pp. 2034–2041.

[10] E. Michael, T. A. Wood, C. Manzie, and I. Shames, "Sensitivity analysis for bottleneck assignment problems," *European Journal of Operational Research*, vol. 303, no. 1, pp. 159–167, 2022.

[11] C. T. Bornstein, N. Maculan, M. Pascoal, and L. L. Pinto, "Multiobjective combinatorial optimization problems with a cost and several bottleneck objective functions: An algorithm with reoptimization," *Computers & Operations Research*, vol. 39, no. 9, pp. 1969–1976, 2012.

[12] P. Hansen, "Bicriterion path problems," *Multiple Criteria Decision Making Theory and Application*, pp. 109–127, 1980.

[13] L. M. Laskov and M. L. Marinov, "List of selected number of optimal solutions of the assignment problem by time criterion," in *2022 International Conference Automatics and Informatics (ICAI)*. IEEE, 2022, pp. 100–106.