Proceedings of the of the 18th Conference on Computer Science and Intelligence Systems pp. 603–613
DOI: 10.15439/2023F3718
ISSN 2300-5963 ACSIS, Vol. 35

List Of Pareto Optimal Solutions of a Biobjective Shortest Path Problem

Lasko M. Laskov
0000-0003-1833-818
Informatics Department
New Bulgarian University
21 Montevideo Str., 1618 Sofia, Bulgaria
Email: llaskov@nbu.bg

Marin L. Marinov
0009-0003-9544-819X
Informatics Department
New Bulgarian University
21 Montevideo Str., 1618 Sofia, Bulgaria
Email: mlmarinov@nbu.bg

***Abstract*—Many applications in practice involve the search for a shortest path in a network by optimizing two conflicting objective functions. Such problems often are referred to as biobjective optimization problems. Their goal is to find special optimal paths that are *nondominated* and are also known in the specialized literature as to as *Pareto optimal*. While most of the existing methods aim to find the minimum complete set of Pareto optimal paths, we propose an approach that is able to generate a list of all Pareto optimal solutions in a given network.**

The described method solves the biobjective optimization problem in the case in which the first objective function is a linear (MINSUM), while the second objective function is from the "bottleneck" type (MAXMIN). The presented approach is based on two modifications of the Dijkstra's shortest path algorithm that solve the MINSUM and the MAXMIN problems respectively.

We prove the correctness and the computational complexity of the presented algorithms. Also, we provide detailed numerical examples that illustrate their execution.

I. Introduction

PARETO optimal solutions of biobjective (bicriterion) optimization problems are a subject of extensive research in combinatorial optimization and operation research disciplines, and in particular the biobjective shortest path problems [1]. These special type of shortest path problems can arise in numerous applications in practice including transportation problems, computer networking, robot motion, and many others.

The search of biobjective Pareto optimal solutions in a shortest path problem is an optimization problem that is a subject of two objective functions, lets say f and g. The Pareto optimal paths (also called nondominated) are a set of paths, such that for any path α in it, it is impossible to improve either f or g criterion, without getting worse the other [2].

By determining the objective functions f and g, different types of bicriterion path problems can be defined. The first notable work that examines these types of problems is [3] in which Hansen defines ten types of bicriterion path problems, and also introduces their abbreviations. In particular, the MINSUM-MAXMIN problem is solved with an algorithm with a polynomial complexity $O(m^2 \log n)$, where n is the number of vertices of the network, and m is the number of directed edges. In the MINSUM-MAXMIN problem the first objective function is a liner one, while the second objective function is from the *bottleneck* type.

In the literature actually there are quite few works that focus on the solution of the MINSUM-MAXMIN problem. One of them is [4] which proposes an extension of the Martin's algorithm [5] for a multiobjective shortest path problem with a MAXMIN objective function. Most of the methods that can be found in the literature focus on the combination of two linear functions, for example [2], [6], [7], [8] solve the MINSUM-MINSUM bicriteria path problem. Other works focus on MINMAX-MINSUM problem (see [9] and [10]), however in their case the authors do not consider the Pareto optimality, rather they aim to define a singe objective function by combining the MINMAX and MINSUM criteria.

Another subject that is rarely considered in the literature is the calculation of all Pareto optimal paths from a source vertex v_0 and a destination (target, terminal) vertex v_t. Most of the existing algorithms aim to find the minimal complete set of Pareto optimal paths, which means that from each equivalent set of Pareto optimal paths a single path is discovered (see [3]). In [11] the authors look for a set of alternative Pareto optimal paths in a method that solves a concrete practical problem for routing of Hazardous materials and propose shortest path algorithm on a network with two criteria: one that corresponds to road length, and the other that corresponds to a risk measure.

The work [2] is one of the few in the literature that pays special attention to the calculation of all Pareto optimal paths. The authors propose two algorithms depending whether the paths may contain or may not contain loops, both of them based on k shortest paths algorithms in graphs. However, as mentioned above, in this work the two objective functions are linear, and the algorithms do not cover the case in which one of the functions is a bottleneck function.

The exact methods that are present in the literature, are generally classified into *labeling* and *ranking paths* algorithms [12]. Labeling algorithms can be split into two categories: *label setting* [3], [5], [8]; and *label correcting* [6], [7], [11]. In the category *ranking paths* we can classify methods that are based on the k shortest paths algorithms, for example [2].

The other major branch of methods are based on heuristics approaches. For example, in [13] the authors propose a so-

lution to the biobjective shortest path problem that is based on a genetic algorithm for which the authors report to find the Pareto optimal set in 77% of the instances. Also, the probabilistic technique ant colony optimization (ACO) and its modifications are adopted in the solution of various complex combinatorial problems (see for example [14]). Heuristic methods often are adopted in various practical problems, like electric vehicle shortest path problem [15].

In this paper we propose an exact method for calculation of all Pareto optimal paths for the MINSUM-MAXMIN problem in a network. We define two helper problems, MINSUM list and MAXMIN list, and we provide two algorithms that solve them, which are based on generalization of the Dijkstra's algorithm [16]. We use the solutions of the two helper problems to formulate the method for general problem solution. The correctness of all algorithms is proved, and their computational complexity is shown. Also, we illustrate the algorithms with detailed examples that show their execution.

The paper is organized as follows. In Sec. II we introduce the notations and problems formulation. In Sec. III we describe the two algorithms that solve the two helper problems. In Sec. IV we show how the Pareto optimal solutions list is constructed based on the solution of the two helper problems. Finally, Sec. V contains conclusions and discussions.

II. Problem formulation

A. Notations

Let $G = (V, E)$ is a directed graph (digraph) with $n = |V|$ number of vertices and $m = |E|$ number of directed edges. Without loss of generality we will assume that $V = \{1, 2, \ldots, n\}$ and $E \subseteq V^2$.

We define the following two functions on the set of edges of the digraph, $f : E \to \mathbb{R}_+$ and $g : E \to \overline{\mathbb{R}}_+$. The function f assigns to each edge $(i, j) \in E$ the positive number $f(i, j)$, which we call the *length* of the edge $e = (i, j)$. The function g assigns to each edge $(i, j) \in E$:

$$g(i,j) = \begin{cases} +\infty, & \text{if} \quad (i,j) \in E \text{ has no restriction} \\ g_{ij} > 0, & \text{if} \quad (i,j) \in E \text{ has a restriction} \end{cases}$$

For convenience, we will call the value $g(i, j)$ the *capacity* of the edge $e = (i, j)$.

The digraph G together with the functions f and g defines the network $G = (V, E, f, g)$ (see [17]). The network is represented by the adjacency list of the outgoing neighbors [18] that is augmented with the length and capacity of the edges in the following way:

$$Adj = \{Adj(1), \ldots, Adj(i), \ldots, Adj(n)\}, \tag{1}$$

where $Adj(i) = \{(j, f(i, j), g(i, j)) : (i, j) \in E\}, \forall i \in V$. In this way, if $q = Adj(i, k)$ for some $i \in V$, and a positive integer k, $q(1)$ will denote the k-th outgoing neighbor of the vertex i, $q(2)$ will denote the length of the edge $(i, q(1))$, and $q(3)$ will denote the capacity of the edge. To denote the adjacency list of outgoing neighbors of a given network G, we will use the notation $G.Adj$.

Path in the network G is the finite sequence of the type

$$v_0, e_1, v_1, e_2, \ldots, v_{(t-1)}, e_t, v_t, \tag{2}$$

where $v_j \in V, \forall j \in \{0, 1, \ldots, t\}$ are distinct vertices, and e_i is an edge with starting vertex $v_{(i-1)}$ and ending vertex v_i, that belongs to E for all $i \in \{1, 2, \ldots, t\}$. The path consists of $(t + 1)$ vertices and t edges, the vertex v_0 is the *source* of the path, and the vertex v_t is the *destination* of the path.

The path (2) with a source v_0 and a destination v_t connects v_0 with v_t and is called a (v_1, v_t)-path, which we will denote with an ordered sequence α of vertices:

$$\alpha = (v_0, v_1, \ldots, v_t). \tag{3}$$

For each path $\alpha = (v_0, v_1, \ldots, v_t)$ we define two functions:

$$x(\alpha) = \sum_{j=1}^{t} f(v_{j-1}, v_j) \tag{4}$$

$$y(\alpha) = \min_{j \in \{1,\ldots,t\}} \{g(v_{j-1}, v_j)\} \tag{5}$$

We will call the number $x(\alpha)$ the *length* of the path, and the number $y(\alpha)$ the *capacity* of the path α. Both functions $x(\alpha)$ and $y(\alpha)$ define the objective functions of the problems that we will discuss.

We denote all $(1, j)$-paths with the shorter W_j. Then, we will call the number

$$r_j = \min_{\alpha \in W_j} \{x(\alpha)\} \tag{6}$$

a *distance* between the vertex 1 and the vertex j. Also, we will call the number

$$c_j = \max_{\alpha \in W_j} \{y(\alpha)\} \tag{7}$$

the *capacity* of the vertex j.

For each W_j the term *Pareto optimal path* is defined as follows.

Definition 1. *We call the path $\alpha \in W_j$ **Pareto optimal** when there does not exist another path $\beta \in W_j$, for which any of the following two conditions is fulfilled:*

- $x(\beta) < x(\alpha)$ *and* $y(\beta) \geq y(\alpha)$*;*
- $x(\beta) \leq x(\alpha)$ *and* $y(\beta) > y(\alpha)$.

We say that α and β are *equivalent* ($\alpha \sim \beta$), when $x(\alpha) = x(\beta)$ and $y(\alpha) = y(\beta)$.

The path β is *dominated* by the path α, when $x(\alpha) < x(\beta)$ and $y(\alpha) \geq y(\beta)$ or $x(\alpha) \leq x(\beta)$ and $y(\alpha) > y(\beta)$.

Besides that, we will denote the distance from vertex 1 to any vertex v with $r(v)$.

B. Problems formulation

Based on the above definitions, we formulate the main problem considered in this paper:

Problem 1 (List of Pareto optimal solutions)**.** *Compute a list of all Pareto optimal solutions for W_n.*

To solve the List of Pareto optimal solutions problem, we will use the solutions of the following two helper problems.

The solution of the first helper problem requires the definition of a function $minsum(G.Adj)$ that computes a *list of all shortest paths* in the network.

Problem 2 (MINSUM list)**.** *Compute a list of all* $(1, n)$*-paths with minimal length, given by:*

$$S_x = \{\alpha \in W_n : x(\alpha) \leq x(\beta), \forall \beta \in W_n\}. \tag{8}$$

The solution of the second helper problem requires the definition of a function $maxmin(G.Adj)$, and it is a version of the first helper problem that computes a *list of all maximum capacity paths* in the network.

Problem 3 (MAXMIN list)**.** *Compute a list of all* $(1, n)$*-paths with maximal capacity, given by:*

$$S_y = \{\alpha \in W_n : y(\alpha) \geq y(\beta), \forall \beta \in W_n\}. \tag{9}$$

III. Solution of the two helper problems

To solve the two helper problems we propose two modifications of the Dijkstra's algorithm [16], in which the results hold as well in the case in which the source vertex is selected $i_0 \neq 1$. In both modifications we assume that the network $G = (V, E, f, g)$ is defined using the adjacency list of the outgoing neighbors.

In the computer program implementation of the modified versions of Dijkstra's algorithm we apply the Fibonacci heap data structure [19] for all priority queue operations. Even though the relative complexity of its implementation, this advanced data structure introduces a significant speedup of the algorithm to $O(n \log n + m)$, which is proved based on amortized analysis [18].

A. List of all shortest paths

We will solve the problem of computing of a list of all $(1, n)$- shortest paths by finding the subnetwork $\widehat{G} = (V, \widehat{E}, f, g)$ of the shortest paths.

Definition 2. *We will say that* $\widehat{G} = (V, \widehat{E}, f, g)$ *is a* ***subnetwork of the shortest paths*** *in the network* $G = (V, E, f, g)$*, if the following two properties hold:*

1) *Every* $(1, n)$*- shortest path in* G *is also a* $(1, n)$*-path in* $\widehat{G}$*.*
2) *Every* $(1, n)$*-path in* $\widehat{G}$ *is a* $(1, n)$*- shortest path in* G*.*

The solution of the MINSUM list helper problem is given by the definition of a function $minsum(G.Adj)$ (Alg. 2), which for a given network G calculates the adjacency list of the outgoing neighbors of the shortest paths subnetwork $\widehat{G}$. The implementation of the $minsum(G.Adj)$ function is based on a modification of the Dijkstra's algorithm [16], as follows.

The algorithm splits the set of network vertices in into subsets. The first subset V_0 denotes the vertices that are not yet traversed by the algorithm. The second subset $U = V \setminus V_0$ stores the traversed vertices. The procedure that traverses the network guarantees that

$$r(v) \geq r(u), \tag{10}$$

for each vertex $v \in V_0$ and each vertex $u \in U$. Initially, $V_0 = V$ and $U = \varnothing$. In each of n consecutive iterations of execution the function $minsum(G.Adj)$ a selected vertex is transferred from V_0 into U.

The algorithm uses two vectors d and pr, both of them with n components. Initially, $d(1) = 0$, and all other components of d are ∞. The initial values of pr are equal to the empty set $\varnothing$. After the completion of the algorithm, d will store the distances from the source vertex to each of the other vertices in the network, in other words $d(j) = r(j), \forall j \in V$, and pr will store the adjacency list of the ingoing neighbors of the digraph $(V, \widehat{E})$. The last step of the $minsum(G.Adj)$ function composes the adjacency list of the outgoing neighbors of the shortest paths subnetwork $\widehat{G}$.

In the proposed variant of the Dijkstra's algorithm that solves Prob. 2, the function that implements the relaxation procedure is modified. We define the function $relaxS(u, v, d, pr)$ that performs relaxation of the edge (u, v) by changing the current state of d and pr, as it is given in Alg. 1.

Algorithm 1 Function $relaxS(u, v, d, pr)$

Input: vertices u, v, and vectors d, pr
Output: vectors d, pr

```
    q ∈ G.Adj(u), such that q(1) = v
2:  r ← d(u) + q(2)
    if d(v) > r then
4:      d(v) ← r
        pr(v) ← {u}
6:  else if d(v) = r then
        pushback(pr(v), u)
8:  end if
    return {d, pr}
```

We use two more helper functions in our modified Dijkstra's implementation of the $minsum(G.Adj)$ procedure: $extract(V_0)$ and $outadj(pr, G.Adj)$.

The input of $extract(V_0)$ is the subset $V_0 \subset V$, and the output is $\{v_1, V_1\}$, where $v_1 \in V_0$, $V_1 = V_0 \setminus \{v_1\}$ and $d(v_1) = \min\limits_{v \in V_0}\{d(v)\}$.

The purpose of the $outadj(pr, G.Adj)$ function is to build the adjacency list of the outgoing neighbors of the network $\widehat{G}$ out of the digraph presented by the adjacency list of the ingoing neighbors pr. The function composes each edge $e = (i, j)$ from the digraph given by pr, and takes the corresponding edge length $f(i, j)$ and capacity $g(i, j)$ from the outgoing adjacency list of the original input network $G.Adj$.

Proposition 1. *The function* $minsum(G.Adj)$ *is correctly defined.*

Proof: The proof of the correctness of the function $minsum(G.Adj)$ is analogous to the proof of the Dijkstra's

Algorithm 2 Function $minsum(G.Adj)$

Input: $G.Adj$
Output: distance d_0 to vertex n, and $\widehat{G}.Adj$

$V_0 \leftarrow \{1, 2, \ldots, n\}$
$d \leftarrow (0, \infty, \ldots, \infty)$
while $V_0 \neq \varnothing$ **do**
  $\{u, V_0\} \leftarrow extract(V_0)$
  for each $q \in G.Adj(u)$ **do**
    $\{d, pr\} \leftarrow relaxS(u, q(1), d, pr)$
  end for
end while
$\widehat{G}.Adj \leftarrow outadj(pr, G.Adj)$
return $\{d(n), \widehat{G}.Adj\}$

algorithm (see [18]). We will only note that after the each iteration of the **while** loop the following properties hold for each vertex $v \in V_0$:

1) $d(v) \geq r(v)$;
2) $pr(v)$ is the set of all vertices $u \in U$ for which there exists a path $\alpha = (i_0, i_1, \ldots, i_k, u, v)$ with length $x(\alpha) = d(v)$.

Besides that, when $v_1 \in V_0$ and $d(v_1) = \min_{v \in V_0}\{d(v)\}$, then following the proof of the Dijkstra's algorithm, we find out that $d(v_1) = r(v_1)$. Then in the set V_0 does not exist a vertex that is the one before the last one in a (i_0, v_1)-path with length $d(v_1)$. In fact, if we assume that for the path $\alpha = (i_0, i_1, \ldots, i_k, v_1)$ holds $i_k \in V_0$ and $x(\alpha) = d(v_1)$, we reach contradiction because

$$r(v_1) = d(v_1) = \sum_{s=1}^{k} f(i_{s-1}, i_s) + f(i_k, v_1) \geq r(i_k) + f(i_k, v)$$

and using the inequality (10), we get $r(v_1) \geq r(v_1) + f(i_k, v)$.

The **while** loop of the algorithm stops when $V_0 = \varnothing$, and then $d(j) = r(j), \forall j \in V$.

We define the network $\widehat{G} = (V, \widehat{E}, f, g)$, where $\widehat{E} = \{(i, j) \in E : i \in pr(j)\}$. In this way, $\widehat{G}$ is the shortest paths subnetwork of the network G, and pr is the adjacency list of the ingoing neighbors of the digraph $(V, \widehat{E})$, which is verified directly, by using the fact that $pr(v)$ is the set of those vertices $u \in V$ for which there exists a path $\alpha = (i_0, i_1, \ldots, u, v)$ such that $x(\alpha) = r(v)$. ■

It is clear that using the adjacency list $\widehat{G}.Adj$, it is easy to compose the list of all $(1, n)$- shortest paths. The following example illustrates this observation.

Example 1. *Let G_1 is the network given on the Figure 1. We will find all $(1, 5)$- shortest paths in G_1.*

Solution: The outgoing adjacency list of G_1 is given by:

$$\begin{aligned} G_1.Adj = \{&\{(2, 2, 4), (3, 5, 3)\}, \\ &\{(3, 3, 5), (4, 6, 4), (5, 5, 3)\}, \\ &\{(4, 3, 6), (5, 1, 1)\}, \{(5, 1, 7)\}, \{\}\}. \end{aligned} \tag{11}$$

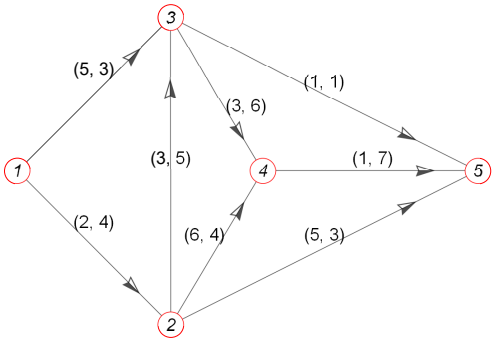

Fig. 1. Example network G_1 composed by five vertices with length and capacity of each edge given next to it

The function $minsum(G_1.Adj)$ produces the following result:

$$\{d_0, \widehat{G}_1.Adj\} \leftarrow minsum(G_1.Adj), \tag{12}$$

where $\widehat{G}_1.Adj = \{\{(2, 2, 4), (3, 5, 3)\}, \{(3, 3, 5), (4, 6, 4)\}, \{(4, 3, 6), (5, 1, 1)\}, \{\}, \{\}\}$ and $d_0 = 6$.

The network with outgoing adjacency list $\widehat{G}_1.Adj$ has exactly two $(1, 5)$-paths

$$\alpha = (1, 3, 5) \text{ and } \beta = (1, 2, 3, 5).$$

Obviously, $x(\alpha) = x(\beta) = 6 = d_0$.

Proposition 2. *The function $minsum(G.Adj)$ has computational complexity $O(n \log n + m)$.*

The proof follows directly from the computational complexity of the Dijkstra's algorithm, in the case in which the Fibonacci heap data structure is used for the implementation of the priority queue operations (refer to [18]).

B. List of all maximum capacity paths

We denote the capacity of each vertex of the network $v \in V$ with $c(v)$.

Definition 3. *We say that $\widetilde{G} = (V, \widetilde{E})$ is a **maximal capacity** digraph of the network $G = (V, E, f, g)$, if the following two properties hold:*

1) *Every $(1, n)$- maximal capacity path in G is also a $(1, n)$-path in $\widetilde{G}$.*
2) *Every $(1, n)$-path in $\widetilde{G}$ is also a $(1, n)$- maximal capacity path in G.*

In this section we will define the function $maxmin(G.Adj)$ which calculates the adjacency list of the outgoing neighbors $\widetilde{G}.Adj$ of the maximal capacity graph $\widetilde{G} = (V, \widetilde{E})$. For its implementation, first we will formulate the function $capacity(G.Adj)$ (Alg. 3) which calculates the capacity of a vertex n in the network G, again following the Dijkstra's algorithm.

We denote with d a vector of n components that initially has all its elements equal to $-\infty$, except the first element $d(1) = \infty$. During the calculations of the function $capacity(G.Adj)$ the components of d are growing, and when the function

Algorithm 3 Function $capacity(G.Adj)$

Input: $G.Adj$
Output: the capacity c_n of the vertex n

```
    V_0 ← {1, 2, ..., n}
 2: d ← (∞, −∞, ..., −∞)
    while V_0 ≠ 0 do
 4:     {u, V_0} ← extract(V_0, d)
        for each q ∈ G.Adj(u) do
 6:         relax(u, q(1), G.Adj, d)
        end for
 8: end while
    return d(n)
```

Algorithm 4 Function $maxmin(G.Adj)$

Input: $G.Adj$
Output: the capacity c of the vertex n and $\widetilde{G}.Adj$

```
    c ← capacity(G.Adj)
 2: for i ← 1 to n do
        for each q ∈ Adj(i) do
 4:         if q(3) ≥ c then
               pushback(G̃.Adj, q(1))
 6:         end if
        end for
 8: end for
    return {c, G̃.Adj}
```

completes, the j-th component of d stores the capacity of the vertex j, or in other words $d(j) = c(j)$. Note that by definition $c(1) = \infty$.

We will use the following helper functions in the implementation of $capacity(G.Adj)$.

The function $extract(V_0, d)$ for an arbitrary subset $V_0 = \{j_1, j_2, \ldots, j_k\} \subseteq V$ and using the vector d, calculates the pair $\{j_s, V_1\}$, where

$$j_s \in V_0,\ d(j_s) = \max_{j_p \in V_0} \{d(j_p)\} \text{ and } V_1 = V_0 \setminus \{j_s\}. \quad (13)$$

The function $relax(i, j, G.Adj, d)$ for each two vertices i and j, and based on $G.Adj$ and d, performs the following calculations:

1) Defines $r = \min\{d(i), g(i, j)\}$.
2) If $d(j) < r$, it sets $d(j) = r$.

Proposition 3. *The function $capacity(G.Adj)$ is correctly defined.*

Proof: Let $1 \leq k < n$. We assume that after k iterations of the **while** loop (see Alg. 3), from the initial set $V_0 = V = \{1, 2, \ldots, n\}$ are excluded k number of vertices u_s, and for the resulting set $V_0 = V \setminus \{u_1, \ldots, u_k\}$ the following properties hold:

1) For each $u \in U = \{u_1, \ldots, u_k\}$ it holds that $d(u) = c(u)$.
2) For each $u \in U$ and each $v \in V_0$ it hods that $d(u) \geq d(v)$.
3) Let $v_k \in V_0$ and $d(v_k) \geq d(v)$ for each $v \in V_0$. Then $d(v_k) = c(v_k)$.

The above three properties are true for $k = 1$.

After the $(k+1)$-st iteration of the **while** loop, the vertex v_k is excluded from v_0 and we get $V_0' = V_0 \setminus \{v_k\}$. We will prove that the three properties hold for the set V_0'.

From the definition of v_k and the fact that the three properties hold for V_0, it follows directly that properties 1) and 2) are fulfilled for V_0'.

Now, let $v_{k+1} \in V_0'$ and $d(v_{k+1}) \geq d(v)$ for each $v \in V_0'$. We denote $U' = U \cup \{v_k\} = V \setminus V_0'$. Also, let α be an arbitrary $(1, v_{k+1})$-path and

$$\alpha = (1, u_1, \ldots, u_s, v_{j_1}, \ldots, v_{j_r}, v_{k+1}),$$

where $u_i \in U', \forall i \in \{1, \ldots, s\}$ and $v_{j_1} \in V_0'$.

From the definition of $d(v_{j_1})$ it follows that for the capacity $y(\beta)$ of the path $\beta = (1, u_1, \ldots, u_s, v_{j_1})$ it is fulfilled that $y(\beta) \leq d(j_1)$. Then, for that capacity $y(\alpha)$ of the path α we have that

$$y(\alpha) \leq y(\beta) \leq d(v_{j_1}) \leq d(v_{k+1}),$$

which proves that $d(v_{k+1}) = c(v_{k+1})$. ■

Having the function $capacity(G.Adj)$, we can define the function $maxmin(G.Adj)$, given in Alg. 4.

Proposition 4. *The function $maxmin(G.Adj)$ is correctly defined.*

Proof: Since the function $capcity(G.Adj)$ that is triggered on the first line of the Alg. 4 is correct, it follows that c is the capacity of the vertex n. The outer **for** loop on the line 2 defines that adjacency list $\widetilde{G}.Adj$ of the graph $\widetilde{G} = (V, \widetilde{E})$, where $\widetilde{E} = \{(i, j) \in E : g(i, j) \geq c\}$. Now we will prove that the graph $\widetilde{G} = (V, \widetilde{E})$ is the maximum capacity graph.

If α is a $(1, n)$-path in the network G with capacity $y(\alpha) = c$, then for each edge (i, j) of α it holds that $g(i, j) \geq c$, and hence, $(i, j) \in \widetilde{E}$. This shows that α is a $(1, n)$-path of $\widetilde{G}$.

Now, let β is a $(1, n)$-path in $\widetilde{G}$. This means that β is a $(1, n)$-path in the network G, and for each its edge (i, j) it is fulfilled that $g(i, j) \geq c$. From here it follows that $y(\beta) \geq c$. However, since c is the capacity of the vertex n, then $y(\beta) = c$. ■

Proposition 5. *The computational complexity of the function $maxmin(G.Adj)$ is $O(n \log n + m)$.*

The proof follows directly from the complexity of the Dijkstra's algorithm implemented with Fibonacci heap, since the function $capacity(G.Adj)$ repeats exactly it steps.

We will introduce the function $list(m, \widetilde{G}.Adj)$ that will help us to clarify the following examples. The argument of the function m is a natural number or ∞. The function maps a list S of $(1, n)$-paths in $\widetilde{G}$ that satisfies the following properties:

1) If $m = \infty$, the list S contains all $(1, n)$-paths of $\widetilde{G}$.
2) If m is a natural number, S contains all $(1, n)$-paths of $\widetilde{G}$ if their number is not greater than m. Otherwise, S contains m number of all $(1, n)$-paths of $\widetilde{G}$.

Example 2. *For the network G_1 (see Fig. 1) we will compose the list S_y (9) of all $(1,5)$-paths with maximal capacity.*

Solution: The solution is composed by the following two steps:

1) $\{c, \widetilde{G}_1.Adj\} \leftarrow maxmin(G_1.Adj)$.
2) $S_y \leftarrow list(\infty, \widetilde{G}_1.Adj)$.

On the first step, using the function $maxmin(G_1.Adj)$, we calculate that:

$$c = 4 \text{ and } \widetilde{G}_1.Adj = \{\{2\}, \{3,4\}, \{4\}, \{5\}, \{\}\}.$$

On the second step, the function $list(\infty, \widetilde{G}_1.Adj)$ calculates the list

$$S_y = \{(1,2,4,5), (1,2,3,4,5)\}.$$

Then, it is directly verified that

$$y((1,2,4,5)) = y((1,2,3,4,5)) = 4 = c.$$

IV. Pareto optimal solutions set

We denote with P the set of Pareto optimal paths in the network $G = (V, V, f, g)$. It is clear that

$$P = \bigcup_{i=1}^{k_0} P_i, \tag{14}$$

where P_i are the classes of Pareto equivalent paths.

We will find the set P by composing a list Q of all classes of Pareto equivalent paths P_i. We will compose the list Q using the following procedure, which we will call *Pareto Optimal Paths (POP)*:

1) Set $W = W_n$.
2) Calculate $d = \min\limits_{\beta \in W}\{x(\beta)\}$ and define $X = \{\alpha \in W : x(\alpha) = d\}$.
3) Calculate $c = \max\limits_{\beta \in X}\{y(\beta)\}$ and define $P_0 = \{\alpha \in X : y(\alpha) = c\}$. Store P_0 into the list Q.
4) Define $Y = \{\beta \in W : y(\beta) > c\}$.
5) Define $Z = W \setminus (Y \cup P_0)$.
6) If $Y = \varnothing$, then end. Otherwise, set $W = Y$ and go back to step 2.

Lemma 1. *The POP procedure correctly composes the list Q.*

Proof: We will prove the correctness of the POP procedure by induction.

Base case. The *first iteration* of POP defines:

$$d_1 = \min_{\beta \in W_n}\{x(\beta)\}, X_1 = \{\alpha \in W_n : x(\alpha) = d_1\},$$
$$c_1 = \max_{\beta \in X_1}\{y(\beta)\}, P_1 = \{\alpha \in X_1 : y(\alpha) = c_1\}, Q = \{P_1\},$$
$$Y_1 = \{\beta \in W_n : y(\beta) > c_1\}, Z_1 = W_n \setminus (Y_1 \cup P_1).$$

The following properties hold:

1) From the definitions of the sets P_1, Y_1 and Z_1 it follows:
$$W_n = Z_1 \cup P_1 \cup Y_1, Z_1 \cap P_1 = \varnothing,$$
$$Z_1 \cap Y_1 = \varnothing, Y_1 \cap P_1 = \varnothing.$$
2) If $\alpha \in P_1$, then by the definition of P_1 the equalities $x(\alpha) = d_1$ and $y(\alpha) = c_1$ hold. Hence, the paths that belong to P_1 are equivalent.
3) If $Z_1 \neq \varnothing$ and $\beta \in Z_1$, then from step 5 it follows that one of the following two statements is fulfilled:
   a) $y(\beta) < c_1$ and $x(\beta) \geq d_1$, or
   b) $y(\beta) = c_1$ and $x(\beta) > d_1$.

   Hence, β is dominated by each $\alpha \in P_1$.

If $Y_1 = \varnothing$, then $W_n = P_1 \cup Z_1$ and P_1 is the set of Pareto optimal paths. This means that in (14) the constant $k_0 = 1$, and the calculations of the POP procedure will stop.

If $Y_1 \neq \varnothing$, then the constant $k_0 > 1$. In this case, for each $\beta \in Y_1$ the inequalities are fulfilled:

$$c_1 < y(\beta) \text{ and } d_1 < x(\beta). \tag{15}$$

As a result, also in this case P_1 is a set of equivalent Pareto optimal paths α, and it is correctly included in the list Q. Here, we set $W = Y_1$, and we go back to step 2 of the second iteration of the procedure.

The *second iteration* of the POP procedure defines:

$$d_2 = \min_{\beta \in Y_1}\{x(\beta)\}, X_2 = \{\alpha \in Y_1 : x(\alpha) = d_2\},$$
$$c_2 = \max_{\beta \in X_2}\{y(\beta)\}, P_2 = \{\alpha \in X_2 : y(\alpha) = c_2\},$$
$$Q = \{P_1, P_2\}, Y_2 = \{\beta \in Y_1 : y(\beta) > c_2\},$$
$$Z_2 = Y_1 \setminus (Y_2 \cup P_2).$$

In analogy with the first iteration of POP, the following properties are proved.

1) From the definitions of the sets P_2, Y_2 and Z_2 it follows:
$$Y_1 = Z_2 \cup P_2 \cup Y_2, Z_2 \cap P_2 = \varnothing,$$
$$Z_2 \cap Y_2 = \varnothing, Y_2 \cap P_2 = \varnothing \tag{16}$$
2) If $\alpha \in P_2$, then by the definition of P_2 the equalities $x(\alpha) = d_2$ and $y(\alpha) = c_2$ hold, and hence the paths that belong to P_2 are equivalent. Besides that, from (15) follows that $d_1 < d_2$ and $c_1 < c_2$.

If $Z_2 \neq \varnothing$, then for each path $\beta \in Z_2$ holds that $x(\beta) \geq d_2$ because $\beta \in Y_1$. Besides that, $y(\beta) \leq c_2$, because $\beta \notin Y_2$. Then, for β one of the following statements hold:

- $y(\beta) < c_2$ and $x(\beta) \geq d_2$, or
- $y(\beta) = c_2$ and $x(\beta) > d_2$, because $\beta \notin P_2$.

Therefore, each path $\beta \in Z_2$ is dominated by any path from P_2.

Let $\alpha \in P_2$, and β is such a $(1,n)$-path, so that $\beta \notin Y_1$. Then $y(\beta) \leq c_1 < y(\alpha)$. The following two cases are possible:

- $x(\beta) \geq d_2 = x(\alpha)$. In this case α dominates β.
- $x(\beta) < d_2 = x(\alpha)$. In this case α and β cannot be compared.

That proves that if $Y_2 = \varnothing$, then the elements of P_2 are Pareto optimal and the equation (14) has the form $P = P_1 \cup P_2$, in other words, $k_0 = 2$. From here it follows that P_2 is correctly included in the list Q, and the termination of the computation of POP procedure is correct.

If $Y_2 \neq \varnothing$, then $k_0 > 2$. In this case, for each $\beta \in Y_2$ the following inequalities are fulfilled.

$$c_2 < y(\beta) \text{ and } d_2 < x(\beta) \tag{17}$$

The first inequality follows from the definition of Y_2, and the second one – from the definition of d_2 and c_2.

Therefore, also in the case in which $Y_2 \neq \varnothing$, the set P_2 is a set of Pareto optimal paths, and it is correctly included in the list Q. In this case, we set $W = Y_2$ and we go back to step 2 in the procedure to start its third iteration.

It is clear that in the third iteration of the procedure $W_n = Z_1 \cup P_1 \cup Z_2 \cup P_2 \cup Y_2$.

Inductive step. We assume that after $k \geq 2$ iterations of the POP procedure, the following components are defined.

- The sets P_i, Z_i, Y_k, $i \in \{1, 2, \ldots, k\}$, that have no common elements.
- The numbers d_i and c_i $i \in \{1, 2, \ldots, k\}$, for which the following five properties are fulfilled.
  1) $W_n = \bigcup_{i=1}^{k} (Z_i \cup P_i) \bigcup Y_k$.
  2) $P_1, P_2, \ldots, P_k$ are sets of equivalent Pareto optimal paths, for which
     a) $d_i = x(\alpha)$ and $c_i = y(\alpha)$, $\forall \alpha \in P_i$ and $\forall i \in \{1, 2, \ldots, k\}$;
     b) $d_i < d_{i+1}$ and $c_i < c_{i+1}$, $\forall i \in \{1, 2, \ldots, (k-1)\}$.
  3) Every path $\beta \in Z_i$ is dominated by every path $\alpha \in P_i$.
  4) For every $\beta \in Y_k$ the inequalities hold:
  $$c_k < y(\beta) \text{ and } d_k < x(\beta). \tag{18}$$
  5) $Q = \{P_1, P_2, \ldots, P_k\}$.

  These five properties follow directly from the proof of the *second iteration* of the procedure.

It is clear that if $Y_k = \varnothing$, then $W_n = \bigcup_{i=1}^{k} (Z_i \cup P_i)$ and the list Q is correctly composed.

If $Y_k \neq \varnothing$, we implement the $(k+1)$-st iteration of the POP procedure. Using steps from 2 to 5, we define:

$$d_{k+1} = \min_{\beta \in Y_k} \{x(\beta)\}, X_{k+1} = \{\alpha \in Y_k : x(\alpha) = d_{k+1}\},$$
$$c_{k+1} = \max_{\beta \in X_{k+1}} \{y(\beta)\},$$
$$P_{k+1} = \{\alpha \in X_{k+1} : y(\alpha) = c_{k+1}\},$$
$$Q = \{P_1, P_2, \ldots, P_{k+1}\}, Y_{k+1} = \{\beta \in Y_k : y(\beta) > c_2\},$$
$$Z_{k+1} = Y_k \setminus (Y_{k+1} \cup P_{k+1}).$$

Repeating the proof of the second iteration, we find out that for

- sets P_i, Z_i, Y_{k+1}, $i \in \{1, 2, \ldots, k, (k+1)\}$, and
- numbers d_i and c_i $i \in \{1, 2, \ldots, k, (k+1)\}$

the five properties are fulfilled.

Since W_n has finite number of elements and $P_i \neq \varnothing, \forall i$, then after a finite number of k_0 iterations the procedure POP stops, and we prove that:

1) $W_n = \bigcup_{i=1}^{k_0} (Z_i \cup P_i)$;
2) $P_1, P_2, \ldots, P_{k_0}$ are sets of equivalent Pareto optimal paths for which
   a) $d_i = x(\alpha)$ and $c_i = y(\alpha)$, $\forall \alpha \in P_i$ and $\forall i \in \{1, 2, \ldots, k_0\}$;
   b) $d_i < d_{i+1}$ and $c_i < c_{i+1}, \forall i \in \{1, 2, \ldots, (k_0 - 1)\}$.
3) Every path $\beta \in Z_i$ is dominated by every path $\alpha \in P_i$.
4) $Q = \{P_1, P_2, \ldots, P_{k_0}\}$.

■

Corollary 1. *For the classes P_i of Pareto optimal paths the following holds*

$$P_i = \{\alpha \in W_n : \ x(\alpha) = d_i \text{ and } y(\alpha) = c_i\},$$

for each $i \in \{1, 2, \ldots, k_0\}$.

Corollary 2. *For P_{k_0} the following equality hods:*

$$P_{k_0} = \{\alpha \in X_1' : \ x(\alpha) = d_1'\},$$

where $X_1' = \{\alpha \in W_n : y(\alpha) = \max_{\beta \in W_n} \{y(\beta)\}$ and $d_1' = \min_{\beta \in X_1'} \{x(\beta)\}$.

Remark 1. *The list Q can be composed by: first apply the Corollary 2 to separate the set P_{k_0}; after that, consecutively separate the sets P_{k_0-1}, P_{k_0-2}, and so on, until P_1 is separated.*

Every subset P_i turns out to be a set of all $(1, n)$-paths in a special digraph G_i. The algorithm that finds a *List of Pareto Optimal Paths* which we will call LPOP (given in Alg. 5), composes a list S of the adjacency lists $G_i.Adj$ for each $i \in \{1, 2, \ldots, k_0\}$ by implementing the POP procedure.

We will note that once we have the list S, we can easily obtain a list P' of Pareto optimal solutions by taking predefined number of elements for each class P_i, as well we can obtain a list P of all optimal solutions.

Besides the previously defined functions $minsum(G.Adj)$, $maxmin(G.Adj)$ and $capacity(G.Adj)$, in the formulation of the LPOP algorithm (Alg. 5), we will use the function $restrict(G.Adj, c)$, that is defined as follows.

The function $restrict(G.Adj, c)$ takes as an input the outgoing adjacency list $G.Adj$ and the number c. It calculates the adjacency list of those edges (i, j) from $G.Adj$, for which $g(i, j) > c$.

Theorem 1. *The LPOP algorithm (Alg. 5) is correct.*

Proof: The correctness of the Alg. 5 follows from the correctness of the functions $capacity(G.Adj)$, $minsum(G.Adj)$, $maxmin(G.Adj)$, and from Lemma 1. It is easily verified by proving that the **while** loop of the $lpop(G.Adj)$ function implements the POP procedure using these functions.

We will examine the first iteration of the **while** loop. The function $minsum(G.Adj)$ calculates $d_0 = \min_{\beta \in W_n} \{x(\beta)\}$ and defines the adjacency list $\widehat{G}.Adj$ of the shortest paths subnetwork $\widehat{G}$. From the correctness of the function $minsum(Adj)$

Algorithm 5 Function $lpop(G.Adj)$

Input: $G.Adj$
Output: the list S with k_0 number of elements

```
    RAdj ← G.Adj
 2: c_0 ← capacity(G.Adj)
    more ← true
 4: while more = true do
      {d_0, G.Adj} ← minsum(G.Adj)
 6:   if d_0 = ∞ then
        more ← false
 8:   else
        {c_1, G̃.Adj} ← maxmin(G.Adj)
10:     pushback(S, G̃.Adj)
        if c_1 = c_0 then
12:       more ← false
        else
14:       RAdj ← restrict(RAdj, c_1)
        end if
16:     G.Adj ← RAdj
      end if
18: end while
    return S
```

we know that α is a $(1,n)$-path in $\widehat{G}$, exactly when α is a $(1,n)$-path in the network G and $x(\alpha) = d_0$. Hence, the set X from the POP procedure is the set of all $(1,n)$-paths of the subnetwork, defined by the adjacency list $G.Adj$.

By using the definition of the function $maxmin(G.Adj)$, we will prove that its function call implements step 3 of the POP procedure.

Indeed, when applied on the adjacency list of the subnetwork $\widehat{G}$, the function $maxmin(Adj)$ calculates the maximal capacity c_1 of a $(1,n)$-path in $\widehat{G}$, and defines the adjacency list $\widehat{G}.Adj$ of the maximal capacity digraph $\widetilde{G}$ of the subnetwork $\widehat{G}$. This means that α is a $(1,n)$-path in the digraph $\widetilde{G}$, if and only if it is a $(1,n)$-path in the network $\widehat{G}$ and $y(\beta) = c_1$. In the notations of the POP procedure, this means that $c_1 = \max\limits_{\beta \in X}\{y(\beta)\}$ and α is a $(1,n)$-path in the digraph $\widetilde{G}$ if and only if $\alpha \in P_0$. For that reason the adjacency list $\widetilde{G}.Adj$ is included in the list S (line 10 of Alg. 5).

The step 6 from the POP procedure is implemented in the alternative branch of the **if** statement which verifies whether the set Y, defined by the step 4 of the procedure, is the empty set. If $c_0 = c_1$, then $Y = \varnothing$, and the algorithm is terminated. Otherwise, we define the adjacency list $RAdj$ of the subnetwork for which β is a $(1,n)$-path if and only if it is a $(1,n)$-path in G, and $y(\beta) > c_1$. ■

We will illustrate the above proof with the following example.

Example 3. *For the input network G_1, given on Fig. 1, we will trace how the algorithm LPOP (Alg. 5) implements the POP procedure.*

Solution: Initially, the algorithm stores a copy of the adjacency list in the variable $RAdj$, which will be modified in the body of the **while** loop. On line 2, the $capacity(G.Adj)$ function calculates that the capacity of the vertex 5 is $c_0 = 4$.

First iteration. The $minsum(G.Adj)$ function calculates the distance $d_0 = 6$ to the vertex 5, and the adjacency list of the subnetwork $\widehat{G}$:

$$\widehat{G}.Adj = \{\{(2,2,4),(3,5,3)\},\{(3,3,5),(4,6,4)\},\{(4,3,6),(5,1,1)\},\{\},\{\}\}.$$

It is apparent that the above defined subnetwork $\widehat{G}$ has exactly two $(1,5)$-paths: $\alpha_1 = (1,3,5)$ and $\beta_1 = (1,2,3,5)$. Besides that, $x(\alpha_1) = x(\beta_1) = 6$. Since the set of all $(1,5)$-paths in G_1 is $W = \{(1,2,5),(1,3,5),(1,2,3,5),(1,2,4,5),(1,3,4,5),(1,2,3,4,5)\}$, it is directly verified that α_1 and β_1 are the only $(1,5)$-paths with length $d_0 = 6$ in the network G_1. The latter follows from the correctness of the function $minsum(G.Adj)$. In the procedure POP the set $\{\alpha_1, \beta_1\}$ is denoted by X.

Since $d_0 = 6 \neq \infty$ the algorithm enters the body of the **else** statement on line 8. The function $maxmin(G.Adj)$ calculates that in the subnetwork $\widehat{G}$ the capacity of the vertex 5 is $c_1 = 1$, and:

$$\widetilde{G}.Adj = \{\{2,3\},\{3,4\},\{4,5\},\{\},\{\}\}.$$

The outgoing adjacency list $\widetilde{G}.Adj$ defines the maximal capacity digraph $\widetilde{G}$ of the subnetwork $\widehat{G}$. It is apparent that $\widetilde{G}$ has exactly two $(1,5)$-paths. In this case these are $\alpha_1 = (1,3,5)$ and $\beta_1 = (1,2,3,5)$. Also, $y(\alpha_1) = y(\beta_1) = 1$. In the procedure POP we denote the set $\{\alpha_1, \beta_1\}$ by P_0. From Lemma 1 it follows that $P_0 = \{\alpha_1, \beta_1\}$ is the first class of equivalent Pareto optimal solutions. For that reason the algorithm includes $\widetilde{G}.Adj$ in the list S.

Since $c_1 = 1 \neq c_0 = 4$, the function $restrict(RAdj, c_1)$ modifies the adjacency list $RAdj$ by removing all edges with capacity not greater than c_1. The new adjacency list is:

$$RAdj = \{\{(2,2,4),(3,5,3)\},\{(3,3,5),(4,6,4),(5,5,3)\},\{(4,3,6)\},\{(5,1,7)\},\{\}\}.$$

It is directly verified that the set Y_1 of all $(1,5)$-paths in the network defined by $RAdj$ is the set of all $(1,5)$-paths in the network G_1 with capacity bigger than $c_1 = 1$, which is verified by Theorem 1. In the procedure POP Y_1 is denoted by Y and is defined in the step 4 of the procedure.

Setting $G.Adj = RAdj$ the LPOP algorithm moves to the next iteration. In the procedure POP, it corresponds to the assignment $W = Y$, and the start of the new iteration by transition to the step 2.

Second iteration. The $minsum(G.Adj)$ function calculates $d_0 = 7$ and

$$\widehat{G}Adj = \{\{(2,2,4),(3,5,3)\},\{(3,3,5),(4,6,4),(5,5,3)\},\{(4,3,6)\},\{\},\{\}\}.$$

The resulting subnetwork $\widehat{G}$ has a single $(1,5)$-path $\alpha_2 = \{1,2,5\}$. The length of the path α_2 is $x(\alpha_2) = 7$. In this case $X = \{\alpha_2\}$.

The result of the function $maxmin(G.Adj)$ is:

- the capacity of the vertex 5 in the network $\widehat{G}$ is $c_1 = 3$, and
- the digraph $\widetilde{G}$ of the maximal capacity has adjacency list

$$\widetilde{G}.Adj = \{\{2,3\},\{3,4,5\},\{4\},\{\},\{\}\}.$$

The above verifies the fact that in this case $P_0 = \{\alpha_2\}$ is the second class of equivalent Pareto optimal solutions. For that reason $\widetilde{G}.Adj$ is included as second element in the list S.

Since the condition for the loop stop is not fulfilled,

$$RAdj = \{\{(2,2,4)\},\{(3,3,5),(4,6,4)\},\{(4,3,6)\},\{(5,1,7)\},\{\}\}.$$

It is immediately apparent that the set Y_2 of all $(1,5)$-paths in the network defined by $RAdj$ is the set of $(1,5)$-paths of G_1, that have capacities grater than $c_1 = 3$.

Third iteration. The $minsum(G.Adj)$ function calculates that this time the distance to the vertex 5 is $d_0 = 9$, and the new minimal paths subnetwork has adjacency list:

$$\widehat{G}Adj = \{\{(2,2,4)\},\{(3,3,5),(4,6,4)\},\{(4,3,6)\},\{(5,1,7)\},\{\}\}.$$

In this case the set of all $(1,5)$-paths in $\widehat{G}$ is $X = \{(1,2,4,5),(1,2,3,4,5)\}$, and these are all $(1,5)$-paths of Y_2 with length $d_0 = 9$.

Using the $maxmin(G.Adj)$ function, we find out that in the network $\widehat{G}$ the vertex 5 has capacity $c_1 = 4$, and the maximal capacity digraph $\widetilde{G}$ of the subnetwork $\widehat{G}$ has adjacency list $\widetilde{G}.Adj = \{\{2\},\{3,4\},\{4\},\{5\},\{\}\}$.

Apparently, the digraph $\widetilde{G}$ has exactly two $(1,5)$-paths $\alpha_3 = (1,2,4,5)$ and $\beta_3 = (1,2,3,4,5)$. According to Lemma 1, the set $P_0 = \{\alpha_3,\beta_3\}$ is the third class of equivalent Pareto optimal paths. In this case $x(\alpha_3) = x(\beta_3) = 9$ and $y(\alpha_3) = y(\beta_3) = 4$. $\widetilde{G}.Adj$ is included as third element in the list S.

The condition of the **if** statement on line 11 $c_0 = c_1$ will be evaluated to true. This means that there does not exist a $(1,5)$-path with a capacity greater than the current c_1. As a result, the **while** loop is terminated. In the procedure POP this means that $Y_3 = \{\beta \in W : y(\beta) > 4\} = \varnothing$, and the procedure stops.

After the end of the calculations

$$S = \{\{\{2,3\},\{3,4\},\{4,5\},\{\},\{\}\},\{\{2,3\},\{3,4,5\},\{4\},\{\},\{\}\},\{\{2\},\{3,4\},\{4\},\{5\},\{\}\}\},$$

where each element of S determines one class of Pareto optimal paths:

- $S(1) = \{\{2,3\},\{3,4\},\{4,5\},\{\},\{\}\}$ defines the class $P_1 = \{(1,3,5),(1,2,3,5)\}$;
- $S(2) = \{\{2,3\},\{3,4,5\},\{4\},\{\},\{\}\}$ defines the class $P_2 = \{\{1,2,5\}\}$; and
- $S(3) = \{\{\{2\},\{3,4\},\{4\},\{5\},\{\}\}\}$ defines the class $P_3 = \{(1,2,4,5),(1,2,3,4,5)\}$.

The network G_1 has the set of Pareto optimal paths $P = P_1 \cup P_2 \cup P_3$.

Theorem 2. *The LPOP algorithm (Alg. 5) has computational complexity $k_0 O(n \log n + m)$, where k_0 is the number of classes of Pareto equivalent paths.*

The proof follows from Prop. 2 and Prop. 5. It is enough to note that on line 2 of Alg. 5 the call to the function $capacity(G.Adj)$ has complexity $O(n \log n + m)$, and the **while** loop has k_0 number of iterations, where k_0 is the number of classes of Pareto equivalent classes (14). Each iteration involves a single call to the functions $minsum(G.Adj)$ and $maxmin(G.Adj)$, where both have complexity $O(n \log n + m)$. Besides that, the function $restrict(R.Adj, c_1)$ has computational complexity that is lower than $O(n \log n + m)$.

Example 4. *Let the network G_2 be defined with the adjacency list*

$$\begin{aligned} G_2.Adj = \{&\{(2,1,17),(3,1,20),(4,1,19)\},\\ &\{(5,1,7),(6,15,15),(7,1,12)\},\\ &\{(6,1,9),(7,1,18),(8,1,19),(4,1,15)\},\\ &\{(5,14,15),(6,1,12),(7,1,12),(8,1,9),\\ &(10,1,2)\},\{(9,10,22),(10,1,2),(6,1,2)\},\\ &\{(11,1,4),(9,14,20),(10,1,6),(7,1,3)\},\\ &\{(8,1,11),(10,2,7),(5,8,15)\},\\ &\{(10,7,10),(11,10,11)\},\\ &\{(11,8,19),(10,1,20)\},\{(11,3,21)\},\{\}\}. \end{aligned} \tag{19}$$

We will find the list of all Pareto optimal solutions using the LPOP algorithm.

Solution: Using the LPOP algorithm, we calculate the list

$$\begin{aligned} S = \{&\{\{2,3,4\},\{5,7\},\{6,7,8\},\{6,7,8\},\{9\},\\ &\{11\},\{\},\{\},\{\},\{\},\{\}\},\\ &\{\{2,3,4\},\{5,7\},\{6,7,8\},\{6,7,8\},\{9\},\{10\},\\ &\{\},\{\},\{\},\{11\},\{\}\},\\ &\{\{2,3,4\},\{5,7\},\{6,7,8\},\{6,7,8\},\{9\},\{\},\\ &\{10\},\{\},\{\},\{11\},\{\}\},\\ &\{\{2,3,4\},\{7\},\{7,8\},\{6,7\},\{\},\{9\},\{5\},\\ &\{11\},\{\},\{11\},\{\}\},\\ &\{\{2,3,4\},\{7\},\{7,8\},\{6,7\},\{\},\{9\},\{5\},\\ &\{\},\{10\},\{11\},\{\}\},\\ &\{\{2,3,4\},\{6\},\{7,8\},\{\},\{9\},\{\},\{5\},\{\},\\ &\{10\},\{11\},\{\}\}\}. \end{aligned} \tag{20}$$

Every element of S defines a class of equivalent Pareto optimal paths. Using the function $list(m, \widetilde{G}.Adj)$ we get the Pareto optimal paths as a sequence of vertices. For example, the first element of S defines the class P_1, that contains two equivalent Pareto optimal paths $\alpha_1 = (1,3,6,11)$ and $\beta_1 = (1,4,6,11)$.

To each $(1,11)$-path in α we will map a point

$$A_\alpha(x(\alpha), y(\alpha)) \tag{21}$$

The points defined in this way are plotted on Fig. 2. For example, to α_1 and β_1 we map a single point $A_1(3,4)$, because

TABLE I
THE ELEMENTS OF THE LIST S WITH THE CORRESPONDING PARETO OPTIMAL CLASSES P_j AND THEIR POINT REPRESENTATIONS A_j

S_j	P_k	A_j
S_1	$P_1 = \{(1,3,6,11),(1,4,6,11)\}$	$A_1(3,4)$
S_2	$P_2 = \{(1,3,6,10,11),(1,4,6,10,11)\}$	$A_2(6,6)$
S_3	$P_3 = \{(1,2,7,10,11),(1,3,7,10,11),$ $(1,4,7,10,11)\}$	$A_3(7,7)$
S_4	$P_4 = \{(1,3,8,11)\}$	$A_4(12,11)$
S_5	$P_5 = \{(1,4,6,9,10,11)\}$	$A_5(20,12)$
S_6	$P_6 = \{(1,3,7,5,9,10,11)\}$	$A_6(24,15)$

$x(\alpha_1) = x(\beta_1) = 3$ and $y(\alpha_1) = y(\beta_1) = 4$. Following this scheme, by using consecutively the elements S_j of the list S, we calculate the remaining classes P_j and we map the point A_j defined by (21), for each $j \in \{2,3,4,5,6\}$. The results are given in Tab. I.

Therefore, in example (20) the set of all Pareto optimal paths is

$$P = P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5 \cup P_6.$$

The network G_2 has 10 Pareto optimal paths, distributed in 6 classes of equivalent paths. To illustrate graphically the result in Fig. 2, we define the set $\mathcal{P}$ of all $(1,11)$ paths in digraph G_2. In the example (20) there are 118 such paths. In this case we obtain 71 points. If the point A_α illustrates the path α that is not a Pareto optimal, we plot it in black. The points given in Tab. I are plotted in white, and they represent the Pareto optimal paths. Each point A_j is a vertex of an angle γ_j with rays given in dashed lines. Let α_j be a $(1,11)$-path that is represented by the point A_j. Inside the angle γ_j lie all points that illustrate paths that are dominated by α_j. The vert. opp. angle of γ_j is plotted in gray color, and inside it might lie points that illustrate paths that dominate α_j. As we may expect, such points does not exist.

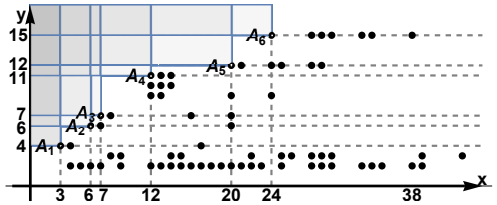

Fig. 2. Plot of the points A_j that illustrate the Pareto optimal classes P_j on the Cartesian plane defined by the values of the functions x and y

V. CONCLUSION

In [3] Hansen solves the MAXSUM-MAXMIN shortest path problem by presenting an algorithm that discovers a special subset of the Pareto optimal solutions, called "minimal complete set of efficient paths (MCS)". The list S that is composed by the LPOP algorithm presented in this paper, gives a more complete information for the Pareto optimal solutions. It is enough to select just one $(1,n)$-path from each element of S to obtain a MCS. For example, using the list S that we get for (11) that defines the network G_1, we get the following MCS: $P^{(1)} = \{(1,3,5),(1,2,5),(1,2,4,5)\}$. Besides that, apparently for the network G_1 three more different MCS can be composed.

Hansen proves that the number of Pareto optimal solutions grows exponentially with the increasing of the number of vertices in the network. However, it has been shown that in various real practical applications this number can be much smaller [20]. In the latter work authors discover key characteristics in the input data that lead to a number of Pareto optimal solutions on each vertex that is restricted by a small constant. In our case, this leads to the restriction of the constant k_0 in Theorem 2.

In Sec. III-A we solve the MINSUM list problem. We prove the correctness of $minsum(G.Adj)$ function (Alg. 2) that helps us to describe all shortest paths by calculation of the shortest paths subnetwork. We prove that its computational complexity is $O(n \log n + m)$.

In Sec. III-B we solve the MAXMIN list problem with Alg. 3 that allows us to calculate the capacity of a vertex with computational complexity $O(n \log n + m)$. Based on it, we define the function $maxmin(G.Adj)$ (Alg. 4) that describes all maximum capacity paths by defining the maximum capacity digraph of the network G. The complexity of the algorithm is again shown to be $O(n \log n + m)$.

Besides the solution of the corresponding MINSUM list and MAXMIN list problems, the two functions $minsum(G.Adj)$ and $maxmin(G.Adj)$ using Alg. 5 allow the solution of Prob. 1. The resulting description of all Pareto optimal solutions separates the classes of Pareto equivalent paths, and allows to visualize a predefined number of elements from each class of Pareto equivalent paths. The correctness of the algorithm is proved (Th. 1), and also its computational complexity is proved to be $k_0 O(n \log n + m)$ (Th. 2).

REFERENCES

[1] R. Beier, H. Röglin, C. Rösner, and B. Vöcking, "The smoothed number of pareto-optimal solutions in bicriteria integer optimization," *Mathematical Programming*, vol. 200, pp. 319–355, September 2022. doi: 10.1007/s10107-022-01885-6

[2] J. C. Namorado Climaco and E. Queirós Vieira Martins, "A bicriterion shortest path algorithm," *European Journal of Operational Research*, vol. 11, no. 4, pp. 399–404, 1982. doi: 10.1016/0377-2217(82)90205-3

[3] P. Hansen, "Bicriterion path problems," *Multiple Criteria Decision Making Theory and Application*, pp. 109–127, 1980. doi: 10.1016/S1097-2765(03)00225-9

[4] X. Gandibleux, F. Beugnies, and S. Randriamasy, "Martins' algorithm revisited for multi-objective shortest path problems with a maxmin cost function," *4OR*, vol. 4, no. 1, pp. 47–59, 2006. doi: 10.1007/s10288-005-0074-x

[5] E. Q. V. Martins, "On a multicriteria shortest path problem," *European Journal of Operational Research*, vol. 16, no. 2, pp. 236–245, 1984. doi: 10.1016/0377-2217(84)90077-8

[6] J. Brumbaugh-Smith and D. Shier, "An empirical investigation of some bicriterion shortest path algorithms," *European Journal of Operational Research*, vol. 43, no. 2, pp. 216–224, 1989. doi: 10.1016/0377-2217(89)90215-4

[7] A. Skriver and K. Andersen, "A label correcting approach for solving bicriterion shortest-path problems," *Computers & Operations Research*, vol. 27, no. 6, pp. 507–524, 2000. doi: 10.1016/S0305-0548(99)00037-4

[8] A. Sedeño-noda and M. Colebrook, "A biobjective dijkstra algorithm," *European Journal of Operational Research*, vol. 276, no. 1, pp. 106–118, 2019. doi: 10.1016/j.ejor.2019.01.007

[9] M. Minoux, "Solving combinatorial problems with combined min-max-min-sum objective and applications," *Mathematical Programming*, vol. 45, no. 1-3, pp. 361–372, 1989. doi: 10.1007/bf01589111

[10] A. P. Punnen, "On combined minmax-minsum optimization," *Computers & Operations Research*, vol. 21, no. 6, pp. 707–716, 1994. doi: 10.1016/0305-0548(94)90084-1

[11] P. Dell'Olmo, M. Gentili, and A. Scozzari, "On finding dissimilar pareto-optimal paths," *European Journal of Operational Research*, vol. 162, no. 1, pp. 70–82, 2005. doi: 10.1016/j.ejor.2003.10.033

[12] F. Guerriero and R. Musmanno, "Label correcting methods to solve multicriteria shortest path problems," *Journal of Optimization Theory and Applications*, vol. 111, no. 3, pp. 589–613, 2001. doi: 10.1023/A:1012602011914

[13] C. Mohamed, J. Bassem, and L. Taicir, "A genetic algorithms to solve the bicriteria shortest path problem," *Electronic Notes in Discrete Mathematics*, vol. 4, no. 1, pp. 851–858, 2010. doi: 10.1016/j.endm.2010.05.108

[14] S. Fidanova, M. Ganzha, and O. Roeva, "Intercriteria analyzis of hybrid ant colony optimization algorithm for multiple knapsack problem," in *2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS)*, 2021. doi: 10.15439/2021F22 pp. 173–180.

[15] A. Cassia, O. Jabali, F. Malucelli, and M. Pascoal, "The electric vehicle shortest path problem with time windows and prize collection," in *2022 17th Conference on Computer Science and Intelligence Systems (FedCSIS)*, 2022. doi: 10.15439/2022F186 pp. 313–322.

[16] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959. doi: 10.1007/bf01386390

[17] R. Diestel, *Graph Theory*, 5th ed. Berlin: Springer Publishing Company, Incorporated, 2017. ISBN 3662536218. doi: 10.1007/978-3-662-53622-3

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, 3rd ed. Cambridge, Massachusetts: The MIT Press, 2009. doi: 10.5555/1614191

[19] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM*, vol. 34, no. 3, pp. 596–615, July 1987. doi: 10.1145/28869.28874

[20] M. Müller-Hannemann and K. Weihe, "Pareto shortest paths is often feasible in practice," *Algorithm Engineering*, pp. 185–197, 2001. doi: 10.1007/3-540-44688-5_15