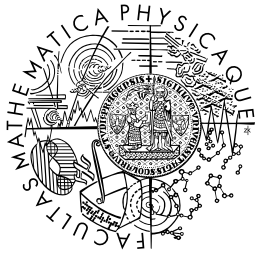


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Jiří Březina

Multiobjective shortest path problem with interval costs

Department of Algebra

Supervisor of the master thesis: prof. Mgr. Milan Hladík, Ph.D.

Study programme: Mathematics for Information Technologies

Prague 2023

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank my supervisor prof. Mgr. Milan Hladík, Ph.D., for his great advice, willingness and the time he invested in resolving issues associated with this thesis.

Title: Multiobjective shortest path problem with interval costs

Author: Jiří Březina

Department: Department of Algebra

Supervisor: prof. Mgr. Milan Hladík, Ph.D., Department of Applied Mathematics

Abstract: The multiobjective shortest path problem with interval costs is a generalization of the single-pair shortest path problem. In this problem, the edge weights are represented as tuples of intervals. The aim is to find the path that minimizes the maximum regret. We present theorems regarding the computation of the regret and the efficiency of a feasible solution to the problem. The main result of the thesis is an algorithm seeking for the solution with the least regret in the interval multiobjective shortest path problem.

Keywords: the interval multiobjective shortest path problem, the minimax regret problem, an efficient solution.

Contents

Introduction

The shortest path problem (SPP) is a well-known problem from the graph theory. It has been extensively studied over the past century, and even in the recent years, researchers continue to seek more efficient algorithms to solve any type of shortest path problems. This interest is obviously driven by various range of real-world applications, including traffic flow optimization, network routing algorithms and robot path planning. As our knowledge about SPP deepens, numerous forms of generalization to the problem naturally arise.

One possible generalization of the SPP is the multiobjective shortest path problem (MOSP), where the weight of each edge consists of more than one number. These numbers, known as criteria, allow us to compare the paths based on multiple aspects, unlike the classical SPP. However, in order to achieve wider comparison, we pay the cost of complexity when comparing vectors with a large number of components.

Another generalization involves the uncertainty regarding the exact weights of edges. There can be situations, where we may not know the precise value of certain edge weight, but we still know that this value lies within specific intervals. In such cases, we are solving the interval shortest path problem instead of the SPP. Different potential values within these intervals can result in different path lengths, and consequently different longest paths.

In recent years, many authors have studied the two aforementioned generalizations, although not to the same extent as the original SPP. However, the combination of interval SPP and MOSP has received relatively less attention in terms of in-depth study. This is the main reason why we have chosen this topic for our thesis.

The thesis is divided into four chapters, with the first one serving as a preliminary section that introduces the concepts used in the following chapters. In this chapter, we define the single-pair shortest path problem, interval multiobjective programming and other necessary definitions and theorems for our work. The key concepts defined in this chapter are the efficiency and the maximal regret of a feasible solution.

In the second chapter we delve deeper into these two concepts, emphasizing that feasible solutions now corresponds to paths in the graph from the starting vertex to the ending vertex. We present several theoretical theorems regarding the efficiency and the regret of a given solution. Furthermore, we simplify the computation of the regret for the interval MOSP. Many theorems discussed in this chapter will help us in a construction of an algorithm solving the minimax regret problem in the interval MOSP.

The construction of this algorithm will accompany us throughout the entire third chapter. Employing the algorithm proposed by Rivaz in [7] for solving general interval multiobjective linear program, we adapt their approach to our specific problem. We explain in detail how each step can be solved. Although the presented algorithm will be written in pseudocode, every reader with basic programming skills can understand it or potentially implement it on his own.

The final chapter focuses on a smaller details that are not so essential to the

studied problem. We present a few observation about the edges of graph, which can be used, for instance, in the preprocessing stage of the algorithm stated in Chapter 3. Our primary focus lies on layered graphs, as they offer nontrivial preprocessing techniques.

1. Preliminaries

1.1 The shortest path problem

Throughout the entire thesis we consider a directed graph denoted as $G = (V, E)$, where V represents the set of its vertices and E represents the set of its edges. We assume that $|V| = n$, $|E| = m$, and $s, t \in V$.

In the graph, each edge is assigned a positive real number $w(e)$. We express these weights as vector $w = (w(e_1), w(e_2), \ldots, w(e_m))^T \in \mathbb{R}^m_+$. For any subset $A \subset E$, we define:

$$w(A) = \sum_{e \in A} w(e).$$

To enhance readability, let P denote the set of all paths in G that initiate at vertex s and terminate at vertex t. We recall the classical single-pair shortest path problem, which can be defined as follows:

Definition 1. *Let $G = (V, E)$ be a directed, weighted graph, $s, t \in V$ two of its vertices and $w \in \mathbb{R}^m_+$ a vector of weights. The shortest path problem from s to t in G is the problem of finding path $p \in P$ such that $w(p)$ is minimal among all paths in P.*

We denote the Shortest path problem shortly as SPP, and the optimal solution for this problem for given G, s and t as $SPP(G, s, t)$. In terms of optimization, SPP can be stated as

$$SPP(G, s, t) = arg \min_{p \in P} w(p).$$

We formulate SPP as an integer linear program. This program arises quite naturally by introducing the binary variable x_{uv} for every edge (u, v) of the graph. The value of x_{uv} expresses if the edge (u, v) belongs to a path, or not. We need to ensure that variables x_{uv} define a path in G starting in s and ending in t. This is stated in the equality constraints bellow, the objective function then asks for the path with minimal weight. Overall, we get the following program:

$$\begin{aligned}
\min \quad & \sum_{(u,v) \in E} w((u, v)) x_{uv} \\
\text{s.t.} \quad & \sum_{u:(u,s) \in E} x_{us} - \sum_{v:(s,v) \in E} x_{sv} = -1, \\
& \sum_{u:(u,t) \in E} x_{ut} - \sum_{v:(t,v) \in E} x_{tv} = 1, \\
& \sum_{u:(u,w) \in E} x_{uw} - \sum_{v:(w,v) \in E} x_{wv} = 0, \text{ for every } w \in V \setminus \{s, t\}, \\
& x_{uv} \in \{0, 1\}, \text{ for every } (u, v) \in E.
\end{aligned} \tag{1.1}$$

The optimal value of the program is equal to the weigh of the shortest path. The shortest path can be decoded from the variables x_{uv}:

$$SPP(G, s, t) = \{(u, v) \mid x_{uv} = 1\}.$$

We transform the above constraints into matrix equations using the incidence matrix of G, denoted as A_G. The matrix A_G is an element of $\{0, \pm 1\}^{n \times m}$ satisfying $(A_G)_{ue} = -1$ if u is a starting vertex of e, $(A_G)_{ue} = 1$ if u is an ending vertex of e and $(A_G)_{ue} = 0$ otherwise. With the additional assumption that the first row of A_G is incident with vertex s and the last one with t, we can transform (1.1) into

$$\begin{aligned} &\min\ w^T x \\ &\text{s.t. } A_G x = \begin{pmatrix} -1 \\ 0 \\ ... \\ 0 \\ 1 \end{pmatrix}, \\ &x \in \{0,1\}^m. \end{aligned} \tag{1.2}$$

This is a good expression of the problem, but we will make it even better. The first observation will be made about an upper bound on x. There is no sense in setting any x_{uv} higher than one due to the program structure. Thus, we can replace the constraint $x \in \{0,1\}^m$ with two modified constraints $x \geq 0$ and $x \in \mathbb{Z}^m$.

More importantly, we can get rid of constraints on integerity. Solving an integer program is in contrast of solving linear program a very hard task, actually it is an NP-hard problem. Fortunately, this is not the case for our problem. We will use two results from integer programming. Firstly, it holds that the incidence matrix of an oriented graph is totally unimodular. Secondly, an integer program with a totally unimodular matrix in constraints is equivalent to its linear relaxation. Problem (1.2) is therefore equivalent to the linear program

$$\begin{aligned} &\min\ w^T x \\ &\text{s.t. } A_G x = \begin{pmatrix} -1 \\ 0 \\ ... \\ 0 \\ 1 \end{pmatrix}, \\ &x \geq 0. \end{aligned} \tag{1.3}$$

Equivalence of programs (1.2) and (1.3) does not mean that every optimum of relaxation is integral. However, it guarantees that there exists an integral solution of the relaxed program which is an optimal for the relaxation. Moreover, every basic solution of the relaxation is integral, and the simplex method, which is the most used algorithm for solving linear programs, seeks a solution among these basic solutions. Hence, when solving (1.3) using the simplex method we obtain an integral solution for every instance. The basic solution and the simplex method are fundamental concepts of linear programming. As we will not use them further in the text, it is unnecessary to provide their definitions here. These definitions can be found in numerous books on linear programming.

Due to its significance, we denote the set of constraints for the shortest path problem and its relaxation using specific names. Let M_G denote the set of x that satisfy the constraints in (1.2), and similarly, let M'_G represent the set of

x that satisfy the constraints in (1.3). This notation will be consistently used throughout the entire thesis.

1.2 Multiobjective programming

We state a definition of a general multiobjective program and define some important terms related to this kind of programs. Then, we will mainly focus on multiobjective linear programs, but let us start with a general definition. A multiobjective program can be formulated as

$$\min_{x \in M} f(x) = \min_{x \in M} (f_1(x), f_2(x), \ldots, f_s(x))^T, \tag{1.4}$$

where M is the set of feasible solutions and s is a positive integer typically greater than one. The function f is a vector-valued function consisting of one-dimensional functions $f_1, f_2, \ldots, f_s$. The set M can be defined by equalities, inequalities, or even other constraints. We say that x is a feasible solution if $x \in M$.

To be precise, the problem (1.4) is not well-defined for $s > 1$. For instance, when considering two solutions x_1 and x_2 with respective function values $f(x_1) = (2, 5)^T$ and $f(x_2) = (4, 3)^T$, it becomes unclear how to determine the minimum of them. We introduce two definitions that partially resolve this issue:

Definition 2. *Let $a, b \in \mathbb{R}^s$, we define*
- $a \leq b$, if $a_i \leq b_i$, for $i = 1, \ldots, s$,
- $a \preceq b$, if $a \leq b$ and there is at least one $1 \leq j \leq s$ such that $a_j < b_j$,
- $a \prec b$, if $a_i < b_i$, for $i = 1, \ldots, s$.

With respect to the above definition, we distinguish three types of solutions for (1.4).

Definition 3. *Assume x_0 is a feasible solution for (1.4), then x_0 is*
- ideal, if $f(x_0) \leq f(x)$ for all $x \in M$,
- efficient, if there is no $x \in M$ such that $f(x) \preceq f(x_0)$,
- weak efficient, if there is no $x \in M$ such that $f(x) \prec f(x_0)$.
For x and y such that $f(x) \preceq f(y)$, we say that x dominates y.

Based on the given definition, we can resolve the issue at hand. Assume that $M = \{x_1, x_2\}$, where $f(x_1) = (2, 5)^T$ and $f(x_2) = (4, 3)^T$. As per the definition, neither of these solution is an ideal solution, but both of them are efficient.

The main goal of solving a multiobjective program is to find an ideal solution, but in the most cases no ideal solution exists. Therefore, we have to try to find at least efficient or weak efficient solutions, which are not as strong but still possess some useful properties.

The standard approach for solving multiobjective programs is scalarization. Obviously, scalarization is meaningful only when using non-negative weights that are not all zero. Two sets suitable for scalarization are the following ones:

$$S = \{\lambda \in \mathbb{R}^s \mid \lambda \succ 0\}, \bar{S} = \{\lambda \in \mathbb{R}^s \mid \lambda \succeq 0\}.$$

By taking the scalar product of λ from S, respectively $\bar{S}$, and $f(x)$, we can transform the program (1.4) into a single-objective program:

$$\min_{x \in M} \lambda^T f(x) = \min_{x \in M} \sum_{i=1}^{s} \lambda_i f_i(x). \tag{1.5}$$

Our primary focus will be on the linear multiobjective programs (MOLP). The programs, where the feasible set M is defined by linear inequalities and the objective functions are also linear. Hence, we can express the program (1.4) as follows:

$$\begin{aligned} &\min\ Cx \\ &\text{s.t. } Ax \leq b, \\ &\quad x \geq 0, \end{aligned} \tag{1.6}$$

where $C \in \mathbb{R}^{s \times n}$, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.

The efficient solutions of MOLP have a strong relationship with the optimal solutions for its scalarization. Specifically, the set of efficient solutions is equal to the union of optimal solutions of scalarizations with positive weights. This result is attributed to Iserman, and we will make use of it in the second chapter.

Theorem 1. *Let $\mathcal{E}$ be a set of efficient solutions for the MOLP* (1.6)*. Let $M_{opt}(\lambda)$ denote the set of optimal solutions for the scalarization of* (1.6) *using weight λ. Then*

$$\bigcup_{\lambda \in S} M_{opt}(\lambda) = \mathcal{E}.$$

We mention one additional result about MOLP, which we will also make use of in the second chapter. This result states that it is possible to evaluate the efficiency of a feasible solution using a linear program.

Theorem 2 (Charnes and Cooper)**.** *Let x_0 be a feasible solution for* (1.6)*. Consider a linear program:*

$$max\ e^T z,\ such\ that\ Ax \leq b,\ x \geq 0,\ Cx + z = Cx_0,\ z \geq 0. \tag{1.7}$$

Then x_0 is an efficient solution for (1.6) *if and only if the optimal value of* (1.7) *is equal to zero.*

At the end of the section we recall that linear programming is efficiently solvable. Several linear algorithms have been proven to run in polynomial time. Although the simplex method can have exponential time complexity in the worst case, on average it also runs in polynomial time and outperforms many other algorithms, which have better theoretical time complexity results. This is why the simplex method remains the most widely used tool in linear programming solvers. Nice survey about this issue is presented in [5].

1.3 Interval MOLP

We will further investigate MOLP. Particularly, we are now interested in interval MOLP (IMOLP). In the IMOLP, the matrix C can attain any value within the interval $[C^l, C^u]$. This means that the coefficient $c_{i,j}$ of C lies within the interval $[c^l_{i,j},\ c^u_{i,j}]$. While the elements of A and b can also have this property,

we will consider them as fixed real numbers. Hence, the uncertainty will solely appear in the matrix C. The formulation of an IMOLP is presented in the following definition:

Definition 4. *Let $\Psi = [C^l, C^u]$ be a set of $s \times n$ matrices. The interval multi-objective linear program is program in the form of* (1.6)*, where the matrix C can attain any value from Ψ.*

By changing the matrix C within Ψ we can obtain different optimal solutions. Thus, we distinguish between solutions that are optimal for every choice of C and solutions that are optimal for at least one choice of C.

Definition 5. *For the IMOLP* (1.6) *and a feasible solution x_0 we refer to x_0 as*
- necessarily efficient, if it is efficient for any $C \in \Psi$,
- possibly efficient, if it is efficient for at least one $C \in \Psi$,
- possibly weak efficient, if it is weak efficient for at least one $C \in \Psi$.

The necessary efficiency is clearly a stronger concept than the possible efficiency. Moreover, determining the necessary efficiency is much harder than the possible efficiency. Loosely speaking, problems involving the necessary efficiency are classified as NP-hard, whereas the same problems with the possible efficiency can be solved in polynomial time. This distinction is formally presented in the following three theorems. The theorem regarding the necessary efficiency can be found in [1]. Note that co-NP hardness is attained even in a specific case described in the theorem.

Theorem 3. *Checking if a given solution in an IMOLP is a necessarily efficient solution is a co-NP-complete problem on a class of IMOLP with one objective function and rational inputs.*

The following theorems focus on possible efficiency. The first is cited from [2], while the second one is taken from [3].

Theorem 4. *Checking the existence of possibly efficient solution in an IMOLP is a polynomial problem. Additionally, if there exists a possibly efficient solution, then finding such a solution can also be done in polynomial time.*

Theorem 5. *Checking whether a given solution to an IMOLP is possibly efficient can be done in polynomial time.*

1.4 Minimax regret

To conclude this chapter, we introduce one of the most important concept in IMOLP, known as the minimax regret criterion.

In the previous sections, we tried to compare vectors in $\mathbb{R}^s$, now we delve deeper into this comparison. Consider two solutions, x and y, to the MOLP problem (1.6). The inequality $Cy \leq Cx$ implies that x is a worse solution than y. But how much worse is x compared to y?

To answer this question, we need to introduce a weight vector $\lambda \in S$, where each component λ_i represents the importance assigned to the i-th objective. Observe that this concept is closely linked to scalarization.

We define the weighted regret of x with weights λ corresponding to $C \in \Psi$ and $y \in M$ such that $Cy \leq Cx$ as follows:

$$r(x, y, C) = \max\{\lambda_i c_i(x - y) \mid i = 1, \ldots, s\},$$

where c_i represents the $i-$th row of the matrix C.

This definition can serve as an intermediate step towards the weighted maximal regret of x. Both, the matrix C and the solution y, can vary in their respective sets, Ψ and M. The most interesting pair of $y \in M$ and $C \in \Psi$ is the one that maximizes the weighted regret, as defined above. Consequently, the weighted maximal regret with weights λ corresponding to x is defined as:

$$r(x) = \max\{\lambda_i c_i(x - y) \mid y \in M, C \in \Psi, Cy \leq Cx, i = 1, \ldots, s\}.$$

The matrix C' and vector y' satisfying $r(x, y', C') = r(x)$ are called the worst-case scenario for x and the worst-case pair for x. We denote them as C^x and y^x. The values C^x and y^x represent the worst-case possibilities for x, as their names suggest. When discussing the weighted maximal regret, we will interchangeably use shorter terms such as "the maximal regret", or simply "the regret".

The aim of minimizing regret is to find a feasible solution x with the lowest regret among all feasible solutions. To achieve this, we need to solve the following program:

$$\min_{x \in M} r(x) = \min_{x \in M} \max\{\lambda_i c_i(x - y) \mid y \in M, C \in \Psi, Cy \leq Cx, i = 1, \ldots, s\}.$$

The above definition will follow us throughout the whole thesis. In the second chapter, we investigate the maximal regret associated with a solution to the multiobjective shortest path problem with interval costs. The third chapter will be entirely dedicated to an algorithm seeking for the path with the lowest maximal regret. Detailed explanations of these terms and more complex statements regarding the maximal regret will be provided within these chapters.

2. Multiobjective shortest path problem with interval costs

Before going straight into the definitions, let us briefly motivate generalization of the shortest path problem. The shortest path problem, as defined in the first chapter, is searching for the least weighted path between two given vertices. A common real-world example is minimizing the time spent on a road from place A to place B. However, minimizing only one criterion may not be sufficient. For instance, imagine you want to travel from one place to another via public transport. The time of travelling is for sure the most important factor, but there are many other quantities to compare, such as the cost of the travelling, the number of transfers involved or the reliability of the connection. Moreover, not all these entities may be known precisely all the time. For example, the travel time may depend on the train schedule or a traffic situation.

There are of course many other areas where the shortest path problem can be useful to solve. In many of these areas, generalization makes sense as in our previous example. We won't list them here because it is not the goal of this thesis. Nonetheless, we hope that our given example is enough to deal with the problem stated in the following section.

2.1 Formulation of the problem

As stated in the first chapter, let $G = (V, E)$ be a directed graph, where V is the set of its vertices and E is the set of its edges. Assume $|V| = n$ and $|E| = m$. For every edge of the graph, we assign s weights, where s is a positive integer. Quantity s is equal to a number of criteria. Moreover each of these weights is not prescribed exactly, we only know that it lies within some given interval. Hence, we define weight vectors $w^l(e) \in \mathbb{R}^s_+$ and $w^u(e) \in \mathbb{R}^s_+$ satisfying $w^l(e) \leq w^u(e)$ and require $w(e) \in [w^l(e), w^u(e)]$ for each edge $e \in E$.

The shortest path problem now becomes more complex with this generalization. We formulate a program for finding the shortest path as we did with the classical SPP. The variables and constraints on these variables remain the same as in (1.2). The only difference is in the objective function. The vector c is replaced by the matrix $C \in \mathbb{R}^{s\times m}_+$ which is an element of $\Psi = [C^l, C^u]$, where C^l and C^u are defined as follows:

$$C^l = \left(w^l(e_1) \mid w^l(e_2) \mid \ldots \mid w^l(e_m)\right), C^u = \left(w^u(e_1) \mid w^u(e_2) \mid \ldots \mid w^u(e_m)\right).$$

To sum up, the program for finding the shortest path in graph G from the first vertex to the last one with weight function w can be formulated as

$$\begin{aligned} &\min Cx \\ &\text{s.t. } A_G x = \begin{pmatrix} -1 \\ 0 \\ ... \\ 0 \\ 1 \end{pmatrix}, \\ &x \in \{0,1\}^m, \end{aligned} \tag{2.1}$$

where $C \in \Psi$. For the purpose of the next sections, we will consider one particular instance of the above interval program. In the following program, matrix C_1 is fixed, which makes this program non-interval. Everything else remains the same as in (2.1):

$$\begin{aligned} &\min C_1 x \\ &\text{s.t. } A_G x = \begin{pmatrix} -1 \\ 0 \\ ... \\ 0 \\ 1 \end{pmatrix}, \\ &x \in \{0,1\}^m. \end{aligned} \tag{2.2}$$

Recall the notation from the previous chapter, where we denoted the set of x satisfying the above programs as M_G. When investigating the multiobjective shortest path problem (MOSP), we cannot simply interchange M_G with M'_G. There could be a problem with a path that is necessarily efficient among all solutions from M_G but not among M'_G. For example, consider three disjoint paths with costs equal to $(5,5)$, $(2,6)$, and $(6,2)$ respectively. Neither the second nor the third path dominates the first one, but a convex combination of these paths with coefficients 0.5 already dominates it. Hence, we are required to work with an integer program and utilize its linearization only in special cases.

2.2 Maximal regret and efficiency

In this section, we present theoretical results of multiobjective shortest path problem with interval costs (IMOSP). Our main focus lies on the regret and the necessary efficiency of a particular solution. We also show the close connection between these two concepts.

2.2.1 Maximal regret

Let us start with the regret, which was defined in the previous chapter in a general setting. Let $\hat{x}$ be an element of M_G, the regret of this solution is, by the definition, equal to:

$$r(\hat{x}) = \max\{\lambda_i c_i(\hat{x} - y) \mid y \in M_G, C \in \Psi, Cy \leq C\hat{x}, i = 1, \ldots, s\}.$$

It is easy to see that $r(\hat{x}) \geq 0$. Moreover, it should be intuitively obvious that the lower regret, the better. Therefore, our aim is to find a feasible solution with

the least regret. This leads us to the minimax problem:

$$\min_{x \in M_G} r(x) = \min_{x \in M_G} \max\{\lambda_i c_i(x - y) \mid y \in M_G, C \in \Psi, Cy \leq Cx, i = 1, \ldots, s\}. \tag{2.3}$$

Before delving into a deep study of the above program, we would like to introduce some notation. First of all, every solution from M_G uniquely determines a path in G from the first vertex to the last one, and vice versa. For a solution $\hat{x} \in M_G$, we denote $P_{\hat{x}}$ as the path induced by this solution, i.e.,

$$P_{\hat{x}} = \{e_i \mid \hat{x}_i = 1\}.$$

In the text, we will sometimes use the interchange of $\hat{x}$ and $P_{\hat{x}}$. For example, when we state that "$P_{\hat{x}}$ is a necessarily efficient path in G", it means that "$\hat{x}$ is a necessarily efficient solution" to the corresponding problem.

This bijection between paths and solutions helps us to make the following observation. Let us consider the best and the worst choices of matrices in Ψ for a given $\hat{x}$. It is not difficult to see that the best-case occurs when we set edge weights of $P_{\hat{x}}$ to the lower bound and the weights of other edges to the upper bound. Similarly, in the worst-case scenario, weights on $P_{\hat{x}}$ are set to the upper bound and those on other edges to the lower bound. Formally, we define two matrices $C^{\hat{x}}$ and $B^{\hat{x}}$ belonging to the worst-case and the best-case scenarios, respectively:

$$(C^{\hat{x}})_i = \begin{cases} (C^u)_i & \hat{x}_i = 1 \\ (C^l)_i & \hat{x}_i = 0 \end{cases}$$

$$(B^{\hat{x}})_i = \begin{cases} (C^l)_i & \hat{x}_i = 1 \\ (C^u)_i & \hat{x}_i = 0, \end{cases}$$

where the index i denotes the i-th column of the respective matrix. Although it may seem obvious, we have not yet proven that $C^{\hat{x}}$ is indeed the worst-case scenario. We do it now by proving the following theorem:

Theorem 6. *Let $\hat{x} \in M_G$, then*

$$r(\hat{x}) = \max\{\lambda_i c_i(\hat{x} - y) \mid y \in M_G, C^{\hat{x}} y \leq C^{\hat{x}} \hat{x}, i = 1, \ldots, s\}.$$

Proof. Let us denote $r'(\hat{x}) = \max\{\lambda_i c_i(\hat{x} - y) \mid y \in M_G, C^{\hat{x}} y \leq C^{\hat{x}} \hat{x}, i = 1, \ldots, s\}$. Our goal is to prove that $r(\hat{x}) = r'(\hat{x})$. The inequality $r(\hat{x}) \geq r'(\hat{x})$ is easy: it follows from the fact that in the definition of the maximal regret we consider all matrices $C \in \Psi$, while in $r'(\hat{x})$ we take in account only the matrix $C^{\hat{x}}$.

For the reverse inequality assume that $r(\hat{x})$ is achieved by $y^1 \in M_G$, $C^1 \in \Psi$ and $k \in \{1, \ldots, s\}$, i.e. $r(\hat{x}) = \lambda_k c^1_k(\hat{x} - y^1)$ and it holds that $C^1 y^1 \leq C^1 \hat{x}$. We will prove that $C^1 y^1 \leq C^1 \hat{x}$ implies $C^{\hat{x}} y^1 \leq C^{\hat{x}} \hat{x}$, meaning that y^1 and $C^{\hat{x}}$ are feasible for the definition of $r'(\hat{x})$.

Since we are comparing two vectors, we need to compare their corresponding indices. Let $i \in \{1, \ldots, s\}$ be arbitrary, the i-th element of $C^{\hat{x}} y^1 - C^{\hat{x}} \hat{x}$ is equal to:

$$c^{\hat{x}}_i y^1 - c^{\hat{x}}_i \hat{x} = \sum_{j=1}^{n} c^{\hat{x}}_{i,j}(y^1_j - \hat{x}_j). \tag{2.4}$$

If $\hat{x}_j = 0$, then $y^1_j \geq \hat{x}_j$, and according to the definition of $C^{\hat{x}}$ we derive $c^{\hat{x}}_{i,j} \leq c^1_{i,j}$. Combining these two inequalities we obtain

$$c^{\hat{x}}_{i,j}(y^1_j - \hat{x}_j) \leq c^1_{i,j}(y^1_j - \hat{x}_j). \tag{2.5}$$

If $\hat{x}_j = 1$, then $y^1_j \leq \hat{x}_j$, and according to the definition of $C^{\hat{x}}$ we derive $c^{\hat{x}}_{i,j} \geq c^1_{i,j}$. Combining these two inequalities results once again in (2.5). Therefore, inequality (2.5) holds for each j from 1 to n, and by summing over all j we obtain:

$$\sum_{j=1}^{n} c^{\hat{x}}_{i,j}(y^1_j - \hat{x}_j) \leq \sum_{j=1}^{n} c^1_{i,j}(y^1_j - \hat{x}_j) = c^1_i(y^1 - \hat{x}) \leq 0, \tag{2.6}$$

where the last inequality follows from the assumption that $C^1 y^1 \leq C^1 \hat{x}$. By combining (2.4) and (2.6), we conclude that

$$c^{\hat{x}}_i y^1 - c^{\hat{x}}_i \hat{x} \leq 0.$$

Since i was arbitrarily chosen, we obtain the desired result that $C^{\hat{x}} y^1 \leq C^{\hat{x}} \hat{x}$. If i is equal to k, (2.5) can be rewritten as:

$$c^{\hat{x}}_{k,j}(\hat{x}_j - y^1_j) \geq c^1_{k,j}(\hat{x}_j - y^1_j).$$

Summing over all j and multiplying by λ_k we achieve:

$$\lambda_k c^{\hat{x}}_k(\hat{x} - y^1) \geq \lambda_k c^1_k(\hat{x} - y^1).$$

Hence, we have shown that y^1, $C^{\hat{x}}$ are feasible for $r'(\hat{x})$ and have greater or equal objective value than y^1 and C^1, therefore

$$r'(\hat{x}) \geq \lambda_k c^{\hat{x}}_k(\hat{x} - y^1) \geq \lambda_k c^1_k(\hat{x} - y^1) = r(\hat{x}).$$

This establishes the reverse inequality and concludes the proof. □

Knowing the worst-case scenario matrix for $\hat{x}$ means that we only need to consider $C^{\hat{x}}$ instead of all choices of C from Ψ in the definition of regret. This simplifies the computation of regret for a given solution in our problem.

2.2.2 Necessary efficiency

Now, we shift our focus to the necessary efficiency. In the definition of the regret of $\hat{x}$, we have a condition stating that y dominates $\hat{x}$ in the scenario C, i.e. $Cy \leq C\hat{x}$. Although this condition does not need to be included, we include it because it makes more sense. Moreover, we can observe the relationship between the necessary efficiency and the maximal regret. The statement of the theorem is widely known and can be found, for instance, in [6].

Theorem 7. *Solution $\hat{x}$ is a necessarily efficient solution for* (2.1) *if and only if* $r(\hat{x}) = 0$.

Proof. Let $\hat{x}$ be a necessarily efficient solution for (2.1). This means that there are no $y \in M_G$ and no $C \in \Psi$ such that $Cy \preceq C\hat{x}$. Therefore, for all $y \in M_G$ and $C \in \Psi$ satisfying $Cy \leq C\hat{x}$, it holds that $Cy = C\hat{x}$. We make use of this observation and compute that

$$\begin{aligned} r(\hat{x}) &= \max\{\lambda_i c_i(\hat{x} - y) \mid y \in M_G, C \in \Psi, Cy \leq C\hat{x}, i = 1, \ldots, s\} \\ &= \max\{\lambda_i c_i(\hat{x} - y) \mid y \in M_G, C \in \Psi, Cy = C\hat{x}, i = 1, \ldots, s\} \\ &= \max\{\lambda_i \cdot 0 \mid y \in M_G, C \in \Psi, Cy = C\hat{x}, i = 1, \ldots, s\} \\ &= 0. \end{aligned}$$

Now assume $r(\hat{x}) = 0$, and for the sake of contradiction, also assume that $\hat{x}$ is not necessarily efficient for (2.1). By the definition of the necessary efficiency, there exist $\hat{y} \in M_G$ and $\hat{C} \in \Psi$ such that $\hat{C}\hat{y} \preceq \hat{C}\hat{x}$. We can easily compute that

$$r(\hat{x}) \geq \max\{\lambda_i \hat{c}_i(\hat{x} - \hat{y}) \mid i = 1, \ldots, s\} > 0,$$

which leads to the desired contradiction. □

The previous theorem confirms that definitions of the necessary efficiency and the maximal regret are reasonable. We strive to find solutions with the least regret or those that are necessarily efficient. According to the previous theorem, these efforts lead to the same solutions.

The task of finding a necessarily efficient solution is hard and potentially impossible in the cases where no necessarily efficient solution exists. In the following lines, we will focus on a slightly easier task, which is to test if a given solution is necessarily efficient. Once again, we make use of the observation about the worst-case scenario matrix and transform the task of checking the necessary efficiency of a solution for the IMOSP to checking the efficiency of a solution for non-interval MOSP. The proof of the theorem is a modification of the proof presented in [9]. We also acknowledge that we get the entire idea about the worst-case scenario matrix from this essay.

Theorem 8. *Solution $\hat{x}$ is a necessarily efficient solution to* (2.1) *if and only if $\hat{x}$ is an efficient solution to* (2.2) *with $C_1 = C^{\hat{x}}$.*

Proof. The forward implication is clear from the definition of necessary efficiency. According to this definition, $\hat{x}$ is efficient for any choice of C from Ψ, so specially it is efficient for $C^{\hat{x}}$.

Now assume that $\hat{x}$ is an efficient solution for (2.2) with $C_1 = C^{\hat{x}}$. Let $y \in M_G$ and $C \in \Psi$ be arbitrary. The efficiency of $\hat{x}$ in the scenario $C^{\hat{x}}$ implies that $C^{\hat{x}}y \not\preceq C^{\hat{x}}\hat{x}$. This means that either there exists a k such that $c_k^{\hat{x}}y > c_k^{\hat{x}}\hat{x}$, or that $c_i^{\hat{x}}y = c_i^{\hat{x}}\hat{x}$ holds for all i. We have two cases, and in both of them, we will prove that $Cy \not\preceq C\hat{x}$ and therefore, in consequence, that $\hat{x}$ is a necessarily efficient solution.

The first case assumes that $c_k^{\hat{x}}y > c_k^{\hat{x}}\hat{x}$, which is equivalent to

$$\sum_{j=1}^{m} c_{k,j}^{\hat{x}} y_j > \sum_{j=1}^{m} c_{k,j}^{\hat{x}} \hat{x}_j. \tag{2.7}$$

The right hand side of this equation is equal to

$$\sum_{j=1}^{m} c_{k,j}^{\hat{x}} \hat{x}_j = \sum_{j:\hat{x}_j=1} c_{k,j}^{\hat{x}} = \sum_{j:\hat{x}_j=1} c_{k,j}^{u} = \sum_{e_j \in P_{\hat{x}}} c_{k,j}^{u}.$$

Similarly the left hand side is equal to

$$\sum_{j=1}^{m} c^{\hat{x}}_{k,j} y_j = \sum_{e_j \in P_y} c^{\hat{x}}_{k,j} = \sum_{e_j \in (P_y \cap P_{\hat{x}})} c^{u}_{k,j} + \sum_{e_j \in (P_y \setminus P_{\hat{x}})} c^{l}_{k,j}.$$

Subtracting the same terms from these sums and utilizing (2.7), we obtain that

$$\sum_{e_j \in (P_y \setminus P_{\hat{x}})} c^{l}_{k,j} > \sum_{e_j \in (P_{\hat{x}} \setminus P_y)} c^{u}_{k,j}. \tag{2.8}$$

Observe that for any indexes q, r holds $c^{l}_{q,r} \leq c_{q,r} \leq c^{u}_{q,r}$. Therefore, we can derive the following set of inequalities, where the second one follows from (2.8) and the remaining ones from the aforementioned observation:

$$\sum_{e_j \in (P_y \setminus P_{\hat{x}})} c_{k,j} \geq \sum_{e_j \in (P_y \setminus P_{\hat{x}})} c^{l}_{k,j} > \sum_{e_j \in (P_{\hat{x}} \setminus P_y)} c^{u}_{k,j} \geq \sum_{e_j \in (P_{\hat{x}} \setminus P_y)} c_{k,j}.$$

The final step of the calculation is to add $\sum_{e_j \in (P_{\hat{x}} \cap P_y)} c_{k,j}$ to both sides of the equation. This leads to the inequality

$$\sum_{e_j \in P_y} c_{k,j} > \sum_{e_j \in P_{\hat{x}}} c_{k,j}. \tag{2.9}$$

As at the beginning of the proof, we can conclude that the left-hand side of the above inequality is equal to the k-th row of Cy, and the right-hand side is equal to the k-th row of $C\hat{x}$. Therefore, the above equation states $c_k y > c_k \hat{x}$ and thus $Cy \not\preceq C\hat{x}$.

The second case assumes that $c^{\hat{x}}_i y = c^{\hat{x}}_i \hat{x}$, for all i. Therefore, for all i holds that

$$\sum_{j=1}^{m} c^{\hat{x}}_{i,j} y_j = \sum_{j=1}^{m} c^{\hat{x}}_{i,j} \hat{x}_j.$$

In the similar manner as we derived (2.9), we can conclude that in this case the following inequality holds for all i:

$$\sum_{e_j \in P_y} c_{i,j} \geq \sum_{e_j \in P_{\hat{x}}} c_{i,j}.$$

This proves that in second case it also holds that $Cy \not\preceq C\hat{x}$. □

Unfortunately, even this does not make the problem efficiently solvable. In fact, checking path efficiency is an NP-hard problem, as we will demonstrate in the final section of this chapter.

Nonetheless, we still present two methods for checking the efficiency of a given solution. The first approach involves a straightforward use of Theorem 2. Although the theorem was formulated for linear programs, its applicability to mixed-integer programs is clearly possible as well. However, when working with integer values, we lose computational efficiency.

Theorem 9. *Let $\hat{x}$ be a feasible solution to* (2.2)*. Consider a mixed-integer linear program*

$$max\ e^T z,\ such\ that\ x \in M_G,\ C_1 x + z = C_1 \hat{x},\ z \geq 0. \tag{2.10}$$

Then $\hat{x}$ is an efficient solution for (2.2) *if and only if the optimal value of* (2.10) *is equal to zero.*

Hence, in order to check the efficiency of $\hat{x}$, we can solve (2.10). The optimal value obtained from this mixed-integer linear program provides the information regarding the efficiency.

The second approach is based on modifying Dijkstra's algorithm to handle the multiobjective shortest path problem. This modified algorithm is referred to as Multi-Dijkstra's algorithm. The complete algorithm was originally presented in [4], here we outline its main principles.

Similar to the classical Dijkstra's algorithm, we maintain a set of temporary labels and permanent labels for each vertex in the graph. However, in Multi-Dijkstra's algorithm, we update these labels slightly differently by considering all efficient paths to a given vertex, rather than just the one with the shortest weight. Another distinction is that we cannot terminate the computation upon reaching the target vertex. Instead, we continue until no temporary labels are available. These characteristics contribute to the computational inefficiency of the algorithm.

2.2.3 Possible efficiency and possible weak efficiency

Now, we get to possibly efficient solutions. This type of solutions still retains many useful properties, although the concept of the possible efficiency is not as strong as the necessary efficiency. In the first chapter, we stated the theorem concerning the efficient search for a possibly efficient solution for a general IMOLP. In the context of the IMOSP, this task remains straightforward. The following theorem is a formulation of Theorem 1 applied to our case.

Theorem 10. *Let $\lambda \in \mathbb{R}^s_+$, $C_1 \in \Psi$ and let $P_{\hat{x}}$ be one of the shortest paths from the first vertex to the last one in graph G with weight vector $\lambda^T C_1$. Then $\hat{x}$ is a possibly efficient solution for* (2.1).

Proof. We assume that path $P_{\hat{x}}$ is one of the shortest paths in G with weight vector $\lambda^T C_1$. This implies that $\hat{x}$ is an optimal solution to the linear program

$$\begin{aligned} &\min\ (\lambda^T C_1)x \\ &\text{s.t. } x \in M_G. \end{aligned} \tag{2.11}$$

Note that (2.11) is a scalarization of (2.2) with positive weights λ. Therefore, by Theorem 1, $\hat{x}$ is an efficient solution to (2.2). Finally, by the definition of possible efficiency, this implies that $\hat{x}$ is a possibly efficient solution for (2.1). □

In other words, the theorem states that solving classical SPP is sufficient for finding a possibly efficient solution. It is important to note that there is always at least one possibly efficient solution. This does not need to be the case for necessarily efficient solutions.

Although finding any possibly efficient solution can be achieved in polynomial time, the task of checking possible efficiency of a given solution remains a hard problem. It can be done similarly to testing the necessary efficiency. The only difference is that we consider the best-case scenario matrix instead of the worst-case scenario matrix.

Theorem 11. *Solution $\hat{x}$ is a possibly efficient solution for* (2.1) *if and only if $\hat{x}$ is an efficient solution for* (2.2) *with $C_1 = B^{\hat{x}}$.*

Proof. The proof can be done in a similar way to the proof of Theorem 8, therefore we will omit it here. □

To evaluate the efficiency of $\hat{x}$ in the above non-interval problem, we can again employ either Theorem 2 or Multi-Dijkstra's algorithm, just as we did in the previous subsection.

Let us revisit the minimax regret problem, focusing primarily on its optimal solution. Theorem 7 states that a solution is necessarily efficient if and only if its regret is zero. The solution with zero regret is definitely the optimal solution for the minimax regret problem. What happens if the optimal value of the minimax regret problem is positive? In such cases, there are no necessarily efficient solution by the aforementioned theorem. Nevertheless, we can still ask about the properties of the optimal solution. It holds that the optimal solution is always at least possibly weak efficient. The proofs of both theorems bellow can be found in [7].

Theorem 12. *If $\hat{x}$ is an optimal solution of* (2.3) *such that $r(\hat{x}) > 0$, then $\hat{x}$ is a possibly weak efficient solution of* (2.1).

In order to attain possible efficiency, we need to add a condition on the uniqueness of the solution.

Theorem 13. *If $\hat{x}$ is a unique optimal solution of* (2.3) *such that $r(\hat{x}) > 0$, then $\hat{x}$ is a possibly efficient solution of* (2.1).

The uniqueness of the solution is a necessary requirement. In the example below, we illustrate a weighted graph where the optimal solution to the minimax problem is not possibly efficient, but only possibly weak efficient. The example also presents concepts discussed in this chapter. Therefore, it can also serve as a guide in the swamp of many definitions and theorems we have already encountered.

2.2.4 Example

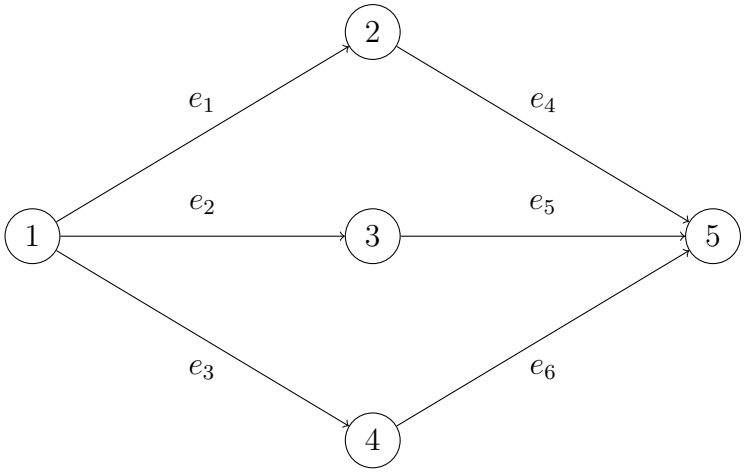
Example 1. Let $G_1 = (V_1, E_1)$, where

$$V_1 = \{1, 2, 3, 4, 5\},\ E_1 = \{(1, 2), (1, 3), (1, 4), (2, 5), (3, 5), (4, 5)\}.$$

The planar embedding of the graph G_1 is shown below. We denote the edges of the graph as e_i and will use this shorter version for edge names.

There are clearly three paths in G_1 from the first vertex to the last one, i.e., vertex 5. Namely, it is $P_1 = \{e_1, e_4\}$, $P_2 = \{e_2, e_5\}$, $P_3 = \{e_3, e_6\}$. These paths correspond to feasible solutions $x_1 = (1, 0, 0, 1, 0, 0)^T$, $x_2 = (0, 1, 0, 0, 1, 0)^T$, and $x_3 = (0, 0, 1, 0, 0, 1)^T$. Let P be the set of appropriate paths, hence $P = \{P_1, P_2, P_3\}$.

It remains to assign weights to the edges. We consider three criteria that are equally important. Using our notation, this can be expressed as $s = 3$ and $\lambda_1 = \lambda_2 = \lambda_3 = 1$. The weights of edges e_1, e_2, e_3 and e_5 are degenerate, which means that their lower weights are equal to their upper weights. We define their weights as follows:



$$w^l(e_1) = \begin{pmatrix} 5 \\ 5 \\ 1 \end{pmatrix}, w^l(e_2) = \begin{pmatrix} 5 \\ 5 \\ 2 \end{pmatrix}, w^l(e_3) = \begin{pmatrix} 5 \\ 5 \\ 1 \end{pmatrix}, w^l(e_5) = \begin{pmatrix} 15 \\ 15 \\ 2 \end{pmatrix}.$$

The weights of the two remaining edges are non-degenerate. Hence, we need to assign a lower weight and an upper weight for each of them:

$$w^l(e_4) = \begin{pmatrix} 5 \\ 15 \\ 1 \end{pmatrix}, w^u(e_4) = \begin{pmatrix} 15 \\ 15 \\ 1 \end{pmatrix}, w^l(e_6) = \begin{pmatrix} 15 \\ 5 \\ 1 \end{pmatrix}, w^u(e_6) = \begin{pmatrix} 15 \\ 15 \\ 1 \end{pmatrix}.$$

Now, we have all the input information and we can examine problems studied in this chapter. Firstly, we can express the matrices C^l and C^u. This only involves rewriting the weights accordingly:

$$C^l = \begin{pmatrix} 5 & 5 & 5 & 5 & 15 & 15 \\ 5 & 5 & 5 & 15 & 15 & 5 \\ 1 & 2 & 1 & 1 & 2 & 1 \end{pmatrix}, C^u = \begin{pmatrix} 5 & 5 & 5 & 15 & 15 & 15 \\ 5 & 5 & 5 & 15 & 15 & 15 \\ 1 & 2 & 1 & 1 & 2 & 1 \end{pmatrix}.$$

Let us start with the properties of feasible solutions. We will show that P_1 and P_3 are possibly efficient, while P_2 is not possibly efficient but at least it is possibly weak efficient. For P_1, we perform the complete calculation. The best-case scenario matrix for x_1 is equal to

$$B^{x_1} = \begin{pmatrix} 5 & 5 & 5 & 5 & 15 & 15 \\ 5 & 5 & 5 & 15 & 15 & 15 \\ 1 & 2 & 1 & 1 & 2 & 1 \end{pmatrix}.$$

Let us denote $w_1(P_i)$ as the weights of the path P_i in this scenario. Therefore, $w_1(P_i) = B^{x_1} x_i$ and it is easy to obtain that

$$w_1(P_1) = \begin{pmatrix} 10 \\ 20 \\ 2 \end{pmatrix}, w_1(P_2) = \begin{pmatrix} 20 \\ 20 \\ 4 \end{pmatrix}, w_1(P_3) = \begin{pmatrix} 20 \\ 20 \\ 2 \end{pmatrix}.$$

Clearly, P_1 is an efficient solution for this scenario, making it a possibly efficient solution. Similarly, we can show possible efficiency of P_3. It is worth noting

that P_1 and P_3 are symmetrical in terms of the first and second criterion, which also implies the possible efficiency of P_3.

Now consider the best-case scenario matrix for x_2, it is equal to

$$B^{x_2} = \begin{pmatrix} 5 & 5 & 5 & 15 & 15 & 15 \\ 5 & 5 & 5 & 15 & 15 & 15 \\ 1 & 2 & 1 & 1 & 2 & 1 \end{pmatrix}.$$

Let us denote $w_2(P_i)$ as the weight of the path P_i in this scenario. We can compute that

$$w_2(P_1) = \begin{pmatrix} 20 \\ 20 \\ 2 \end{pmatrix}, w_2(P_2) = \begin{pmatrix} 20 \\ 20 \\ 4 \end{pmatrix}, w_2(P_3) = \begin{pmatrix} 20 \\ 20 \\ 2 \end{pmatrix}.$$

Hence, we have $w_2(P_1) \preceq w_2(P_2)$, indicating that P_2 is not efficient for this scenario. According to Theorem 11, P_2 is not a possibly efficient solution. However, we observe that P_2 is a possibly weak efficient solution.

Moving from the efficiency of solutions, let's now focus on the maximal regret. We calculate the maximal regrets for all paths. This allows us to prove the desired claim that the solution to the minimax regret problem does not need to be possibly efficient. Let us begin with the regret of x_1. With a little help of Theorem 6 and reformulation of maximum, we obtain that

$$\begin{aligned} r(x_1) &= \max\{c_i^{x_1}(x_1 - y) \mid P_y \in P, C^{x_1}y \leq C^{x_1}x_1, i = 1,2,3\} = \\ &= \max_{y \in \{x_1,x_2,x_3\}} \max_{i=1,2,3} \{c_i^{x_1}(x_1 - y) \mid C^{x_1}y \leq C^{x_1}x_1\}. \end{aligned}$$

We can easily determine that the condition $C^{x_1}y \leq C^{x_1}x_1$ is not satisfied for $y = x_2$, but it is satisfied for both of remaining cases. The case when $y = x_1$ is trivial, as it is clear that $\max_{i=1,2,3}\{c_i^{x_1}(x_1 - x_1)\} = 0$. The second case is more interesting. It holds that

$$C^{x_1}(x_1 - x_3) = \begin{pmatrix} 5 & 5 & 5 & 15 & 15 & 15 \\ 5 & 5 & 5 & 15 & 15 & 5 \\ 1 & 2 & 1 & 1 & 2 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ -1 \\ 1 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 10 \\ 0 \end{pmatrix}.$$

From the above computation, it follows that $\max_{i=1,2,3}\{c_i^{x_1}(x_1 - x_3)\} = 10$. By plugging this value into the formula above, we can determine the maximal regret of x_1:

$$r(x_1) = \max_{y \in \{x_1,x_3\}} \max_{i=1,2,3} \{c_i^{x_1}(x_1 - y) \mid C^{x_1}y \leq C^{x_1}x_1\} = \max\{0, 10\} = 10.$$

Similarly, we could determine the regret of x_2 and x_3. However, for brevity, we skip calculations and proceed directly to the results, which are as follows

$$r(x_2) = 10,\ r(x_3) = 10.$$

All solutions have the same regret and therefore all of them are solutions with the least regret. In particular, x_2 is the optimal solution to the minimax regret problem. This answers the question of whether a solution that is only possibly weak efficient could be optimal for the minimax regret. Some may argue that x_2 is not an unique solution to the minimax problem. Indeed, according to Theorem 13, such a situation cannot occur.

Additionally, note that we determined the necessary efficiency of the possibly efficient solutions x_1 and x_3. Both of these solutions have a positive regret, and therefore, according to Theorem 7, they cannot be necessarily efficient.

2.3 Checking path efficiency is NP-hard

In the previous section, we dealt with a concept of the necessary efficiency and stated that checking whether a path in a graph is an efficient path is an NP-hard problem. In this section, we provide two reductions to establish this statement. The first reduction transforms the knapsack problem into the problem of finding a path in a graph with upper bounded weights. The second reduction goes from the problem of finding this path to checking the efficiency of a given path. By combining these two reductions with the well-known fact about NP-hardness of the knapsack problem, we obtain the desired claim.

Let us start with the knapsack problem. In this problem we are given n objects, where i-th objects has a weigh equal to b_i and a utility equal to a_i. Moreover, we can assume for simplicity that a_i and b_i are positive integers. The decision form of the knapsack problem asks if it is possible to choose a subset of objects with a total weight less than or equal to K_1 and a total utility greater than or equal to K_2. In other words, it poses the question about the existence of binary variables x_i satisfying the following conditions:

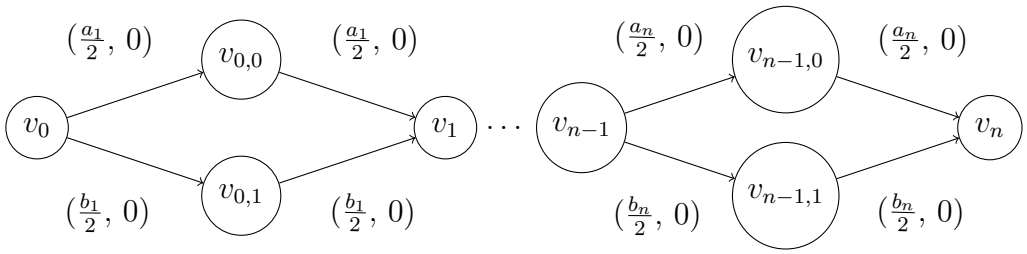
$$\begin{aligned} \sum_{i=1}^{n} a_i x_i &\geq K_2, \\ \sum_{i=1}^{n} b_i x_i &\leq K_1, \\ x_i &\in \{0,1\}. \end{aligned}$$

Theorem 14. *The decision form of the knapsack problem is an NP-hard.*

Now, we turn our attention to the first reduction, originally proposed by Serafini in his article [8]. The reduction is based on a simple yet brilliant idea, which we also use here. Given the values a_i, b_i, we construct a graph using the same principles as Serafini's construction, with only minor differences in formalism. Let us denote this graph as G_2 and define its set of vertices V_2, set of edges E_2, and weights of these edges as follows:

$$\begin{aligned} &V_2 = \{v_i, v_{i,0}, v_{i,1} \mid i = 0, 1, \ldots, n-1\} \cup \{v_n\}, \\ &E_2 = \{(v_i, v_{i,j}), (v_{i,j}, v_{i+1}) \mid i = 0, 1, \ldots, n-1, j = 0, 1\}, \\ &w((v_i, v_{i,0})) = w((v_{i,0}, v_{i+1})) = (\frac{a_{i+1}}{2}, 0)^T, i = 0, \ldots, n-1, \\ &w((v_i, v_{i,1})) = w((v_{i,1}, v_{i+1})) = (0, \frac{b_{i+1}}{2})^T, i = 0, \ldots, n-1. \end{aligned}$$

A planar embedding of graph G_2:



We are interested in investigation of the paths in G_2 from vertex v_0 to vertex v_n. Finding such path in G_2 with a weight less or equal to a given constant vector is equivalent to solving the knapsack problem:

Theorem 15. *Determining whether there exists a path in G_2 from v_0 to v_n such that $w(P) \leq (\sum_{i=1}^{n} a_i - K_2, K_1)$ is an NP-hard problem.*

Proof. We observe that from vertex v_i there are two options: we can either go through vertex $v_{i,0}$, or through $v_{i,1}$. Let x_{i+1} be a binary variable that determines whether we go through $v_{i,0}$, or $v_{i,1}$. If we pass through $v_{i,0}$, we set $x_{i+1} = 0$, otherwise we set $x_{i+1} = 1$. The values of x_1 up to x_n uniquely determine the path, which we denote as P_x. The weight of this path is equal to:

$$(\sum_{i=1}^{n} a_i(1 - x_i),\ \sum_{i=1}^{n} b_i x_i)^T.$$

Therefore, the path P_x has a weight less or equal to $(\sum_{i=1}^{n} a_i - K_2, K_1)$ if and only if

$$\sum_{i=1}^{n} a_i(1 - x_i) \leq \sum_{i=1}^{n} a_i - K_2,\ \sum_{i=1}^{n} b_i x_i \leq K_1.$$

Hence, in order to find a path satisfying the given condition, we need to search for $x_1, x_2, \ldots, x_n$ such that

$$\sum_{i=1}^{n} a_i x_i \geq K_2,$$
$$\sum_{i=1}^{n} b_i x_i \leq K_1,$$
$$x_i \in \{0, 1\}.$$

This is precisely the decision form of the knapsack problem. Based on the NP-hardness of this problem, it follows that finding a path in G with a weight less or equal to a given vector is also NP-hard problem. □

Now, let's consider a graph G_2 with an additional edge (v_0, v_n). We will denote this edge as e and refer to the modified graph as G_2'. Assign the weight of e as $(c_1 + 0.5, c_2 + 0.5)$, where c_1, c_2 are two positive integers. The second reduction arises from the following observation:

Lemma 16. *The path e is an efficient path in G_2' if and only if there is no path P_1 in G_2 such that $w(P_1) \leq (c_1, c_2)$.*

In conclusion, the NP-hardness of checking path efficiency is established.

Theorem 17. *The problem of determining if a path in a directed graph is an efficient path is an NP-hard problem.*

Proof. Consider the graph G_2' and assign weight of the edge e as

$$w(e) = (\sum_{i=1}^{n} a_i - K_2 + 0.5,\ K_1 + 0.5).$$

According to Lemma 16, the edge e is an efficient path if and only if there is no path P_1 in G_2 such that $w(P_1) \leq (\sum_{i=1}^{n} a_i - K_2,\ K_1)$. According to Theorem 15, proving the existence, and hence also the non-existence, of this path is NP-hard. Therefore, checking the efficiency of the path e in G is also NP-hard. □

Note that in G_2 and hence also in G_2' we allowed zero edge weights. This issue can be solved by adding any positive value ϵ to both criteria of all edges, and setting an upper bound of $w(P)$ as $(\sum_{i=1}^{n} a_i - K_2 + 2n\epsilon, K_1 + 2n\epsilon)$. We have intentionally omitted this in order to maintain the clarity of the construction.

3. An algorithm solving minimax regret problem in IMOSP

In this chapter we present an algorithm for solving the minimax regret problem in the IMOSP. We employ the algorithm from [7] which solves the minimax regret problem for general IMOLP. Building on the foundations laid in the previous chapter, we adapt this algorithm to our specific problem.

In the following section we outline the main idea of the algorithm and provide a detailed description of it. In Section 3.2, we analyze specific steps of the algorithm and propose possible solutions to our problem. In the remaining three sections, we will briefly discuss the strength of the outputted solution and the computational time complexity of the algorithm.

3.1 An algorithm for general interval MOLP

3.1.1 Main idea

Let us consider general IMOLP in the form:

$$\begin{aligned} &\min Cx \\ &\text{s.t. } x \in M, \end{aligned}$$

where $C \in \Psi = [C^l, C^u]$. Our goal is to find a feasible solution for this problem with the least regret among all feasible solutions. The minimax regret problem was introduced in the first chapter, we recall that it involves solving the following program:

$$\min_{x \in M} \max\{\lambda_i c_i(x-y) \mid y \in M, C \in \Psi, Cy \leq Cx, i = 1, \ldots, s\}. \tag{3.1}$$

The first thing we should do is to convert the above problem into a more convenient form. Adding a new variable σ is a simple way to achieve this:

$$\begin{aligned} &\min \sigma \\ &\text{s.t.} \max_{1 \leq i \leq s} \lambda_i c_i(x-y) \leq \sigma, \text{ if } C(y-x) \leq 0, y \in M, C \in \Psi, \\ &\quad x \in M, \sigma \geq 0. \end{aligned} \tag{3.2}$$

This is the problem with infinity number of constraints. The main idea of the algorithm in [7] is to take just the finite number of particular constraints and solve this relaxed version of the problem. By considering k specific choices of y from M, and C from Ψ we obtain:

$$\begin{aligned} &\min \sigma \\ &\text{s.t.} \max_{1 \leq i \leq s} \lambda_i c_i^h(x-y^h) \leq \sigma, \text{ if } C^h(y^h-x) \leq 0, h = 1, \ldots, k, \\ &\quad x \in M, \sigma \geq 0, \end{aligned} \tag{3.3}$$

where y^h are fixed elements from M and C^h are fixed elements from Ψ for all h. The key observation is the following theorem.

Theorem 18. *Let* (x^k, σ^k) *be an optimal solution for* (3.3) *and feasible for* (3.2). *Then it is an optimal solution for* (3.2) *and* x^k *is an optimal solution to the minimax regret problem* (3.1).

Proof. The statement is a simple observation following from the fact that the set of constraints in (3.3) is a subset of the constraints in (3.2). However, due to its importance, we will prove it in detail.

Assume, for a contradiction that (x^k, σ^k) is not optimal for (3.2). Therefore, there exists a feasible solution (x^1, σ^1) for (3.2) such that $\sigma^1 < \sigma^k$. Since (x^1, σ^1) is feasible for (3.2), it is also feasible for (3.3) due to the aforementioned relation between the constraints of the two programs. Moreover, (x^1, σ^1) has a lower objective value than (x^k, σ^k) and thus (x^k, σ^k) cannot be optimal for (3.3), resulting in the desired contradiction.

By derivation of (3.2), the above means that σ^k is the minimal regret of the problem and x^k is the solution that achieves this minimal regret. In other words, x^k is the optimal solution for (3.1). □

This provides a straightforward idea of the algorithm. We solve (3.3) and then check if the optimal solution is feasible for (3.2). If it is feasible, we have an optimum for (3.2) and therefore also for the minimax regret problem. If the optimum is not feasible, then there exists at least one violated constraint from (3.2). We identify the most violated constraint, add it to k constraints of (3.3), and solve the updated problem again.

3.1.2 The algorithm

In the following lines, we present the complete algorithm adapted from [7], although our algorithm will slightly differs from theirs. The difference arises primarily from their focus on a maximization problem, in contrast to our minimization problem. Another modifications are cosmetic adjustments.

Algorithm 1
Input: A set of feasible solutions M, an interval $\Psi = [C^l, C^u]$, and weights $\lambda_1, \ldots, \lambda_s$ for criteria, where $\lambda_i > 0$ for each i.
Output: The optimal solution for the minimax regret problem (3.1).

Step 1:
  for $i = 1, \ldots, s$:
    solve the linear programming problem $\min_{x \in M} c_i^l x$
    $x^{i*} \leftarrow$ the optimal solution of the i–th problem
  $x^{i_0 *} \leftarrow \arg\min_{1 \le i* \le s} c_i^l x^{i*}$
  $y^1 \leftarrow x^{i_0 *}$
  $C^1 \leftarrow C^l$
Step 2:
  $k \leftarrow 2$
  $\sigma^1 \leftarrow 0$
  $x^1 \leftarrow y^1$
Step 3:
  solve $\max\{\lambda_i c_i (x^{k-1} - y) \mid y \in M, C \in \Psi, C(y - x^{k-1}) \le 0, i = 1, \ldots, s\}$
  $\Phi^{k-1} \leftarrow$ optimal value of the above program

$(y^k, C^k) \leftarrow$ optimal solution of the above program

Step 4:

If $\Phi^{k-1} \leq \sigma^{k-1}$:

return x^{k-1}

Else:

solve min σ

s.t. $\max_{1 \leq i \leq s} \lambda_i c_i^h (x - y^h) \leq \sigma$, if $C^h(y^h - x) \leq 0, h = 1, \ldots, k$

$x \in M$, $\sigma \geq 0$

$(x^k, \sigma^k) \leftarrow$ optimal solution of the above program

$k \leftarrow k + 1$

go to Step 3

We can observe that Step 1 and Step 2 serve as initialization steps, the main part of the algorithm consists of Step 3 and Step 4. Step 3 is devoted to computing the regret of a fixed solution, whereas Step 4 involves solving the relaxed maximum regret problem.

3.2 An algorithm for interval MOSP

Now we shift back from the general IMOLP to the IMOSP. Therefore, the input now consists of a weighted graph G along with two of its vertices, and once again weights $\lambda_1, \ldots, \lambda_s$ for the criteria. Let us again recall the form of the program solving IMOSP:

$$\begin{aligned} &\min Cx \\ &\text{s.t. } A_G x = \begin{pmatrix} -1 \\ 0 \\ ... \\ 0 \\ 1 \end{pmatrix}, \\ &x \in \{0,1\}^m, \end{aligned} \tag{3.4}$$

where C is an element of $\Psi = [C^l, C^u]$, $C^l = \left(w^l(e_1) \mid \ldots \mid w^l(e_m)\right)$ and $C^u = \left(w^u(e_1) \mid \ldots \mid w^u(e_m)\right)$. Also recall that the set of feasible solutions of (3.4) was denoted as M_G. With this notation and knowledge from the second chapter, we can improve the steps of the algorithm. We will go step by step demonstrating how each one can be solved.

3.2.1 Step 1

The initial step is straightforward. It requires computing s linear programs and selecting minimum of their optimal values.

The i-th problem is equal to

$$\min_{x \in M_G} c_i^l x.$$

The objective function is fixed and one-dimensional. This means we obtained the classical shortest path problem with weights given by

$$c_i^l = ((w^l(e_1))_i, \ldots, (w^l(e_m))_i).$$

While it can be solved as a linear program, it is more efficient to use algorithm solving single-pair shortest path problem, e.g. Dijkstra's algorithm. Hence, Dijkstra's algorithm provides the shortest path of the i-th problem. The second part of the step involves comparing the optimal values of each of these s shortest paths.

Overall, Step 1 can be solved very efficiently. Step 2 is just an assignment, so there is nothing to study and we can proceed to more complex steps.

3.2.2 Step 3

The central aspect of Step 3 lies in computing the regret of a given solution x^{k-1}. Hence, we are solving the following problem:

$$\max\{\lambda_i c_i(x^{k-1} - y) \mid y \in M_G, C \in \Psi, C(y - x^{k-1}) \leq 0, i = 1, \ldots, s\}.$$

Let us denote the value of the above problem as $r(x^{k-1})$. In the previous chapter, we studied the problem of the maximum regret and one of the main results was Theorem 6, which states that

$$r(x^{k-1}) = \max\{\lambda_i c_i^{x^{k-1}}(x^{k-1} - y) \mid y \in M_G, C^{x^{k-1}}(y - x^{k-1}) \leq 0, i = 1, \ldots, s\},$$

where $C^{x^{k-1}}$ is the worst-case scenario matrix for x^{k-1} defined in the text before Theorem 6. This simplifies our task.

The next simplification involves dividing the problem into s subproblems. The i-th subproblem can be expressed in the language of optimization as follows:

$$\begin{aligned} &\max \lambda_i c_i^{x^{k-1}} x^{k-1} - \lambda_i c_i^{x^{k-1}} y \\ &\text{s.t. } C^{x^{k-1}}(y - x^{k-1}) \leq 0, \\ &\qquad y \in M_G. \end{aligned} \tag{3.5}$$

We transform this problem into the equivalent problem in order to find its optimal solution y^{i*}. Firstly, we get rid of constant term $\lambda_i c_i^{x^{k-1}} x^{k-1}$ in the objective function, which does not affect an optimality of solutions. By same reasoning we can also get rid of a positive multiplicative factor λ_i before non-constant term in the objective. Finally, we transform the maximization problem into the minimization problem. Therefore, we have come to the problem

$$\begin{aligned} &\min c_i^{x^{k-1}} y \\ &\text{s.t. } C^{x^{k-1}} y \leq C^{x^{k-1}} x^{k-1}, \\ &\qquad y \in M_G. \end{aligned} \tag{3.6}$$

Note also that we slightly change inequality constraints for better understanding of the following thoughts.

Problem (3.6) is classical shortest path problem but with an additional constraint on y dominating x^{k-1} in the scenario $C^{x^{k-1}}$. To solve this problem, we can use any solver for mixed integer programming. This is of course a correct approach, but it does not take advantage of any additional properties of our problem. An alternative and potentially more efficient method is to use Yen's algorithm as presented in [10].

Yen's algorithm is a generalization of algorithms solving the shortest path problem. It not only identifies the shortest path in the graph from the starting vertex to the ending vertex, but also finds k shortest paths for any positive integer k. Additionally, it outputs the shortest path first, followed by the second shortest paths, the third one, and so forth. This property makes it an ideal algorithm for our problem.

To solve (3.6) we can simply use Yen's algorithm in G with weights equal to

$$c_i^{x^{k-1}} = ((w^{x^{k-1}}(e_1))_i, (w^{x^{k-1}}(e_2))_i, \ldots, (w^{x^{k-1}}(e_m))_i).$$

For each path outputted by the algorithm, we check if it satisfies the dominance condition. The first path satisfying this condition will be the optimum for (3.6). This process surely terminates, because there is at least one feasible solution for (3.6), namely x^{k-1}.

Let us denote y^{i*} the optimal solution for (3.6), the regret of x^{k-1} is then equal to

$$r(x^{k-1}) = \max\{\lambda_i c_i^{x^{k-1}}(x^{k-1} - y^{i*}) \mid i = 1, 2, \ldots, s\}.$$

As in Step 1, this simply involves selecting the maximum from the set of s fixed real numbers.

3.2.3 Step 4

The final step of the algorithm corresponds to the problem (3.3). Given matrices $C^h \in \Psi$ and vectors $y^h \in M_G$, for $h = 1, 2, \ldots, k$, our goal is to find a feasible x that minimizes the relaxed maximum regret. Thus, we are solving the problem:

$$\begin{aligned} &\min \sigma \\ &\text{s.t.} \max_{1 \le i \le s} \lambda_i c_i^h (x - y^h) \le \sigma, \text{ if } C^h(y^h - x) \le 0, h = 1, \ldots, k, \\ &\quad x \in M_G, \sigma \ge 0, \end{aligned} \tag{3.7}$$

When solving this problem, we will follow the same method as employed in [7]. We will also use logical operator $\Rightarrow$ and $\vee$ as they do in the aforementioned article. The expression "A $\Rightarrow$ B" signifies "if A, then B", while "A $\vee$ B" means that "A holds or B holds".

The problematic part of (3.7) is the inequality constraint involving σ, which must be satisfied when y^h dominates x in scenario C^h. We want to reformulate (3.7) without employing any logical operators. To achieve this, we will introduce new binary variables. But before that, a slight modification of the constraint is required. The constraint ensures that the following must be met for each h:

$$C^h(y^h - x) \le 0 \Rightarrow \max_{1 \le i \le s} \lambda_i c_i^h (x - y^h) \le \sigma. \tag{3.8}$$

The left-hand side consists of s inequalities. We write each of them individually. A similar procedure will be applied to the right-hand side - an inequality with the max operator will be rewritten as s inequalities. This refinement will transform (3.8) into

$$(c_i^h(y^h - x) \le 0, \forall i = 1, \ldots, s) \Rightarrow (\lambda_i c_i^h (x - y^h) \le \sigma, \forall i = 1, \ldots, s).$$

Now, we utilize the well-known fact that "A $\Rightarrow$ B" is equivalent to "not A $\vee$ B". Furthermore, let us denote the set $\{1, 2, \ldots, s\}$ as S. With this notation and the aforementioned fact from propositional calculus, we can rewrite the above proposition into the form:

$$(\exists i \in S : c_i^h(y^h - x) > 0) \vee (\lambda_i c_i^h(x - y^h) \leq \sigma \ \forall i \in S).$$

The first part can be expressed as a disjunction of s inequalities. Following this approach, we eliminate the existential quantifier and obtain that

$$(c_1^h(y^h - x) > 0) \vee \ldots \vee (c_s^h(y^h - x) > 0) \vee (\lambda_i c_i^h(x - y^h) \leq \sigma, \forall i \in S). \quad (3.9)$$

Finally, it is time to introduce the binary variables b_i^h, for $i = 1, \ldots, s$. The variable b_i^h determines whether the inequality $c_i^h(y^h - x) > 0$ is satisfied. Additionally, we introduce another variable denoted as K, which is a positive real number. It is important to choose K sufficiently large such that $c_i^h(y^h - x) > -K$ holds for any value of x. The choice of K will be discussed at the end of this section. We define the equation

$$c_i^h(y^h - x) > -Kb_i^h.$$

If the original equation is not satisfied, we need to assign $b_i^h = 1$. If none of the s equations $c_i^h(y^h - x) > 0$ holds, then it must be the case that $\lambda_i c_i^h(x - y^h) \leq \sigma$ for all i from S. In this case, all b_i^h are equal to one and we require the validity of the inequalities involving σ. Conversely, if any of the inequalities $c_i^h(y^h - x) > 0$ is satisfied, there are no objections to other inequalities, even these involving σ. A simple trick allows us to achieve this condition and enables us to rewrite (3.9) as follows

$$\begin{aligned} & c_i^h(y^h - x) > -Kb_i^h, \ i \in S, \\ & \lambda_i c_i^h(x - y^h) \leq \sigma + \lambda_i K(s - \sum_{j=1}^{s} b_j^h), \ i \in S. \end{aligned}$$

We have already eliminated logical operators. The final issue we have to deal with is the strict inequalities, which are not well-suited for program solvers. To achieve this, we can introduce a small positive value denoted as ϵ, and add it to the right-hand sides of the inequalities. With this adjustment, we have successfully reformulated the constraint in (3.7), and we can substitute it into this program. Putting it all together we obtain a mixed integer program

$$\begin{aligned} & \min \ \sigma \\ & \text{s.t. } c_i^h(y^h - x) + Kb_i^h \geq \epsilon, \ i = 1, \ldots, s, \ h = 1, \ldots, k, \\ & \quad \lambda_i c_i^h(x - y^h) - \lambda_i K(s - \sum_{j=1}^{s} b_j^h) \leq \sigma, \ i = 1, \ldots, s, \ h = 1, \ldots, k, \\ & \quad \sigma \geq 0, \ x \in M_G, \ b_i^h \in \{0, 1\}, \ i = 1, \ldots, s, \ h = 1, \ldots, k. \end{aligned} \quad (3.10)$$

To solve this program, we can employ any integer program solver. Note that the matrix of constraints is significantly more complicated compared to the matrix

from the shortest path problem. In general, the matrix is not unimodular, which means we cannot relax the condition on the binary variables x. Also note that for a set of k given vectors y^h and matrices C^h, the resulted program involves $ks + m + 1$ variables. As a consequence, the difficulty of the program increases with a large value of k.

To conclude the section, we derive which choice of K is sufficiently large. We require K to satisfy the following equation for all values of i and h:

$$c_i^h(y^h - x) > -K.$$

Our aim is clear: we will estimate a lower bound of the left-hand side. Upon obtaining the expression, which is always less than the left-hand side, we simply define K as this expression. Given the conditions that $x, y \succeq 0$ and $c^h \succ 0$, it holds that:

$$c_i^h y^h - c_i^h x > -c_i^h x \geq -c_i^u x.$$

We rewrite the scalar product $c_i^u x$ as the sum and estimate it from above. Recall that x_j are binary variables.

$$\sum_{j=1}^{m} c_{i,j}^u x_j \leq \sum_{j=1}^{m} c_{i,j}^u \leq \sum_{j=1}^{m} \max_{\substack{1 \leq i \leq s \\ 1 \leq j \leq m}} c_{i,j}^u = m \cdot \max_{i,j} c_{i,j}^u.$$

The two previous sets of inequalities, when combined, yield that

$$c_i^h y^h - c_i^h x > -m \cdot \max_{i,j} c_{i,j}^u.$$

Therefore, it is sufficient to choose K as

$$K = m \cdot \max_{i,j} c_{i,j}^u.$$

In other words, we defined K as the product of the largest value in the matrix C^u and the number of edges. Although tighter estimations are possible, they are not crucial in this context. For instance, we can assign different values of K_i for each i. However, to keep things simple, we stick with our value of K derived above.

Similarly, we can derive an appropriate choice for ϵ. We won't follow the entire process of deriving this bound, the derivation is similar to the above one. Without providing a formal prove, we state that we can choose ϵ as follows:

$$\epsilon = \min_{i,j} c_{i,j}^l.$$

3.2.4 The algorithm

In the previous section, we provided a detailed description of how the steps of the Algorithm 1 can be utilized to solve the shortest path problem. Now, we add everything together and present the algorithm for solving the minimax regret problem in the IMOSP. The input to the algorithm is a graph G with interval weights on its edges and criteria weights λ indicating the relative importance of each criterion.

Algorithm 2
Input: Graph G with edges weights in the interval matrix $[C^l, C^u]$ and criteria weights $\lambda = (\lambda_1, \ldots, \lambda_s)$.
Output: The optimal solution to the minimax regret problem (2.3).

Step 1:
  for i = 1 to s:
    solve the shortest path problem with weights c_i^l
    $x^{i*} \leftarrow$ the shortest path
  $i_m \leftarrow \arg\min_{1 \le i* \le s} c_i x^{i*}$
  $y^1 \leftarrow x^{i_m}$
  $C^1 \leftarrow C^l$
Step 2:
  $k \leftarrow 2$
  $\sigma^1 \leftarrow 0$
  $x^1 \leftarrow y^1$
Step 3:
  for i = 1 to s:
    Dominance $\leftarrow$ False
    $l \leftarrow 1$
    while Dominance = False:
      $z^i \leftarrow$ the l–th shortest path, weights are $c_i^{x^{k-1}}$
      $l \leftarrow l + 1$
      if $C^{x^{k-1}} z^i \le C^{x^{k-1}} x^{k-1}$:
        Dominance $\leftarrow$ True
  $i_m \leftarrow \arg\min_{1 \le i \le s} \lambda_i c_i^{x^{k-1}} z^i$
  $(y^k, C^k) \leftarrow (z^{i_m}, C^{x^{k-1}})$
  $\phi^{k-1} \leftarrow \lambda_i c_{i_m}^{x^{k-1}} z^{i_m}$
Step 4:
  if $\phi^{k-1} \le \sigma^{k-1}$:
    return x^{k-1}
  else:
    solve $\min \sigma$
    s.t. $c_i^h(y^h - x) + K b_i^h \ge \epsilon$
      $\lambda_i c_i^h (x - y^h) - \lambda_i K (s - \sum_{j=1}^s b_j^h) \le \sigma$
      $\sigma \ge 0,\ x \in M_G,\ b_i^h \in \{0,1\},\ i = 1, \ldots, s,\ h = 1, \ldots, k$
    $(x^k, \sigma^k) \leftarrow$ optimal solution of the above program
    $k \leftarrow k + 1$
    go to Step 3

In the description of the algorithm, by solving the shortest path problem we of course mean the shortest path problem in G from the first vertex to the last one. The solution to more complex commands was discussed in the previous section. We remind that the single shortest path problem in Step 1 can be solved using Dijkstra's algorithm, finding the l-th shortest path can be accomplished by Yen's algorithm, and the mixed integer program in Step 4 can be tackled using any mixed integer program solver.

3.3 Property of outputted solution

In this section, we briefly examine the characteristics of the algorithm's outputted solution. In particular, we determine whether the solution is necessarily efficient, possibly efficient, or possibly weak efficient. The groundwork for these determinations was laid out in the second chapter, now we just utilize the knowledge gained there.

Let x^o and σ^o be the output of the Algorithm 2, i.e. x^o is the solution with the least regret and σ^o is the regret of this solution. We state three theorems that determine the character of the solution x^o. At the end of the section, we present a simple algorithm that incorporates these theorems.

First of all, according to Theorem 12, x^o is always at least possibly weak efficient. Additionally, based on the definitions, a necessarily efficient solution is also possibly efficient, and a possibly efficient solution is also possibly weak efficient. Therefore, our task is to determine the specific type to which the solution belongs among these three possibilities.

Theorem 19. *If $\sigma^o = 0$, then x^o is a necessarily efficient solution for* (2.1)*.*

Proof. It follows directly from Theorem 7. □

The determination of the possible efficiency depends on the efficiency in the best-case scenario. To make a decision about this efficiency, we define the following program:

$$\begin{aligned} &\max\ e^T z \\ &\text{s.t. } B^{x^o} x + z = B^{x^o} x^o, \\ &\qquad x \in M_G,\ z \geq 0. \end{aligned} \tag{3.11}$$

The optimal value of this program decides about possible efficiency and possible weak efficiency in the following way:

Theorem 20. *If $\sigma^o > 0$ and the optimal value of* (3.11) *is zero, then x^o is a possibly efficient solution for* (2.1)*, which is not necessarily efficient for this program.*

Proof. According to Theorem 7, x^o is not necessarily efficient. By Theorem 9, x^o is an efficient solution for (2.2) with $C_1 = B^{x^o}$. Therefore, x^o is a possibly efficient solution for (2.1). □

There remains the last case, when σ is positive and optimal value of (3.11) is also positive. If this happens, x^o is only possibly weak efficient:

Theorem 21. *If $\sigma^o > 0$ and the optimal value of* (3.11) *is greater than zero, then x^o is a possibly weak efficient solution for* (2.1)*, which is not possibly efficient for this program.*

Proof. The possible weak efficiency follows from Theorem 12. The part with the possible efficiency is the conclusion of Theorem 11 and Theorem 9. □

Previous observations lead us to the following algorithm. We can use this algorithm to determine a strength of the solution outputted by Algorithm 2.

Algorithm 3
Input: x^o and σ^o outputted by Algorithm 2.
Output: efficiency of x^o.

```
if σ^o = 0:
    return ''x^o is necessarily efficient''
else:
    solve (3.11)
    m* ← the optimal value of (3.11)
    if m* = 0:
        return ''x^o is possibly efficient''
    else:
        return ''x^o is possibly weak efficient''
```

By understanding the efficiency of the optimal solution, we also acquire valuable information about other solutions as well. For instance, if the optimal solution is possibly efficient but not necessarily efficient, then there is no necessarily efficient solution. This observation follows from Theorem 7.

However, if the optimal solution is possibly weak efficient, we cannot say anything about possibly efficient solutions. At the end of the second chapter, we provided the example, where the optimal solution was possibly weak efficient, despite the existence of two other solutions that were possibly efficient.

3.4 Time complexity

In this section, we discuss the time complexity of operations in Algorithm 2. We will not calculate the complexity of the entire algorithm due to the presence of several instances of mixed integer programs. These programs are difficult to solve and even harder to estimate their time complexity.

In our examination, the complexity of the algorithm depends on the values of inputs, namely the number of vertices denoted as n, and the number of criteria denoted as s. Additionally, we consider the number of paths in the graph from the source vertex to the target vertex, denoted as p. This number is finite, but it can be exponentially large in comparison to n.

Starting from Step 1 of the algorithm, we observe that the nontrivial operation in this step is solving the classical SPP. It is known that Dijkstra's algorithm is capable of solving SPP in $O(n^2)$ time complexity. In our case, we need to solve s instances of this problem, resulting in a total time complexity of $s \cdot O(n^2) = O(sn^2)$. The second step solely consists of an assignment, which can be done in constant time.

Step 3 is similar to the first step, with the difference in utilization of Yen's algorithm instead of Dijkstra's algorithm. In the original paper by Yen [10], it is stated that finding k shortest paths requires approximately $\frac{1}{2}kn^3$ operations. However, the problem lies in not knowing the duration of the search for an appropriate path. Therefore, although k is typically small, the only estimation we

can derive is the obvious upper bound obtained by the number of paths, p. As a result, the time complexity of Step 3 can be bounded as $O(spn^3)$.

Even more challenging situation occurs in Step 4, where we are required to solve a mixed integer program. It is widely known that solving integer programs is generally NP-hard. While this does not necessarily imply that the program in Step 4 is also difficult to solve, we currently lack a better estimation for the program. Let us just state that in the k-th iteration, the program under consideration has $sk + m + 1$ variables and $2sk + n + 1$ constraints. Hence, with every iteration, the complexity of the program increases.

The final issue is about the number of returns to Step 3. There are infinite number of scenarios, hence the number of returns can also be infinite. However, this is not the case, because there are only finite number of paths and if we were to return to Step 3 with the same path twice, we would already come to halt. This observation is formally proven in the following theorem. As a corollary, we deduce that the number of returns to Step 3 is bounded above by p.

Theorem 22. *Consider the solution (x^k, σ^k) obtained in the k-th iteration of Step 4, and the solution (x^l, σ^l) obtained in the l-th iteration, where $1 < k < l$ and $x^k = x^l$. In this case, Algorithm 2 terminates in the $(l+1)$-th iteration, outputting x^l.*

Proof. If (x^k, σ^k) is the optimal solution to the program in Step 4, then after returning to Step 3, we compute the regret of this solution in this step. Additionally, within this step, we determine the worst-case scenario and worst-case pair for x^k, and add the constraints on them to the relaxed maximal regret problem being solved in Step 4. As a result, during the l-th iteration of Step 4, we compute the exact regret of x^l, which is equal to σ^l. Clearly, the same regret comes up from the computation after returning to Step 3. Hence, the condition in if-else branch is fulfilled during the $(l+1)$-th iteration, yielding the output x^l. □

3.5 Example

Let us illustrate the execution of Algorithm 2 on a particular instance. Let's consider a graph G_1 from the example provided at the end of the second chapter. In this graph, there exists three appropriate paths, hence there are three solutions that could be the solution with the least regret. We already know that each solution has the value of regret equals to ten, indicating that all of them are optimal. Nonetheless, it is still worth to observe how the computation through Algorithm 2 works.

In the difference to the mentioned example, let us now consider that $\lambda_1 = 1$, $\lambda_2 = 1$, and $\lambda_3 = 10$. The execution of Algorithm 2 will proceed as follows:
Step 1: Finding the shortest path in the scenario c_i^l for $i = 1, 2, 3$:
$i = 1$: The optimum is x_1 with a value of $c_1^l x_1 = 10$.
$i = 2$: The optimum is x_3 with a value of $c_2^l x_3 = 10$.
$i = 3$: The optimum is x_1 with a value of $c_3^l x_1 = 2$.
Selecting the argument of the minimum value: $i_m = 1$.
Setting $(y^1, C^1) = (x_1, C^l)$.
Step 2:

$k = 2$, $\sigma^1 = 0$, $x^1 = x_1$.

Step 3: Finding the shortest paths in the scenario C^{x_1}, which also satisfy the dominance condition:

$i = 1$: The shortest path is x_1, and it satisfies the condition.

$i = 2$: The shortest path is x_2, and it satisfies the condition.

$i = 3$: The shortest path is x_1, and it satisfies the condition.

Selecting the minimum of $\{\lambda_1 c_1^{x_1} x_1, \lambda_2 c_2^{x_1} x_2, \lambda_3 c_3^{x_1} x_1\}$, which is equal to the minimum of $\{20, 10, 20\}$.

Therefore $i_m = 3$ and $\phi^1 = 10$.

Setting $(y^2, C^2) = (x_3, C^{x_1})$.

Step 4: Since $\phi^1 = 10 \not\leq 0 = \sigma^1$, we need to solve the mixed-integer program. (The form of this program is for illustration purposes listed below the example.)

The optimal solution is $x = x_3$, $\sigma = 0$.

Setting $x^2 = x_3$, $\sigma^2 = 0$, $k = 3$ and going to Step 3.

Step 3: Similar to the first execution of this step, but now considering the scenario C^{x_3}.

We obtain $i_m = 1$ and $\phi^2 = 10$.

Setting $(y^3, C^3) = (x_1, C^{x_3})$.

Step 4: Once again $\phi^2 \not\leq \sigma^2$, hence we proceed to solve another mixed-integer program.

The optimal solution is $x = x_1$, $\sigma = 10$.

Setting $x^3 = x_1$, $\sigma^3 = 10$, $k = 4$ and going to Step 3.

Step 3:

We obtain $i_m = 1$ and $\phi^3 = 10$.

Setting $(y^4, C^4) = (x_3, C^{x_1})$.

Step 4: Since $\phi^3 \leq \sigma^3$, the algorithm outputs x_1, with the regret equal to $\sigma^3 = 10$.

Let us now present the mixed-integer program obtained in the first execution of Step 4. We have $\lambda = 3$ and $k = 2$, and the two constraints given by $(y^1, C^1) = (x_1, C^l)$ and $(y^2, C^2) = (x_3, C^{x_1})$. This by the way means that the program consists of 13 variables and 18 constraints. The program takes the following form:

$$
\begin{aligned}
&\min\ \sigma \\
&\text{s.t. } c_i^l(x_1 - x) + Kb_i^1 \geq \epsilon,\ i = 1,2,3, \\
&\quad c_i^{x_1}(x_3 - x) + Kb_i^2 \geq \epsilon,\ i = 1,2,3, \\
&\quad \lambda_i c_i^l(x - x_1) - \lambda_i K(3 - \sum_{j=1}^{3} b_j^1) \leq \sigma,\ i = 1,2,3, \\
&\quad \lambda_i c_i^{x_1}(x - x_3) - \lambda_i K(3 - \sum_{j=1}^{3} b_j^2) \leq \sigma,\ i = 1,2,3, \\
&\quad \begin{pmatrix} -1 & -1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} x = \begin{pmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \\
&\quad \sigma \geq 0,\ x \in \{0,1\}^6,\ b_i^h \in \{0,1\},\ i = 1,2,3,\ h = 1,2.
\end{aligned}
$$

The values c_i^l, $c_i^{x_1}$, x_1, and x_3 are known and therefore we can substitute the

corresponding real numbers for them. Considering the discussion about sufficiency of value K and ϵ, we can choose K as $6 \cdot 15 = 90$ and ϵ as 1.

We will not rewrite the entire program here, but we will demonstrate the substitution for two of the inequalities. After substitution, the inequality on the first row for $i = 1$ and the inequality on the fourth row for $i = 3$ become:

$$\begin{aligned} \begin{pmatrix} 5 & 5 & 5 & 5 & 15 & 15 \end{pmatrix} x - 90b_1^1 &\leq 9, \\ \begin{pmatrix} 10 & 20 & 10 & 10 & 20 & 10 \end{pmatrix} x + 900b_1^2 + 900b_2^2 + 900b_3^2 - \sigma &\leq 2680. \end{aligned}$$

The rewriting of the remaining ten inequalities would follow a similar process. Note that even in the first execution of this step, the program is already quite extensive. Furthermore, the extensiveness of the program increases with each subsequent iteration of this step.

4. Efficient paths and layered graphs

In the second chapter, we delved into examination of path efficiency. Now, we revisit this concept again, focusing on a smaller aspect: the efficiency of individual edges. Some of gained results in this chapter have only theoretical aspect, while others can also have a practical application, mainly in preprocessing for Algorithm 2.

The first section will be devoted to the analysis of edge efficiency in general directed graphs. In the subsequent section, we shift our attention to layered graphs. For these graph, it is easier to derive useful theorems and algorithms that help us in studying the efficiency and solving the minimax regret problem.

Several results from this chapter are generalizations of theorems and algorithms from [9].

4.1 Efficient paths

Let us start with a notation we will use in this chapter. As before, let P be the set of paths in G from the starting vertex to the ending vertex, and for a fixed edge $e \in E$ let P_e denote such paths in P that contain the edge e. The weight of path p in the scenario s will be denoted as $w^s(p)$. With this in mind, we can proceed to the definition of edge efficiency.

Definition 6. *Let e be the edge of G, we say that e is:*

- necessarily efficient, if for each scenario S there exists a path $p_S \in P_e$ such that p_S is an efficient path in scenario S,

- possibly efficient, if there exists scenario S and path $p_S \in P_e$ such that p_S is an efficient path in scenario S,

- possibly weak efficient, if there exists scenario S and path $p_S \in P_e$ such that p_S is a weak efficient path in scenario S.

It should be intuitive that a necessarily efficient edge between two vertices is also a necessarily efficient path between these two vertices. However, in order to establish a connection between these two types of the necessary efficiency, we prove this statement properly.

Theorem 23. *If an edge $e = (u, v)$ is a necessarily efficient edge, then e is a necessarily efficient path from u to v in G.*

Proof. Let's assume, for the sake of contradiction, that e is not a necessarily efficient path from u to v. According to the definition, this means that there exists a scenario s_1 and a path p_1 from u to v such that:

$$w^{s_1}(p_1) \preceq w^{s_1}(e). \tag{4.1}$$

The necessary efficiency of edge e implies that for all scenarios there exists a path $p_S \in P_e$ which is an efficient path in that scenario. Denote the path

satisfying this condition for scenario s_1 as p_2. Moreover, let $p_{2,1}$ be the part of p_2 from s to u, and let $p_{2,2}$ be the part of p_2 from v to t.

Consider the walk $p_3' := p_{2,1} \cup p_1 \cup p_{2,2}$, inequality (4.1) then yields:

$$w^{s_1}(p_3') = w^{s_1}(p_{2,1}) + w^{s_1}(p_1) + w^{s_1}(p_{2,2}) \preceq w^{s_1}(p_{2,1}) + w^{s_1}(e) + w^{s_1}(p_{2,2}) = w^{s_1}(p_2).$$

The walk p_3' does not need to be a path, but by removing cycles we can transform it to a path, denoted as p_3. The path p_3 has smaller weight than p_3' and therefore $w^{s_1}(p_3) \preceq w^{s_1}(p_2)$. This contradicts the fact about the efficiency of p_2 in scenario s_1. □

We state an easy observation regarding possibly weak efficient edges. Recall that B^{x_p} represents the best-case scenario matrix for x_p and $b_i^{x_p}$ denotes the i-th row of this matrix.

Theorem 24. *An edge e is possibly weak efficient if and only if*

$$\min_{p \in P_e} \max_{q \in P} \min_{1 \leq i \leq s} \{b_i^{x_p}(x_p - x_q)\} \leq 0.$$

Proof. According to the respective definitions, an edge e is possibly weak efficient if and only if there exists possibly weak efficient path p containing e. By Theorem 11, the latter states that there exists $p \in P_e$ such that for all $\tilde{p} \in P$ holds:

$$B^{x_p} x_{\tilde{p}} \not\preceq B^{x_p} x_p.$$

This vector inequality can be reformulated as follows:

$$\min_{i=1,\ldots,s} \{b_i^{x_p}(x_p - x_{\tilde{p}})\} \leq 0.$$

The soundness of this inequality for all paths $\tilde{p} \in P$ implies that

$$\max_{\tilde{p} \in P} \min_{i=1,\ldots,s} \{b_i^{x_p}(x_p - x_{\tilde{p}})\} \leq 0.$$

We seek for a path satisfying this condition, which means we aim to minimize the expression above over all paths that contain e. This can be expressed as follows:

$$\min_{p \in P_e} \max_{\tilde{p} \in P} \min_{i=1,\ldots,s} \{b_i^{x_p}(x_p - x_{\tilde{p}})\} \leq 0. \tag{4.2}$$

Altogether, e is possibly weak efficient if and only if the condition (4.2) is satisfied, which concludes the proof.

□

For a general directed graph, we are currently not aware of any algorithms capable of efficient aid to the minimax problem. This limitation lead us to shift our attention towards layered graphs.

4.2 Efficient paths in layered graphs

A layered graph consists of a starting vertex, a fixed number of layers of vertices and an ending vertex. The important restriction is that edges can only connect vertices from one layer to the following layer. Formally, we define a layered graph as follows:

Definition 7. *A layered graph with l layers is defined as $G = (V, E)$, where*

$$V = \{s\} \cup \{(k, j) \mid k = 1, \dots l,\ j = 1, \dots, n_k\} \cup \{t\},$$

and edges exist only from vertex s to vertices in layer 1, from vertices in layer k to vertices in layer $k + 1$, for all $k = 1, 2, \dots m - 1$, and from vertices in layer l to vertex t.

Note that the graph presented in section 2.2.4 is a layered graph and so the graph we used to prove the NP-hardness of checking path efficiency. It allows us to reformulate Theorem 17 based on our new definitions to stronger version. This formulation illustrate that layered structure of graph does not help us with complexity associated with checking path efficiency, nor with checking edge efficiency:

Theorem 25. *The problem of determining if an edge in a layered, directed graph is an efficient edge is an NP-hard problem.*

Proof. The proof follows from Lemma 16 and Theorem 15, just as the proof of Theorem 17. □

Despite the above theorem, we will still try to utilize the potential benefits of graph's layered structure. We start with a straightforward observation regarding the possible efficiency of all graph's edges. We will discuss the application of this observation immediately afterwards.

Theorem 26. *Assume that there exist vectors c_s, c_t and c_k, for $k = 1, \dots, l - 1$ such that:*

$$c_s \in [w^l(e_s), w^u(e_s)],\ \textit{for all } e_s \in E \textit{ incident with } s,$$

$$c_t \in [w^l(e_t), w^u(e_t)],\ \textit{for all } e_t \in E \textit{ incident with } t,$$

$$c_k \in [w^l(e_k), w^u(e_k)],\ \textit{for all } e_k \in E \textit{ from the layer } k \textit{ to the layer } (k+1).$$

Moreover assume that for each edge there exists a path from s to t containing this edge. Then all edges of G are possibly efficient.

Proof. Assume the existence of the vectors in the theorem statement. Then there exists a scenario S in which all edges from s have a weight equal to c_s, all edges from layer k has a weight equal to c_k, for $k = 1, \dots, l - 1$, and all edges from layer l have a weight equal to c_t.

In this scenario, every path from s to t has the same weight. More precisely, the weight $w(p)$ of each path $p \in P$ is equal to:

$$w(p) = c_s + \sum_{k=1}^{l-1} c_k + c_t. \tag{4.3}$$

Hence, every path in G is possibly efficient, and consequently due to the technical assumption about non-emptiness of P_e for every edge e, every edge is possibly efficient as well.

□

The existence of vectors c_s, c_k and c_t can be efficiently verified. We can proceed by traversing the graph layer by layer and computing the intersection of weight intervals for all edges starting at each layer. For layer k, we calculate:

$$\bigcap_{e \in E, e \text{ starts at layer } k} [w^l(e), w^u(e)].$$

If this intersection yields a nonempty set, then it is certain that there exists a vector c_k satisfying the assumption of the theorem. Otherwise, no such c_k exists.

We repeat this procedure for every layer, including the one containing vertex s, to determine if the assumption of Theorem 26 is satisfied. If it is so, then we conclude that all edges in the graph are possibly efficient. If this is not the case, we can utilize the preprocessing method explained in the following section. This method tries to identify edges that are not possibly weak efficient.

4.2.1 An algorithm for preprocessing

We present an algorithm determining if an edge from vertex s is possibly weak efficient. This algorithm can serve as a useful preprocessing step to Algorithm 2 (if the given graph is layered). We will discuss these applications further later on, now let's delve into the algorithm. The algorithm is a generalization of an algorithm from [9].

Consider the edge from s to vertex $(1, 1)$ and let us denote it as e. Our goal is to find out whether e is possibly weak efficient or not. To begin, we set the edge values of all paths from s to t passing through e to the lower bound. All other edge values are set to the upper bound.

In the first iteration we consider only vertex s and vertices in layers 1 and 2. We find all weak efficient paths from s to every vertex in layer 2. This can be achieved using Multi-Dijkstra's algorithm. Once we compute these paths, we encounter three possible cases:

Case 1: None of the weak efficient paths contains e. In this case, we conclude that e is not possibly weak efficient.

Case 2: For all vertices in layer 2 there exists at least one weak efficient path that contains e. Then e is possibly weak efficient.

Case 3: Weak efficient paths that contains e exists for some vertices in layer 2 but not for everyone. In this case, we need to proceed to the next iteration:

We shrink the graph between s and layer 2 by considering only the relevant vertices - those in layer 2 and vertex s. For each vertex in layer 2, we maintain a set of all efficient paths from s to that vertex. If for a vertex any of efficient paths from s to this vertex contains e, we discard the paths that do not include e and keep only those that do. We set edge values from these vertices to the lower bound, while setting other edge values to the upper bound.

Next, we compute all efficient paths from s to layer 3, taking advantage of the fact that graph has been shrunk. This computation is similar to the initial computation of efficient paths. The only difference is that "edge" from s to vertex

in layer 2 can attain more than one value - the values of all efficient paths from s to this vertex. This can slow down the process of computation, but does not affect correctness of the algorithm.

We then reevaluate which of the three cases applies. After at most l iterations, we determine the truth about possible weak efficiency of e.

Due to the increased complexity of our algorithm compared to the one proposed in [9], we will prove its correctness in a more formal way:

Theorem 27. *The algorithm above correctly determines whether edge e is a possibly weak efficient edge or not.*

Proof. Observe that setting the values of edges as it is done in the algorithm clearly corresponds to determining possible weak efficiency of edge e. Therefore, our task is narrowed down to proving the correctness of solving the three cases. We will demonstrate that the solution for each case is justified.

We begin with Case 2, assuming that there is at least one weak efficient path for each vertex in layer 2. For the sake of contradiction, let us assume that e is not possibly weak efficient. Therefore, there exists $\tilde{p} \in P \setminus P_e$ such that for every $p_e \in P_e$ holds

$$w(\tilde{p}) \prec w(p_e), \tag{4.4}$$

where $w(p)$ denotes the weight of path p in the scenario constructed via the algorithm.

Let $(2, j)$ be the vertex in layer 2 that is incident with the path $\tilde{p}$, and q_1 be one of the weak efficient paths from s to this vertex. Furthermore, let $\tilde{p} = \tilde{p}_1 \cup \tilde{p}_2$, where $\tilde{p}_1$ is the part of the path from s to vertex $(2, j)$, and $\tilde{p}_2$ is the remaining part of the path. Then, based on efficiency of q_1, we obtain:

$$w(\tilde{p}_1) \not\prec w(q_1).$$

By concatenating q_1 with $\tilde{p}_2$, we obtain path from P_e. Let's denote this path as q, hence $q = q_1 \cup \tilde{p}_2$ and by the above inequality we have:

$$w(\tilde{p}) = w(\tilde{p}_1) + w(\tilde{p}_2) \not\prec w(q_1) + w(\tilde{p}_2) = w(q).$$

This is contradiction with (4.4), and therefore the claim regarding the possible weak efficiency of e holds true.

Case 1 follows from the observation that if a path is weak efficient, then any part of it must also be weak efficient. A more detailed proof can be done in a similar way as we proved Case 2.

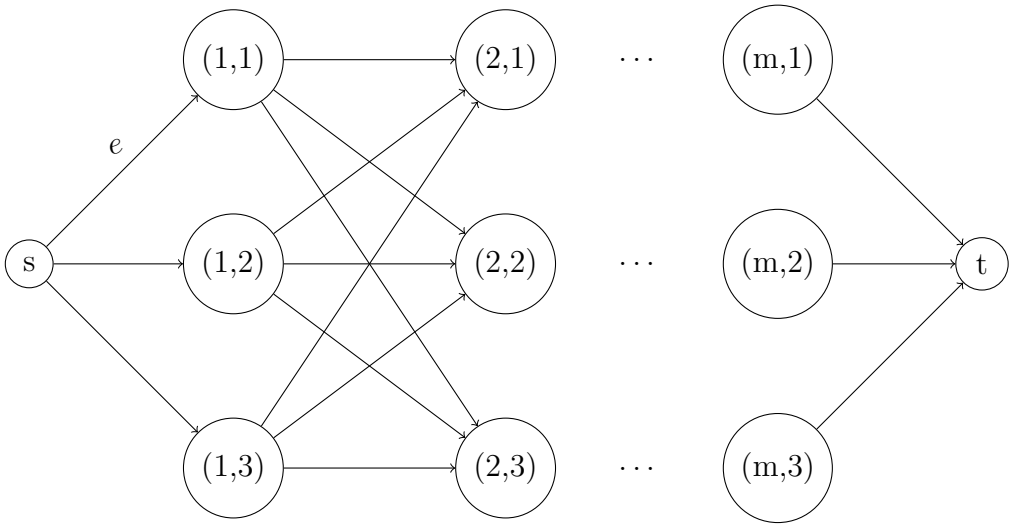
In the last case, we shrink the graph and essentially end up with a situation similar to the one we had at the beginning. Therefore, for vertices in layer 3, 4 and so on, we can apply the same proof technique as we used above. The final issue to address is the elimination of weak efficient paths for vertices where at least one weak efficient path via e exists. We leave this as an observation, as the proof again follows the same pattern we used in proving correctness of Case 2. □

The algorithm is correct, and it seems to perform only efficiently computable operations. However, the number of these operations could become huge due to the presence of numerous efficient paths from s to intermediate vertices. This problem is illustrated in the following example.

Example 2. Consider a layered graph with l layers, where each layer consists of three vertices and the set of edges contains all possible edges for this type of graph. Formally, let $G_3 = (V_3, E_3)$, where

$$\begin{aligned} V_3 =& \{s\} \cup \{(i,j) \mid 1 \leq i \leq l,\, 1 \leq j \leq 3\} \cup \{t\}, \\ E_3 =& \{(s,(1,j)) \mid 1 \leq j \leq 3\} \cup \{((l,j),t) \mid 1 \leq j \leq 3\} \\ & \cup \{((i,j),(i+1,l)) \mid 1 \leq i < l, 1 \leq j,l \leq 3\}. \end{aligned}$$

The planar embedding of G_3:



Consider that all weight intervals are degenerate, and the weight assigning is defined as follows:

$$w((s,(1,j))) = e_j,\ w(((i,j),(i+1,l))) = 2^i e_l,\ w(((l,j),t)) = 2^l e_j.$$

Therefore, an edge from the i-th layer has one coordinate equal to 2^i, while the other coordinates are equal to zero. Moreover, the non-zero coordinate is determined by the out-neighbour of the edge. The same principle applies to the starting and ending vertices.

First, observe that every path from s to any vertex in layer i has the same total weight, i.e. the sum of the three weights. This weights is equal to:

$$\sum_{l=0}^{i-1} 2^l = 2^i - 1.$$

In particular, all paths from s to t have a total weight equals to $2^{l+1} - 1$.

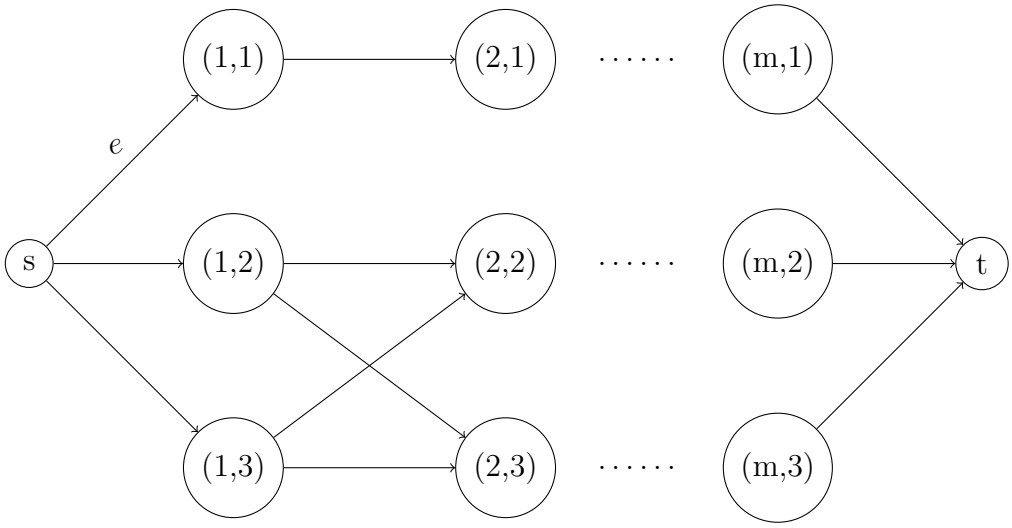
It is easy to derive from this observation that any partial path from s to any vertex is weak efficient. How many paths are there from s to a vertex in the i-th layer? We can prove by induction that this number is equal to 3^{i-1}. Therefore, for vertex in layer l we have 3^{l-1} efficient paths, each with a different weight. Considering the fact that the graph G_3 has $3l+2$ vertices, we conclude that there are exponentially many efficient paths to vertices in the last layers.

The presented algorithm does not need to compute this. In the first iteration, it already determines that all vertices in layer 1 have one efficient path containing the edge $(s,(1,1))$ and outputs that this edge is possibly weak efficient.

However, if we consider a slightly modified graph, a problem arises. Consider G_3 without the edges from vertices $(i,1)$ to vertices $(i+1,2)$ and $(i+1,3)$, as well as edges from $(i,2)$ and $(i,3)$ to $(i+1,1)$, for $i = 1,\ldots,l-1$. Let's examine what happens in layer i for this modified graph. There exists exactly one efficient path from s to $(i,1)$, and this path contains the edge $(s,(1,1))$. All other paths lead to vertices $(i,2)$ and $(i,3)$, and none of them contains the investigated edge. Clearly, these paths are still efficient, and although their number is now 2^{i-1} instead of 3^{i-1}, it is still exponentially large.

Hence, the algorithm must go through all iterations and also memorize exponentially many efficient paths and for each of these path it needs to start the computation.

The planar embedding of modified G_3:



Now, let us demonstrate how this algorithm can help us in the preprocessing. For each edge e from s, we can decide if it is possibly weak efficient or not. If e is not possibly weak efficient, then there is no possibly weak efficient path from s to t that contains e. According to Theorem 12, the path with the minimal regret is at least possibly weak efficient. Therefore, this path does not contain edge e. Hence, when solving the minimax regret problem, we can remove e from consideration.

We can apply the same procedure to any edge from s. After removing all edges originating in s that are not possibly weak efficient, we can also remove the vertices in layer 1 that are incident to these edges. Furthermore, we can also remove edges that are incident to these vertices. We can continue this process by removing vertices in layer 2 that have no in-neighbors along with the edges originating from these vertices, and so on.

Also notice that the same strategy can be employed with edges from layer l to vertex t: we can consider the mirrored image of graph G and follow the same procedure we used to eliminate edges from s. This way, we can obtain a graph with significantly fewer vertices and edges, which can simplify the problem and enhance the efficiency of Algorithm 2.

Conclusion

In the first chapter, we present essential definitions, including two concepts that are fundamental for our thesis - the efficiency of a solution and the minimax regret problem.

In the second chapter, we formulated the interval multiobjective shortest path problem as a mixed-integer linear program. Then, we dealt with feasible solutions to this program. We simplified the computation of the maximal regret and checking the necessary and the possible efficiency of a given solution. Moreover, we established the relationship between the necessary efficiency and the regret of solution. We outlined an approach for checking necessary or possible efficiency. We provided the proofs for the properties of the solution with the least regret. Lastly, we demonstrate that checking tha efficiency of a path in interval multiobjective shortest path problem is an NP-hard problem.

The third chapter was devoted to the algorithm solving the minimax regret problem in the IMOSP. We introduced an algorithm solving the general MOLP and adjust it to solve IMOSP efficiently. We provided a brief overview of the algorithm's time complexity.

In the final chapter, we examine additional valuable information regarding the IMOSP. We defined and explored the efficiency of edges and present preprocessing algorithms that can enhance the efficiency of the algorithm discussed in the third chapter.

Notation

$\mathbb{R}$	the set of real numbers
$\mathbb{R}_+$	the set of positive real numbers
$\mathbb{Z}$	the set of integers
$c_{i,j}$	the element on i-th row and j-th column of matrix C
$(C)_i$	the i-th column of matrix C
c_i	the i-th row of matrix C
e_i	vector with value of one in the i-th coordinate and zeros in all other coordinates, i.e. $e_i = (0, \ldots, 0, 1, 0, \ldots, 0)^T$
e	the vector with all ones, i.e. $e = (1, \ldots, 1)^T$

Bibliography

[1] M. Hladík. "Complexity of necessary efficiency in interval linear programming and multiobjective linear programming". In: *Optimization Letters* 6.5 (2012), pp. 893–899.

[2] M. Hladík. "On relation of possibly efficiency and robust counterparts in interval multiobjective linear programming". In: *Optimization and Decision Science: Methodologies and Applications* 217 (2017), pp. 335–343.

[3] M. Inuiguchi and M. Sakawa. "Possible and necessary efficiency in possibilistic multiobjective linear programming problems and possible efficiency test". In: *Fuzzy Sets and Systems* 78 (1996), pp. 231–241.

[4] Ernesto Queirós Vieira Martins. "On a multicriteria shortest path problem". In: *European Journal of Operational Research* 16 (1984), pp. 236–245.

[5] Nimrod Megiddo. "On the complexity of linear programming". In: *Advances in Economic Theory. Fifth World Congress* (1987).

[6] M. A. Yaghoobi S. Rivaz. "Minimax regret solution to multiobjective linear programming problems with interval objective functions coefficients". In: *Springer* 21 (2012), pp. 625–649.

[7] M.Hladík S. Rivaz M. A. Yaghoobi. "Using modified maximum regret for finding a necessarily efficient solution in an interval MOLP problem". In: *Fuzzy Optim Decis Making* 15 (2015), pp. 237–253.

[8] Paolo Serafini. "Some considerations about computational complexity for multi objective combinatorial problems". In: *Recent Advances and Historical Development of Vector Optimization* (1987).

[9] Hande Yaman. "Essay on some combinatorial optimization problems with interval data". PhD thesis. Bilkent University, 1999.

[10] Jin Y. Yen. "Finding the K Shortest Loopless Paths in a Network". In: *Management Science* 17.11 (1971), pp. 712–716.