HYPERVOLUME SCALARIZATIONS FOR MULTIOBJECTIVE GRAPH OPTIMIZATION PROBLEMS Ana Catarina Quitério Lourenço

1 2 9 0 UNIVERSIDADE D COIMBRA

UNIVERSIDADE D
COIMBRA

Ana Catarina Quitério Lourenço

HYPERVOLUME SCALARIZATIONS FOR MULTIOBJECTIVE GRAPH OPTIMIZATION PROBLEMS

Dissertation in the context of the Master in Informatics Engineering, Specialization in Intelligent Systems, advised by Professor Luís Paquete and presented to Faculty of Sciences and Technology / Department of Informatics Engineering.

June 2021

Faculty of Sciences and Technology

Department of Informatics Engineering

Hypervolume scalarizations for multiojective graph optimization problems

Ana Catarina Quitério Lourenço

Dissertation in the context of the Master in Informatics Engineering, Specialization in Intelligent Systems advised by Prof. Luís Paquete and presented to the Faculty of Sciences and Technology / Department of Informatics Engineering..

June 2021

UNIVERSIDADE D
COIMBRA

This page is intentionally left blank.

Abstract

Many real-life situations result in optimization problems with multiple objectives that are very often conflicting. For example, in finance, we want to choose a portfolio where the expected value is as high as possible and the risk is as low as possible. One way of solving these problems is to scalarize the several objectives in order to obtain a single objective optimization problem. In this thesis we use the notion of hypervolume scalarization to solve graph optimization problems, with particular emphasis for the Shortest Path Problem and Minimum Spanning Tree Problem. For the first problem, we introduce the hypervolume scalarization for the Biobjective Shortest Path Problem and its mixed integer linear programming formulation. In addition, we propose a label setting algorithm to solve it and pruning conditions to improve its performance. Moreover, we prove the computational complexity of solving this problem and present numerical results for a wide range of graph problem instances. For the Spanning Tree Problem, we introduce the hypervolume scalarization for the Biobjective Minimum Spanning Tree Problem, its mixed integer linear programming formulation, a branch-and-bound algorithm, and the computation complexity of this problem.

Keywords

Hypervolume scalarization, shortest Path Problem, Minimum Spanning Tree Problem, multiobjective combinatorial optimization, integer linear programming.

This page is intentionally left blank.

Resumo

Muitas situações da vida real resultam em problemas de optimização com objectivos múltiplos que muitas vezes estão em conflito. Por exemplo, em finanças, queremos escolher um portfolio em que o retorno esperado é tão alto como possível e o risco é tão baixo como possível. Uma forma de resolver estes problemas é escalarizar os vários objectivos para obter um problema de optimização com um único objectivo. Nesta tese, usamos a notação de escalarização de hipervolume para resolver problemas de optimização em grafos, com um ênfase particular no Problema do Caminho Mais Curto e no Problema da Árvore de Abrangência Mínima. Para o primeiro problema, introduzimos escalarizações de hipervolume para o Problema do Caminho Mais Curto Biobjectivo e a sua formulação de programação linear inteira mista. Além do mais, propomos um algoritmo de "label setting" para o resolver e condições de corte para melhorar a sua performance. Para além disso, demonstramos a complexidade computacional da resolução deste problema e apresentamos resultados numéricos numa ampla gama de instâncias de problemas de grafos. Para o Problema da Árvore de Abrangência Mínima, introduzimos a escalarização de hipervolume para o Problema da Árvore de Abrangência Mínima Biobjectivo, a sua formulação de programação linear inteira mista, um algoritmo de "branch-and-bound" e a complexidade computacional deste problema.

Palavras-Chave

Escalarizações de hipervolume, Problema do Caminho Mais Curto, optimização combinatória multiobjectivo, programação linear inteira, Problema da Árvore de Abrangência Mínima.

This page is intentionally left blank.

Acknowledgements

I would like to express my gratitude to my supervisor, Professor Luís Paquete, for his support, guidance and availability during this year.

I would like to acknowledge the Centre for Informatics and Systems of the University of Coimbra for providing financial support for this thesis, through a research grant with reference number UIDB/00326/2020_L.717529 within the project with reference UIDB/00326/2020.

Lastly, I would like to thank all the members of ALGO Lab for helping with keeping me motivated and learning more about related research subjects.

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Acronyms

BMSTP Biobjective Minimum Spanning Tree Problem. 31–35

BSPP Biobjective Shortest Path Problem. 11, 14, 15, 17–19, 22, 26, 27, 35

CMSTP Constrained Minimum Spanning Tree Problem. 33, 34

CP Counting Problem. 9

CSPP Constrained Shortest Path Problem. 16, 17

DP Decision Problem. 8, 9, 17

ILP Integer Linear Programming. 9–11, 14–16, 22, 24, 31–33, 39

MOCO Multiobjective Combinatorial Optimization. ix, 5–9, 15, 38

MSTP Minimum Spanning Tree Problem. 31, 33

SPP Shortest Path Problem. 11, 12, 20

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

Optimization problems are some of the fundamental problems in Computer Science. Although many real-life applications result in optimization problems with multiple objectives, in general, there is no single solution that is optimal for all objectives. For instance, usually there is no single path between two cities that optimizes cost and time since the fastest path may not be the cheapest, just consider highways. It is possible to solve a multiobjective optimization problem by scalarizing it, under some assumptions on the decision maker preferences. When scalarizing a multiobjective optimization problem we formalize a related single-objective problem such that an optimal solution to this problem is also optimal for the multiobjective problem. Some of the most common scalarizations are weighted sum, ϵ-constraint method and achievement scalarizing functions.

The hypervolume scalarization has been proposed recently in [20] and it consists of a particular product of the objective functions. This scalarization presents some interesting properties, such as the fact that no convexity assumptions are required, but it leads to particular formulations (which are quadratic for biojective problems) for which very little is known. In [20], an application of this scalarization technique to a Biobjective Constrained Knapsack Problem, which results in a quadratic knapsack formulation, is presented.

The goal of this dissertation is to formalize hypervolume scalarized formulation of known multiobjective graph problems, such as the Minimum Spanning Tree Problem and the shortest path problem, as well as to develop exact and/or approximation algorithms for these problems. We start to extend the previous work for biobjective knapsack problems in [20], for which a linearization of the quadratic scalarized formulation was achieved and an approximation algorithm with provable quality guarantee was devised. Focusing on the Biojective Shortest Path Problem, we start by introducing a linearization of the quadratic scalarized formulation, prove its computational complexity, propose an algorithm for solving the quadratic scalarized formulation and pruning conditions that may improve the running times. This work has recently been published in [16]. Then, we discuss the implementation of the proposed algorithm and perform an in-depth experimental analysis on a wide range of graph problem. We relate the properties of these instances with the performance of our approach. At last, we extend the work for the Biojective Minimum Spanning Tree Problem, once again introducing the linearization of the quadratic scalarized formulation, proving its computational complexity, proposing a branch-and-bound algorithm for solving the quadratic scalarized formulation and pruning conditions.

The work developed shows that, while it is hard to solve the linearized version of hypervolume scalarizations using integer linear programming solvers, algorithms dedicated to the specific problem are much more efficient.

We first review some basic concepts about graphs, multiobjective optimization problems, computational complexity, integer linear programming and the Shortest Path Problem in Chapter 2. Chapter 3 deals with the Biobjective Shortest Path Problem. We present its integer linear programming formulation, its hypervolume scalarized formulation and a linearization of the quadratic scalarized formulation. We prove the computational complexity of the hypervolume scalarized formulation, propose a label setting algorithm for solving it and pruning conditions for improving the running time. In Chapter 4, we discuss the methods used for implementing the label setting algorithm, the benchmark instances used for testing and the numerical results, focusing on running times, speedup values and the instances characteristics. We discuss the efficiency of the pruning conditions for each set of instances as well as which problem characteristics have more impact on the performance. We also discuss the use of an integer linear programming solver and compare the performance. Chapter 5 deals with the Biobjective Minimum Spanning Tree problem. We present its integer linear programming formulation, its hypervolume scalarized formulation and a linearization of the quadratic scalarized formulation. We prove the computational complexity of the hypervolume scalarized formulation, propose an algorithm for solving it and pruning conditions for improving the running time. In Chapter 6 we summarize the main ideas presented in this thesis and how this work can be applied to other problems.

This page is intentionally left blank.

Chapter 2

Basic Concepts

In this chapter, we introduce basic concepts that are relevant for the remainder of this work, such as concepts about graphs, multiobjective combinatorial optimization and the algorithms more commonly used for solving multiobjective optimization problems, computational complexity, integer linear programming and the Shortest Path Problem. As reference works we refer to [4], [19], [11] and [13].

2.1 Graphs

A *directed graph* $G = (V, A)$ is a set V of *nodes* and a set A of *arcs* whose elements are ordered pairs of distinct nodes. We denote an *arc* a from node u to node v by $a = (u, v)$. A *directed network* is a directed graph whose arcs and/or nodes have associated numerical values, such as costs, capacities, etc.

$|V| = n$ denotes the number of nodes and $|A| = m$ denotes the number of arcs in a graph G. For an arc $a = (u, v)$, u is the *source* and v the *target*, both u and v are called *endpoints* of the arc. Two arcs are *adjacent* if they share a common endpoint. The *degree of a node* is the sum of the number of its incoming arcs and the number of its outgoing arcs.

A graph $G' = (V', A')$ is a *subgraph* of $G = (V, A)$ if $V' \subseteq V$ and $A' \subseteq A$. G' is a spanning subgraph if $V' = V$ and $A' \subseteq A$.

A *path* between two nodes u and v in a graph G is an alternating tuple $(u, (u, w_1), w1, \ldots,$ $w_r, (w_r, v), v)$ of nodes of V and arcs of A starting in u and ending in v so that every node is adjacent to its neighbouring edges. If all nodes are pairwise distinct we have a *simple or loopless path.*

A *cycle* is a simple path $(i_1, (i_1, i_2), i_2, \ldots, (i_{r-1}, i_r), i_r)$ together with the arc (i_r, i_1). If a graph contains no cycle, it is called *acyclic.*

Two nodes i and j are *connected* if the graph contains at least one path from node i to node j. If every pair of nodes is connected, the graph is connected.

A *tree* is a connected graph that contains no cycle. A tree T is a *spanning tree* of G if T is a spanning subgraph of G.

Topological sorting for graphs is a linear ordering of nodes such that for every directed arc (u, v), node u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a directed acyclic graph.

2.2 Multiobjective Combinatorial Optimization

The goal of a combinatorial optimization problem is to find a combination of items of a finite set $A = \{\alpha_1, \ldots, \alpha_n\}$ that is feasible for the problem constraints and optimizes the objective of the problem. This is modeled by using binary decision variables x_i for $i = 1, \ldots, n$. We have $x_i = 1$ if item i belongs to the subset and $x_i = 0$ otherwise. A *solution* of the problem is a vector $x = (x_1, \ldots, x_n)^T \in \{0, 1\}^n$.

A combinatorial optimization problem is given by the *set of feasible solutions* $\mathcal{X} \subseteq 2^{\{0,1\}^n}$ defined as a subset of the power set A, and an objective function $f : \mathcal{X} \to \mathbb{R}$. Thus, the problem is written using the notation

$$\min_{x \in \mathcal{X}} \; f(x).$$

In this thesis, we consider only the sum objective functions where each item α_i has one associated coefficient c_i :

$$f(x) = \sum_{i=1}^{n} c_i x_i.$$

In Multiobjective Combinatorial Optimization (MOCO) each item α_i is associated with several objective function coefficients c_i^j, $j = 1, \ldots, m$ with $m \geq 2$. In this way, m objective functions $f_j(x)$ are formulated. The MOCO-problem can now be modeled as

$$\underset{x \in \mathcal{X}}{vmin}\, f(x) = \left(f_1(x), \ldots, f_m(x)\right). \tag{2.1}$$

The image of the feasible set $\mathcal{X}$ in the objective space is called the *set of feasible points* and is denoted by $\mathcal{Y} := f(\mathcal{X})$.

In order to compare images we need to be able to order vectors in $\mathbb{R}^m$. Since there is no canonical ordering for $m \geq 2$, the vectors may be ordered based on several definitions. Given two points $y^1, y^2 \in \mathbb{R}^m$, we have

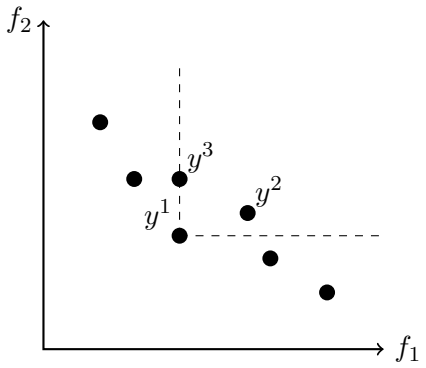

Figure 2.1: Example for $\mathbb{R}^2$. We have $y^1 < y^2$ and $y^1 \leq y^3$.

- *weak componentwise order*:

  $$y^1 \leqq y^2 \text{ if } y_j^1 \leq y_j^2 \text{ for } j = 1, \ldots, m;$$

- *componentwise order*:

  $$y^1 \leq y^2 \text{ if } y^1 \leqq y^2 \text{ and } y^1 \neq y^2,$$

  as illustrated for points y^1 and y^3 in Figure 2.1;

- *strict componentwise order*:

  $$y^1 < y^2 \text{ if } y_j^1 < y_j^2 \text{ for } j = 1, \ldots, m,$$

  as illustrated for points y^1 and y^2 in Figure 2.1;

- *lexicographic order*:

  $$y^1 \leq_{\text{lex}} y^2 \text{ if } y^1 = y^2 \text{ or } y_k^1 < y_k^2 \text{ for } k = \min\{j : y_j^1 \neq y_j^2,\ j = 1, \ldots, m\}.$$

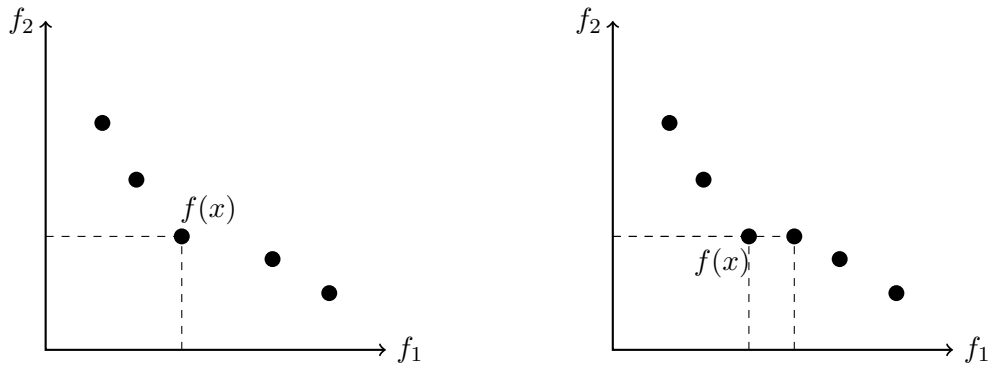

Figure 2.2: Example for a minimization problem in $\mathbb{R}^2$. On the left, solution x is efficient and $f(x)$ is a nondominated point. On the right, solution x is weakly efficient.

Using the component wise order, a feasible solution $x \in \mathcal{X}$ is called *efficient* or *Pareto optimal* if there is no other solution $\overline{x} \in \mathcal{X}$ such that

$$f(\overline{x}) \leq f(x).$$

If x is efficient, such as seen on the left in Figure 2.2, $f(x)$ is called a *nondominated point*. If $x^1, x^2 \in \mathcal{X}$ and $f(x^1) \leq f(x^2)$, we say x^1 *dominates* x^2 and $f(x^1)$ *dominates* $f(x^2)$. The set of all efficient solutions $x \in \mathcal{X}$ is denoted by $\mathcal{X}_E \subseteq \mathcal{X}$ and is called the *efficient set*. The set of all nondominated points $y = f(x) \in \mathcal{Y}$, where $x \in \mathcal{X}_E$ is denoted by $\mathcal{Y}_N \subseteq \mathcal{Y}$ and is called the *nondominated set*.

Using the weak componentwise order, a feasible solution $x \in \mathcal{X}$ is called *weakly efficient* or *weakly Pareto optimal* if there is no $\overline{x} \in \mathcal{X}$ such that $f(\overline{x}) < f(x)$, such as seen on the right in Figure 2.2.

Let $\mathcal{A} + \mathcal{B} = \{a + b \ : \ a \in \mathcal{A}, \ b \in \mathcal{B}\}$ denote the Minkovski-sum of sets $\mathcal{A}$ and $\mathcal{B}$ in $\mathbb{R}^m$, let $\mathbb{R}^m_{\geq} := \{y \in \mathbb{R}^m \ : \ y_j \geq 0, \ j = 1, \ldots, m\}$ denote the non-negative orthant of $\mathbb{R}^m$ and let $\text{conv}(\mathcal{Y}_N)$ denote the convex hull of the nondominated set.

The set of efficient solutions $\mathcal{X}_E$ and the set of nondominated points $\mathcal{Y}_N$ are divided into two subsets: if there is some $\lambda \in \mathbb{R}^m_{>}$ such that $x \in \mathcal{X}_E$ is an optimal solution of $\min_{x \in \mathcal{X}} \lambda^T f(x)$, then x is called a *supported solution* and $y = f(x)$ is called a *supported point*. Otherwise x and y are called *unsupported solution* and *unsupported point*, respectively. The sets of all supported solutions is denoted $\mathcal{X}_{sE}$, the set of all supported points $\mathcal{Y}_{sN}$, the set of all unsupported solutions $\mathcal{X}_{uE}$ and the set of all unsupported points $\mathcal{Y}_{uN}$. The set of supported points that are also extreme points of $\text{conv}(\mathcal{Y}_N)$ is called the set of *extreme supported points* $\mathcal{Y}_{eN}$. The corresponding efficient solutions are called *extreme supported solutions* $\mathcal{X}_{eE}$.

For MOCO it is possible to define upper and lower bounds on the set of nondominated points because the set of feasible points $\mathcal{Y}$ is compact by definition.

The *ideal point* $y_{\mathcal{I}} = (y^1_{\mathcal{I}}, \ldots, y^m_{\mathcal{I}})^T$ is a tight lower bound on $\mathcal{Y}_N$ and is defined as the individual minima of the m objective functions, that is,

$$y^j_{\mathcal{I}} := \min\{y^j \ : \ y \in \mathcal{Y}\}, \quad j = 1, \ldots, m.$$

Thus, the ideal point $y_{\mathcal{I}}$ usually is not an element of $\mathcal{Y}$, such as in Figure 2.3. If $y_{\mathcal{I}} \in \mathcal{Y}$, it dominates all other feasible points and $\mathcal{Y}_N = \{y_{\mathcal{I}}\}$.

The *nadir point* $y_{\mathcal{N}} = (y^1_{\mathcal{N}}, \dots, y^m_{\mathcal{N}})^T$ is a tight upper bound on $\mathcal{Y}_n$ as is defined by the maximal components of all nondominated points for the m objective functions, that is,

$$y^j_{\mathcal{N}} := \max\{y^j \ : \ y \in \mathcal{Y}_N\}, \quad j = 1, \dots, m.$$

The computation of the ideal point is considered easy (from a multiobjective point of view) because the ideal point is found by solving m single objective optimization problems. On the other hand, computation of the nadir point involves optimization over the efficient set, which is a very difficult problem and no efficient method to determine it in general is known.

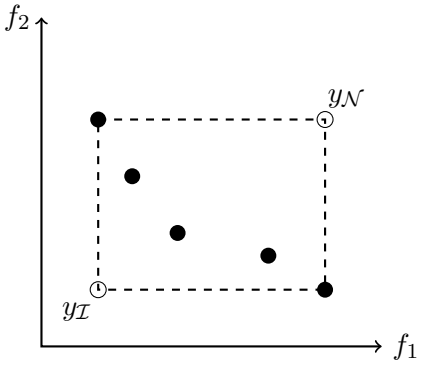

Figure 2.3: Example for $\mathbb{R}^2$. We have the ideal point $y_{\mathcal{I}}$ and the nadir point $y_{\mathcal{N}}$.

2.3 Techniques for Solving MOCO Problems

Scalarization methods are commonly used to solve MOCO problems. These methods are based on the formulation of one or several parametric single objective optimization problems that can be solved using single objective solution methods. The parameters choice usually allow to compute a subset of the efficient set $\mathcal{X}_E$ of the original multiobjective problem.

2.3.1 Weighted Sum Scalarization

In this method, a single objective problem is created by a weighted sum of the objective functions of the MOCO problem. The feasible set and the number of constraints remains equal. The weighted sum scalarization is formulated as follows:

$$\min_{x \in \mathcal{X}} \sum_{j=1}^{m} \lambda_j f_j(x), \tag{2.2}$$

where the weights λ are in $\mathbb{R}^m$.

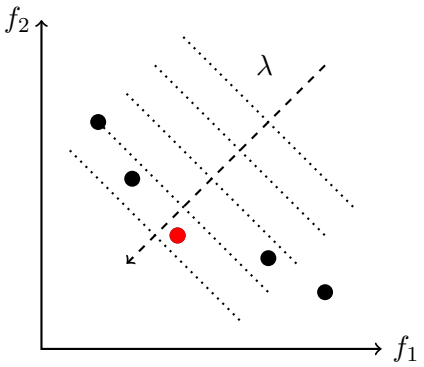

Figure 2.4: Example of weighted sum scalarization for $\mathbb{R}^2$. The solution of the scalarized problem is highlighted in red.

It has been shown [7] that for $\lambda \in \mathbb{R}^m_{\geq} := \{\lambda \in \mathbb{R}^m \ : \ \lambda_j \geq 0, \ j = 1, \dots, m\}$ every optimal solution of (2.2) is a weakly efficient solution of the initial problem and for $\lambda \in \mathbb{R}^m_{>} := \{\lambda \in \mathbb{R}^m \ : \ \lambda_j > 0, \ j = 1, \dots, m\}$ every optimal solution of (2.2) is a supported efficient solution of the initial problem. Using appropriate weights $\lambda \in \mathbb{R}^m_{>}$, every supported efficient solution of the initial problem can be found as an optimal solution of (2.2). The biggest drawback is the fact that no unsupported solution can be obtained using the weighted sum method.

2.3.2 Weighted Chebycheff Scalarization

The weighted Chebycheff scalarization is formulated as follows:

$$\min_{x\in\mathcal{X}} \max_{j=1,\dots,m} \{\lambda_j (f_j(x) - z_j)\}, \tag{2.3}$$

where the weights λ are in $\mathbb{R}^m$ and z is the ideal point. It has been shown that for $\lambda \in \mathbb{R}^m_> := \{\lambda \in \mathbb{R}^m \ : \ \lambda_j > 0, \ j = 1, \dots, m\}$ every optimal solution of 2.3 is a weakly efficient solution of the initial problem.

2.3.3 ϵ-Constraint Method

In the ϵ-constraint method there is no aggregation of criteria. Instead only one of the original objectives $f_k, \ k = 1, \dots, m$ is used as the objective function, while the others are transformed to constraints by introducing a bound on the respective objective function values. This results in the following problem:

$$\begin{aligned} \min_{x\in\mathcal{X}} \quad & f_k(x) \\ s.t. \quad & f_j(x) \leq \epsilon_j, \quad j = 1, \dots, m, j \neq k. \end{aligned} \tag{2.4}$$

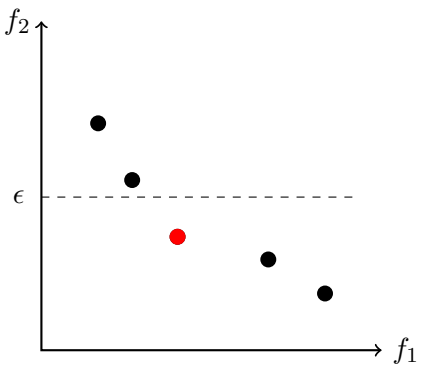

Figure 2.5: Example of ϵ-constraint method for $\mathbb{R}^2$. For $f_2 \leq \epsilon$, the solution of the scalarized problem is highlighted in red.

It has been shown that for any $\epsilon \in \mathbb{R}^m$, every optimal solution of (2.4) is weakly efficient for the initial problem and it is an efficient solution of the initial problem if it is the unique optimal solution of (2.4). Choosing an appropriate vector $\epsilon \in \mathbb{R}^m$, such as for $\overline{x} \in \mathcal{X}_E, \ \epsilon = f(\overline{x})$, every efficient solution of the initial problem can be found by the ϵ-constraint method for any $k \in \{1, \dots, m\}$.

2.4 Complexity Theory

This is a brief review of computational complexity of MOCO problems. For further details we refer to [6]. Complexity theory allows us to evaluate how difficult it is to solve a problem based on the number of operations an algorithm needs to find the correct answer to the problem in the worst case. Complexity theory requires the problem to be stated as a Decision Problem (DP), where a decision problem is a question that has either a yes or no answer.

Optimization and decision problems are closely related. For example, if we have the following optimization problem: minimize $f(x)$ for $x \in \mathcal{X}$, then the decision version is: given a constant $\beta \in \mathbb{Z}$, does there exist $x \in \mathcal{X}$ such that $f(x) \leq \beta$.

To measure the number of operations required to find the correct answer, we use the "big O" notation. The running time of an algorithm is $\mathcal{O}(g(n))$ if there is a constant c such that the number of operations performed by the algorithm is less than or equal to $cg(n)$ for all instances of the decision problem, where g is some function and n is the size of the instance.

If there is a deterministic algorithm that answers the decision problem and needs $\mathcal{O}(p(n))$ operations, where p is a polynomial in n, we say that the DP belongs to the *class* $\mathcal{P}$ of problems.

If there is a polynomial p such that the running time of the algorithm is $\mathcal{O}(p(n))$, the algorithm is said to be a *polynomial time algorithm*. Otherwise, if there is no such polynomial, the algorithm is said to be exponential.

If there is a nondeterministic polynomial time algorithm that solves the DP, the DP belongs to the *class NP*. This means that, for the decision version of an optimization problem, given x it is possible to check whether $x \in \mathcal{X}$ and $f(x) \leq \beta$ in polynomial time. Clearly, $\mathcal{P} \subset \mathcal{NP}$.

Let DP_1 and DP_2 be two decision problems. A polynomial time transformation of DP_1 to DP_2 is a polynomial time algorithm A that constructs an instance I_2 of DP_2 from an instance I_1 of DP_1 with the property that x_1 yields a "yes" answer for the instance I_1 if and only if $A(x_1)$ yields a "yes" answer for the instance of DP_2 [4]. When this happens, we write $DP_1 \propto DP_2$. DP_1 and DP_2 are *equivalent* if $DP_1 \propto DP_2$ and $DP_2 \propto DP_1$.

A decision problem DP is $\mathcal{NP}-$complete if $DP \in \mathcal{NP}$ and $DP' \propto DP$ for all $DP' \in \mathcal{NP}$. The relation is transitive and transitivity means that $\mathcal{NP}-$completeness of DP follows if $DP' \propto DP$ for one $DP' \in \mathcal{NP}$ [4]. Thus, showing $DP' \propto DP$ actually means that DP' is a special case of DP, so DP is at least as difficult as DP'. A problem DP is called $\mathcal{NP}-$hard if $DP' \propto DP$ for all $DP' \in \mathcal{NP}$ but it is not known if $DP \in \mathcal{NP}$ (such as optimization problems).

There is a *Counting Problem (CP)* associated to each DP: How many "yes" answers does the DP have. For the decision version of an optimization problem, we have the following CP: Given $\beta \in \mathbb{Z}$, how many $x \in \mathcal{X}$ satisfy $f(x) \leq \beta$?

If there is a nondeterministic algorithm that correctly finds the answer to the CP and such that the longest computation that confirms a "yes" answer is bounded by a polynomial in the size of the instance, the CP belongs to the class $\#\mathcal{P}$.

A counting problem CP is $\#\mathcal{P}-$complete, if it is in $\#\mathcal{P}$ and for all $CP' \in \#\mathcal{P}$ there is a *parsimonious transformation* such that $CP' \propto_p CP$. $\propto_p$ denotes a parsimonious transformation, which is a polynomial time transformation that maintains the number of "yes" answers to DP.

Most MOCO problems are not efficiently solvable [4] because:

1. Most MOCO problems are *intractable*, that is, the size of the set of nondominated points $|\mathcal{Y}_N|$ grows exponentially in the size of the problem instance n. Thus, there is no polynomial p such that $|\mathcal{Y}_N|$ is bounded by $\mathcal{O}(p(n))$.

2. Scalarization methods that also compute unsupported efficient solutions often introduce new capacity constraints to the optimization model and the resulting problems are usually $\mathcal{NP}-$hard and not efficiently solvable.

2.5 Integer Linear Programming

An Integer Linear Programming (ILP) problem is an optimization problem where the objective function and constraints are linear and the decision variables are restricted to be integers. If we have n decision variables and m linear constraints, these problems may be

stated in canonical form:

$$\begin{aligned} \max \quad & f(x) = c^T x \\ s.t. \quad & Ax \leq b, \\ & x \geq 0, \\ & x \in \mathbb{Z}^n, \end{aligned} \tag{2.5}$$

or in standard form:

$$\begin{aligned} \max \quad & f(x) = c^T x \\ s.t. \quad & Ax + z = b, \\ & x \geq 0, \\ & x \geq 0, \\ & x \in \mathbb{Z}^n, \end{aligned} \tag{2.6}$$

where x is the vector of decision variables, c is a vector in $\mathbb{Z}^n$, b is a vector in $\mathbb{Z}^m$ and A is a matrix in $\mathbb{Z}^{m \times n}$. ILP problems are usually analyzed in standard form, thus ILP problems can be converted to standard form by eliminating inequalities, introducing slack variables z and replacing variables that are not sign-constrained with the difference of two sign-constrained variables.

It is known that a general ILP problem is $\mathcal{NP}-$complete [11].

A first approach to solving an ILP problem is to solve its relaxation, which results by removing the constraint that x takes only integer values. However, the solution obtained for the relaxation may not take only integer values and rounding each variable may result in a solution that not only is suboptimal but may also be infeasible. Alternatively, ILP problems may be solved using exact algorithms that find an optimal solution when the problem is feasible or report that there is no solution if the problem is infeasible, and algorithms that provide suboptimal solutions, which may be more tailored for a specific problem (heuristic algorithms) or are higher level methodologies that are not problem dependent (metaheuristics) [13].

The running time for heuristic algorithms is not guaranteed to be polynomial, but empirical evidence suggests that some of these algorithms find a good solution fast. Greedy Randomized Adaptive Search, Simulated Annealing, Tabu Search, ant Colony Optimization, Genetic Algorithms and Neural Network techniques are some of the most important metaheuristics for ILP problems [13].

Some of the exact algorithms are the Cutting-Planes approach, the Branch-and-Bound approach, the Additive Algorithm, the Branch-and-Cut algorithm, the Branch-and-Price algorithm, the Branch-Cut-and-Price algorithm, Lagrange Relaxation and Decomposition methods [13]. In order to get a solution for an ILP problem, one of these algorithms may be implemented or we may use an already existing ILP solver.

In the ILP version of branch-and-bound approach, the ILP is relaxed and solved for the continuous optimal solution. In a maximization problem, the value of the objective function for the relaxed ILP problem is an upper bound on the optimal ILP objective and any feasible point $f(x)$ where x takes only integer values may be used as a lower bound on the optimal ILP objective value. These bounds are used to add constraints to the sub-problems created during the branch-and-bound approach [14].

If the solution for the relaxed ILP does not take only integer values, a variable with fractional value is selected and two sub-problems are created by adding constraints that

define the round up and round down values as a lower bound and upper bound of the selected variable, respectively. Thus, the prior solution becomes infeasible but no feasible integer values are eliminated. These two new sub-problems are solved and the procedure repeated for all sub-problems found unless one of the following situations occur:

- the linear sub-problem is infeasible;
- the optimal linear sub-problem solution only takes integer values;
- the value of the objective function for the optimal linear sub-problem solution is lower than the the current lower bound.

The lower bound on the optimal objective value is updated each time an optimal linear sub-problem solution that only has integer values and has an higher objective value is found.

2.6 The Shortest Path Problem

The Shortest Path Problem (SPP) is a combinatorial optimization problem. Given a directed graph $G = (V, A)$ with source node s, target node t and distance d_{ij} for each arc (i, j) in A (a positive integer), the SPP finds the path $P = (s = i_0, i_1, \ldots, i_{k-1}, i_k = t)$ that minimizes the distance. Its ILP formulation is as follows.

$$\min f(x) = \sum_{(i,j)\in A} d_{ij}x_{ij} \tag{2.7a}$$

$$s.t \quad \sum_{j:(i,j)\in A} x_{ij} = \sum_{j:(j,i)\in A} x_{ji} + b_i, \quad \forall i \in V, b_i = \begin{cases} 1, \text{ if } i = s \\ -1, \text{ if } i = t \\ 0, \text{ otherwise} \end{cases} \tag{2.7b}$$

$$x_{ij} \in \{0, 1\}, \ \forall (i, j) \in A. \tag{2.7c}$$

Constraint (2.7b) guarantees that all feasible solutions are acyclic paths that start in s and end in t. For $(i, j) \in A$, the variable x_{ij} is set to 1 if arc (i, j) is included in the path, otherwise x_{ij} is set to 0.

In this thesis, we examine SPP with two objective functions. Therefore, we define the Biobjective Shortest Path Problem (BSPP): Given a directed graph $G = (V, A)$ with source node s, target node t, distance d_{ij} and cost c_{ij} for each arc (i, j) in A, we want to find the path $P = (s = i_0, i_1, \ldots, i_{k-1}, i_k = t)$ that minimizes the cost and the distance. Thus we get the ILP formulation (2.8)

$$\min f(x) = \left(\sum_{(i,j)\in A} d_{ij}x_{ij}, \sum_{(i,j)\in A} c_{ij}x_{ij} \right)$$

$$s.t \quad \sum_{j:(i,j)\in A} x_{ij} = \sum_{j:(j,i)\in A} x_{ji} + b_i, \quad \forall i \in V, b_i = \begin{cases} 1, \text{ if } i = s \\ -1, \text{ if } i = t \\ 0, \text{ otherwise} \end{cases}$$

$$x_{ij} \in \{0, 1\}, \ \forall (i, j) \in A.$$

It is known that finding an efficient solution for the the BSPP (2.8) is $\mathcal{NP}-$complete and $\#\mathcal{P}-$complete in acyclic directed graphs [21]. It is also known that for the multiobjective

SPP the efficient set may grow exponentially in the size of the problem, even for the biobjective case [10].

A label setting algorithm, presented in Algorithm 1, has been proposed in [15], under the assumption that the cost and distance are non-negative for all arcs. Let v_i be a node of G. A label of v_i is a $2+3$ tuple (d, c, v_j, ℓ, k) where $v_j \neq v_i$ is a node of G, ℓ is the number of a label of node v_j, and k is the index of the label at node v_i. Thus, a label is a vector made up of a distance component, a cost component, a node predecessor label, identifying the node from which the label was obtained, a further label indicating from which of the several labels of the predecessor it was computed, and a label number at the current node. A list of temporary labels $\mathcal{TL}$, which is kept in lexicographic order for the first two components of the labels are concerned, and a list of permanent labels $\mathcal{PL}$, which will identify efficient paths, are used.

Algorithm 1: Biobjective label setting algorithm

Input : A graph $G = (V, A)$ with 2 arc costs (distance and cost)
Initialization: Create label $\mathcal{L} = (0, 0, 0, 0, 1)$ at node s and let $\mathcal{TL} := \{\mathcal{L}\}$
while $\mathcal{TL} \neq \varnothing$ **do**
  Let label $L = (d, c, v_h, \ell, k)$ of node v_i be the lexicographically smallest label in $\mathcal{TL}$;
  Remove L from $\mathcal{TL}$ and add it to $\mathcal{PL}$;
  for *all* $v_j \in V$ *such that* $(v_i, v_j) \in A$ **do**
    Create label $L' = (d + d_{ij}, c + c_{ij}, v_i, k, w)$ as the next label at node v_j, where w is the is the number of the label at node v_j, and add it to $\mathcal{TL}$;
    Delete all temporary labels of node v_j dominated by L', delete L' if it is dominated by another label of node v_j ;
  end
end
Use the predecessor labels in the permanent labels to recover all efficient paths from s to other nodes of G;
Output : All efficient paths from node s to all other nodes of G.

In the original algorithm in [15] the output consists of all efficient paths from node s to all other nodes of G, but we are only interested in the paths from node s to node t.

This page is intentionally left blank.

Chapter 3

Hypervolume Scalarization of the Biobjective Shortest Path Problem

In this chapter, we present the hypervolume scalarized formulation of the BSPP in terms of ILP and introduce its linearization. We also prove the computational complexity of the hypervolume scalarized formulation and propose an algorithm for solving it, with pruning conditions to improve its performance.

3.1 Hypervolume Scalarizations

For a given point set $S \subset \mathbb{R}^m$ and reference point r, the *hypervolume indicator* of S is the measure of the region $D(S, r)$, weakly dominated by S and (assuming minimization) bounded above by r [8], that is,

$$D(S, r) = \bigcup_{p \in S} \{z \in \mathbb{R}^m \mid p \leqq z \leqq r\},$$

such as represented in Figure 3.1.

For $m = 2$ the measure corresponds to the area and for $m = 3$ the measure corresponds to the volume.

For a given point r, named reference point, an hypervolume scalarization consists of finding the point $p \in \mathcal{Y}_N$ that maximizes the measure of $\{z \in \mathbb{R}^m \mid p \leqq z \leqq r\}$. Thus the hypervolume scalarization is formulated as follows:

$$\begin{aligned} &\max_{x \in \mathcal{X}} \prod_{j=1}^{m} (r_j - f_j(x)) \\ &s.t. \quad f_j(x) \leq r_j, \quad \text{for } j = 1, \ldots, m, \end{aligned} \tag{3.1}$$

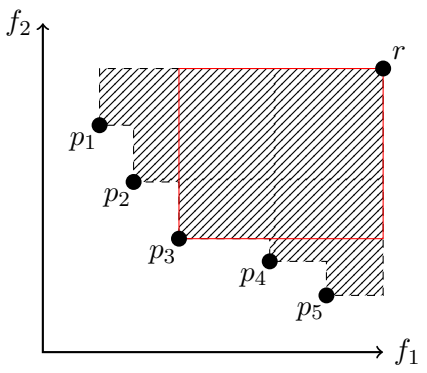

Figure 3.1: Example of region $D(S, r)$ for $\mathbb{R}^2$. For $S = \{p_1, p_2, p_3, p_4, p_5\}$, the hypervolume indicator of S is the area of the shaded region. The point p_3 maximizes the measure.

where, for a feasible solution x such that $f(x)$ weakly dominates r, the objective function computes the measure of the region weakly dominated by $f(x)$ and bounded above by r.

It has been shown [20] that for $r \in \mathbb{R}^m$ every optimal solution of the scalarized problem (3.1) is weakly efficient for (2.1) and every efficient solution $\overline{x}$ of (2.1) can be determined as an

optimal solution of a corresponding scalarized problem using $r = f(\overline{x})$. The hypervolume scalarization does not rely on convexity assumptions, thus it can be used for MOCO problems without being restricted to supported efficient solutions.

3.2 Hypervolume Scalarization of the Shortest Path Problem

The hypervolume scalarization of the BSPP (2.8) leads to the following optimization problem with $r = (r_1, r_2)^T \in \mathbb{R}^2_{\geq}$ as the reference point, presented in its integer programming version.

$$
\begin{aligned}
\max\ & h(x) := \left(r_1 - \sum_{(i,j)\in A} d_{ij}x_{ij}\right) \cdot \left(r_2 - \sum_{(i,j)\in A} c_{ij}x_{ij}\right) \\
s.t\quad & \sum_{(i,j)\in A} d_{ij}x_{ij} \leq r_1 \\
& \sum_{(i,j)\in A} c_{ij}x_{ij} \leq r_2 \\
& \sum_{j:(i,j)\in A} x_{ij} = \sum_{j:(j,i)\in A} x_{ji} + b_i, \quad \forall i \in V, b_i = \begin{cases} 1, \text{ if } i = s \\ -1, \text{ if } i = t \\ 0, \text{ otherwise} \end{cases} \\
& x_{ij} \in \{0,1\},\ \forall (i,j) \in A.
\end{aligned}
\tag{3.2}
$$

3.3 Linearization of the Hypervolume Scalarization

In order to be able to use an ILP solver, we need to linearize the problem above. We have

$$
\begin{aligned}
& \left(r_1 - \sum_{(i,j)\in A} d_{ij}x_{ij}\right) \cdot \left(r_2 - \sum_{(i,j)\in A} c_{ij}x_{ij}\right) \\
&= \sum_{(i,j)\in A} \sum_{(\ell,k)\in A} d_{ij}c_{\ell k}x_{ij}x_{\ell k} - \sum_{(i,j)\in A} \left(r_2 d_{ij} + r_1 c_{ij}\right) x_{ij} + r_1 r_2 \\
&= \sum_{(i,j)\in A} \sum_{(\ell,k)\in A} Q_{ij\ell k}x_{ij}x_{\ell k} - \sum_{(i,j)\in A} \left(r_2 d_{ij} + r_1 c_{ij}\right) x_{ij} + r_1 r_2,
\end{aligned}
$$

where we define $Q_{ij\ell k} = d_{ij}c_{\ell k}$ for all $(i,j) \in A,\ (\ell,k) \in A$.

The objective function is linearized by introducing m^2 (because $|A| = m$) new variables $y_{ij\ell k} = x_{ij}x_{\ell k}$, that attain value 1 if and only if $x_{ij} = 1$ and $x_{\ell k} = 1$, which is ensured by the following constraints:

$$
\begin{aligned}
& y_{ij\ell k} \leq y_{iji j}, \quad \forall (i,j),(\ell,k) \in A,\ (i,j) \neq (\ell,k) \\
& y_{ij\ell k} \geq y_{iji j} + y_{\ell k\ell k} - 1, \quad \forall (i,j),(\ell,k) \in A.
\end{aligned}
$$

Additional constraints are required to handle symmetry, that is, $y_{ij\ell k} = y_{\ell kij}$. Thus, we

get the ILP formulation (3.3) with $\mathcal{O}(m^2)$ constraints:

$$
\begin{aligned}
\max \; & \sum_{(i,j)\in A} \sum_{(\ell,k)\in A} Q_{ij\ell k} y_{ij\ell k} - \sum_{(i,j)\in A} \left(r_2 d_{ij} + r_1 c_{ij}\right) y_{ijij} + r_1 r_2 \\
s.t \quad & \sum_{(i,j)\in A} d_{ij} y_{ijij} \leq r_1 \\
& \sum_{(i,j)\in A} c_{ij} y_{ijij} \leq r_2 \\
& \sum_{j:(i,j)\in A} y_{ijij} = \sum_{j:(j,i)\in A} y_{jiji} + b_i, \quad \forall i \in V, b_i = \begin{cases} 1, \text{ if } i = s \\ -1, \text{ if } i = t \\ 0, \text{ otherwise} \end{cases} \\
& y_{ij\ell k} = y_{\ell kij}, \quad \forall (i,j),(\ell,k) \in A \\
& y_{ij\ell k} \leq y_{ijij}, \quad \forall (i,j),(\ell,k) \in A, \; (i,j) \neq (\ell,k) \\
& y_{ij\ell k} \geq y_{ijij} + y_{\ell k\ell k} - 1, \quad \forall (i,j),(\ell,k) \in A \\
& y_{ij\ell k} \in \{0,1\}, \; \forall (i,j),(\ell,k) \in A.
\end{aligned}
\tag{3.3}
$$

A possible approach for solving the hypervolume scalarization is to use this linearized formulation within an ILP solver.

3.4 Complexity

In order to analyze the complexity of problem (3.2) we introduce the Constrained Shortest Path Problem (CSPP) [23]. The goal of the CSPP is to find the least cost path obeying a set of resource constraints. Given a graph $G = (V, A)$ with $|V| = n$ and $|A| = m$, a source node s and a target node t, and k resource limits $\lambda^{(1)}$ to $\lambda^{(k)}$. Each arc $a = (i, j)$ has a cost c_{ij} and uses $r_{ij}^{(p)}$ units of resource p, $1 \leq p \leq k$. Costs and resources are assumed to be nonnegative and are additive along paths.

Let us consider the case where $k = 1$. If the resource is the distance, with a resource limit $\lambda^{(1)}$ and each arc $a = (i, j)$ with an associated use of resource $r_{ij}^{(1)} = d_{ij}$ units of resource , then the ILP formulation of the CSPP with a single resource is:

$$
\begin{aligned}
\min \; & f(x) := \sum_{(i,j)\in A} c_{ij} x_{ij} \\
s.t \quad & \sum_{(i,j)\in A} d_{ij} x_{ij} \leq \lambda^{(1)} \\
& \sum_{j:(i,j)\in A} x_{ij} = \sum_{j:(j,i)\in A} x_{ji} + b_i, \quad \forall i \in V, b_i = \begin{cases} 1, \text{ if } i = s \\ -1, \text{ if } i = t \\ 0, \text{ otherwise} \end{cases} \\
& x_{ij} \in \{0,1\}, \; \forall (i,j) \in A.
\end{aligned}
\tag{3.4}
$$

Now, let us consider the following decision version of the CSPP. For a constant β, is there a solution x such that $f(x) \leq \beta$ within the constraints of (3.4)? Thus we have the following

problem:

$$\begin{aligned} & \text{Is there a solution } x \\ s.t \quad & \sum_{(i,j)\in A} d_{ij}x_{ij} \leq \lambda^{(1)} \\ & \sum_{(i,j)\in A} c_{ij}x_{ij} \leq \beta \\ & \sum_{j:(i,j)\in A} x_{ij} = \sum_{j:(j,i)\in A} x_{ji} + b_i, \quad \forall i \in V, b_i = \begin{cases} 1, \text{ if } i = s \\ -1, \text{ if } i = t \\ 0, \text{ otherwise} \end{cases} \\ & x_{ij} \in \{0,1\}, \ \forall (i,j) \in A? \end{aligned} \tag{3.5}$$

The decision version of (3.2) is: For a constant γ, is there a solution x such that $h(x) \geq \gamma$ within the constraints of (3.2)? Thus we have the following problem:

$$\begin{aligned} & \text{Is there a solution } x \\ s.t \quad & \left(r_1 - \sum_{(i,j)\in A} d_{ij}x_{ij}\right) \cdot \left(r_2 - \sum_{(i,j)\in A} c_{ij}x_{ij}\right) \geq \gamma && \text{(3.6a)} \\ & \sum_{(i,j)\in A} d_{ij}x_{ij} \leq r_1 && \text{(3.6b)} \\ & \sum_{(i,j)\in A} c_{ij}x_{ij} \leq r_2 && \text{(3.6c)} \\ & \sum_{j:(i,j)\in A} x_{ij} = \sum_{j:(j,i)\in A} x_{ji} + b_i, \quad \forall i \in V, b_i = \begin{cases} 1, \text{ if } i = s \\ -1, \text{ if } i = t \\ 0, \text{ otherwise} \end{cases} && \text{(3.6d)} \\ & x_{ij} \in \{0,1\}, \ \forall (i,j) \in A? && \text{(3.6e)} \end{aligned}$$

In the DP (3.6), for $\gamma = 0$ we can ignore constraint (3.6a) because, if (3.6b) and (3.6c) hold, then (3.6a) is true.

Therefore, if we assume $\lambda^{(1)} = r_1$, $\beta = r_2$ and $\gamma = 0$, the decision version of the CSPP and the decision version of the hypervolume scalarized formulation of the BSPP have the same set of constraints. Thus, since CSPP is $\mathcal{NP}-$complete [23], the hypervolume scalarized formulation of the BSPP is also $\mathcal{NP}-$complete.

3.5 Label Setting Approach

In the following, we propose an algorithm for solving the hypervolume scalarized formulation of the BSPP based on the label setting approach discussed in Section 2.6 and similar to the labeling approach used to solve the CSPP problem in [23].

Labeling approaches follow the line of Pareto-optima approaches, in which none of the objectives can be improved without sacrificing at least one of the other objectives, and do not consider dominated paths, that is, if a path is dominated by an already found path, it is immediately discarded. An v-w-path p is dominated if there exists a v-w-path q such that $c(p) \geq c(q)$ and $d(p) \geq d(q)$ with at least one strict inequality, that is, q is more efficient than p with respect to cost and distance.

The standard labeling approaches use a set of labels for each node. Each label represents a path q from s to this node and consists of a tuple of numbers (c, d, h, v_j, ℓ, k) where c, d, h are the cost, distance and hypervolume of path q, respectively, $v_j \neq v_i$ is a node of G, ℓ is the number of a label of node v_j, and k is the number of the label at node v_i. A labeling algorithm now finds all non-dominated labels on every node. Starting with no labels at every node, except for $(0, 0, r_1 r_2, 0, 0, 1)$ at node s, the algorithm extends the label lists by extending non-dominated s-v-subpaths along their outgoing arcs for every node $v \in V$.

Algorithm 1 could be used for solving the hypervolume scalarized formulation of the BSPP by choosing the path with the highest hypervolume from the set of efficient paths returned that hold constraints (3.6b) and (3.6c). However, we can reduce the running time by using the distance and cost constraints for discarding the subpaths that are infeasible at each node. Therefore, a path does not need to be extended and is called non-promising if its minimal cost and minimal distance exceeds the limits. A list of temporary labels $\mathcal{TL}$, which is kept in lexicographic order for the first two components of the label, and a list of permanent labels $\mathcal{PL}$, which will identify efficient paths, are used.

Algorithm 2: Hypervolume scalarization label setting algorithm

Input : A graph $G = (V, A)$ with 2 arc costs (distance and cost)
Initialization: Create label $\mathcal{L} = (0, 0, r_1 r_2, 0, 0, 1)$ at node s and let $\mathcal{TL} := \{\mathcal{L}\}$
while $\mathcal{TL} \neq \varnothing$ **do**
  Let label $L = (d, c, h, v_h, \ell, k)$ of node v_i be the lexicographically smallest label in $\mathcal{TL}$;
  Remove L from $\mathcal{TL}$ and add it to $\mathcal{PL}$;
  for *all* $v_j \in V$ *such that* $(v_i, v_j) \in A$ **do**
    if $d + d_{ij} \leq r_1$ *and* $c + c_{ij} \leq r_2$ **then**
      Create label $L' = (d + d_{ij}, c + c_{ij}, (r_1 - d - d_{ij}) \cdot (r_2 - c - c_{ij}), v_i, k, w)$ as the next label at node v_j, where w is the is the number of the label at node v_j, and add it to $\mathcal{TL}$;
      Delete all temporary labels of node v_j dominated by L' for distance and cost, delete L' if it is dominated by another label of node v_j ;
    end
  end
end
Use the predecessor labels in the permanent labels to recover all efficient paths from s to other nodes of G;
Output : Efficient path from node s to node t a with the highest hypervolume

There are at most n iterations, where labels for n nodes are computed. However, in the worst case, the running time can be exponential, because there might exist an exponential number of labels at each node.

3.6 Pruning Conditions

The label setting algorithm can be improved by considering stronger pruning conditions instead of testing only the feasibility of the path identified by the label.

For a given path q, we introduce the following notation

$$d(q) \coloneqq \sum_{(i,j) \in q} d_{ij} \quad \text{and} \quad c(q) \coloneqq \sum_{(i,j) \in q} c_{ij} \tag{3.7}$$

to denote the total distance and the total cost of path q, respectively, and

$$\overline{h}(q) = (r_1 - d(q)) \cdot (r_2 - c(q)) \tag{3.8}$$

to denote the hypervolume defined by path q with respect to the reference point $r = (r_1, r_2)$.

We define a lower bound $\mathcal{L}^A(q)$ for a given s-v-path q of the s-t-path BSPP as

$$\mathcal{L}^A(q) \coloneqq \left(d(q) + d(q^d), c(q) + c(q^c)\right), \tag{3.9}$$

where q^d and q^c are the shortest v-t-path with respect to distance and cost, respectively. The value of the shortest v-t-path for any node v and for a given objective can be precomputed by reversing the direction of the arcs in G and building the shortest path tree with source in node t.

Additionally, we define the upper bound $\mathcal{U}^B(q)$ on the hypervolume scalarization for path q

$$\mathcal{U}^B(q) \coloneqq \left(r_1 - d(q) - d(q^d)\right) \cdot \left(r_2 - c(q) - c(q^c)\right), \tag{3.10}$$

as the hypervolume is at most the value of the measure of the region weakly dominated by the lower bound $\mathcal{L}^A(q)$ and bounded above by r.

Now, let $P^A(q)$ be the set of feasible s-t-paths for the BSPP, each of which contains s-v-path q. Let $P^B(q)$ be the set of feasible s-t-paths for the hypervolume scalarization of the BSPP, each of which contains s-v-path q. We have $P^B(q) \subseteq P^A(q)$. Then, the following implications hold with respect to path q, given a feasible s-t-path p and a reference point (r_1, r_2) :

Pruning Condition D1. $\mathcal{U}^B(q) \leq \overline{h}(p) \Longrightarrow \overline{h}(\hat{q}) \leq \overline{h}(p)$, *for* $\hat{q} \in P^B(q)$

Proof. If $\hat{q} \in P^B(q)$, then $\overline{h}(\hat{q}) \leq \mathcal{U}^B(q)$. But $\mathcal{U}^B(q) \leq \overline{h}(p)$, thus $\overline{h}(\hat{q}) \leq \overline{h}(p)$. ■

Therefore, if an incumbent path q fulfills condition D1, it cannot improve the best known solution for the hypervolume scalarization and q can be pruned.

Pruning Condition D2. $(r_1, r_2) \ngeq \mathcal{L}^A(q) \Longrightarrow P^B(q) = \emptyset$

Proof. For each path $\hat{q} \in P^B(q)$, we must have $d(\hat{q}) \leq r_1$ and $c(\hat{q}) \leq r_2$ in order to have a feasible solution for the hypervolume scalarization. If $(r_1, r_2) \ngeq \mathcal{L}^A(q)$, then for $\forall \hat{q} \in \mathcal{L}^A(q)$, we have $d(\hat{q}) > r_1 \vee c(\hat{q}) > r_2$. Thus, $\hat{q} \notin P^B(q)$ and $P^B(q) = \emptyset$. ■

Therefore, if an incumbent path q fulfills condition D2, none of the path extensions is feasible for the hypervolume scalarization and it can be pruned.

3.7 Particular Case

One may wonder whether a greedy algorithm, which selects only the path with the larger hypervolume at each node, could solve problem (3.2). In the following we show that is not possible.

Let us consider the problem (3.2) where the reference point is far enough that all solutions are feasible for the first two constraints. For example, let us consider any case where the reference point $r = (r_1, r_2)$ is such that

$$r_1 \geq 1 + \sum_{(i,j)\in A} d_{ij}$$
$$r_2 \geq 1 + \sum_{(i,j)\in A} c_{ij}.$$

Using such a reference point, we have

$$\sum_{(i,j)\in A} d_{ij}x_{ij} \leq \sum_{(i,j)\in A} d_{ij} < r_1$$
$$\sum_{(i,j)\in A} c_{ij}x_{ij} \leq \sum_{(i,j)\in A} c_{ij} < r_2,$$

thus all s-t-paths are feasible according to the distance and cost constraints and do not need to be considered when solving the problem.

Even in this case, we cannot say that there is a polynomial time algorithm for solving (3.2) because if we tried to use an algorithm that stores only the best hypervolume in each step (such as the algorithm introduced in [1] for the SPP in directed acyclic graphs), it would not find the optimal solution for the scalarized problem. The only way we could store only the best hypervolume in each step is if we could prove that, if the hypervolume of the s-v-path x is higher than the hypervolume of the s-v-path y, then any extension of path x has higher hypervolume than the same extension for path y. However this can be disproven by counterexample.

Example. Let $r_1 = 100$ and $r_2 = 90$. Let paths p and q start in node s and end in node v. Let the distance and cost of path p be $d(p) = 6$ and $c(p) = 7.7$, respectively. Let the distance and cost of path q be $d(q) = 9$ and $c(q) = 5$, respectively. Thus,

$$(r_1 - d(p))(r_2 - c(p)) = 7736.2 > 7735 = (r_1 - d(q))(r_2 - c(q))$$

and the hypervolume of path p is higher than the hypervolume of path q.

Now let us consider the arc (v, w) as an extension of paths p and q. Let the distance and cost of arc (v, w) be $d_{vw} = 3$ and $c_{vw} = 5$, respectively. Thus,

$$(r_1 - d(p) - d_{vw})(r_2 - c(p) - c_{vw}) = 7034.3 < 7040 = (r_1 - d(q) - d_{vw})(r_2 - c(q) - c_{vw})$$

and the hypervolume of the extension of path q is higher than the hypervolume of the extension of path q.

Therefore, we have disproven that if the hypervolume of the s-v-path x is higher than the hypervolume of the s-v-path y, then any extension of path x has higher hypervolume than the same extension for path y.

Thus, with this counter example, we show that we need to store all non-dominated label on every node.

This page is intentionally left blank.

Chapter 4

Implementation and Results

In this chapter, we discuss the implementation of the label setting algorithm for the hypervolume scalarization of the BSPP and of the linearization of the hypervolume scalarization of the BSPP using an ILP solver. The benchmark instances used for testing are available in [18]. We discuss the numerical results obtained, focusing on comparing the performance of the algorithm with the performance of the ILP solver and comparing the performance of the algorithm using the pruning conditions. We also analyze the influence of the problem parameters on the performance of the algorithm, focusing on the size and type of graphs, and the influence of restricting the reference point and the resulting decrease on the number of efficient paths.

4.1 Implementation of the Label Setting Algorithm and the Integer Linear Programming Formulation

The label setting algorithm for the hypervolume scalarization of the BSPP, as presented in Algorithm 2, was implemented in C++ and compiled with `gcc` version 8.3.0. The algorithm was implemented with and without the pruning conditions discussed in Section 3.6. The running time was measured only with respect to the label setting algorithm.

SCIP version 7.0.1, with default paramenters, was used as ILP solver. Solving Constraint Integer Programs (SCIP) is a framework to solve constraint integer programs and mixed-integer nonlinear programs implemented as a C callable library [5].

A problem is initiated using **SCIPcreate()** and the default plugins are included using **SCIPincludeDefaultPlugins()**. Then, **SCIPcreateProbBasic()** is used to define the problem and **SCIPsetObjsense()** is used to set the objective function direction to either SCIP_OBJSENSE_MAXIMIZE or SCIP_OBJSENSE_MINIMIZE. Variables are created using **SCIPcreateVarBasic()** and added to the problem with **SCIPPaddVar()**. Linear constraints are created using **SCIPcreateConsBasicLinear()**. We use **SCIPaddCoefLinear()** to define the coefficients of each variable in a constraint and **SCIPcreateConsBasicLinear()** to add a constraint to the problem. Once this process is defined, it is solved using **SCIPsolve().** After this process finishes, using **SCIPgetBestSol()** returns the best feasible primal solution of the problem and **SCIPprintBestSol()** prints the best solution and which variables were chosen. At last, we use **SCIPreleaseCons()** and **SCIPreleaseVar()** to release all the variables and constraints and we use **SCIPfree()** to release the SCIP environment.

4.2 Benchmark Instances

For the experimental analysis, we have used the graphs stored in a website [18], which contains three types of network topologies. Given n nodes, we have

- Random networks: an Hamiltonian cycle is firstly generated to assure there are no nodes that are never visited. The remainder arcs as well as the arc coefficients are randomly generated.
- Square grid: Each node is connected to his nearest neighbors in a square mesh. Coefficient are randomly generated.
- Complete networks: every node in the graph is connect by an arc. Coefficients are randomly generated.

Each instance is characterized by the number of nodes n, the number of arcs m, the network density n/m, the number of objectives and the maximum arc coefficient 1000, that is, the arc coefficient for each objective are uniformly distributed over the set $\{1, \ldots, 1000\}$. We have considered only the instances with two objectives.

For random networks, we have $(n, m) = \{(1000, 3000), (2500, 15000), (5000, 30000)\}$. For square grids, we have $(n, m) = \{(49, 168), (81, 288), (100, 360)\}$. For complete networks, we have $(n, m) = \{(25, 600), (50, 245), (100, 9900)\}$. There is a total of 50 instances for each set. Table 4.1 summarizes the main data about the benchmark instances used. For each set of instances, the table presents the average number of efficients paths EF, computed using Algorithm 1, and the average number of efficient paths when the reference point is such that that we get $\max\{0.5EF_i, 1\}$ efficient paths for each instance (EF 50%) and $\max\{0.1EF_i, 1\}$ efficient paths for each instance (EF 10%), where EF_i is the number of efficient paths for instance i. The reference points were restricted in order to decrease the number of efficient paths and analyze its influence on average running times.

Type	n	m	EF	EF 50%	EF 10%
Random	1000	3000	3.14 ± 1.69	1.44 ± 0.68	1.00 ± 0.00
	2500	15000	6.25 ± 2.91	2.96 ± 1.37	1.00 ± 0.00
	5000	30000	6.10 ± 2.43	2.86 ± 1.21	1.00 ± 0.00
Square	49	168	8.46 ± 3.32	4.04 ± 1.67	1.00 ± 0.00
	81	288	12.22 ± 4.17	5.88 ± 2.05	1.06 ± 0.24
	100	360	17.22 ± 7.22	8.32 ± 3.60	1.42 ± 0.61
Complete	25	600	6.90 ± 3.23	3.22 ± 1.62	1.00 ± 0.00
	50	2450	8.62 ± 3.65	4.04 ± 1.85	1.02 ± 0.14
	100	9900	12.64 ± 4.47	6.02 ± 2.21	1.08 ± 0.27

Table 4.1: Benchmark instances used for testing. Type of graph, number of nodes $|V| = n$, number of arcs $|A| = m$, average number of efficients paths of problem (2.8) (EF, EF 50% and EF 10%). 50 instances were used for each set of type of graph, n and m.

4.3 Numerical Results

All experiments were conducted on a computer cluster with the following specifications:

- Dell PowerEdge R740 Server
- 2 Intel Xeon Silver 4210R 2.4G, 10 Cores / 20 Threads, 9.6GT/s, 13.75M Cache
- 2 32GB RDIMM, 2933MT/s
- 2 480GB SSD SATA
- OS Debian GNU/Linux 10 (buster)
- Slurm Workload Manager

using a `gcc` version 8.3.0 compiler and SCIP version 7.0.1 as ILP solver. We defined a cut-off limit of 3600 seconds for the total time of each instance.

Due to the size of the graphs, the only results obtained using the ILP solver were for the complete network with $n = 25$. On average the running time for the ILP was 470.4 ± 719.6 seconds, versus 0.016 ± 0.010 seconds for Algorithm 2, thus, on average we have a 72566 speedup. The ILP solver also presents a very large variance.

Table 4.3 presents the average running times for Algorithm 2, with and without pruning conditions. For each set of benchmark instances, defined by type of graph, number of nodes $|V| = n$ and number of arcs $|A| = m$, we have the average running time for the label setting Algorithm 2 (time), the average running time and the average speedup for Algorithm 2 using pruning condition D1 (time D1 and speedup D1, respectively), the average running time and average speedup for Algorithm 2 using pruning condition D2 (time D2 and speedup D2, respectively) and the average running time and speedup for Algorithm 2 using pruning conditions D1 and D2 (time D12 andspeedup D12, respectively).

Type	n	time	time 50%	speedup 50%	time 10%	speedup 10%
Random	1000	0.8 ± 0.7	0.4 ± 0.4	3.9	0.2 ± 0.3	4.7
	2500	28.9 ± 20.4	9.3 ± 10.5	7.5	1.6 ± 2.0	42.4
	5000	158.9 ± 93.3	43.9 ± 43.2	17.7	7.8 ± 11.6	102.2
Square	49	0.0 ± 0.0	0.0 ± 0.0	1.4	0.0 ± 0.0	2.1
	81	0.1 ± 0.0	0.0 ± 0.0	2.0	0.0 ± 0.0	3.53
	100	0.1 ± 0.0	0.0 ± 0.0	1.9	0.0 ± 0.0	2.9
Complete	25	0.0 ± 0.0	0.0 ± 0.0	8.1	0.0 ± 0.0	40.5
	50	0.0 ± 0.0	0.0 ± 0.0	9.8	0.0 ± 0.0	134.9
	100	0.3 ± 0.1	0.1 ± 0.0	9.1	0.3 ± 0.1	313.7

Table 4.2: Results for testing with stricter reference points. For each set of benchmark instances, defined by type of graph, number of nodes $|V| = n$ and number of arcs $|A| = m$, we have the average running time for the label setting algorithm (2) (time) and the average running times and speedups for Algorithm (2) when the reference point is such that we get only a percentage of the efficient paths.

Type	n	time	time D1	speedup D1	time D2	speedup D2	time D12	speedup D12
Random	1000	0.8 ± 0.7	0.8 ± 0.7	1.0	0.0 ± 0.1	56.7	0.0 ± 0.1	56.6
	2500	28.9 ± 20.4	26.0 ± 16.1	1.5	8.4 ± 12.2	60.0	6.8 ± 9.6	60.4
	5000	158.9 ± 93.3	154.5 ± 85.7	1.0	53.9 ± 80.4	96.5	42.5 ± 59.0	100.6
Square	49	0.0 ± 0.0	0.0 ± 0.0	1.1	0.0 ± 0.0	2.2	0.0 ± 0.0	2.3
	81	0.1 ± 0.0	0.0 ± 0.0	2.0	0.0 ± 0.0	3.8	0.0 ± 0.0	3.7
	100	0.1 ± 0.0	0.1 ± 0.0	1.5	0.1 ± 0.0	2.3	0.1 ± 0.0	2.2
Complete	25	0.0 ± 0.0	0.0 ± 0.0	1.4	0.0 ± 0.0	1.5	0.0 ± 0.0	1.9
	50	0.0 ± 0.0	0.0 ± 0.0	2.0	0.0 ± 0.0	1.4	0.0 ± 0.0	2.5
	100	0.3 ± 0.1	0.1 ± 0.0	2.7	0.3 ± 0.1	1.1	0.1 ± 0.1	2.7

Table 4.3: Results for the pruning conditions testing. For each set of benchmark instances, defined by type of graph, number of nodes $|V| = n$ and number of arcs $|A| = m$, we have the average running time for the label setting Algorithm 2 (time) and the average running times and speedups using pruning conditions.

Table 4.2 presents the average running time for Algorithm 2 using reference points such that all efficient paths of BSPP are feasible for its hypervolume scalarization (time) and the average running times and speedups using reference points such that only 50% (time 50% and speedup 50%) and 10% (time 10% and speedup 10%) of the efficient paths are feasible.

4.3.1 Pruning Conditions

The speedup values for random networks in table 4.3 are not very significant when using only pruning condition D1, but the speedup results are much better for pruning condition D2. The speedup value obtained using both conditions is only slightly higher than the speedup value obtained using only D2. Thus, pruning condition D2 is the main contributor to the speedup in random networks and using condition D1 may not be essential to obtain better running times.

The speedup values for square grids and complete networks are not very high, but may be more significant when dealing with larger graphs. For square grids, pruning condition D2 is the main contributor to the speedup and for complete networks pruning condition D1 is the main contributor.

Combining pruning conditions does not offer better results than using only the pruning condition with the highest speedup average.

4.3.2 Reference Points and Efficient Paths

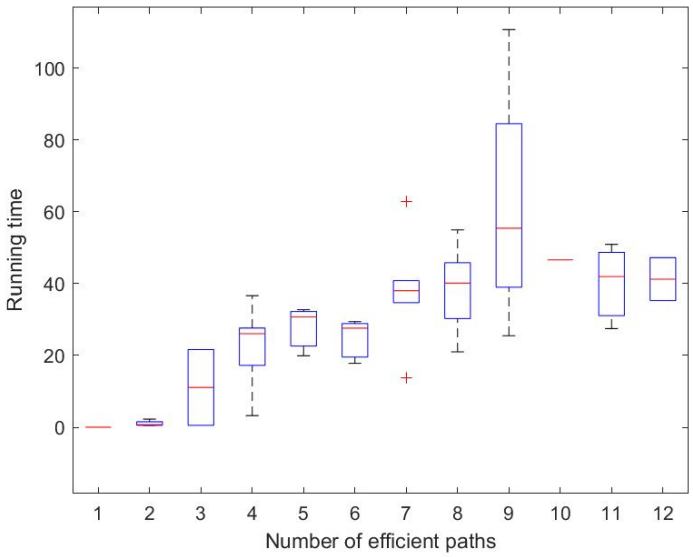

Figure 4.1: Running time for the label setting algorithm 2 for random networks with $n = 2500$ grouped by number of efficient paths.

Looking at Table 4.2, selecting a stricter reference point seems to result in a more significant speedup for random networks and complete networks. These results must be due to the fact that restricting the reference point results in decreasing the number of efficient paths and the number of efficient paths seems to have a bigger impact on random and complete networks than on square grids, as can be observed in the examples in Figures 4.1, 4.2 and 4.3, where we observe the boxplots of running times grouped by number of efficient

paths for some sets of instances. Thus, it seems that trying to restrict the reference point may not be very effective for square grids but results in significant speedups for the other graphs.

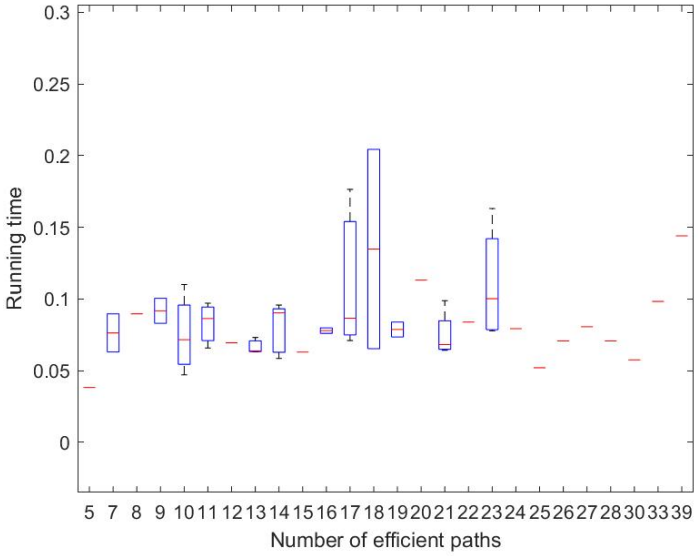

Figure 4.2: Running time for the label setting algorithm 2 for square grids with $n = 100$ grouped by number of efficient paths.

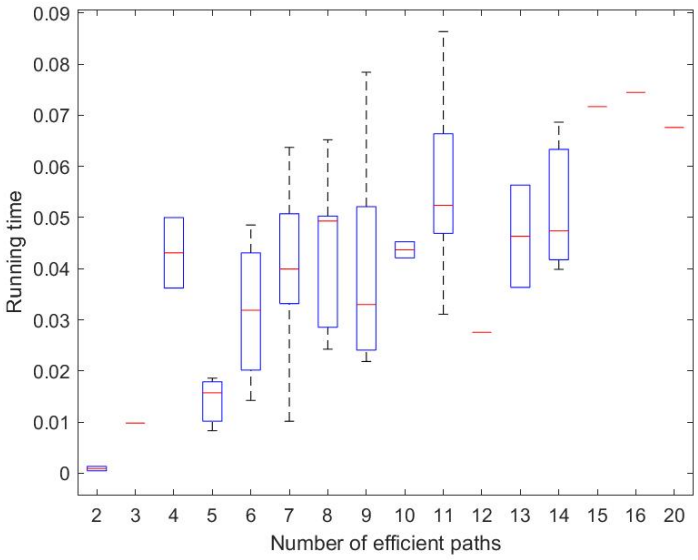

Figure 4.3: Running time for the label setting algorithm 2 for complete networks with $n = 50$ grouped by number of efficient paths.

We must take into consideration the fact that the speedup values obtained are much higher for a very restrictive reference point and these reference points were computed by first solving the BSPP and then choosing points that allowed only for the chosen percentage of efficient paths. The goal of restricting the reference point is to improve the performance of the algorithm, so we would need to find another way of selecting a strict reference point. However, any other way of severely restricting the reference point could result in an unfeasible problem.

4.3.3 Problem Parameters

Besides the type of graph, we may analyze the influence of the number of nodes and arcs in the performance of the algorithm. Figures 4.4, 4.5 and 4.6 show the average running time versus the number of nodes on the left and the number of arcs on the right for random networks, square grids and complete networks, respectively. In those figures we have the average running time for the label setting algorithm 2, the average running time for algorithm 2 using pruning condition D1, the average running time for algorithm 2 using pruning condition D2 and the average running time for algorithm 2 using pruning conditions D1 and D2.

The figures show that the improvements in the running time that result from using the best pruning condition for each type of network are more significant for larger graphs, with more nodes and arcs.

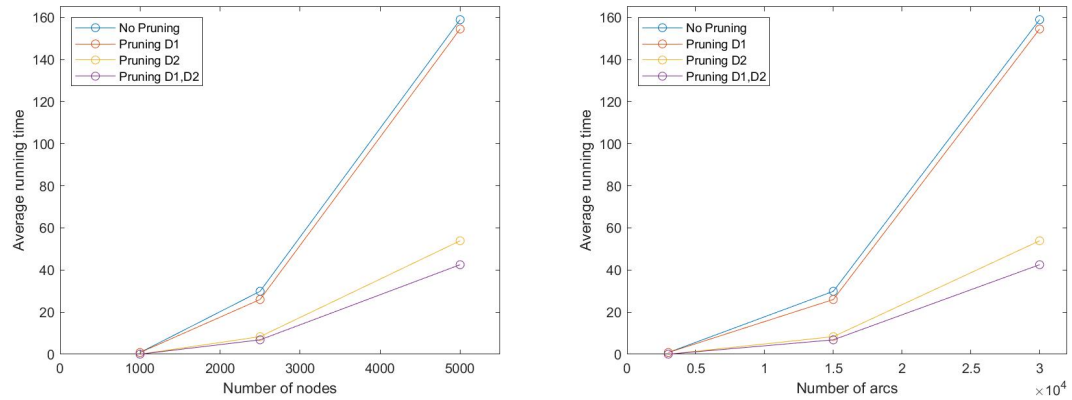

Figure 4.4: Average running time for the label setting algorithm 2 for random networks versus the number of nodes on the left and the number of arcs on the right, with and without pruning conditions.

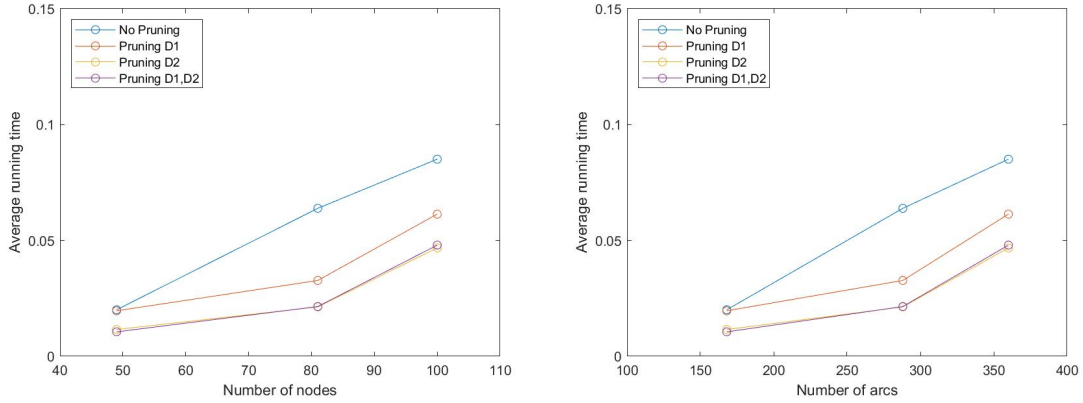

Figure 4.5: Average running time for the label setting algorithm 2 for square grids versus the number of nodes on the left and the number of arcs on the right, with and without pruning conditions.

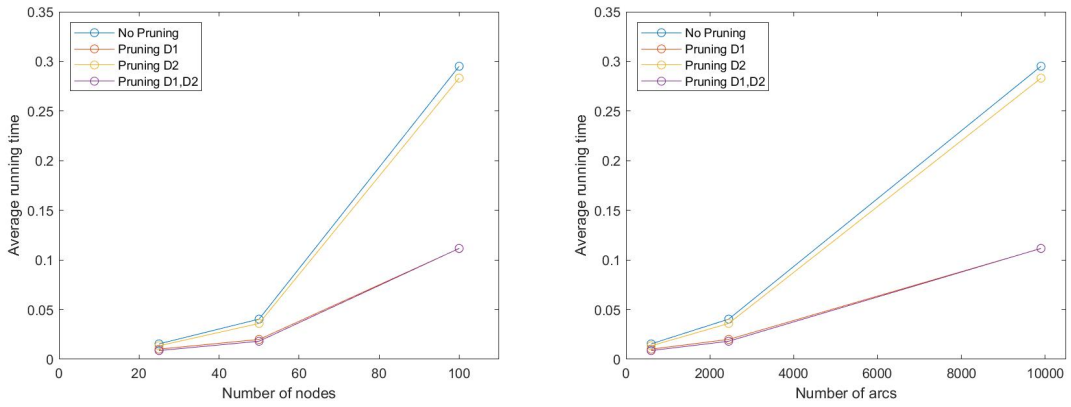

Figure 4.6: Average running time for the label setting algorithm 2 for complete networks versus the number of nodes on the left and the number of arcs on the right, with and without pruning conditions.

4.4 Discussion

Pruning condition D2 is the main contributor to the speedup in random networks and square grids. For complete networks pruning condition D1 is the main contributor. If we combine pruning conditions, the results offered are similar to the results obtained using only the pruning condition with the highest speedup average. Using the best pruning condition for each type of network offer better improvements in the running time for larger graphs, with more nodes and arcs.

Selecting a stricter reference point seems to result in a more significant speedup for random networks and complete networks due to the number of efficient paths having a bigger impact on running times for random and complete networks than on square grids. However, finding a way to severely restrict the reference point while avoiding an unfeasible problem may be computationally expensive.

This page is intentionally left blank.

Chapter 5

Hypervolume Scalarization of the Biobjective Minimum Spanning Tree Problem

In this chapter, we present the hypervolume scalarized formulation of the Biobjective Minimum Spanning Tree Problem (BMSTP) and introduce its linearization in order to further illustrate the application of the previous concepts to a structurally different biojective graph problem. We also prove the computational complexity of the hypervolume scalarized formulation and propose an algorithm for solving this formulation.

5.1 Minimum Spanning Tree Problem

Given a graph $G = (V, E)$, with $|V| = n$, $|E| = m$, cost c_e and distance d_e for each unordered edge $e \in E = \{1, \ldots, m\}$ (a positive integer), the goal of the BMSTP is to find the spanning tree of minimum cost and distance. Thus we get the Integer Linear Programming (ILP) formulation (5.1)

$$\min f(x) = \left(\sum_{e \in E} c_e x_e, \ \sum_{e \in E} d_e x_e \right) \tag{5.1a}$$

$$s.t \quad \sum_{e \in E} x_e = n - 1, \tag{5.1b}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \quad \forall \ S \subset V, S \neq \emptyset, \ S \neq V \tag{5.1c}$$

$$x_e \in \{0, 1\}, \ \forall e \in E, \tag{5.1d}$$

where constraint (5.1b) guarantees that the solution is a spanning subgraph and constraint (5.1c) guarantees that there are no cycles in the solution, that is, any subset of vertices S of V contains at most $|S| - 1$ edges. For $e \in E$, the variable x_e is set to 1 if edge e is included in the spanning tree T, otherwise x_e is set to 0.

It is known that the multiobjective MSTP (5.1) is $\mathcal{NP}$-complete [3] and that the efficient set may grow exponentially in the size of the problem, even for the biobjective case [9]. However, the $\mathcal{P}$-completeness of the multiobjective MSTP is still open because it does not follow from intractability due to the fact that counting the spanning trees of a graph is easy [12].

Practical applications of the BMSTP include the construction of powerline networks, in which you may want the lenght of the network to be as low as possible while having a cost of construction as low as possible.

5.2 Hypervolume Scalarization of the Biobjective Minimum Spanning Tree Problem

The hypervolume scalarization of the BMSTP (5.1) leads to the following optimization problem with $r = (r_1, r_2)^T \in \mathbb{R}^2_{\geq}$ as the reference point, presented in its integer programming version.

$$\max h(x) := \left(r_1 - \sum_{e \in E} c_e x_e\right) \cdot \left(r_2 - \sum_{e \in E} d_e x_e\right) \tag{5.2a}$$

$$s.t \quad \sum_{e \in E} c_e x_e \leq r_1 \tag{5.2b}$$

$$\sum_{e \in E} d_e x_e \leq r_2 \tag{5.2c}$$

$$\sum_{e \in E} x_e = n - 1, \tag{5.2d}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \quad \forall\ S \subset V, S \neq \emptyset,\ S \neq V \tag{5.2e}$$

$$x_e \in \{0, 1\},\ \forall e \in E. \tag{5.2f}$$

5.3 Linearization of the Hypervolume Scalarization

In order to be able to use an ILP solver, we need to linearize the problem above. We have

$$\begin{aligned}
&\left(r_1 - \sum_{i \in E} c_i x_i\right) \cdot \left(r_2 - \sum_{j \in E} d_j x_j\right) \\
=&\sum_{i \in E} \sum_{j \in E} c_i d_j x_i x_j - \sum_{i \in E} \left(r_2 c_i + r_1 d_i\right) x_i + r_1 r_2 \\
=&\sum_{i \in E} \sum_{j \in E} Q_{ij} x_i x_j - \sum_{i \in E} \left(r_2 c_i + r_1 d_i\right) x_i + r_1 r_2,
\end{aligned}$$

where we define $Q_{ij} = c_i d_j$ for all $i,\ j \in E$. The objective function is linearized by introducing m^2 (because $|E| = m$) new variables $y_{ij} = x_i x_j$, that attain value 1 if and only if $x_i = 1$ and $x_j = 1$, which is ensured by the following constraints:

$$\begin{aligned}
&y_{ij} \leq y_{ij}, \quad \forall i, j \in E,\ i \neq j \\
&y_{ij} \geq y_{ii} + y_{jj} - 1, \quad \forall i, j \in E,\ i < j.
\end{aligned}$$

Additional constraints are required to handle symmetry, that is, $y_{ij} = y_{ji}$. Thus, we get the ILP formulation (5.3) with $\mathcal{O}(m^2)$ constraints:

$$\begin{aligned}
\max & \sum_{i\in E}\sum_{j\in E} Q_{ij}y_{ij} - \sum_{i\in E}\left(r_2c_i + r_1d_i\right)y_{ii} + r_1r_2 \\
s.t \quad & \sum_{i\in E} c_iy_{ii} \leq r_1 \\
& \sum_{i\in E} d_iy_{ii} \leq r_2 \\
& \sum_{i\in E} y_{ii} = n-1, \\
& \sum_{i\in E(S)} y_{ii} \leq |S|-1, \quad \forall\ S\subset V, S\neq\emptyset,\ V \\
& y_{ij}\in\{0,1\},\ \forall i,j\in E \\
& y_{ij} = y_{ba}, \quad \forall i,j\in E \\
& y_{ij} \leq y_{ii}, \quad \forall i,j\in E,\ i\neq j \\
& y_{ij} \geq y_{ii} + y_{jj} - 1, \quad \forall i,j\in E,\ i<j.
\end{aligned} \tag{5.3}$$

A possible approach for solving the hypervolume scalarization is to use this linearized formulation within an ILP solver.

5.4 Complexity

In this section we will use a combinatory formulation for the MSTP. Given a graph $G = (V, E)$, with $|V| = n$, $|E| = m$, cost c_e and distance d_e for each unordered edge $e \in E = \{1, \ldots, m\}$ (a positive integer), the goal of the BMSTP is to find the spanning tree of minimum cost and distance, that is, to find the spanning tree $T = (V, E_1),\ E_1 \subseteq E$ such that

$$c(T) = \sum_{e\in T} c_e \quad \text{and} \quad d(T) = \sum_{e\in T} d_e \tag{5.4}$$

are minimized. Thus we get the following formulation for the BMSTP

$$\min_{T\in\mathcal{T}}\ \left(c(T), d(T)\right), \tag{5.5}$$

where $\mathcal{T}$ is the set of spanning trees of graph G. Using this notation, we have the following formulation for the hypervolume scalarization of the BMSTP with $r = (r_1, r_2)^T \in \mathbb{R}^2_{\geq}$ as the reference point:

$$\max_{T\in\mathcal{T}}\ \left(r_1 - c\left(T\right)\right)\cdot\left(r_2 - d\left(T\right)\right) \tag{5.6a}$$

$$s.t \quad c(T) \leq r1 \tag{5.6b}$$

$$d(T) \leq r_2. \tag{5.6c}$$

In order to analyze the complexity of problem (5.5) we introduce the Constrained Minimum Spanning Tree Problem (CMSTP) [17]. The goal of the CMSTP is to find the least cost spanning tree obeying a set of resource constraints, such as a maximum value for the total distance of the tree. Given a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, and distance limit β, let $\mathcal{T}$ denote the set of spanning trees of G. Each edge e has a cost c_e and distance

d_e. Costs and distances are assumed to be nonnegative and are additive along trees. Thus we have the following combinatory formulation:

$$\min_{T \in \mathcal{T}} \; c(T) \tag{5.7a}$$

$$s.t \quad d(T) \leq \beta. \tag{5.7b}$$

The decision version of the CMSTP is: For a constant γ, is there a solution T such that T is a spanning tree of G with cost $c(T) \leq \gamma$ and distance $d(T) \leq \beta$?

Now, let us consider the decision version of (5.6): For a constant λ, is there a solution T such that $(r_1 - c\,(T)) \cdot (r_2 - d\,(T)) \geq \lambda$ within the constraints of (5.6)?

For $\lambda = 0$ we can ignore the constraint $(r_1 - c\,(T)) \cdot (r_2 - d\,(T)) \geq \lambda$ because, if (5.6b) and (5.6c) hold, then $(r_1 - c\,(T)) \cdot (r_2 - d\,(T)) \geq \lambda$ is true.

Therefore, if we assume $\lambda = 0$, $\beta = r_2$ and $\gamma = r_1$, the decision version of the CMSTP and the decision version of the hypervolume scalarized formulation of the BMSTP have the same set of constraints. Thus, since CMSTP is $\mathcal{NP}-$complete [2], the hypervolume scalarized formulation of the BMSTP is also $\mathcal{NP}-$complete.

5.5 Branch-and-Bound Approach

A branch-and-bound approach may be used to solve both the BMSTP [22] and its hypervolume scalarization. The initial upper bound for the BMSTP may be computed by summing the first $n-1$ highest edge values for cost and summing the first $n-1$ highest edge values for distance. Using this initial upper bound, a corresponding initial lower bound for the hypervolume scalarized version is computed.

The branching scheme is defined by, at each node, selecting an edge e of G and creating two subproblems. In the first one, edge e is mandatory, that is, it must be in the spanning tree, while in the second problem edge e is forbidden. The heuristic to select e may search for the edge not yet considered such that $\min\{c_e, d_e\}$, where c_e and d_e are the cost and the distance of edge e, respectively, is minimal. Each subproblem is solved using the same method, unless one of the following situations occur:

- the last mandatory or forbidden edge results in a feasible solution or in an unfeasible problem;
- in the BMSTP, the objective value for the solution of the subproblem is dominated by the current upper bound;
- in the case of its hypervolume scalarization, the objective value for the solution of the subproblem is lower than the lower bound.

When a feasible solution for BMSTP is found for one of the subproblems and its objective value dominates the upper bound, the upper bound is updated. When a feasible solution for its hypervolume scalarization is found and its objective value is higher than the lower bound, the lower bound for the hypervolume is updated.

5.6 Pruning Conditions

The branch-and-bound approach can be improved by considering stronger pruning conditions similar to the conditions used for the knapsack problem [16].

Let us assume that all instances of (5.1) are defined such that

$$c_1 \leq c_2 \leq \ldots \leq c_m. \tag{5.8}$$

Let $\mathcal{S}_m$ denote the symmetric group of order m and $\pi \in \mathcal{S}_m$ denote a permutation $\{1, \ldots, m\}$. Consider π such that

$$d_{\pi(1)} \leq d_{\pi(2)} \leq \ldots \leq d_{\pi(m)}. \tag{5.9}$$

Using the sorted coefficients c_i and $b_{\pi(i)}$, we derive the following lower bound $\mathcal{L}^A$ for any efficient solution of (5.1):

$$\mathcal{L}^A \coloneqq \left(\sum_{i=1}^{n-1} c_i,\ \sum_{i=1}^{n-1} d_{\pi(i)} \right), \tag{5.10}$$

which results from the fact that a spanning tree has $n-1$ edges. A related upper bound for (5.2) can be derived:

$$\mathcal{U}^B \coloneqq \left(r_1 - \sum_{i=1}^{n-1} c_i \right) \cdot \left(r_2 - \sum_{i=1}^{n-1} d_{\pi(i)} \right). \tag{5.11}$$

Now, in the context of the branch and bound approach, let us introduce the following definitions. Let $\overline{x} \in \{0,1\}^m$ such that $\overline{x}_j = 0$, for $j = \ell+1, \ldots, m$, and $\sum_{i=1}^{\ell} \overline{x}_i \leq n-1$, and let

$$\overline{n} \coloneqq \min \left\{ n - 1 - \sum_{i=1}^{\ell} \overline{x}_i,\ m - \ell \right\}, \tag{5.12}$$

that is, let $\overline{x}$ be a solution such that in the first ℓ edges, at most $n-1$ edges belong to the spanning tree and the last $m-\ell$ edges do not yet belong to the tree and let $\overline{n}$ be the number of edges that have to be added to solution $\overline{x}$ in order to obtain a spanning tree.

We define a lower bound $\mathcal{L}^A(\overline{x})$ as follows

$$\mathcal{L}^A(\overline{x}) \coloneqq \left(\sum_{i=1}^{\ell} c_i \overline{x}_i + \sum_{j=l+1}^{\ell+\overline{n}} c_j,\ \sum_{i=1}^{\ell} d_i \overline{x}_i + \sum_{j \in J} d_{\pi(j)} \right), \tag{5.13}$$

where $J \coloneqq \{j_1, \ldots, j_{\overline{n}}\} \subseteq \{\ell+1, \ldots, m\}$ for which it holds that

$$\pi(j_1) < \pi(j_2) < \ldots < \pi(j_{\overline{n}}) \tag{5.14}$$

and

$$\pi(j_{\overline{n}}) < \pi(j), \quad \forall j \in \{\ell+1, \ldots, m\} \setminus \{j_1, \ldots, j_{\overline{n}}\}. \tag{5.15}$$

The different method of defining lower bound $\mathcal{L}^A(q)$ for the BSPP and lower bound $\mathcal{L}^A(\overline{x})$ results of the fact that a feasible solution for the BMSTP has a fixed number of edges.

Similarly, we define the following upper bound $\mathcal{U}^B(\overline{x})$

$$\mathcal{U}^B(\overline{x}) \coloneqq \left(r_1 - \sum_{i=1}^{\ell} c_i \overline{x}_i - \sum_{j=l+1}^{\ell+\overline{n}} c_j,\ r_2 - \sum_{i=1}^{l} d_i \overline{x}_i - \sum_{j \in J} d_{\pi(j)} \right). \tag{5.16}$$

Both lower bound $\mathcal{L}^A(\overline{x})$ and upper bound $\mathcal{U}^B(\overline{x})$ can be used for pruning incumbent solutions within the branch-and-bound framework. A solution $\hat{x} \in \mathcal{X}$ is a feasible extension of $\overline{x}$ if and only if it is feasible for (5.1) and $\hat{x}_i = \overline{x}_i$ for $i = 1, \ldots, \ell$. Let $E^A(\overline{x})$ denote the set of all feasible extensions of $\overline{x}$ and let $E^B(\overline{x}) \subseteq E^A(\overline{x})$ denote the set of feasible extensions of $\overline{x}$ that are also feasible for its hypervolume scalarizarion. Then, the following implications hold with respect to path $\overline{x}$, given a feasible $x^* \in \mathcal{X}$ and a reference point (r_1, r_2) :

Pruning Condition E1. $U^B(\overline{x}) \leq h(x^*) \Longrightarrow h(\hat{x}) \leq h(x^*)$, *for* $\hat{x} \in E^B(\overline{x})$

Proof. If $\hat{x} \in E^B(\overline{x})$, then $h(\hat{x}) \leq U^B(\overline{x})$. But $U^B(\overline{x}) \leq h(x^*)$, thus $h(\hat{x}) \leq h(x^*)$. ■

Therefore, if an incumbent solution $\overline{x}$ fulfills condition E1, it cannot improve the best known solution for the hypervolume scalarization and $\overline{x}$ can be pruned.

Pruning Condition E2. $(r_1, r_2) \ngeq L^A(\overline{x}) \Longrightarrow E^B(\overline{x}) = \emptyset$

Proof. For each path $\hat{x} \in P^B(\overline{x})$, we must have $c(\hat{x}) \leq r_1$ and $d(\hat{x}) \leq r_2$ in order to have a feasible solution for the hypervolume scalarization. If $(r_1, r_2) \ngeq L^A(\overline{x})$, then for $\forall \hat{x} \in L^A(\overline{x})$, we have $c(\hat{x}) > r_1 \vee d(\hat{x}) > r_2$. Thus, $\hat{x} \notin P_B(\overline{x})$ and $P^B(\overline{x}) = \emptyset$. ■

Therefore, if an incumbent path $\overline{x}$ fulfills condition E2, none of the path extensions is feasible for the hypervolume scalarization and it can be pruned.

This page is intentionally left blank.

Chapter 6

Conclusion

6.1 Tasks Scheduling

The timeline of the first semester is illustrated in Figure 6.1. During the first few weeks of work, I reviewed the state of art of Multiobjective Combinatorial Optimization and some graph problems (namely, the Shortest Path Problem). Then, I wrote about the basic concepts about Multiobjective Combinatorial Optimization and studied the hypervolume scalarization and linearization of the graph problems. For the following weeks I proved the computational complexity of the hypervolume scalarized problem using the Constrained Shortest Path Problem and studied the dynamic programming algorithms used for solving this problem. After that, I analyzed a labeling approach for hypervolume scalarized problem and studied some particular cases of the problem. In the last few weeks I wrote the intermediate report.

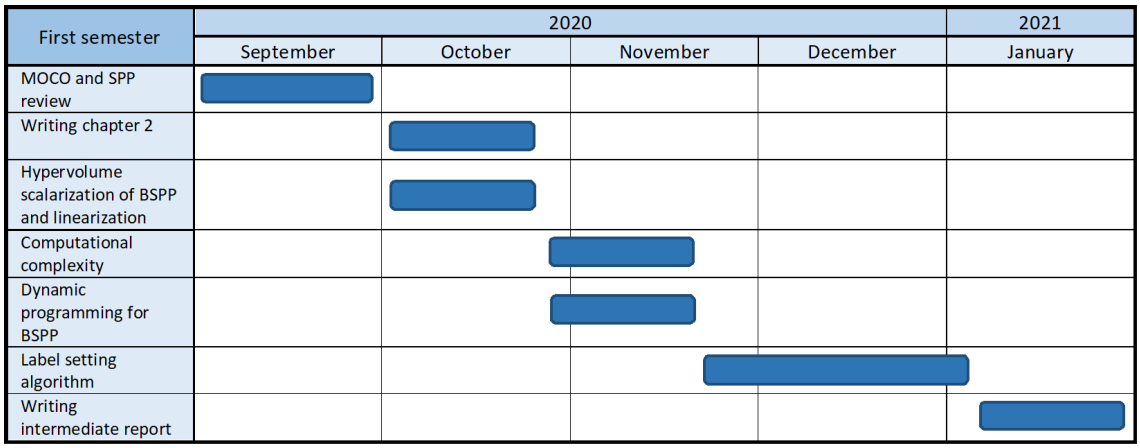

Figure 6.1: Gantt chart for tasks completed during the first semester

The timeline of the second semester is illustrated in Figure 6.2. The first few weeks were spent on reviewing C++ and implementing both versions of the label setting algorithm, focusing on optimizing the code in order to improve the execution time. At the same time, I also focused on adapting the pruning conditions for the Biojective Knapsack Problem to the hypervolume scalarization of the Biojective Shortest Path Problem. The next step was to study the C++ SCIPOPT libraries and solving the linearized version of the hypervolume scalarization of the Biojective Shortest Path Problem using SCIP. After the implementation was completed, the testing phase, using different types of networks, began. During those

weeks, I also focused on the Biojective Minimum Spanning Tree Problem, studying its hypervolume scalarization and linearization. I proved the computational complexity of the problem and analyzed a branch-and-bound approach to solve the problem and pruning conditions to improve performance. The last few weeks were spent analyzing the numerical results and writing the final report.

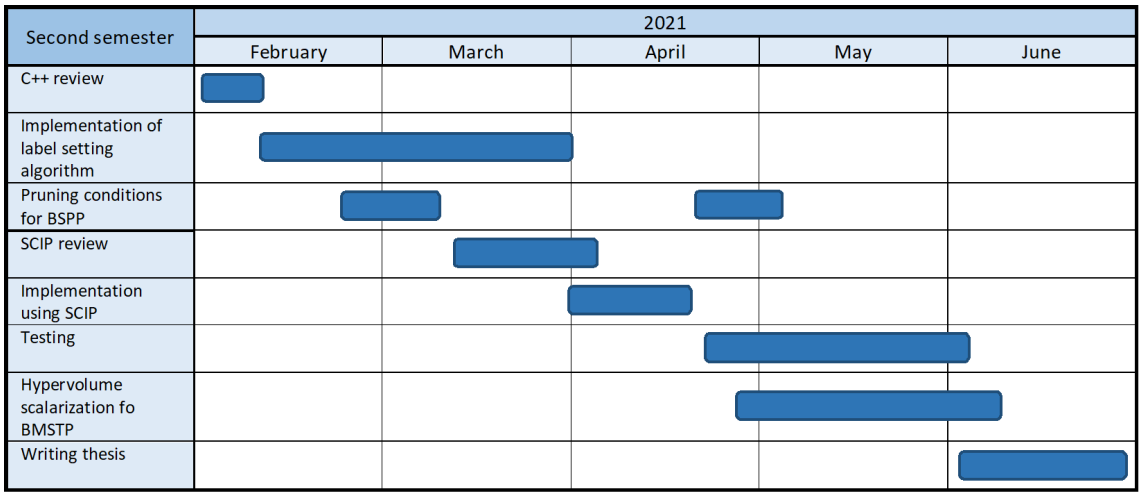

Figure 6.2: Gantt chart for tasks completed during the second semester

6.2 Main Conclusions and Future Work

In this thesis, we introduced the hypervolume scalarization of the Biobjective Shortest Path Problem and it linearization. We proposed a label setting algorithm for solving the hypervolume scalarization and pruning conditions to improve performance. Numerical results showed that the proposed algorithm provides significantly better running times than using the linearization and an ILP solver. Numerical results also show that the best pruning condition depends on the type of network used and that the speed up resulting from the use of pruning conditions is significantly higher for larger graphs.

The Biobjective Minimum Spanning Tree Problem was also studied. We extended the work developed for the Biobjective Knapsack Problem in [16] and for the Biobjective Shortest Path Problem by introducing the hypervolume scalarization and its linearization and proposing an algorithm for solving the hypervolume scalarization and pruning conditions. This algorithm was not implemented, therefore there are no numerical results for this problem. However, given that the Minimum Spanning Tree contains an exponential number of constraints, we expect the branch-and-bound approach to have a better performance.

Extending the work developed for the Biobjective Knapsack Problem to the Biobjective Shortest Path Problem and the Biobjective Minimum Spanning Tree Problem shows that applying the hypervolume scalarization to biobjective combinatorial problems is simple and we may adapt algorithms already used for solving the biobjective versions of the problems in order to solve the hypervolume scalarizations. On the other hand, the pruning conditions seem to be general enough to be adapted to other problems.

The next step of this work would to generalize hypervolume scalarizations to problems with higher dimensions and to compare the results obtained using hypervolume scalarizations

with other scalarizations.

References

[1] GeeksforGeeks shortest path in directed acyclic graph. `https://www.geeksforgeeks.org/shortest-path-for-directed-acyclic-graphs/`. Accessed: 2020-12-27.

[2] V. Aggarwal, Y.P. Aneja, and K.P.K. Nair. Minimal spanning tree subject to a side constraint. *Computers Operations Research*, 9(4):287–296, 1982.

[3] P. Camerini, G. Galbiati, and F. Maffioli. The complexity of multi-constrained spanning trees. In L. Lovasz, editor, *Theory of Algorithms*, pages 53–101, North-Holland, Amsterdam, 1984.

[4] M. Ehrgott. *Multicriteria Optimization*. Springer-Verlag, Berlin, Heidelberg, 2005.

[5] G. Gamrath, D. Anderson, K. Bestuzheva, W.K. Chen, L. Eifler, M. Gasse, P. Gemander, A. Gleixner, L. Gottwald, K. Halbig, G. Hendel, C. Hojny, T. Koch, P. Le Bodic, S.J. Maher, F. Matter, M. Miltenberger, E. Mühmer, B. Müller, M.E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, C. Tawfik, S. Vigerske, F. Wegscheider, D. Weninger, and J. Witzig. *The SCIP Optimization Suite 7.0*. ZIB-Report. Zuse Institut Berlin, 2020.

[6] M.R. Garey and D.S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman Co., USA, 1990.

[7] A.M. Geoffrion. Proper efficiency and the theory of vector maximization. *Journal of Mathematical Analysis and Applications*, 22(3):618 – 630, 1968.

[8] A. Guerreiro, C. Fonseca, and L. Paquete. The hypervolume indicator: Problems and algorithms. *ACM Computing Surveys (in press)*, 2021.

[9] H. Hamacher and G. Ruhe. On spanning tree problems with multiple objectives. *Annals of Operation Research*, 52:209–230, 1994.

[10] P. Hansen. Bicriterion path problems. In G. Fandel and T Gal, editors, *Multiple Criteria Decision Making Theory and Application*, volume 177, pages 109–127, Berlin, 1979. Springer Verlag.

[11] R. Kannan and Clyde L. Monma. On the computational complexity of integer programming problems. In Rudolf Henn, Bernhard Korte, and Werner Oettli, editors, *Optimization and Operations Research*, pages 161–172, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg.

[12] G. Kirchhoff. On the resolution of the equations to which one is lead in the investigation of the linear distribution of galvanic currents (German). *Annalen der Physik and Chemie*, 72:497–508, 1847.

[13] S. Kumar, M.K. Luhandjula, E. Munapo, and B.C. Jones. Fifty years of integer programming: A review of the solution approaches. *Asia Pacific Business Review*, 6(3):5–15, 2010.

[14] A.H. Land and A.G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[15] E.Q.V. Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2):236 – 245, 1984.

[16] L. Paquete, B. Schulze, M. Stiglmayr, and A.C. Lourenço. Computing representations using hypervolume scalarizations. *Computers Operations Research*, page 105349, 2021.

[17] R. Ravi and M.X. Goemans. The constrained minimum spanning tree problem. In R. Karlsson and A. Lingas, editors, *Algorithm Theory — SWAT'96*, pages 66–75, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[18] J.L.E. Santos. Public library of benchmark instances. `http://www.mat.uc.pt/~zeluis/INVESTIG/MSPP/DataBase/instanceDataBase.htm`. Accessed: 2021-04-17.

[19] B. Schulze. *New perspectives on multi-objective knapsack problems*. PhD thesis, University of Wuppertal, 2017.

[20] B. Schulze, M. Stiglmayr, L. Paquete, C. Fonseca, D. Willems, and S. Ruzika. On the rectangular knapsack problem: approximation of a specific quadratic knapsack problem. *Mathematical Methods of Operations Research*, 92:107–132, 2020.

[21] P. Serafini. Some considerations about computational complexity for multi objective combinatorial problems. In J. Jahn and W. Krabs, editors, *Recent Advances and Historical Development of Vector Optimization*, pages 222–232, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[22] F. Sourd and O. Spanjaard. A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem. *INFORMS Journal on Computing*, 20:472–484, 08 2008.

[23] M. Ziegelmann. *Constrained Shortest Paths and Related Problems*. PhD thesis, Universität des Saarlandes, 2001.