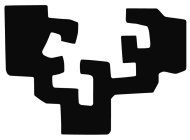
eman ta zabal zazu



Universidad del País Vasco
Euskal Herriko Unibertsitatea

Konputazio Zientziak eta Adimen Artifizialaren Saila
Departamento de Ciencias de la Computación e Inteligencia Artificial

Advances in Branch-and-Fix methods to solve the Hamiltonian cycle problem in manufacturing optimization

by

Maialen Murua Etxeberria

Supervised by Roberto Santana and Diego Galar

Donostia - San Sebastián, January 2022

Etxekoei.

Acknowledgments

Primero de todo, me gustaría agradecer a mis supervisores Roberto y Diego por su ayuda, orientación y amplios consejos. A Roberto, porque sería imposible enumerar aquí todo lo que me ha enseñado. Más allá de los conceptos me ha transmitido lo que es realmente "investigar". También me ha enseñado a escribir y la importancia que tiene repasar una frase no dos veces sino cuatro. Finalmente, ha sabido animarme en los momentos que me he sentido más perdida en este camino. A Diego, por nuestras amplias y enriquecedoras discusiones en las reuniones de Zaragoza y por guiarme tan bien en este camino. Su conocimiento y experiencia en los momentos clave de la tesis han sido fundamentales.

Bigarrenik, eskerrak eman nahi dizkiet Euskal Herriko Unibertsitako Intelligent Systems Group ikerketa taldeko ikasle eta irakasleei. Irakasleei une oro beraien sostengua sentitu ahal izan dudalako eta ikasleei askotan tesiaren zama arintzen lagundu didatelako. Azpimarratu nahiko nuke tesi hau aurrera eramateko beharrezkoak izan direla unibertsitatearen baliabide konputazionalak. Honi dagokionean, bereziki eskertu nahiko nuke Jose Pascual eta Borja Calvok beti erakutsi didaten prestutasuna.

Bestalde, nire esker hona adierazi nahi diot Tecnaliari eta bertako hainbat pertsonei. Tesi honen lehen zatian lankide izan nituen Fabrikazio Aurreratuko kideei eta bereziki WAAM teknologiako lan taldeari. Une oro teknologia horrekiko erakutsi zuten grinagatik eta behin eta berriz nire zalantzak argitzeko pazientzia izateagatik. Behar izan dudanean hor egon diren Smart Systems-eko nire taldekideei ere, eskerrik asko. Master amaierako garaietatik hasita azpimarratzekoa izan da Alberto Diez eta Fernando Botoren babesa. Fernandori zinez eskertu nahi diot edozein zalantzaren aurrean beti eman didan pazientziazko laguntza eta kodearekin ezin ditudan arazoetan luzatutako ezagutza. Tesi honen kodean izandako arazoetan laguntzegatik Gaizka Arizmendiarrieta ere eskertu nahiko nuke. Azkenik, Smart Systems-eko zuzendaritzari eta Tecnaliako Pertsonak sailari eskertza luzatu nahi diet.

Moreover, I owe a debt of gratitude to Prof. Bjarne Bergquist for taking me as a visiting researcher so warmly in Luleå University of Tecnology. Also, to Marduch Tadaros for our discussions on Optimization and Operations Research that have in some way enriched this dissertation. I would always remember the people that I met in Luleå that contributed in some way to the last period of this dissertation.

I would also like to acknowledge the great work of John Kennedy and Elizabeth Thompson. Their comments and suggestions related to the English writing have enhanced the readability of the published research work and this dissertation.

Azkenik, ingurukoen babesa ezinbestekoa izan da lan hau aurrera ateratzeko, beraiengatik ez balitz ez nintzen honaino iritsiko. Bihotz-bihotzez mila esker!

Contents

Part V Appendices

Glossary

AI Artificial Intelligence.
AM additive manufacturing.
ASTM American Society for Testing Materials.

BB branch-and-bound.
BF Branch-and-Fix.

CAD computer-aided design.
CAM computer-aided manufacturing.
CIM computer-integrated manufacturing.
CNC computer numerical control.

DED direct energy deposition.
DSS decision support system.

EA evolutionary algorithm.

FDM fused deposition modeling.

GA genetic algorithm.
GMAW gas metal arc welding.
GTAW gas tungsten arc welding.

HC Hamiltonian cycle.
HCP Hamiltonian cycle problem.
HV hypervolume.

LP linear program.

MAT medial axis transformation.
MDP Markov decision process.
MO multi-objective.

NSGA-II non-dominated sorting genetic algorithm.

OR Operations Research.

PAW plasma arc welding.
PDG part decomposition graph.
PS Pareto set.

RQ research question.

SB strong branching.

TSP traveling salesman problem.

UI unsolved instance.

WAAM wire arc additive manufacturing.

1

Introduction

Optimization problems deal with the selection of the best element (with respect to a certain criterion) from a set of available alternatives. Many engineering problems related to manufacturing, such as the design of machine tools, airplanes and automobiles, are multi-criteria optimization problems. The manufacturing industry [151] has a profound impact on the economy and societal progress through its development of technological changes and innovations. In the paradigm of Industry 4.0 [67], solving optimization problems through Operations Research (OR) and building decision support systems (DSSs) for the manufacturing industry are crucial.

Additive manufacturing (AM) is a manufacturing process that consists of depositing material layer-by-layer to create three dimensional objects [96]. The ability to produce customized products and complex and lightweight designs makes AM one of technology trends of Industry 4.0. The technology has attracted the interest of the research community in the past few years; for example, researchers have examined repeatability and reproducibility [49], lack of control over certain defects [135] and process planning [87]. One of the most studied problems in the field of process planning has been the tool-path problem [41] [126]. To this point, studies of tool-path generation in AM have mainly been based on geometric analysis, but this is not usually optimal from a manufacturing engineering point of view. There is a need to further examine the optimization of the tool-path in AM technology.

The Hamiltonian cycle problem (HCP) [19] is a problem of Graph Theory with many applications in OR, especially because of its close relationship with the traveling salesman problem (TSP) [74]. The HCP consists of finding a cycle in a given graph that passes through every single vertex exactly once, or determining that this cannot be achieved. Most approaches to solve the problem only work for undirected graphs, or those considered for directed graphs are not fully implemented and not tested for large graphs.

One of the approaches to solve the HCP in directed graphs is the Branch-and-Fix (BF) method [51], an exact method based on linear programming

that uses embedding to convert the discrete optimization of the HCP into a continuous one. It is a versatile algorithm because, while using directed graphs for its internal representation, it can also solve undirected graphs by representing them as double connected, without the need of adding extra vertices. However, one of the limitations of linear programming based methods is that the number of constraints increases with the size of the problem. Another limitation is that they consume time exploring solution spaces that lead to infeasible solutions. The BF consists of constructing a logical tree, where at each node of the tree, two linear programs (LPs) are solved. In previous research, this method has been tested using graphs with a small number of vertices (the maximum number of vertices considered is 64 [19]). Since the method has not been tested in large graphs, the weaknesses and the parts of the algorithm where enhancements can be applied have not been studied. The efficiency of the algorithm is related to the LPs and how the tree structure and the recursive calls are implemented. Proper exploitation of the characteristics of the graphs can improve the efficiency of the algorithm in terms of time and computational cost.

One important point to consider in the design of the algorithm is the branching rule, as the BF belongs to the family of branching algorithms. This means that the search space is recursively split into smaller spaces [107]. The branching rules proposed in the literature are based on the results obtained from the LPs [19], but exploiting more global characteristics of the BF is probably a better option. To the best of our knowledge, this has not yet been proposed. In addition, no previous research on an in-depth investigation of the role played by the branching methods (based on LPs and more global ones) in the behavior of the BF has been conducted.

Finally, in many real-world problems, including manufacturing optimization problems, various criteria should be considered simultaneously, turning optimization problems into multi-objective (MO) optimization problems. The importance of the MO optimization has attracted the interest of scholars [36] [112]. Thus, we can find MO variants of well-known optimization problems, such as the TSP. The HCP has a close relationship with the TSP, but, finding cycles in a graph can be difficult depending on the number of vertices and sparsity. Essentially, it is desirable to define the MO HCP and to find an efficient method to solve it. Here, the graph density plays a major role. Depending on the density of a graph, an HCP instance can be less or more similar to a TSP instance. With higher density, the graph is closer to a complete graph, so that to the TSP.

To summarize, this dissertation will start with the optimization problem of the tool-path and continue with a contribution of a DSS to generate optimal tool-paths in AM technology. Then, it will address the HCP giving the following four methodological contributions: 1) enhancement of the efficiency of the BF in various aspects; 2) the global branching method; 3) a BF collapse algorithm; 4) an extension of the BF to deal with the MO HCP.

1.1 Layout of the dissertation

This dissertation is divided into four parts. Part I introduces basic notions related to the context and the theoretical framework. It defines the objectives of the research, and lists the main contributions. Part I is divided into four separate chapters. Chapter 2 gives some context of manufacturing industry related to the dissertation. Chapter 3 explains mathematical optimization, some basics of Graph Theory, the embedding of the HCP in a continuous problem and the BF method. Chapter 4 gives some context of the HCP by reviewing the relevance of the problem and the approaches to solve it. Chapter 5 describes the purposes of the research. Part II (Chapter 6) describes a contribution related to a manufacturing optimization problem. Part III focuses on methodological contributions to the BF method. This part also is divided into four chapters. Chapter 7 proposes an improvement in the efficiency of the algorithm in terms of time by adding an early subcycle detection step and a degree-based simplification step. Chapter 8 proposes the global branching method and compares different branching rules. Chapter 9 proposes a BF collapse algorithm, whose some of the components were proposed in previous chapters. Chapter 10 introduces the MO HCP and a method to solve it. Part IV gives the main conclusions of the research and suggests future research.

Part I

Preliminaries

2

Manufacturing industry

The manufacturing industry has a profound impact on the economy and societal progress [16]. Starting with the Industrial Revolution, technological changes have mainly taken place in the manufacturing sector. Technological changes and innovations are essential sources for social and economic progress, as more profitable and efficient sectors and firms displace less productive and less profitable ones. Thus, technological change is at the center of modern economic growth. With global competition, change and innovation become even more essential, as there is a need for fast adaptation of production to the ever-changing market requests [130].

The concept of Industry 4.0, or the fourth industrial revolution, is now well understood by companies and research community [67]. It originally was a strategic initiative introduced by the German government to transform the manufacturing industry through digitization and exploitation of new technologies. At this point, Industry 4.0 has been extensively studied in academic research for many years and is also well accepted in industry [123]. On the one hand, academic research focuses on conceptualizing the development of related technologies, business models and methodologies. On the other hand, the industrial sector is interested in changes in industrial machines, intelligent products and potential customers.

This chapter explains the background of AM technology and gives some insights into DSSs in manufacturing environments.

2.1 Additive manufacturing

Technological trends related to Industry 4.0 [66] include: Internet of things, cloud computing, big data and analytics, blockchain, augmented and virtual reality, automation and industrial robotics, cybersecurity, AM, simulation and modeling techniques and cyber physical systems.

Additive manufacturing or 3D printing consists of depositing material layer-by-layer to create a three dimensional object. AM technologies will be widely used to produce customized products and complex and lightweight designs [83]. They offer manufacturers the opportunity to produce prototypes and proof of concept designs, processes which used to slow down product design and manufacturing. There are a variety of processes that belong to the family of AM technologies that can be classified depending on how the material is deposited, the material used and the feedstock state [128].

These new emerging technologies will enable mass production while reducing the costs and increasing the parts functionality [75]. However, physical phenomena have a considerable impact on the product quality that is not yet under control [149] [98]. This occurs mainly because the manufacturing paths are not considered from the beginning when the parts are in the design stage. Moreover, the high variability of AM technologies makes the process of creating general automated design and manufacturing processes more difficult, as the physical phenomena differ from one technology to other.

2.2 Decision support systems in manufacturing

DSSs are information systems that enable decision-making activities, such as operations and planning, and decisions about problems that are not easily to predict in advance. There has been a growth in recent years of the need for quick and precise decision-making; this has made it essential to embrace new technologies [150]. To have insight into operations and assets, firms need to have suitable DSSs, but, it is almost impossible to run a business without using OR to optimize activities and resources. OR deals with advanced analytical methods for decision making. It can be understood as the application of systematic and mathematical systems to study and investigate problems and support humans with data to make correct decisions [148].

There has been an increase of computing capacity and developments in machine learning and big data. Consequently, Artificial Intelligence (AI) has been embraced by research on DSSs [14]. Common AI techniques used to develop DSSs are fuzzy logic, genetic algorithms (GAs), agent-based systems, data mining and neural networks. In the digital era and Industry 4.0 scenario, more and more businesses are implementing AI in their DSS and improving their OR capabilities. AI-based DSSs are used to solve OR problems in manufacturing, including product and process design, scheduling of machines and equipment for optimal utilization, quality, maintenance and fault identification.

The adoption of computer-integrated manufacturing (CIM) offers manufacturers innumerable benefits, allowing them to produce high quality products at low cost [27]. Nevertheless, selecting a suitable CIM is a complex task,

as it involves many parameters to assure that it meets the requirements of a company. DSSs are good approaches to design suitable CIM systems. CIM includes the following stages: design, planning, manufacturing and distribution [64]. It represents the highest level of automation in manufacturing. The next section discusses planning, as this stage is related to the dissertation.

2.2.1 Planning

Process planning is one of the most complex phases of the manufacturing process, as it comprises several sequences that depend on the type of product to be manufactured. For example in AM technologies, the process planning should include 2D slicing into layers, bead geometry, tool-path generation and process parameter selection [117]. Traditional manufacturing technologies, such as machining operations, include design interpretation, selection of machining operations, machine tools and cutting tools and determination of cutting conditions.

Automated process planning is a key factor achieving effectiveness in the era of smart manufacturing in Industry 4.0 [2]. Automated process planning involves connecting computer-aided design (CAD) and computer-aided manufacturing (CAM). Scholars have worked on developing automated process planning systems to provide high production quality and a quick response to firms [133].

Tool-path generation problem is an engineering problem that consists of planning the cutter trajectory relative to the part. It is based on the part model, machining method and a tolerance requirement [76]. The engineering problem has been extensively studied for industrial processes [103], [28], including machining and cutting operations [122], and it can be posed as a multicriteria optimization problem. It is a necessary stage in the automation of control programs of computer numerical control (CNC) systems [55]. Depending on the field, it can be called the tool-path planning problem, cutting path problem, drilling path problem or tool routing problem.

The tool-path cutting problem is directly applicable to a number of processes, including laser cutting operations, where optimal torch path generation has a considerable impact on production time. In this case, the problem consists of finding the path that minimizes the total time required to cut all the parts from a sheet while respecting the precedence constraints [40]. Consequently, several commercial CAD/CAM packages offer automatic torch path sequencing [110]. This problem is studied in other machining operations as well, because optimizing the process has the potential to minimize the financial and environmental costs of producing a part. For instance, in a multi-hole drilling process [39], most of the total time is employed in tool movement and switching. Thus, tool-path optimization is important in cost minimization. Another important factor to consider in optimizing the process is the minimization of the idling time [55].

3

Theoretical background

This chapter gives a theoretical background for the principal methods used throughout the dissertation. First, it presents some notions related to mathematical optimization and common methods to solve the optimization problems. Then, it offers insights into Graph Theory. Finally, it explains the BF method, as well as the embedding used to convert the discrete optimization of the HCP into continuous optimization problem.

3.1 Mathematical Optimization

Mathematical Optimization or Mathematical Programming is a subfield of applied mathematics that deals with the selection of the best element (with respect to a certain criterion) from a set of available alternatives. It is used to solve problems in different disciplines, such as Physics, Biology, Engineering, Economy and Business.

Given a vector space of dimension n C^n, $\boldsymbol{x} \in C^n$ is a vector of decision variables, where b is the number of inequality constraints (g_j) and c is the number of equality constraints (h_l). F is the objective function $F(\boldsymbol{x}) : C^n \rightarrow C^1$. An optimization problem can be posed as follows:

$$\begin{aligned} &\text{optimize} \quad F(\boldsymbol{x}) \\ \text{subject to } &g_j(\boldsymbol{x}) \leq 0,\ j = 1, 2, ..., b, \\ &h_l(\boldsymbol{x}) = 0,\ l = 1, 2, ..., c. \end{aligned} \tag{3.1}$$

Linear programming is a subfield of Mathematical Programming. When $F(\boldsymbol{x})$ is a linear function and the restrictions are linear, we have a linear programming problem. The most common method to solve this problem is the Simplex method [13].

Integer programming is a specific type of linear programming, where some decision variables can take integer values instead of continuous ones. A specific case of integer programming is the 0-1 programming, where the decision variables take value 0 or 1. These programs are constructed adding extra restrictions that limit the decision variables to values lower than or equal to 1. Branch-and-bound (BB) methods [100] are used to solve these type of problems.

3.1.1 Multi-objective optimization

In many real-world problems, various criteria should be considered simultaneously, turning the problem into an MO optimization problem. An MO optimization problem [112] is posed in a similar way to a single-objective optimization problem. The difference is that there are m objective functions and $\boldsymbol{F}(\boldsymbol{x}) \in C^m$ is a vector of objective functions, where $F_i(\boldsymbol{x}) : C^n \rightarrow C^1$. It can be stated as:

$$\begin{aligned} \text{optimize } \boldsymbol{F}(\boldsymbol{x}) &= [F_1(\boldsymbol{x}), F_2(\boldsymbol{x}), ..., F_m(\boldsymbol{x})]^T \\ \text{subject to } g_j(\boldsymbol{x}) &\leq 0, \ j = 1, 2, ..., b, \\ h_l(\boldsymbol{x}) &= 0, \ l = 1, 2, ..., c. \end{aligned} \tag{3.2}$$

In MO optimization, however, there is not usually a globally optimal solution, thus the following concepts are used to represent optimal solutions. A point is Pareto optimal, if there is no other point that improves at least one of its objectives without detriment to another objective. A Pareto set (PS) is the set of all Pareto optimal points [112]. These concepts are mathematically formulated as follows.

Definition 1. *A point, $\boldsymbol{x}^* \in C^n$, is Pareto optimal if there does not exist another point, $\boldsymbol{x} \in C^n$, such that $\boldsymbol{F}(\boldsymbol{x}) \leq \boldsymbol{F}(\boldsymbol{x}^*)$, and $F_i(\boldsymbol{x}) < F_i(\boldsymbol{x}^*)$ for at least one function F_i.*

Definition 2. *A Pareto set is the set of all Pareto Optimal points.*

Definition 3. *A vector of objective functions, $\boldsymbol{F}(\boldsymbol{x}^*) \in C^m$, is non-dominated if there does not exist another vector, $\boldsymbol{F}(\boldsymbol{x}) \in C^m$, such that $\boldsymbol{F}(\boldsymbol{x}) \leq \boldsymbol{F}(\boldsymbol{x}^*)$ with at least one $F_i(\boldsymbol{x}) < F_i(\boldsymbol{x}^*)$. Otherwise, $\boldsymbol{F}(\boldsymbol{x}^*)$ is dominated.*

3.1.2 Methods to solve optimization problems

There are numerous methods to solve optimization problems. These, can be classified depending on the problem type that they solve. The following sections give an overview of some of the methods used in this dissertation.

3.1.2.1 Branch-and-bound methods

BB methods [22] are commonly used to solve integer programming problems, but they can be applied to numerous problems. BB is based on the principle that the total space of feasible solutions can be partitioned into subsets of solutions. These subsets are evaluated systematically in an efficient way (the evaluation is not exhaustive), with the minimization objective function until reaching the optimum solution. The method is usually combined with a linear programming technique, such as Simplex [152].

The entire search space is called the root node, and a decision tree or logical tree is built with branches that lead to nodes corresponding to the subsets. Each child node is a partial solution and belongs to the solution set. Before constructing the logical tree, the upper and lower bounds of a given problem are set based on the optimal solution. In the case of integer programming problems, a relaxation is applied (the restrictions that imply that some decision variables must be integers are not considered). When solving the linear programming problem of the relaxed problem, the upper and lower bounds are calculated. At each level, a decision is made about which node to include in the solution set, the node with the best bound. Hence, the optimal solution can be found by discarding subspaces proved not to contain optimal solutions. The logical tree is illustrated in Figure 3.1.

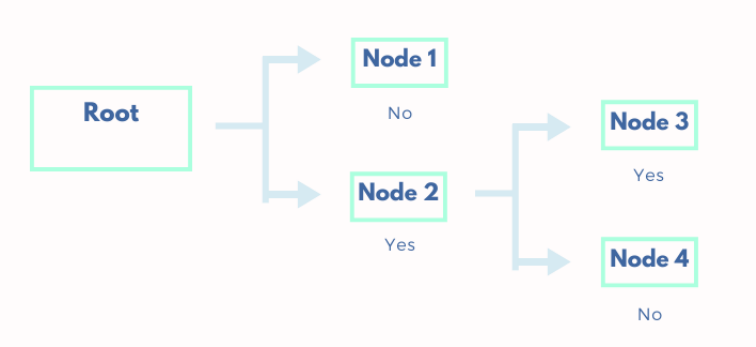


Fig. 3.1: A logical tree based on Yes/No questions where the root and the nodes are shown.

3.1.2.2 Metaheuristic methods

Combinatorial problems have been solved by both exact and metaheuristics methods [26] [119], but in the case of the MO combinatorial problems, the latter method predominates. MO combinatorial optimization problems have

commonly been addressed using metaheuristic techniques because of the computational complexity of MO combinatorial optimization problems [85]. The main advantages of metaheuristics are that they are computationally efficient, general and simple to implement.

Evolutionary algorithms (EAs) are population-based optimization methods based on the theory of natural evolution [50]. The main types of EAs are: GA [70], genetic programming [12], evolutionary programming [9] and estimation of distribution algorithms [99]. This dissertation applies GAs which make use of genetic operators such as selection, crossover and mutation. The idea of these methods is to bias the search process to more promising regions of the search space. There are different kinds of GAs depending on the elements of the population (individuals). The dissertation uses permutation-based GAs [65]. They share the general characteristics of GAs, but have particular characteristics related to the type of solution representation used (permutation-based).

3.2 Graph Theory

Many real-world situations can be represented by a diagram consisting of a set of points joined by certain lines. A mathematical abstraction of that diagram gives rise to the concept of graph.

A *graph* is an ordered triple $(V(G), E(G), \psi_G)$ consisting of a nonempty set $V(G)$ of *vertices*, a set $E(G)$ disjoint of $V(G)$, of *edges*, and an *incidence function* ψ_G that associates with each edge of G an unordered pair of (not necessarily distinct) vertices of G. A *directed graph* is a graph with an incidence function that associates with each arc an ordered pair (not necessarily distinct) vertices. In the context of directed graphs, we substitute $E(G)$ with $A(G)$ where its elements are called arcs instead of edges.

If e is an arc and i and j are vertices such that $\psi_G(e) = ij$, then e is said to *join* i and j; the vertices i and j are called the *ends* of e. For an arc (i, j), the first vertex i is its *tail* and the second vertex j is its *head*. We say that the arc (i, j) *leaves* i and *enters* j. For a vertex i, the *out-degree*, $d^+_G(i)$, is the number of arcs with tail i and the *in-degree*, $d^-_G(i)$, is the number of arcs with head i. We define *outgoing vertices* of a vertex i, $\mathcal{O}(i)$, as the set of heads of the arcs that leave i. The *ingoing vertices* of a vertex i, $\mathcal{I}(i)$, are defined as the set of tails of the arcs that enter i.

An arc with identical ends is called a *loop*, and an arc with distinct ends a *link*. A graph is *simple* if it has no loops. This dissertation considers simple graphs. A simple graph in which each pair of distinct vertices ij is joined by two arcs (i, j) and (j, i) is called a *directed complete graph*.

The adjacency matrix $\boldsymbol{A}(G) = [a_{ij}]$ is an $N \times N$ matrix, where $N = |V(G)|$, defined in Equation (3.3).

$$a_{ij} = \begin{cases} 1 & \text{for } (i,j) \in A, \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

Graph H is a *subgraph* of G (written $H \subseteq G$) if $V(H) \subseteq V(G)$, $A(H) \subseteq A(G)$, and ψ_H is the restriction of ψ_G to $A(H)$.

The graph *density* of a simple graph is the ratio of the number of arcs $|A(G)|$ to the maximum number of possible arcs. A *dense* graph is a graph in which the number of arcs is close to the maximal number. The opposite is a *sparse* graph. For directed simple graphs the graph density is defined in Equation (3.4).

$$D = \frac{|A(G)|}{|V(G)|(|V(G)|-1)} \tag{3.4}$$

A *walk* in G is a finite non-null sequence $W = \{i_0 e_1 i_1 e_2 ... e_k i_k\}$, whose terms are alternately vertices and arcs, such that, for $1 \leq l \leq k$, the ends of e_l are i_{l-1} and i_l. We say that W is a walk from i_0 to i_k, or a (i_0, i_k)-walk. The vertices i_0 and i_k are called the origin and terminus of W, respectively. The integer k is the length of W.

If the arcs $e_1, e_2, ..., e_k$ of a walk W are distinct, W is called a *trail*. If, in addition, the vertices are also distinct, W is called a *path*. A walk is *closed* if it has positive length and its origin and terminus are the same. A closed trail whose origin and internal vertices are distinct is a *cycle*.

The HCP consists of finding a cycle in a given graph that passes through every single vertex exactly once, or determining that this cannot be achieved [19]. These cycles are called Hamiltonian cycle (HC)s. If a graph contains at least one HC, we call it a *Hamiltonian graph*. *Hamiltonicity* is a graph's possession of an HC [57].

Figure 3.2 shows a Hamiltonian graph of six vertices. For instance, one of the HCs of the graph is $1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 1$ that can be represented as $h = (1, 2, 3, 6, 5, 4, 1)$. The set of arcs in the cycle is $\{[1,2], [2,3], [3,6], [6,5], [5,4], [4,1]\}$.

3.3 Embedding the Hamiltonian cycle problem in a Markov decision process

There are innumerable investigations [77] [19] [56] that relate stochastic and discrete optimization by embedding the HCP in an Markov decision process (MDP). An MDP is a particular stochastic process that provides mathematical framework for modeling decision making. In aforementioned research works, it is proven that some classical optimization problems can be analyzed by

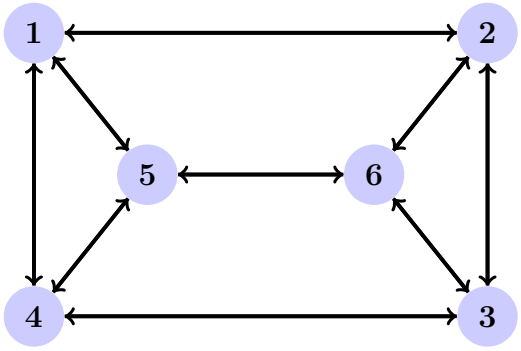


Fig. 3.2: A Hamiltonian graph of six vertices.

embedding suitably in MDPs and building on this basis, new algorithmic approaches are proposed that take advantage of MDPs.

Let $\mathcal{M}$ be an MDP where $\mathcal{S}$ is the *state space*, and $\mathcal{A}$ is the *action space*. The embedding of the HCP is defined in such a way that the vertices in the graph G correspond to the states in $\mathcal{M}$, $\mathcal{S} = V(G)$, and the arcs in G correspond to the actions in the MDP, $\mathcal{A}$. The set of arcs or actions is defined as $\mathcal{A} = \{(i,a) \mid i,a \in \mathcal{S} \text{ and } (i,a) \in E(G)\}$, where $\mathcal{A}(i)$ is the set of states that can be reached by the actions (i,a).

We define *transition probabilities* as $\{p(j|i,a) = \delta_{aj} \mid (i,j) \in \mathcal{A}\}$, where δ_{aj} is the Kronecker delta. This is interpreted in the HCP context as the probability of entering vertex j in one step by traversing arc (i,a).

We define $\zeta(i,a)$ as the probability of selecting action a in state i. For each state i, we define the vectors $\zeta(i) = (\zeta(i,1), \zeta(i,2), ..., \zeta(i,N))$. The set of those N vectors constitute a *stationary policy* $(\boldsymbol{\zeta})$. In the HCP context, $\zeta(i,a)$ is the probability of traversing arc (i,a) when node i is reached. As at every state an action must be selected, then the components of $\zeta(i)$ add up to 1.

Any stationary policy $\boldsymbol{\zeta}$ induces a *probability transition matrix* $\boldsymbol{P}(\boldsymbol{\zeta})$, of dimension $N \times N$.

$$\boldsymbol{P}(\boldsymbol{\zeta}) = [p(j|i,\zeta)]_{i,j=1}^{N,N}$$

where for all $i, j \in \mathcal{S}$,

$$p(j|i,\zeta) = \sum_{a=1}^{N} p(j|i,a)\zeta(i,a).$$

Let $r(i,a)$ be the reward associated to each action a taken in state i. When the actions are prescribed by a policy, we can define,

$$r(i,\zeta) = \sum_{a=1}^{N} r(i,a)\zeta(i,a),\ i \in \mathcal{S}$$

.

The *reward vector* of $\mathcal{M}$ is the vector containing $r(i,\zeta)$ for each state. $\boldsymbol{r(\zeta)} = [r(1,\zeta), r(2,\zeta), ..., r(N,\zeta)]^T$.

In the context of HCP, the reward vector is used to distinguish between visiting the home vertex (vertex 1 for the sake of simplicity). We define $\boldsymbol{r(\zeta)} = [1, 0, , , 0]^T$.

There are different ways to evaluate the rewards, but in this case the discounted MDP is used, where β is the discounted factor. The discounted factor is the rate at which rewards depreciate with time. That is, a desired policy is the one that achieves the largest reward in short time. β takes values between $(0, 1)$.

We define $\boldsymbol{\nu} = [\nu_1, ..., \nu_N]$ as the *initial probability distribution.* For every stationary policy $\boldsymbol{\zeta}$, we define $x_{ia}(\boldsymbol{\zeta})$ as the *discounted occupational measure of the state-action pair* (i, a) induced by $\boldsymbol{\zeta}$.

$$x_{ia}(\boldsymbol{\zeta}) = \{\boldsymbol{\nu}[I - \beta \boldsymbol{P}(\boldsymbol{\zeta})]^{-1}\}_i \zeta(i,a)$$

where I is the identity matrix.

We define $x_i(\boldsymbol{\zeta})$ as the *discounted occupational measure* of the state i, where $x_i(\boldsymbol{\zeta}) = \sum\limits_{a \in \mathcal{A}(i)} x_{ia}(\boldsymbol{\zeta})$.

Feinberg [56] proved that every HC corresponds to an extreme point in a polytope $\mathbf{X}_\beta$ defined by Equations (3.5-3.7) that uses the described embedding for $\mu = 0$. Later, it was proved that an extreme point of the defined polytope by Equations (3.5-3.7) is an HC (Proposition 4.2. by Borkar et al. [19]).

$$\sum_{i=1}^{N} \sum_{a \in \mathcal{A}(i)} (\delta_{ij} - \beta p(j|i,a)) x_{ia} = \nu_j, \quad j \in \mathcal{S} \tag{3.5}$$

$$\sum_{a \in \mathcal{A}(1)} x_{1a} = \frac{(1-(N-1)\mu)(1-\beta) + \mu(\beta - \beta^N)}{(1-\beta)(1-\beta^N)} \tag{3.6}$$

$$x_{ia} \geq 0, \quad i \in \mathcal{S}, a \in \mathcal{A}(i). \tag{3.7}$$

where $\boldsymbol{\nu} = [\nu_1, ..., \nu_N]$ is the initial probability distribution defined as follows for $\mu \in (0, \frac{1}{N})$. The selection of μ as a small positive parameter is to ensure the mapping between the policies and discounted occupational measures.

$$\nu_i = \begin{cases} 1-(N-1)\mu, \text{ if } i = 1, \\ \mu, \qquad \text{otherwise.} \end{cases}$$

3.4 Branch-and-Fix method

The BF method is based on the embedding of the HCP in an MDP; it uses the polytope $\boldsymbol{X}_\beta$ defined by Feinberg [56]. In that investigation, it was shown that the extreme points of the defined polytope correspond to HCs. Later, it was found by Nguyen that the extreme points can also be 1-randomized policies [121]. Let $\boldsymbol{x}$ be an extreme point in $\boldsymbol{X}_\beta$, the associated policy is either a deterministic policy (thus it is an HC by Proposition 4.2 , Borkar et al. [19]), or it is a 1-randomized policy. This policy has $N+1$ positive entries, instead of having N. So that, there is exactly one $i \in \{1, ..., N\}$ and two different a, $a' \in \mathcal{A}(i)$ satisfying that x_{ia}, $x_{ia'} > 0$.

The BF method avoids arriving at extreme points that induce 1-randomized policies by solving sequences of LPs, two at each branching point of the logical tree. The splitting node is the node (state) where randomization occurs when the first LP is solved. When the solution is a 1-randomized policy (instead of deterministic), for one $i \in \{1, ..., N\}$ there are two positive entries $x_{i,a}$, $x_{i,a'}$ where a, $a' \in \mathcal{A}(i)$. In other words, there is one node i and two arcs that emanate from it. That node is called the splitting node. The BF method is compound in the following stages.

1. **Initialization**. The first LP is solved to find $\boldsymbol{x} \in \boldsymbol{X}_\beta$. If the feasible solution $\boldsymbol{x_0}$ induces a deterministic policy $\boldsymbol{\zeta}_0$, the solution is found; if not, $\boldsymbol{\zeta}_0$ is a 1-randomized policy.
2. **Branching**. The 1-randomized policy $\boldsymbol{\zeta}_0$ serves to identify the splitting node i. Let d be the number of arcs that leave i. d subgraphs $G_1, G_2, ..., G_d$ can be constructed, where in each G_k, (i, a_k) is fixed $k = 1, ..., d$. Notice, that the subgraphs are identical to G in all other vertices and arcs. Fixing an arc implies eliminating other arcs as it is explained in `Update_Adjacency_Matrix` function.
3. **Fixing**. Some rules are applied to fix more arcs in the current subgraph. This is explained in greater detail in `Update_Fixed_Arcs` function.
4. **Iteration**. The second LP is solved to check the feasibility of the current subgraph. If it is found feasible, the algorithm returns to Step 1 with the updated graph. If not, the algorithm returns to Step 2, fixing the following arc that leaves i. When all the branches are explored without finding any HC, the algorithm terminates.

The algorithm finishes when an HC is found or when all the branches have been explored. In this last case, the graph is not Hamiltonian. It takes as an input the adjacency matrix defined in Equation (3.3).

The first LP includes the constraints defined in Equations (3.5-3.7) and the objective function defined in Equation (3.8). The optimization problem consists of minimizing the objective function $F(\mathbf{x})$.

$$F(\mathbf{x}) = \sum_{(i,j)\in\mathcal{U}} \left\{ \sum_{a\in\mathcal{A}(j)} x_{ja} - \beta x_{ij} \right\} \tag{3.8}$$

where $\mathcal{U}$ is the set of fixed arcs.

In the first iteration, as $\mathcal{U}$ is an empty set, no objective function is considered. When the set of fixed arcs contains the final arc $(i_N, 1)$, the term in Equation (3.9) has to be added to the objective function.

$$-\beta^N \sum_{a\in\mathcal{A}(1)} x_{1a} + \beta x_{i_N 1} \tag{3.9}$$

The second LP includes constraints defined in Equation (3.5) and (3.7) and the additional constraints of the fixed arcs defined in Equation (3.10).

Let (i_k, i_{k+1}) be a fixed arc from $\mathcal{U}$. Next, for each fixed arc, we add the constraint posed in Equation (3.10).

$$\begin{cases} \sum\limits_{a\in\mathcal{A}(i_{k+1})} x_{i_{k+1}a} - \beta x_{i_k i_{k+1}} = \mu & \text{if } i_{k+1} \neq 1, \\ -\beta^N \sum\limits_{a\in\mathcal{A}(1)} x_{1a} + \beta x_{i_N 1} = \frac{\mu\beta(1-\beta^{N-1})}{1-\beta} & \text{Otherwise.} \end{cases} \tag{3.10}$$

In this LP, the objective function is defined by Equation (3.11). The optimization problem consists of minimizing the objective function.

$$F(\mathbf{x}) = \sum_{a\in\mathcal{A}(1)} x_{1a} \tag{3.11}$$

The feasible regions of the optimization problems (the first and second LPs) are related to the polytope $\boldsymbol{X}_\beta$ proposed by [51]. In the case of the first LP, the constraints of the optimization problem are defined by Equations (3.5-3.7), whereas, for the second LP, the constraints are defined by Equations (3.5), (3.7) and (3.10).

The pseudocode of the implemented BF algorithm is shown in Algorithm 1. It takes as an input an adjacency matrix and an empty set of fixed arcs ($\emptyset$). It returns value *True* of the boolean variable `found` and the HC, or value *False* of the boolean variable `found` and $\emptyset$, when all the branches have been explored without finding an HC. In the following lines, the constructed functions of the pseudocode are explained in a high level of detail.

- `Second_LP`: implements the second LP mentioned in this section.
- `First_LP`: implements the first LP mentioned in this section.

- `Identify_Splitting_Node`: identifies the node where the branching occurred (splitting node).
- `Identify_HC`: converts the obtained solution from the first LP, $\boldsymbol{x}$ to a deterministic policy $\boldsymbol{\zeta}$.
- `Get_ith_Outgoing_Arc` calculates the i-th arc leaving the splitting node.
- `Update_Adjacency_Matrix`: implements graph reduction by eliminating arcs. If the fixed arc is (i, j), all other arcs that leave i are eliminated. All other arcs that enter vertex j are also eliminated, as well as the arc (j, i) if it exists.
- `Update_Fixed_Arcs`: updates the set of fixed arcs $\mathcal{U}$ by applying the following rules: 1) if there is only one arc leaving a vertex, that arc is fixed; 2) if there is a vertex that can be entered from only one arc, that arc is fixed. This update is performed every time that the adjacency matrix is updated.
- `Refined_Fixed_Arcs`: computes the set of arcs added to $\mathcal{U}$ after calling to `Update_Fixed_Arcs`.

The output of the LPs are the status (feasible/infeasible), the vector of decision variables ($\boldsymbol{x}$) and the objective function (F). The variable `found` takes two values, *True* or *False*, depending if the HC is found or not.

The bound refers to the right-hand term of Equation (3.6) that appears in Step 4 of Algorithm 1.

$$\text{bound} = \frac{(1-(N-1)\mu)(1-\beta)+\mu(\beta-\beta^N)}{(1-\beta)(1-\beta^N)}$$

3.4.1 Extensions to the Branch-and-Fix

An extension of the BF by adding some additional constraints was proposed in [19]. This method also uses the embedding of the HCP in an MDP described in Section 3.3. The main difference is that the parameter μ that appears in the initial probability distribution takes value 0 and has some additional constraints apart from those in Equations (3.5-3.7).

There are some additional constraints proposed by Eshragh et al. [53] that allow using $\mu = 0$, still ensuring the mapping between the policies and discounted occupational measures.

$$\sum_{a\in\mathcal{A}(i)} x_{ia} \leq \frac{\beta}{1-\beta^N},\ i = 2, ..., N, \tag{3.12}$$

Algorithm 1 Branch-and-Fix

```
Input: adjacency_matrix, ∅
Output: True/False, HC/∅
function BRANCH_AND_FIX(adjacency_matrix, 𝒰)
    status, x_2, F(x_2) ← Second_LP(adjacency_matrix, 𝒰)
    if status ≠ feasible or F(x_2) > bound then
        return False,∅
    status, x, F(x) ← First_LP(adjacency_matrix,𝒰)
    if status ≠ feasible then
        return False,∅
    splitting_node ← Identify_Splitting_Node(x)
    if splitting node = ∅ then
        HC←Identify_HC(x)
        return True,HC
    else
        d ← |𝒜(splitting_node)|
        found ←False
        i ← 0
        while (found = False) and (i < d) do
            fixed_arc ← Get_ith_Outgoing_Arc(splitting_node,i)
            new_adjacency_matrix ← Update_Adjacency_Matrix(fixed_arc)
            𝒰 ← Update_Fixed_Arcs(new_adjacency_matrix)
            last_fixed_arcs ← Refined_Fixed_Arcs(𝒰)
            while |last_fixed_arcs|> 0 do
                for arc ∈ last_fixed_arcs do
                    new_adjacency_matrix ←Update_Adjacency_Matrix(arc)
                𝒰 ← Update_Fixed_Arcs(new_adjacency_matrix)
                last_fixed_arcs ← Refined_Fixed_Arcs(𝒰)
            found, HC ← BRANCH_AND_FIX( new_adjacency_matrix, 𝒰)
            i ← i + 1
        if found=True then
            return True, HC
        else
            return False, ∅
```

$$\sum_{a \in \mathcal{A}(i)} x_{ia} \geq \frac{\beta^{N-1}}{1-\beta^N}, \ i = 2, ..., N. \tag{3.13}$$

The authors of the BF extension refer to the constraints (3.12-3.13) as *wedge constraints*. For this reason, we will refer to this extended BF as wedge constraints-based BF.

The wedge constraints, could be added in the original LP. However, these constraints would destroy the 1-randomized policies obtained in the solution, which is undesirable. For that reason, the wedge constrains are included in the second LP which attempt to fathom a branch earlier than in the case

that wedge constraints are not used. The second LP in the case of the wedge constraints-based BF is comprised by Equations (3.14-3.19).

$$F(\mathbf{x}) = \sum_{a \in \mathcal{A}(1)} x_{1a} \tag{3.14}$$

$$\sum_{i=1}^{N} \sum_{a \in \mathcal{A}(i)} (\delta_{ij} - \beta p(j|i,a)) x_{ia} = \delta_{1,j},\ j \in \mathcal{S}, \tag{3.15}$$

$$x_{ia} \geq,\ i \in \mathcal{S},\ a \in \mathcal{A}(i), \tag{3.16}$$

$$\sum_{a \in \mathcal{A}(i)} x_{ia} \leq \frac{\beta}{1-\beta^N},\ i = 2, ..., N, \tag{3.17}$$

$$\sum_{a \in \mathcal{A}(i)} x_{ia} \geq \frac{\beta^{N-1}}{1-\beta^N},\ i = 2, ..., N, \tag{3.18}$$

$$\begin{cases} \sum\limits_{a \in \mathcal{A}(i_{k+1})} x_{i_{k+1}a} - \beta x_{i_k i_{k+1}} = 0, & \text{if } i_{k+1} \neq 1, \\ -\beta^N \sum\limits_{a \in \mathcal{A}(1)} x_{1a} + \beta x_{i_N,1} = 0, & \textit{Otherwise.} \end{cases} \tag{3.19}$$

Note that the objective function (3.14) and constraint (3.16) are the same as in the BF method. In the case of constraints (3.15) and (3.19), they are the same as in the BF method for $\mu = 0$. Constraints (3.17-3.18) are the wedge constraints introduced before. The bound in this case, is the same as in the BF, but for $\mu = 0$,

$$\text{bound}^* = \frac{1}{1-\beta^N}$$

The authors of [19] carried out some tests to compare the performance of the wedge constraints-based BF with the BF. It was observed that for smaller-size graphs there was no improvement for the wedge constraints-based BF, respect to the BF. However, for larger-size graphs, specially for 30-40 vertices, the improvement was noticeable.

In the dissertation we did not consider the wedge constraints, because for the BF they are only applicable in the second LP. As mentioned before, including them in the first LP involves destroying the 1-randomized policies, and hence the branching nature of the BF. These constraints have lead to an improved polytope of the $\boldsymbol{X}_\beta$ [53]. This polytope at the same time was used to build an algorithmic approach called the Cross-entropy/optimization hybrid approach that will be presented in Section 4.2.2.

4

Hamiltonian cycle problem

The HCP is an important combinatorial problem with applications in many areas. This chapter gives some context of the HCP by reviewing its relevance and the approaches used in the literature to solve it.

4.1 Applications of the problem and relevance

The HCP is a Graph Theory problem which is related to other problems such as the Icosian game and the Knight's tour, but the most closely related and best-known problem is the TSP.

4.1.1 Traveling salesman problem

The TSP is the problem of finding the shortest route to N different cities visiting each city once and returning to the city of origin [33]. The distances between all pairs of cities are known. It has often been formulated as a permutation-based problem. A solution of the TSP problem can be represented by a permutation σ of length N, where $\sigma(i) = j$ indicates that the city j is visited at the i-th stage. The HC presented in Section 3.2 related to Figure 3.2 can be represented as a permutation, $\sigma = (236145)$.

Given a matrix $\boldsymbol{C} = [c_{ij}]$ of dimension $N \times N$, with the distances between all pairs of cities, the objective function F is the sum of the distances between all pairs of cities in the order specified by σ:

$$F(\sigma) = \sum_{i=2}^{N} c_{\sigma(i-1)\sigma(i)} + c_{\sigma(N)\sigma(1)} \tag{4.1}$$

The TSP has also been modeled as a graph-based problem. Let G be an undirected graph of N vertices with a weight c_{ij} for each arc (i, j). Then, the solution for the TSP corresponds to an HC of minimum total weight [102].

The total weight is the sum of the weights of arcs comprising the HC. We use the term distance in the context of the TSP, but we refer to the weights of an arc when we define the TSP as a graph-based problem.

The TSP is by definition a symmetric problem. That is, the distance between cities i and j is the same as the distances between cities j and i, $c_{ij} = c_{ji}$. For this reason, it is defined in an undirected graph. However, the asymmetric variant of the TSP, where $c_{ij} \neq c_{ji}$, is defined in a directed graph. A directed graph can be converted to an undirected graph; however, this process involves doubling the number of vertices [92].

An important difference between the HCP and TSP is that finding an HC in a given graph might not be trivial. Therefore, traditional search methods that minimize the total weight by generating and evaluating a large number of TSP routes are not necessarily efficient approaches. Moreover, minimizing the total weight of a tour constitutes a simple, linear objective function and it can be argued that much of the difficulty of the TSP is embedded in the HCP [11].

There are numerous applications of the TSP beyond the route planning problem, and they span different areas, such as Mathematics, Computer Science, OR, Genetics, Engineering and Electronics [129]. An efficient solution of the TSP would have an enormous impact in OR, Optimization and Computer Science [11].

There are several approaches to address the TSP, and software implementations are also available, although most of them [6] [21] [142] [113] focus on the symmetric TSP. As mentioned before, converting the asymmetric TSP to the symmetric one involves doubling the number of vertices.

4.1.1.1 Application of the Concorde solver to directed HCP instances

Concorde [5] is a program implemented in ANSI C programming language for the symmetric TSP and some related Graph Theory optimization problems. This solver has been applied to TSP instances with up to 85900 vertices and is the best TSP solver at the moment.

To apply the Concorde TSP solver to directed HCP instances, the adjacency matrices must be transformed. Jäger and Zhang [84] proposed a transformation to convert a directed HCP instance into a symmetric TSP instance using the 2-point reduction [92]. Considering a directed graph G of N vertices, the transformation consists of the following steps.

- Create a copy of the vertex set V, called V', such that $V' = \{i'_1, i'_2, ..., i'_N\}$.
- Define a new undirected graph G', whose vertex set is $V \cup V'$ with symmetric distance function $c' : V \cup V' \rightarrow \{0, 1, 2\}$ for $k, l \in \{1, ..., N\}$, where c' is defined in Equations (4.2-4.3).

$$c'(i_k, i'_l) = \begin{cases} 0, & k = l; \\ 1, & k \neq l \text{ and } (i_k, i'_l) \in A; \\ 2, & k \neq l \text{ and } (i_k, i'_l) \notin A. \end{cases} \tag{4.2}$$

$$c'(i_k, i_l) = 2 \text{ and } c'(i'_k, i'_l) = 2 \tag{4.3}$$

Then, a directed HC exists on G if and only if the optimal tour of G' has total distance N.

4.1.2 Other scenarios with the HCP

Other scenarios apart from the TSP are related to the HCP.

- **Icosian game.** This game was designed by Sir William Rowan Hamilton in 1857. It consists of 20 connected cities, each represented by a hole in a wooden pegboard. The aim is to visit each city exactly once and return to the city of origin. If we formulate the problem as a graph-based problem, the resulting graph is the dodecahedral graph (see Figure 4.1). The mathematically generalized version of the Icosian game is the HCP.

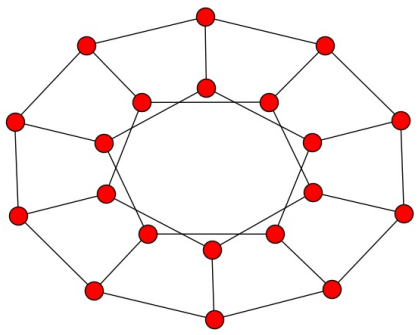


Fig. 4.1: The dodecahedral graph.

- **Knight's tour.** A Knight's tour is a collection of movements made by a knight to visit every square of an $N \times N$ chessboard. The Knight's tour problem is the problem of constructing such a tour. A Knight's graph for an $N \times N$ chessboard is a graph $G = (V, E)$, where $V = \{(i, j) \mid 1 \leq i, j \leq N\}$ and $E = \{(i, j), (o, p) \mid \{|i-o|, |j-p|\} = \{1, 2\}\}$. This graph has N^2 vertices and $4N^2 - 12N + 8$ edges [124].

Mathematically, a closed Knight's tour (a tour that returns to the origin position on the chessboard) is defined as an HC on a Knight's graph. The formal study of the Knight's tour problem started in 1759 with Euler, who considered a standard chessboard of dimension 8×8.

4.2 Approaches to solve the Hamiltonian cycle problem

This section reviews some approaches to solve the HCP. The algorithms can be broadly classified as deterministic and stochastic.

4.2.1 Deterministic algorithms

The following algorithms are deterministic.

- **Snakes and ladders heuristic**. This algorithm for solving HCP on undirected graphs has polynomial complexity. It is a heuristic method, as it is not theoretically guaranteed to find an HC [11].

  All vertices of the graph are located in a circle in some order. This heuristic uses transformations similar to the ones employed by the k-opt algorithms [81] to reorder the connections in the circle. If two adjacent vertices of the circle are connected in the graph, these vertices are connected by arcs in the circle. In contrast, all the other connections of the graph are represented as chords. The arcs of the graph, are classified as *snakes* (arcs of the circle) or *ladders* (chords of the circle), and the algorithm transforms some ladders into snakes and vice versa. If two adjacent vertices on the circle have no snake between them, there is a *gap* between them. An HC is found when there is no gap in the circle. The stopping criterion is the lack of improvement in N^3 iterations.

  Baniasadi et al. [11] carried out a number of experiments in order to measure the performance of the algorithm on many Hamiltonian graphs. The algorithm succeeded in finding HCs in all of them. The maximum number of vertices considered in the experiments was 5000. Recent experiments with the snakes and ladders heuristic [140] has shown that the time and memory can significantly increase for challenging graphs of moderate size.

- **Determinant interior point algorithm**. This algorithm solves a Mathematical Programming problem shown to be equivalent to the HCP [77]. The Mathematical Programming problem is posed as follows:

$$\min_{\boldsymbol{P} \in \mathcal{DS}} \; -\det\left(\boldsymbol{I} - \boldsymbol{P} + \frac{1}{N}\boldsymbol{J}\right)$$

  where $\boldsymbol{P}$ is the probability transition matrix, $\boldsymbol{I}$ is the $N \times N$ identity matrix and $\boldsymbol{J}$ is an $N \times N$ matrix with unit entries. The $\mathcal{DS}$ is the set

of probability transition matrices induced by doubly stochastic policies. A doubly stochastic policy is a stationary policy in which the probability transition matrix induced also has column sums of 1.

It has been proven that the optimization problem is bounded from below by 0 and from above by s, where s is the length of the longest cycle in the graph. For Hamiltonian graphs, $s = N$, and for non-Hamiltonian graphs, $s < N$.

Until now, no algorithm has been designed to solve the posed optimization problem. However, Haythorpe [77] used a logarithmic barrier by adding logarithmic terms to the objective function, thus applying a convex function to the objective function. An interior point method to solve the problem used a special lower-upper decomposition of the matrix.

This approach was tested in Hamiltonian graphs with different numbers of vertices (20, 40, 60, 80-vertices graphs). For each number of vertices, 50 randomly generated graphs were considered. Researchers found the number of unsolved instances and the average run time increased as the number of the vertices increased.

4.2.2 Stochastic algorithms

The following methods feature stochasticity in some of their components.

- **Variance of first hitting times**. This approach is related to a research line that maps discrete problems into a convex domain where continuous analysis can be carried out. This convexification of domains is done by assigning probabilistic interpretation to some elements of the original problem.

  This method uses the embedding of the HCP in a singularly perturbed MDP. The embedding employed in the methods presented in Section 4.2.2 is slightly different, as the HCP is embedded in a discounted MDP. The subgraphs traced out by deterministic policies (including HCs if any) are the extreme points of a convex polyhedron [58]. More precisely, the HCs of a given graph correspond to the global minima of an indefinite quadratic program. This method has never been fully implemented or tested with a set of Hamiltonian graphs.

- **Random walk approach**. The approach is based on the polytope $\boldsymbol{H}_\beta$ defined by Feinberg and introduced in Section 3.3. This polytope contains all the HCs among its extreme points. The random walk approach consists of constructing two smaller polytopes, $\boldsymbol{WH}_\beta$ and $\boldsymbol{WH}_\beta^p$, where $\boldsymbol{WH}_\beta^p \subset \boldsymbol{WH}_\beta \subset \boldsymbol{H}_\beta$. This method is a random walk that pivots on the extreme points of $\boldsymbol{WH}_\beta^p$ and $\boldsymbol{H}_\beta$ [52]. It has been proven that the algorithm detects the Hamiltonicity of the graph in a finite number of iterations.

This approach has been tested with random Hamiltonian undirected graphs of different density. Twelve graphs with 10-200 vertices considered to be dense were used. The approach solved the HCP in one iteration in these graphs. Sparser graphs were also considered in the analysis: precisely ten graphs with 10-80 vertices. The analysis found the number of iterations of the algorithm increasead when the number of vertices increasead. Finally, some non-Hamiltonian graphs have been used to test the method. The executions of the algorithm were terminated when no HC was found.

- **Cross-entropy/optimization hybrid approach**. The approach consists on hybridizing the cross-entropy method [34] and analytical methods based on MDPs [53]. It uses the polyhedral domain proposed by Ejov et al. [51] addressing two deficiencies of it. This improved polytope considers two normalized constraints of $\boldsymbol{X}_\beta$ and includes additional constraints (wedge constraints introduced in Section 3.4.1).

  The cross-entropy method is usually applied to the TSP. As this is a special case of the HCP, it can be also applied to the latter problem. When applied to the TSP, the cross-entropy consists of generating random samples of tours and constructing sequences of transition probability matrices. The entries of these matrices will finally be concentrated on the arcs of the optimal tours.

  The analytical method relies on solving a global optimization problem in the frequency space of MDPs, using a polytope constructed by embedding the HCP in an MDP. The hybrid approach consists of two parts, one involving cross-entropy and the other involving the optimization approach. The cross-entropy component can be used separately or in conjunction with the optimization approach.

  The method was tested using directed Hamiltonian graphs of 6-256 vertices. It was shown that for graphs with a smaller number of vertices (less than 50 vertices), the cross-entropy part of the algorithm was enough to solve the problem. However, for larger graphs, the optimization part of the algorithm was needed to terminate the execution.

- **Wedged-MIP heuristic**. This method uses a mixed (non-linear) integer program to formulate the HCP. This formulation uses the embedding of the HCP in an MDP described in Section 3.3 and the wedge constraints introduced in Section 3.4.1. The IBM ILOG OPL-CPLEX solver was used by the authors to solve the proposed formulation [19]. This solver produces different solutions with different running times, making it a heuristic approach.

  This approach was tested in graphs with different numbers of vertices. The largest graph considered was a 2000-vertex graph. It was compared with two TSP formulations, the modified single commodity flow model and the third stage dependent model. The wedged-MIP heuristic performed much better than the TSP formulations.

4.2.3 Other approximations

Methods such as **genetic theory** or **fractal-like structure** also exploit specific characteristics of cubic graphs giving interesting information about those graphs, but without solving the HCP [121].

In addition, **linear feasibility models** are an indirect method to address HCP, as they identify non-Hamiltonian graphs by infeasibility of suitable constructed linear systems [77].

Table 4.1: Summary of the reported methods to solve the HCP.

Method	Graph	Max. $\|V\|$	Reference	Type
Snakes and ladders heuristic	Undirected	5000	[11]	Determinant
Determinant interior point	Directed	80	[78]	Determinant
Variance of first hitting times	Directed	-	[58]	Stochastic
Random walk approach	Undirected	80	[52]	Stochastic
Cross-entropy/optimization hybrid approach	Directed	256	[53]	Stochastic
Wedged-MIP heuristic	Unspecified	2000	[19]	Stochastic

Table 4.1 presents an overview of the reviewed methods. It should be noted that although some methods are defined using directed graphs, the experimental results are derived from directed graphs that are doubly connected. From a practical point of view, this kind of graphs can be considered undirected. An implementation of the code is not available for any of the methods, to the best of our knowledge. In the case of the snakes and ladders heuristic, a web interface [1] is available to apply the algorithm to a problem.

[1] Available at: `https://sites.flinders.edu.au/flinders-hamiltonian-cycle-project/snakes-and-ladders-heuristic-web-interface/`

5

Purpose and objectives of the research

The main objective of this dissertation is to contribute methodologically to the fields of Mathematics and Computer Science. Specifically, it contributes to a linear programming based method to solve the HCP, a Graph Theory problem whose main applications are in OR. It addresses two main limitations that present linear programming based methods: reduction of the number of constraints of the LPs and fathoming solution spaces that lead infeasible solutions. The dissertation focuses on optimization problems with direct application to the manufacturing industry.

First, it addresses the solution of the optimal tool-path problem in AM. The AM technology was selected because it is one of the technology trends of Industry 4.0. The dissertation poses an optimization problem to build an AI-based DSS that generates the optimal tool-path for AM. With this purpose in mind, it describes the AM tool-path problem and discusses the approaches used in the literature to solve it. Based on the discussion, it introduces a novel problem called sequence strategy generation and solves the problem using a benchmark of parts that have different characteristics.

Second, the dissertation makes methodological contributions to the HCP and the BF. The contribution to optimization problems in manufacturing was an inspiration and starting point for investigating the HCP. Therefore, the methodological contributions with the following specific objectives represent the core of the research:

1. Enhance the efficiency of the BF method.
2. Investigate the role of branching strategies for the BF approach to solve the HCP.
3. Introduce a new method that addresses the limitations of previously reported linear programming based methods.
4. Extend the BF to deal with the MO HCP.

Two main components can be identified in the BF. One is related to the solution of the LPs, where the main role is played by the LP solver. The other

is related to the way branching is implemented and how the recursive calls to the LPs are invoked. The efficiency of this second component is closely related to the choice of the data structure and other design decisions of the algorithm. The first two objectives are related to this second component.

To address the first objective, two enhancements are applied to the BF: an early subcycle detection step and a graph simplification based on vertices of degree two. To address the second objective, a novel branching method is proposed. The BF is based on the principle that the search space or total set of feasible solutions can be partitioned into small subsets of solutions. This operation is called branching; it is done recursively by minimizing the corresponding objective function. The proposed branching method is adaptive within the tree evolution and depends on the subgraph built on each level of the logical tree. The dissertation carries out an exhaustive analysis of the performance of different branching methods in the literature and the proposed branching method.

To address the third objective, the dissertation proposes a BF collapse algorithm that takes into account some of the limitations of previous HCP algorithms based on linear programming. The BF collapse algorithm is built on the BF and incorporates a number of enhancements developed throughout the dissertation.

Apart from single targets, advanced decision-making ecosystems need to address multiple-objective problems. Hence, an MO HCP is defined as the problem of finding HCs that minimize determined multiple criteria. Note, that although the HCP and TSP are closely related, and some investigations deal with an MO variant of the TSP, finding an HC in a graph is not always trivial. For this reason, the dissertation considers that the MO HCP needs a different treatment. Once the problem has been defined, a method to solve the MO HCP is developed to address the fourth objective.

5.1 Research questions

Based on the defined objectives, the following research questions (RQs) were formulated.

- **RQ 1** Is it possible to improve the efficiency (mainly in terms of time) of the BF method? Is it possible to apply it to larger graphs than the ones addressed in the literature?
- **RQ 2** How does the employed branching rule influence the performance of the algorithm? Does any branching method outperform the other methods?
- **RQ 3** Is it possible to extend the BF into an MO scenario? Is it more efficient than an MO GA commonly used to solve the MO TSP?

Table 5.1 shows the chapters in which the RQs are addressed.

Table 5.1: Relationship between RQs and chapters of the dissertation.

RQs	Chapter 7	Chapter 8	Chapter 9	Chapter 10
RQ 1	✓		✓	
RQ 2		✓		
RQ 3				✓

Part II

Contributions to manufacturing optimization

6

Tool-path problem in additive manufacturing

6.1 Introduction

AM or 3D printing is an emerging technology with an array of possibilities to manufacturing technology [139]. Despite its early promise, the technology still has some weaknesses. One is the lack of control over certain defects, such as pores or lack of fusion; another is the repeatability and reproducibility of the part quality [4]. A third major constraint is the amount of time required to fabricate parts [1]. To rectify the latter problem, researchers have proposed the application of better process planning algorithms [44] [46] [48]. In the process planning of CAM, the selection of an appropriate tool-path is critical [101], and, for that reason, the tool-path generation problem is dominating research. In AM, the manufacturing series are shorter than in other technologies employing mass production [62]. AM allows more than one part to be manufactured in the same substrate (batches of workpieces), therefore it is important to generate optimal sequences.

The tool-path required for AM technologies is a predefined trajectory of the nozzles to deposit material in the boundary and interior of each sliced layer [90]. Some AM technologies have several features in common with existing CNC milling machines. The machines are given the instructions in a similar way, the movements are similar, and they both have a rigid tool inside the machine [117]. Accordingly, the approaches used to generate tool-paths in milling can be adapted for AM processes. To this point, tool-path generation in AM has mainly been based on geometric analysis, but this is not usually optimal from a manufacturing engineering point of view [101]. Geometric analysis involves following the boundary of a trend to generate paths without considering an optimality criterion. In actual CAM software, for example, standard tool-paths are generated, such as zig, zigzag or radial, and these are far from optimal [111].

This chapter analyzes the tool-path problem for AM technologies, specifically direct energy deposition (DED) technology. It describes the AM tool-

path problem, and discusses the approaches used in the literature to solve it. Based on the discussion, it introduces a novel problem called sequence strategy generation and solves it using a benchmark of parts that have different characteristics.

6.2 DED process characteristics and analysis of the state-of-the-art

AM or 3D printing consists of depositing material layer-by-layer [30] to create a three dimensional object. The American Society for Testing Materials (ASTM) [128] divides AM processes into seven categories: binder jetting [116], DED [23], material extrusion [141], material jetting [35], powder bed fusion [71], sheet lamination [105] and vat photopolymerization [125]. These processes vary in how the material is deposited (binder, laser, heated), what material is employed (plastics, metals, ceramics) and whether the feedstock state is, solid (powder, wire, sheet) or liquid. This dissertation considers only DED technology. In DED, a nozzle mounted on a multi axis arm, deposits material layer-by-layer. This process can be divided into three main groups: arc welding-based, laser-based and electron-beam based [93]. Wire arc additive manufacturing (WAAM), an arc welding-based technology, employs different types of electrogenic weldings; gas tungsten arc welding (GTAW), gas metal arc welding (GMAW) and plasma arc welding (PAW). The parts in the benchmark used to solve the problem are manufactured by PAW. Fig 6.1 shows the torch and wire of a PAW process.

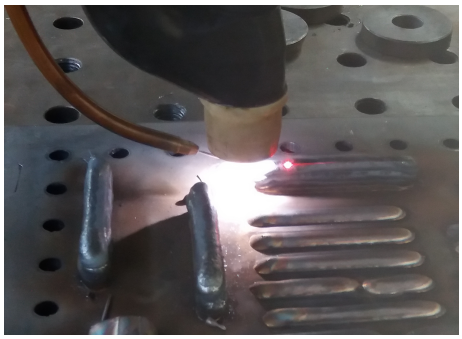


Fig. 6.1: A plasma arc welding process showing the metallic wire and the torch.

6.2.1 Tool-path problem characteristics

This section explains the technical characteristics of the processes defined above. The **total time** plays a major role in WAAM technology, as a long cooling time is needed. The deposition must be performed at a fixed temperature to ensure consistent deposition conditions [138]. The total time is divided into adding time, cooling time and machine movement time. The cooling time can be reduced by applying optimal tool-paths, as the deposition can be carried out in a bead that has already been cooled.

The literature identifies two **precedence constraints** to DED processes [145]. The first is accessibility constraints related to the nozzle. These depend on the process itself and the capabilities of the machine used [138]. The second is heat dissipation. Different WAAM technologies have significant differences in torch movement limitations. For instance, in PAW technology, the torch is more limited in movements than in GMAW or GTAW technology, as the wire is coaxial to the torch. For that reason, the trajectories of the path can be predefined for a technology and machine. The process of finding the predefined trajectories is currently not automated. Figure 6.2 shows the head of the machine and the predefined trajectories, indicated in blue, for a specific part. The machine will follow the traces indicated in blue in the xy-plane to deposit material.

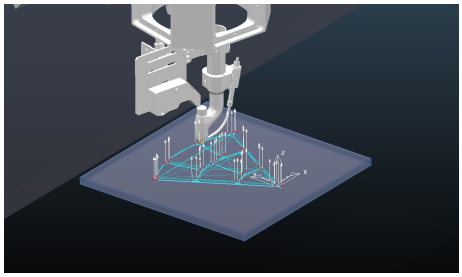


Fig. 6.2: Tool-paths generated for a workpiece following the boundary trend.

Figure 6.3 shows the geometry of a part compounded by three beads in X, Y and Z axis. The lines of the walls represent the layers of the part and the colour indicates the temperature reached by those layers. The temperature was monitored by a pyrometer. The adding sequence is connected with the temperature, because the heat propagates differently depending on the location of the beads. As indicated in Figure 6.3, the temperature in the central bead is lower than in the extremities. Figure 6.4 shows the time required by each bead to cool (until it reaches 400°C). The time to reach a given temperature varies depending on the bead and layer (Figure 6.4).

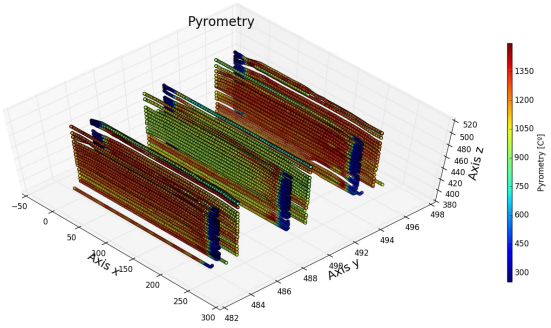


Fig. 6.3: A part manufactured using PAW technology and with the temperature monitored by a pyrometer.

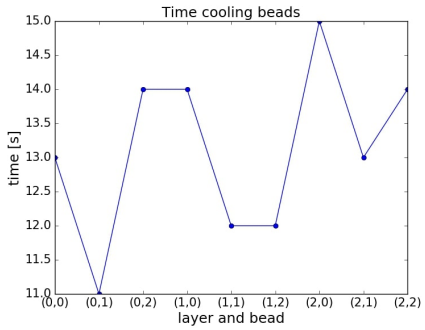


Fig. 6.4: Time to reach 400°C (in this particular example the bead is cooled at 400°C) after performing a deposition. The notation employed (m, n) referres to the bead m and layer n.

For **optimality**, AM requires a better weld bead geometry and surface accuracy, as significant differences, related to the quality of the part, can appear at the start and the end of a weld path [45]. Optimality can be achieved by generating tool-paths that optimize both quality characteristics.

6.2.2 Proposed approaches in the literature

An automated process planning algorithm for AM should take into account the 2D slicing into layers, the bead geometry, tool-path generation and process parameter selection. A slicing algorithm is a method of converting each triangular facet into each respective line segment. The steps of the process planning of all AM technologies are identical, but it is difficult to design an optimal

algorithm to generate the tool-paths for all AM technologies [117]. For that reason, the CAM packages for AM offer only slicing algorithms rather than specialized algorithms that include the bead geometry, tool-path generation and process parameter selection. Another difficulty is that more mature CAM approaches, such as CNC, require an experienced user to make decisions.

Many CAD/CAM packages offer automatic torch-path sequencing for conventional manufacturing, but several constraints in AM are difficult to satisfy using an optimization algorithm; these include the surface quality and the effect of the workpiece heating on the adding sequence. AM's limited capabilities depend, among other things, on the characteristics of AM processes, the current capacity of AM machines and the impact of the AM technology on the material properties [138].

There are some software packages for metal AM, but this is an emerging technology with a wide variety of processes. Most of this software is related to 3D design (CAD) and not to manufacturing (CAM). To be optimal, the software should include such options as changing the process parameters and simulating the piece that will be manufactured. However, a fully automated CAD/CAM software has not been developed for WAAM technology, as there is not yet an automatic way to link the generation of robotic welding paths to the CAD model [43].

One of the AM processes for which the tool-path problem has been studied is fused deposition modeling (FDM). FDM, a popular AM technology, uses a plastic filament as feedstock extruded through a nozzle [89]. In an investigation of the path generation for FDM [91], researches compared FDM and conventional milling. They analyzed the specific features of FDM, identified the three most critical ones and proposed a parallel-based tool-path generation method. In another study Jin et al. [88] proposed a novel tool-path generation method for FDM for thin-wall structures, noting that it is difficult to obtain the desired quality using the commonly employed tool-paths.

The literature has proposed various types of path patterns for AM technologies, including raster, zigzag, contour or spiral [48]. Although these patterns are suitable for powder-based technologies, they have limitations for wire-feed AM technologies, because in these technologies, the deposition width is thicker. In addition, it is important to avoid frequent start/stop points and to avoid changing the deposition path direction as the welding process requires a certain time to stabilize [44].

These path patterns (raster, zigzag, contour and spiral) are based on scan lines [131] and follow the geometrical trend of the boundary [45]. They are not suitable for WAAM because WAAM must meet the following requirements: geometrical accuracy, minimization of the number of tool-path passes and minimization of line segments representing the travel path. Given these requirements, several investigations [46] [44] [47] have used medial axis transformation (MAT) to generate tool-paths. This technique allows the geometry

to be filled from the inside to the boundary (as opposed to the contour path pattern), avoiding the narrow gaps. The extra material is removed in post-process machining.

To conclude, in WAAM technology, the approaches proposed to solve the tool-path problem are geometric-based and do not consider optimality criteria. The techniques do not take the sequence strategy into account; for example, in the MAT approach, the generated paths for each domain go in a counter-clockwise direction [46]. To the best of our knowledge, no research has addressed the other main DED technologies. Moreover, there is a lack of commercial software for AM technology, especially software related to CAM. To fill the gap in the research, we propose a novel problem, sequence strategy generation, in which we consider the previously defined problem characteristics.

6.3 Multicriteria optimization approach to solve the tool-path problem in DED

As previously mentioned, some research on machining and cutting operations has used multicriteria optimization approaches to address the tool-path generation problem. Castelino et al. [24] proposed an algorithm to minimize non-productive time in milling by optimally connecting the segments of the tool-path. This work indicates a possible path for the design of AM strategies; the problem was formulated as a generalized TSP and solved using a heuristic algorithm. Similarly, Chan and Na [25] presented a tool-path algorithm based on simulated annealing; the model improved on the previous TSP model. It included the incorporation of the heat into the cost function, together with the minimization of the tool-path length and the effect of the minimum heat.

Nassehi et al. [120] formulated a tool-path optimization model for a milling process considering three different objective functions: optimization of the cutting time, minimization of the changes in acceleration and constant cutter engagement. Other researches considered the tool-path optimization problem for a drilling process to increase productivity and reduce costs [127]. They reduced the optimization problem to the TSP. Still, other researches modeled the problem of finding the optimum path for a CNC turret typing system using an asymmetric TSP [106]. The aim was to enhance the productivity of the machine by reducing tool changes and optimizing tool routes. A GA, a heuristic optimization approach inspired by natural selection, was used in all of these studies. Together, they suggest that the TSP model is relevant for AM, as some AM technologies have several features in common with CNC machining machines.

A review [40] of tool-path algorithms for laser cutters identified six types of problems: continuous cutting problem, endpoint cutting problem, intermittent

cutting problem, touring polygons problem, TSP and generalized TSP. Most were solved using heuristics and metaheuristics (74%); a few (17%) used exact algorithms and the remainder used approximation algorithms or constraint programming techniques.

Bearing all this in mind, we propose a mathematical framework that models various relevant aspects of DED processes. Using this framework, we formulate a multicriteria optimization problem for DED and solve it for parts manufactured by PAW technology.

6.3.1 Graph representation of DED

A graph is used to represent the part to be manufactured; a graph can express relationships between pairs of variables and show other interesting structures, such as cycles and paths, making it a very useful tool for abstraction. In the following lines, we offer some definitions before introducing the problem.

Definition 4. *A bead S is defined as a set comprising two elements, a vectorial function g that takes a real variable as argument and a layer number l:*

$$S = \{g, l \mid g : [a, b] \subseteq \mathbb{R} \to \mathbb{R}^2, l \in \mathbb{N}^*\} \tag{6.1}$$

where g is the parametrization of a curve C, a continuous line traced on the plane. The initial point $(a, g(a))$ and the final point $(b, g(b))$ are called extreme points.

Definition 5. *An intersection of a bead S_i is a point p, an extreme point of S_i that belongs to another bead S_j for $i, j = \{1, ..., nl\}$ $i \neq j$ where nl is the total number of beads in the part.*

Definition 6. *A segment is a bead in which at least one extreme point is an intersection.*

Figure 6.6 is the graph representation of a part manufactured by PAW, shown in Figure 6.5. The blue lines represent the segments, the red circles represent the intersections and the black arrows are the beads. In this fashion, the part can be represented as a graph with vertices and edges. For instance, for the first layer, the first bead, S_1, is $S_1 = \{g_1, 1 \mid g_1 : [-150, 150] \subseteq \mathbb{R} \to \mathbb{R}^2, 1 \in \mathbb{N}^*\}$. The extreme points of S_1 are $(-150, 250)$ and $(150, 250)$.

Definition 7. *Given a part, a part decomposition graph (PDG) is a planar graph where each vertex is an intersection and an edge represents a segment.*

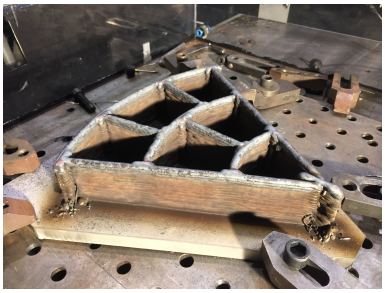


Fig. 6.5: Part manufactured by PAW technology.

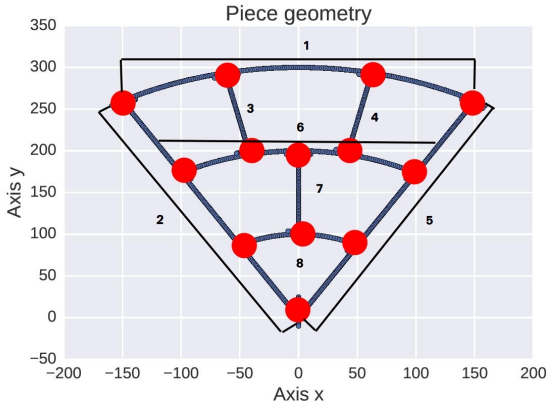


Fig. 6.6: Graph representation of the part where the vertices in red are the intersections and the edges in blue are the segments.

Definition 8. *An adding option, $I_{i,j}$, is a $(i,j)-$walk in the Part decomposition graph (PDG) where i is the origin vertex and j is the terminus vertex.*

Definition 9. *A manufacturing scheme, $\mathcal{M}_S = \{I_{i,j}\ i,j = 1,...,N\}$, is a set of adding options fixed before a workpiece is manufactured.*

Definition 10. *A manufacturing graph, G_m, is a complete graph where the set of nodes is equivalent to the manufacturing scheme $V(G_m) = \mathcal{M}_S$.*

Figure 6.7 shows the PDG of the previously introduced part (Figure 6.5). The manufacturing scheme, $\mathcal{M}_S$, is expressed in Equation (6.2). For each part, these $(i,j)-$ walks are predefined, minimizing the number of the starts and stops of the machine by joining the segments in which the machine can add the material without stopping. Predefining the adding options of the graph

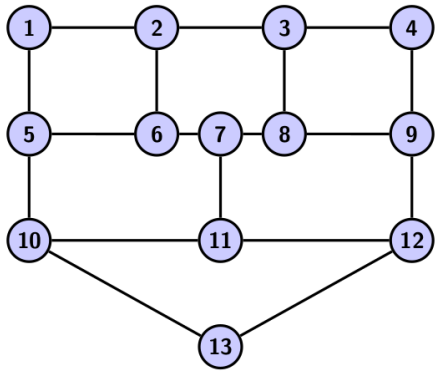


Fig. 6.7: A part decomposition graph of the part shown in Figure 6.5.

in such a way helps to achieve better quality parameters such as weld bead geometry and surface accuracy (see Section 6.2.1).

$$\mathcal{M}_S = \{I_{1,4}, I_{1,13}, I_{2,6}, I_{3,8}, I_{4,13}, I_{5,9}, I_{7,11}, I_{10,12}\} \tag{6.2}$$

Note that a PDG is a graph showing the predefined trajectories (adding options). The set of adding options, $\mathcal{M}_S$, is used to build the G_m. The sequence strategy problem is formulated in the G_m, as shown in Section 6.3.2.

6.3.2 Formulation of a novel problem

This framework allows us to propose a novel scenario related to the tool-path generation for every part manufactured by DED. It takes into account the particular specifications of the tool-path problem described in Section 6.2.1.

Definition 11. *The sequence strategy problem consists of finding a cycle of length $N = |\mathcal{M}_S|$ in a manufacturing graph G_m, which is optimal with respect to one or more predefined criterion.*

The **solution space** of the problem, Ω, is the set of all variable assignments that satisfies the constraints of the problem. In this specific problem, any combination of all the adding options in the manufacturing scheme, or in other words, a permutation of the vertices in the manufacturing graph, G_m is a **feasible solution**. A feasible solution corresponds to the previously mentioned geometric-based approach, as it does not consider the order in which the material is deposited.

$$\Omega = \{(i_1, i_2, ..., i_N) \mid k, l \in \{1, 2, ..., N\} \text{ and } i_k \neq i_l \ \forall \ k \neq l\}$$

as the order in which the deposition is carried out means $|\Omega| = N!$

A **vector of objective functions**, $\boldsymbol{F(x)} = [F_1(\boldsymbol{x}), F_2(\boldsymbol{x}), ..., F_m(\boldsymbol{x})]$ associates m real values with each feasible solution $\boldsymbol{x}$. When $m = 1$, an **optimal solution** to the problem (Definition 11) optimizes the objective function F_1. When $m > 1$, as mentioned in Section 3.1, there is no global optimum solution and the concept of PS is used. Different objective functions can be considered depending on the process characteristics. In this specific case, $\boldsymbol{F(x)} = (F_1(\boldsymbol{x}), F_2(\boldsymbol{x}))$, where F_1 and F_2 are defined as follows.

- Distance (F_1): The distance between two adding options is computed as the Euclidean distance from the final vertex of the first adding option to the initial vertex of the second adding option: $c(I_{i,j}, I_{k,l}) = d_{\text{euclidean}}(j, k)$. The distance between the adding options is traveled by the machine without adding material, and for that reason, the torch has freedom to make movements. This distance was chosen because there is no limitation on the torch's movements, and it represents the shortest distance between two points.
- Waiting time (F_2): The waiting time of two adding options depends on the distance between them. The adding options that are nearer to each other have a longer waiting time (as the temperature has to decrease to a certain value after deposition), while adding options which are further apart have less waiting time. In this study, the waiting time is computed using the temperature monitored by the pyrometer in realistic process conditions. This is made using empirical research based on experience. The machine and monitoring system employed to perform the experiments are detailed in [7].

6.3.3 An evolutionary optimization approach to the formulated problem

Any combination of the elements in $\mathcal{M}_S$ is a feasible solution, but the optimal one(s) can be found using two optimization criteria: distance and waiting time. In this case, the problem is posed as a bi-objective minimization problem, in the solution space of permutations, where the PS of the solutions is computed.

The problem is addressed as a multi-objective TSP. Traditional methods used to solve single-objective TSPs cannot be directly applied to the bi-objective case. Therefore, we use a GA [70] based on the permutation representation. The genetic operators employed in the MO-GA do not violate the restrictions of the multi-objective TSP. Some GA approaches to permutation problems have been tested on large instances (e.g., up to $N = 500$ in [8] [20]). The fact that GA approaches can deal with permutation problems of this large dimensionality enables the possibility of addressing tool-path problems in very complex parts.

Algorithm 2 shows the pseudocode of the algorithm used to solve the problem. The algorithm starts from a set of randomly generated solutions

Algorithm 2 Permutation-based MO-GA

1: $D_0 \leftarrow$ Generate M individuals randomly and evaluate them using the objective functions.
2: $l = 1$
3: **while** stopping criterion not met **do**
4: Select a population D_l^s from D_{l-1} using NSGA-II selection method
5: Create a population D_l applying ordered crossover to individuals in D_l^s with a given probability
6: Apply *shuffle mutation* to individuals in D_l with a given probability
7: Evaluate the individuals in D_l
8: $l \rightarrow l + 1$

and evaluates them using the objective functions. Non-dominated sorting genetic algorithm (NSGA-II) is used as selection algorithm, with the addition of a crowding distance step [37]. This efficient method of selection sorts solutions according to the non-dominated front which they belong to; the first solutions belong to the set of non-dominated solutions. Solutions within each front are also sorted, taking into account the *crowding distance*, a metric that determines how isolated solutions are in the Pareto front. Prioritizing solutions in a less crowded region promotes the spread of the solutions in the Pareto front.

The ordered crossover, a specialized crossover operator that guarantees the offspring will be valid permutations, is applied, and the shuffle mutation operator is applied to the offspring. The latter works by shuffling two positions of the permutation and thus guarantees valid permutations. The overall complexity of Algorithm 2 for a problem of m objectives is $O(g \cdot m \cdot N^2)$, where g is the number of generations and m is the number of objectives. This cost is governed by the selection operator used by the algorithm, as it has complexity $O(m(2N)^2)$ [37]. For the optimization problems addressed in this chapter, we use a population of 500 individuals and 100 generations. The EA is implemented using the DEAP library programmed in Python [59].

6.4 Experiments

The main objective of the experiments is to illustrate the possibility to optimize the tool-paths using the carried out formalization in PAW technology. For that purpose, parts of different geometry and number of beads are used. Note, that the objective is not to find the best optimization method. This section, starts posing the sequence strategy generation problem for the example shown in Section 6.3.1, introduces the benchmark used in the experiments and presents the obtained results. The analysis of the results will be performed analyzing the spread of the obtained PSs and the computation times of the proposed EA approach.

6.4.1 Example of how to pose the optimization problem

The sequence strategy generation problem for the example shown in Section 6.3.1 can be posed as follows. To make the notation of the formulation easier, the adding options are renamed.

- $i_1 = I_{1,4}$
- $i_2 = I_{1,13}$
- $i_3 = I_{2,6}$
- $i_4 = I_{3,8}$
- $i_5 = I_{4,13}$
- $i_6 = I_{5,9}$
- $i_7 = I_{7,11}$
- $i_8 = I_{10,12}$

The objective functions corresponding with the example are presented in Equations (6.3-6.4). The constraints are shown in Equation (6.5).

$$
\begin{aligned}
min \qquad F_1 = {} & 250i_1i_2 + 158.11i_1i_3 + 70.71i_1i_4 + \\
& 0i_1i_5 + 213.6i_1i_6 + 111.8i_1i_7 + \\
& 230.49i_1i_8 + 291.54i_2i_1 + 304.13i_2i_3 + \\
& 304.13i_2i_4 + 269.25i_2i_5 + 201.56i_2i_6 + \\
& 200i_2i_7 + 90.14i_2i_8 + 111.8i_3i_1 + \\
& 111.8i_3i_2 + 141.42i_3i_4 + 158.11i_3i_5 + \\
& 55.9i_3i_6 + 50i_3i_7 + 125i_3i_8 + \\
& 213.6i_4i_1 + 213.6i_4i_2 + 160.08i_4i_3 + \\
& 90.14i_4i_5 + 150i_4i_6 + 55.9i_4i_7 + \\
& 141.42i_4i_8 + 291.55i_5i_1 + 291.55i_5i_2 + \\
& 304.13i_5i_3 + 304.13i_5i_4 + 201.56i_5i_6 + \\
& 200i_5i_7 + 90.14i_5i_8 + 261.01i_6i_1 + \\
& 261.01i_6i_2 + 195.26i_6i_3 + 134.63i_6i_4 + \\
& 75i_6i_5 + 103.08i_6i_7 + 180.28i_6i_8 + \\
& 212.13i_7i_1 + 212.13i_7i_2 + 206.16i_7i_3 + \\
& 206.16i_7i_4 + 180.28i_7i_5 + 292.62i_7i_6 + \\
& 55.9i_7i_8 + 250i_8i_1 + 250i_8i_2 + \\
& 223.61i_8i_3 + 200i_8i_4 + 158.11i_8i_5 + \\
& 167.71i_8i_6 + 111.8i_8i_7
\end{aligned}
\tag{6.3}
$$

$$
\begin{aligned}
min \quad F_2 = {} & 11i_1i_2 + 24i_1i_3 + 49i_1i_4 + 49i_1i_5 + \\
& 11i_1i_6 + 30i_1i_7 + 11i_1i_8 + 6i_2i_1 + \\
& 6i_2i_3 + 6i_2i_4 + 6i_2i_5 + 11i_2i_6 + \\
& 24i_2i_7 + 49i_2i_8 + 30i_3i_1 + 30i_3i_2 + \\
& 30i_3i_4 + 24i_3i_5 + 49i_3i_6 + 49i_3i_7 + \\
& 30i_3i_8 + 11i_4i_1 + 11i_4i_2 + 24i_4i_3 + \\
& 49i_4i_5 + 24i_4i_6 + 49i_4i_7 + 30i_4i_8 + \\
& 6i_5i_1 + 6i_5i_2 + 6i_5i_3 + 6i_5i_4 + \\
& 11i_5i_6 + 24i_5i_7 + 49i_5i_8 + 6i_6i_1 + \\
& 6i_6i_2 + 24i_6i_3 + 30i_6i_4 + 49i_6i_5 + \\
& 30i_6i_7 + 24i_6i_8 + 11i_7i_1 + 11i_7i_2 + \\
& 11i_7i_3 + 11i_7i_4 + 24i_7i_5 + 6i_7i_6 + \\
& 49i_7i_8 + 11i_8i_1 + 11i_8i_2 + 11i_8i_3 + \\
& 24i_8i_4 + 24i_8i_5 + 24i_8i_6 + 30i_8i_7
\end{aligned} \tag{6.4}
$$

$$
\begin{aligned}
\{(i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8) | k, l = \{1, 2..., 8\} \; and \; i_i \neq i_j \\
\forall k \neq l\}
\end{aligned} \tag{6.5}
$$

6.4.2 Benchmark definition

The benchmark employed in these experiments consists of ten parts: the part shown in Section 6.3.1 and other nine parts. Table 6.1 gives the information about the parts, including the manufacturing schemes for each part, the figures of the PDGs (indicated in Figures 6.7 and 6.8) and the number of vertices in each of the manufacturing graphs, ($|V(G_m)|$). Note that $|V(G_m)|$ is also the length of the permutations in the solution space of each of the graphs.

The manufacturing schemes were built to minimize the start and end points of the sections where the machine can add material without stopping, taking into account the movement limitations of a PAW torch. As Figure 6.8 shows, the segments joined by a straight line can be considered adding options. This is noticeable in the graph in Figure 6.8a; the graph has 24 edges, but only eight adding options are in the manufacturing scheme. In contrast, in the graphs shown in Figure 6.8b and Figure 6.8d, all the segments correspond to a different adding option. The objective functions of distance and waiting time are generated for each of the parts as explained in Section 6.3.2.

6.4.3 Results and discussion

In this section, we present the results from the experiments using a benchmark compound by ten parts and described in Section 6.4.2.

The PSs obtained by applying Algorithm 2 to the bi-objective problems defined for each of the parts are shown in Figure 6.9 and Figure 6.10. The

Table 6.1: Description of the benchmark, indicating the figure number of the graph, an associated manufacturing scheme and the number of vertices in the manufacturing graph.

Graph	**Figure**	**Manufacturing scheme**	$\lvert V(G_m)\rvert$
1	Figure 6.7	$\mathcal{M}_S = \{I_{1,4}, I_{1,13}, I_{2,6}, I_{3,8}, I_{4,13}, I_{5,9}, I_{7,11}, I_{10,12}\}$	8
2	Figure 6.8a	$\mathcal{M}_S = \{I_{1,4}, I_{1,13}, I_{2,14}, I_{3,15}, I_{4,16}, I_{5,8}, I_{9,12}, I_{13,16}\}$	8
3	Figure 6.8c	$\mathcal{M}_S = \{I_{1,4}, I_{1,7}, I_{2,5}, I_{3,5}, I_{3,9}, I_{4,6}, I_{4,9}, I_{5,7}, I_{5,8}, I_{6,8}, I_{7,8}\}$	11
4	Figure 6.8e	$\mathcal{M}_S = \{I_{1,4}, I_{1,6}, I_{2,7}, I_{3,11}, I_{4,10}, I_{5,9}, I_{6,7}, I_{6,11}, I_{7,10}, I_{9,12}, I_{10,13}, I_{11,12}, I_{12,13}\}$	13
5	Figure 6.8g	$\mathcal{M}_S = \{I_{1,2}, I_{1,5}, I_{2,4}, I_{4,8}, I_{5,7}, I_{7,8}\}$	6
6	Figure 6.8h	$\mathcal{M}_S = \{I_{1,2}, I_{1,4}, I_{1,6}, I_{2,4}, I_{2,8}, I_{3,5}, I_{4,7}, I_{6,8}\}$	8
7	Figure 6.8b	$\mathcal{M}_S = \{I_{1,2}, I_{1,3}, I_{1,7}, I_{2,4}, I_{2,8}, I_{3,4}, I_{3,5}, I_{4,6}, I_{5,6}, I_{5,7}, I_{6,8}, I_{7,8}\}$	12
8	Figure 6.8d	$\mathcal{M}_S = \{I_{1,2}, I_{1,3}, I_{1,4}, I_{1,7}, I_{2,3}, I_{2,5}, I_{2,8}, I_{3,4}, I_{3,5}, I_{4,6}, I_{4,7}, I_{5,6}, I_{5,8}, I_{6,7}, I_{6,8}, I_{7,8}\}$	16
9	Figure 6.8f	$\mathcal{M}_S = \{I_{1,2}, I_{1,4}, I_{1,5}, I_{2,4}, I_{2,5}, I_{3,6}, I_{4,6}, I_{5,6}\}$	8
10	Figure 6.8i	$\mathcal{M}_S = \{I_{1,4}, I_{1,5}, I_{2,4}, I_{3,6}, I_{4,8}, I_{5,7}, I_{6,7}, I_{6,8}, I_{7,8}\}$	9

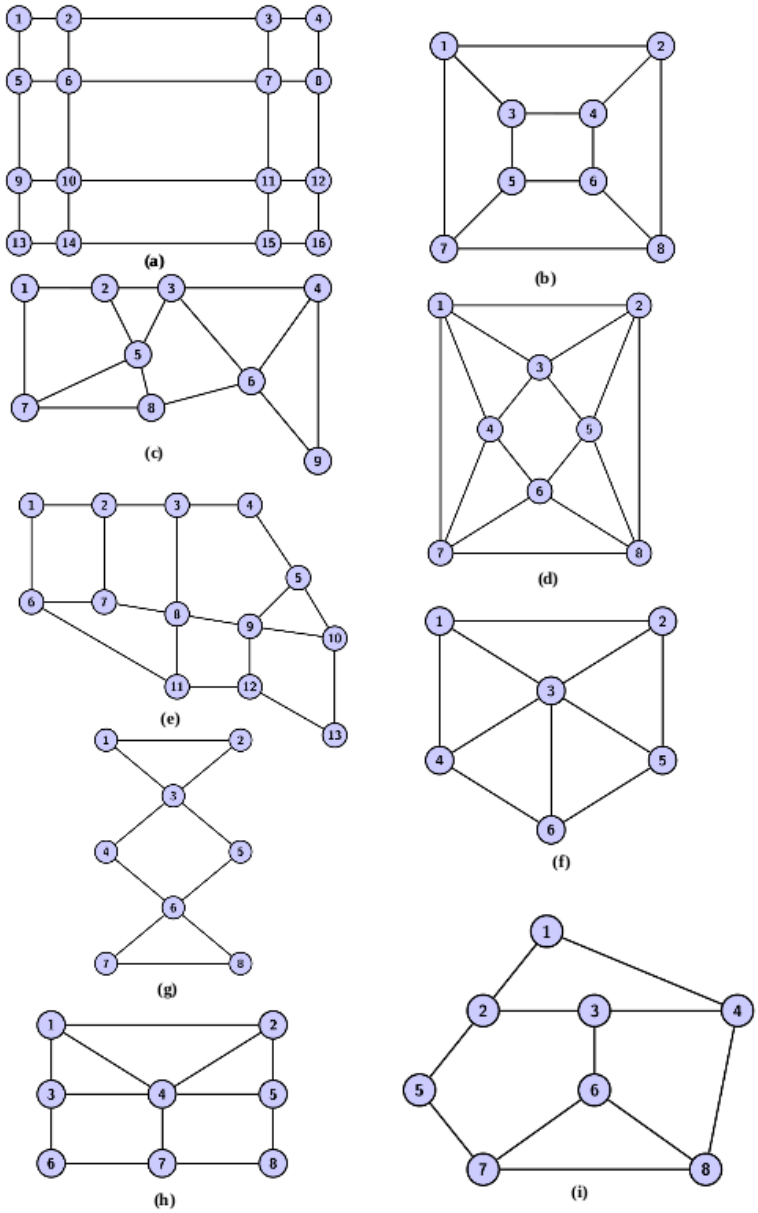


Fig. 6.8: The PDGs of the parts from the benchmark.

Table 6.2: Summary of the information on the PSs, showing the figures related to PSs, execution time in seconds and the minimum and the maximum values related to the two objectives in the PSs.

Graph	Figure	Time [s]	Min (F_1)	Min (F_2)	Max (F_1)	Max(F_2)
1	Fig. 6.9a	105.08	1022.14	99	1883.57	234
2	Fig. 6.9b	105.65	1212.10	48	2175.17	220
3	Fig. 6.9c	130.32	1018.17	318	1591.76	410
4	Fig. 6.9d	151.06	1030.46	265	2540.52	503
5	Fig. 6.9e	83.40	1042.68	140	1106.48	152
6	Fig. 6.10a	103.30	1013.05	260	1130.41	277
7	Fig. 6.10b	136.82	1016.81	378	1727.92	451
8	Fig. 6.10c	168.83	1312.25	405	2601.72	582
9	Fig. 6.10d	100.22	1048.67	217	1354.72	258
10	Fig. 6.10e	109.67	1007.19	269	1348.18	315

legends in the figures indicate the number of individuals in the PSs that reach the same values in the objective functions. For instance, for the PS of the Graph 1, shown in Figure 6.9a, the points in red indicate that those values are reached by only one individual in the PS, whereas the crosses in blue indicate they are reached by two individuals in the PS.

The distribution of the points in the PSs varies significantly from one case to another. The most significant case is the one related to Graph 5 (shown in Figure 6.9e), with some noticeable gaps between the points represented in the PS. In comparison, in Graph 4 (Figure 6.9d), the points cover almost the whole PS. It should be noted that in almost all cases, with the exception of Graph 1, one point is repeated several times (from 4 to 91), indicating that those values are reached by many individuals.

To clarify the results, Table 6.2 links each graph with its corresponding PS figure, indicating the execution time of the MO-GA and minimum and maximum values related to the two objectives in the PSs. In all cases, the execution time is quite similar with a mean value of 119.44 seconds. The minimum execution time is achieved in Graph 5 and the longest execution time in Graph 8. As observed in Table 6.1, the number of vertices in the manufacturing graphs, therefore the length of the permutations in the solution spaces, are the lowest and the highest for the Graphs 5 and 8, respectively. Accordingly, the execution time of the problems is related to the length of the individuals in the solution space. The minimum and maximum values of the two objectives for all the PSs are also indicated in the table.

For illustrative purposes, a solution with the minimum waiting time in the PS for Graph 1 is given. $\boldsymbol{x}_1 = (4, 1, 8, 3, 5, 6, 2)$, where $F_1(\boldsymbol{x_1}) = 99$ and $F_2(\boldsymbol{x_1}) = 1883.57$. The solution translated to the sequence of adding options is shown in Equation (6.6).

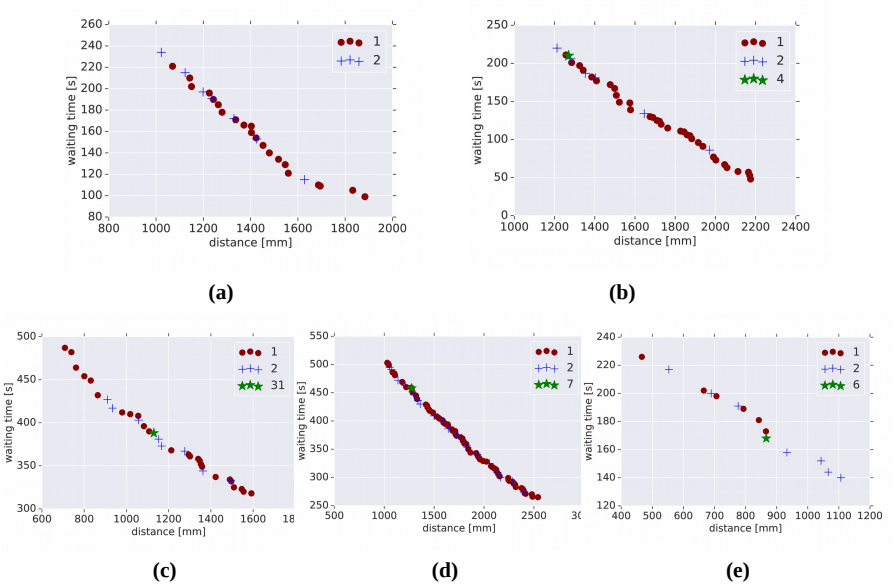


Fig. 6.9: The PSs obtained for the Graphs 1-5 for the objectives of distance and waiting time.

$$I_{3,8} \rightarrow I_{1,4} \rightarrow I_{10,12} \rightarrow I_{2,6} \rightarrow I_{4,13} \rightarrow I_{7,11} \rightarrow I_{5,9} \rightarrow I_{1,13} \tag{6.6}$$

From the performed experiments, it was shown that our proposed formalization serves to optimize the tool-paths. It is possible to compute the PDGs of the parts and finding the corresponding manufacturing schemes, following the criterion of minimizing the starts and stops of the machine in PAW technology. It can be observed in the PSs that a nice spread is obtained in most of the cases with the exception of Figure 6.9e and Figure 6.10a, where more gaps between the Pareto points can be seen.

This analysis suggests it is feasible to compute the optimization before manufacturing a part, as the computation times shown here are affordable. Although the algorithm offers more than one choice for each part, the user can select the most appropriate solution in the PS according to his/her criteria. For example, depending on the material or geometry, one objective function may be more critical than another, and the user could select the solution with minimum value in the preferred criterion. Moreover, the minimum and maximum values of the two objectives indicate the limits of the solutions

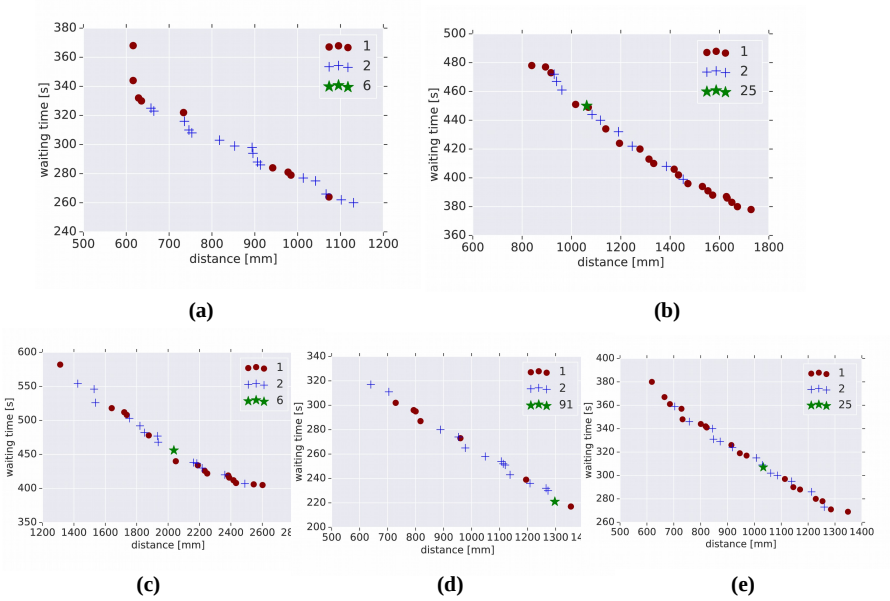


Fig. 6.10: The PSs obtained for the Graphs 6-10 for the objectives of distance and waiting time.

obtained in the PSs. Solutions that are better than the minimum one will not be reached, nor will solutions that are worse than the maximum one.

6.5 Conclusions

This chapter has introduced a sequence strategy generation problem for DED processes that can be used to solve the tool-path optimization problem. Before this, it has proposed a mathematical framework to model a DED process based on a critical review of the previous work. The experiments carried out with ten parts related to the specific technology of PAW, showed that it is possible to address the sequence strategy generation problem with the proposed mathematical framework. It was also found that the problem can be solved using parts with different geometry and number of beads by a MO-GA approach.

On the one hand, in the obtained PSs from the experiments it was observed a nice spread of them. This helps the decision-maker to get a better overview of the available solutions. Regarding to the execution times obtained for the ten parts, affordable execution times were found when applying the MO-GA approach. On the other hand, some limitations of the methodology were also

observed. The construction of the PDGs from CAD files is not automated, and to construct the manufacturing schemes, the need of an expert person in the process is essential.

This study reveals the need to go deeper into the tool-path problem in DED and to extend the proposed framework to specific characteristics of other DED processes. Moreover, the proposed novel problem can be solved using a preference-based EA, where, at each iteration, the decision maker is asked to give preference information in terms of optimality. Finally, the limitations that appeared in this contribution lead to two future research lines. The first limitation is the automated construction of a graph given a CAD file. The second one is related to the construction of the manufacturing scheme from a given PDG based on the specific characteristics of the DED technology.

Part III

Methodological contributions to the Branch-and-Fix method

7

An early subcycle detection step and other enhancements

7.1 Introduction

Most of the algorithms described in the literature to solve the HCP only work for undirected graphs [11], or those considered for directed graphs are not fully implemented or tested for large size graphs [58]. The BF is a linear programming based method to solve the HCP that can be applied to directed graphs. In previous research conducted on the BF [51] [19] [77], the authors did not analyze how different characteristics of a given graph influence on the algorithm. Therefore, the features of a graph, such as the degree of the vertices and its density [72] were not exploited in the BF. This chapter focuses on the second limitation related to linear programming based methods, on fathoming solution spaces [146] that will not lead to HCs.

One of the aspects that was not considered before is the effect of increasing the number of vertices of the graph to be solved by the BF. The authors in [19], only considered graphs up to 64 vertices. Related to the density of a graph, the effect of the third step (Fixing) of the BF has not been previously addressed in the literature. It could be thought that in a denser graph, when an arc is fixed in the second step (Branching), more arcs could be fixed in the third step than in a sparser graph. Another aspect that was not studied before is how the labeling of the vertices of a graph influences on the BF. In other words, it is not clear whether two adjacency matrices that use two different labeling of the vertices of the same graph will obtain different execution times of the BF.

Since the BF was not tested in large graphs, we conducted a number of preliminary experiments in graphs with this characteristic. However, we observed that the method was not scaling well to those graphs. Consequently, after conducting this analysis and to improve the efficiency of the BF, we have introduced two enhancements. It has been shown in the literature that enhancements and modifications to the components of different optimization methods can result in important gains in terms of efficiency, extending the

applicability of the algorithms [17] [32]. This chapter describes an investigation of the BF for large graphs and proposes enhancements to improve the efficiency of the BF. In addition to the two enhancements, we investigate the role of using different adjacency matrix representations depending on the labeling of the vertices.

The first enhancement is an early subcycle detection step that fathoms the branches that lead to subcycles. The presence of subcycles can lead the algorithm to spend a long time before rejecting a branch, a phenomenon that is particularly unfavourable for large graphs. The second one consists of a simplification of the graph by eliminating arcs that are not compatible with other arcs that must be on any HC. In graphs related to real-world problems, a simplification can give on to important improvements, although rarely, can even avoid the use of the BF method. To conduct the experiments, we use graphs with more than 192 vertices, tripling the number of vertices considered in previous investigations. Also, we apply these enhancements to two specific graphs of 400 and 1123 vertices.

The remainder of this chapter is organized as follows. First, Section 7.2 describes the first enhancement of early subcycle detection. Then, Section 7.3 introduces the second enhancement of degree-based simplification. After, Section 7.4 describes the permutation-based BF. Next, Section 7.5 introduces the benchmark of Hamiltonian graphs that will be employed throughout the dissertation, the evaluation metrics in the comparison of the approaches and the experiments carried out to analyze the effect of the proposed modifications. Finally, in Section 7.6, we present the main conclusions of the chapter.

7.2 The impact of subcycles and an algorithm for their early detection

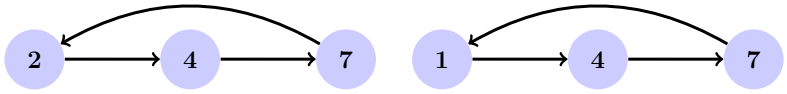
This section analyzes the effect of the subcycles generated in the execution of the BF and introduces a method to address them.

7.2.1 Subcycle generation in the BF

The second LP used by the BF (Step 4) checks whether it is possible to continue exploring the current branch. If the LP is infeasible or the objective function is higher than the right-hand side of Equation (3.6), then there is no HC with the arcs of $\mathcal{U}$. However, subcycles can be generated by adding an arc to the set $\mathcal{U}$ in Step 2.

If the subcycle contains an arc returning to the home node, the second LP is able to detect it; however, if such an arc is not included in the subcycle, the algorithm will continue exploring the branch until the infeasibility is detected later in a deeper level of the tree. Figure 7.1 illustrates those two kind of

subcycles that can be generated in the set $\mathcal{U}$. Figure 7.1a shows a subcycle generated by arcs $[2,4],[4,7]$ and $[7,2]$. In this case, as there is no any arc returning to the home node, the second LP will not detect an infeasibility. However, in Figure 7.1b the subcycle is generated by arcs $[1,4],[4,7]$ and $[7,1]$ (including an arc that is returning to the home node). So, in this latter case, the infeasibility will be detected by the second LP.



(a) Subcycle without an arc returning to the home node

(b) Subcycle that includes the arc returning to the home node

Fig. 7.1: Two different kind of subcycles that can be generated in the set $\mathcal{U}$.

7.2.2 An early subcycle detection step

The phenomenon described in Section 7.2.1 influences in the execution time of the algorithm; therefore, we propose an additional checking step where some branches are fathomed. This step is based on the following theorem and conjecture.

Theorem 1. *Let* $\{(i_1,i_2),(i_2,i_3),...,(i_k,i_1)\} \subset \mathcal{U}$ *be a subcycle of length* k *in the set of fixed arcs, where* $i_1,i_2,...,i_{k-1} \neq 1$ *and* $x_{i_1i_2},x_{i_2i_3},...,x_{i_ki_1}$ *are the entries of the solution vector* $\boldsymbol{x}$ *of the second LP. Then,* $x_{i_1i_2} = x_{i_2i_3} = ... = x_{i_ki_1} = \frac{\mu}{1-\beta}$.

Proof. If we write Equation (3.5), for the defined subset $\mathcal{U}$, taking into account that the arcs are fixed, then

$$\begin{cases} x_{i_1i_2} - \beta x_{i_ki_1} = \mu, \\ x_{i_2i_3} - \beta x_{i_1i_2} = \mu, \\ ... \\ x_{i_ki_1} - \beta x_{i_{k-1}i_k} = \mu. \end{cases} \tag{7.1}$$

We write Equation (3.10), as the arcs are fixed and $i_1,i_3,...,i_{k-1} \neq 1$,

$$\begin{cases} x_{i_1i_2} - \beta x_{i_ki_1} = \mu, \\ x_{i_2i_3} - \beta x_{i_1i_2} = \mu, \\ ... \\ x_{i_ki_1} - \beta x_{i_{k-1}i_k} = \mu. \end{cases} \tag{7.2}$$

These constraints (Equation (7.2)) are redundant, as they are the same as the system in Equation (7.1). Each variable appears in successive constraints with coefficients 1 and $-\beta$. Therefore, the solution of the system must be

$$x_{i_1 i_2} = x_{i_2 i_3} = ... = x_{i_k i_1} = \frac{\mu}{1-\beta}.$$

□

Remark 1. If entries of vector $\boldsymbol{x}$ take value $\frac{\mu}{1-\beta}$, the arcs related to them create a subcycle, but they do not necessarily belong to $\mathcal{U}$. This phenomenon is illustrated in Example 1.

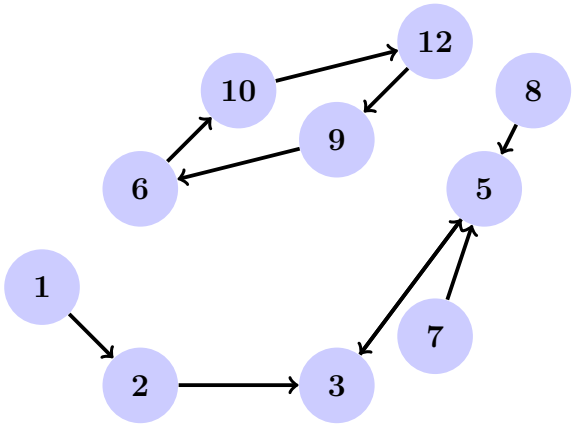


Fig. 7.2: One possible solution of the second LP for a Hamiltonian graph with 12 vertices.

Example 1. Let G be a 12-node Hamiltonian graph. In the second iteration, when the only fixed arc is $(9, 6)$, the second LP returns the solution shown in Figure 7.2. The variables related to the arcs $\{(6, 10), (9, 6), (10, 12), (12, 9)\}$ take value $\frac{\mu}{1-\beta}$, but not all of them belong to the set of fixed arcs.

Conjecture 1. Let $\boldsymbol{x} = (..., x_{i_1 i_2}, x_{i_3 i_4}, ..., x_{i_k i_{k+1}}, ...)$ be the solution of the second LP, where $x_{i_1 i_2} = x_{i_3 i_4} = ... = x_{i_k i_{k+1}} = \frac{\mu}{1-\beta}$ and $\{(i_1, i_2), (i_3, i_4), ..., (i_k, i_{k+1})\} \subset \mathcal{U}$. Then, $\{(i_1, i_2), (i_3, i_4), ..., (i_k, i_{k+1})\}$ is a subcycle of length k in the set $\mathcal{U}$.

We include a function (Algorithm 3) after the second LP to check (when the second LP is feasible) whether there are arcs related to entries that take the value $\frac{\mu}{1-\beta}$. A branch is fathomed if those arcs belong to $\mathcal{U}$.

As this step of the BF is not proved mathematically (it is a conjecture), a number of experiments will be conducted in Section 7.5 to analyze its validity.

Algorithm 3 Inspect-subcycles

```
1: function INSPECT SUBCYCLES(x)
2:     val ← μ/(1−β)
3:     candidates ← {x_ia | x_ia = val and x_ia ∈ x}
4:     arcs_in_loops ← {(i, a) related to x_ia ∈ candidates}
5:     return arcs_in_loops
```

7.3 Exploiting fixed arcs with a degree-based simplification algorithm

In this section, we analyze the effect of fixing arcs in the BF and how some vertices with a specific degree contribute to that fixing. We also propose, a degree-based simplification step to exploit those mentioned effects.

7.3.1 Effect of the fixed arcs in the BF

The Fixing step of the BF aims to fix as many arcs as possible, when in the Branching step, one specific arc is added to the set $\mathcal{U}$. This process is carried out iteratively as described in the Algorithm 1. It is important to fix as many arcs as possible at every stage of the algorithm to simplify the graph, and hence making the problem simpler. The degree of a vertex plays an important role in this Fixing step. It is clear that in an undirected graph, an arc connected to a vertex of degree two, it must be in any HC, hence fixed. This means that only two arcs are connected to the 2-degree vertex, one of them must be the one reaching the vertex and the other arc must be leaving it.

In a previous research work [29], an algorithm for undirected graph simplification was presented. It was shown that an arc for which at least one of the connected vertices has degree 2 must appear in any HC. In order to apply BF to an undirected graph, an auxiliary directed graph has to be constructed where each edge is represented by two arcs with different directions. Therefore, the algorithm introduced in [29] is not of direct application within BF, neither to original directed graphs nor to undirected graphs. For this reason, we introduce a variant of this algorithm to deal with directed graphs. The simplification process is more cumbersome, since the directions of the arcs determine more possible combinations to be treated.

7.3.2 Degree-based simplification algorithm

After analyzing in Section 7.3.1 the role of the fixed arcs in the BF, a degree-based simplification algorithm is proposed here. The pseudocode of the method for the simplification of graphs based on 2-degree vertices is shown in Algorithm 4. The algorithm consists of recursively eliminating arcs and thus,

Algorithm 4 Degree-based simplification

```
1: function Degree Simplification(G)
2:     eliminated←True
3:     feasible←True
4:     while eliminated=True do
5:         eliminated ← False
6:         for i ∈ V(G) do
7:             O(i) ← Outgoing(i)
8:             I(i) ← Ingoing(i)
9:             O2(i) ← 2-Degree(O(i))
10:            I2(i) ← 2-Degree(I(i))
11:            if |O2(i)| > 2 or |I2(i)| > 2 then
12:                eliminated←False
13:                feasible←False
14:                return feasible, G
15:            if |O2(i)| = 2 then
16:                arcs_leave ← Elim(O(i), O2(i))
17:                G ← Update_Graph(G,arcs_leave)
18:                eliminated ← True
19:            if |I2(i)| = 2 then
20:                arcs_enter ← Elim(I(i), I2(i))
21:                G ← Update_Graph(G,arcs_enter)
22:                eliminated ← True
       return feasible, G
```

updating the graph. This degree-based simplification is performed in Step 4 of the BF algorithm before solving the second LP. In the following lines, its main steps are summarized:

- For every vertex, the sets of outgoing $\mathcal{O}(i)$ and ingoing $\mathcal{I}(i)$ vertices are computed. Their 2-degree vertices are identified, $\mathcal{O}_2(i)$ and $\mathcal{I}_2(i)$, respectively (lines 6-10 of the pseudocode).

  This is explained in a deeper extend in Definition 12 and Figure 7.3. Definition 12 is summarized in Table 7.1, where for each condition an illustrative example (Figure 7.3) is indicated for the four conditions involving 2-degree vertices.

- If there are more than two vertices in $\mathcal{O}_2(i)$ or $\mathcal{I}_2(i)$ with 2-degree, it is not possible to find an HC in the current graph. The algorithm would terminate and return that it is not feasible. In this case, the BF would fathom the current branch (lines 11-14 of the pseudocode).

- If there are two vertices in $\mathcal{O}_2(i)$, j and m, with 2-degree, then all arcs that leave i are eliminated, with the exception of arcs (i, j) and (i, m). The graph is updated removing those arcs. A similar process is carried out when there are two vertices in $\mathcal{I}_2(i)$ with 2-degree (lines 15-22).

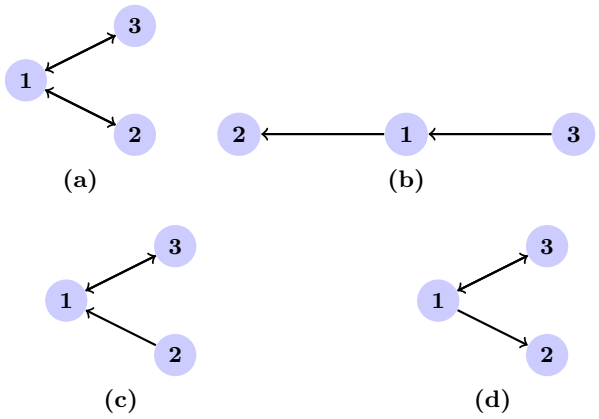


Fig. 7.3: Illustrative example of different casuistries for degree-2 vertices in directed graphs.

Definition 12. *A vertex i is considered a 2-degree vertex if one of the following conditions are fulfilled:*

1. $d_G^+(i) = 2$ *and* $d_G^-(i) = 2$
2. $d_G^+(i) = 1$ *and* $d_G^-(i) = 1$
3. $d_G^+(i) = 2$ *and* $d_G^-(i) = 1$
4. $d_G^+(i) = 1$ *and* $d_G^-(i) = 2$

Table 7.1: Combinations of in-degree and out-degree of a vertex that define a 2-degree vertex. A figure that illustrates each casuistry is also indicated.

In-degree	Out-degree	Example
2	2	Figure 7.3a
1	1	Figure 7.3b
2	1	Figure 7.3c
1	2	Figure 7.3d

The worst computational complexity of Algorithm 4 is $O(N^3)$. The external *while* loop is repeated N times maximum, as it is not possible to fix more than N arcs belonging to the HC. The internal *for* loop is repeated N times as we have N vertices. In the worst case, the complexity of the `Update_Graph` function is $O(N)$ when only one arc is eliminated. In the average case, the computational cost of Algorithm 4 is much less since within each internal loop several arcs can be removed. The effect of the simplification in the quantity of removed arcs will be analyzed in depth in Section 7.5.3.

The pseudocode of the BF algorithm with the carried out modifications (subcycle detection and degree-based simplification) is shown in Algorithm 5. It takes as an input an adjacency matrix and an empty set of fixed arcs ($\emptyset$). It returns value *True* of the boolean variable `found` and the HC, or value *False* of the boolean variable `found` and $\emptyset$, when all the branches have been explored without finding an HC.

The main functionalities of the algorithm were explained in Section 3.4. The `candidate_vertices` are calculated using the Algorithm 3 created by a result of Conjecture 1. This algorithm and the following ones related to the BF presented in the dissertation were implemented in Python programming language (version 3.5.2) and CPLEX (version 12.7.1) was used to solve the LPs.

7.4 The role of the labeling of the vertices and a permutation-based BF

We observed that different representations of the graph (different orderings of the labels of the vertices) could produce different outputs of the original BF (different computational times or even not succeeding to find an HC in a given period of time).

In the construction of the HCs, candidate arcs are selected according to the order of the labels of the vertices; therefore, the BF is sensitive to the way the vertices of the graph are labeled. Different label assignments to the vertices will generate different results. We create a variant of the BF, a permutation-based BF, that receives as an input the adjacency matrix and a permutation of length $|V| = N$.

This permutation is used to sort the rows and the columns of the original adjacency matrix in the order indicated by the permutation and the BF is applied on the rearranged matrix. This permutation-based BF is used to implement it as a parallel approach. First, t random permutations are generated; then, each of them is used as an input for a BF instantiation. The t instances of the BF algorithm start at the same time and run in parallel, hence the output of each individual algorithm can contribute a solution. This permutation-based BF is illustrated in Figure 7.4.

7.5 Experiments

The main goal of our experiments is to evaluate how the enhancements made to the BF algorithm impact its performance and to analyze the effect of using different adjacency matrix representations in the BF. To this end, we use a benchmark of Hamiltonian graphs of different complexity. This section

Algorithm 5 Enhanced Branch-and-Fix

Input: adjacency_matrix, $\emptyset$
Output: True/False, HC/$\emptyset$

function ENHANCED_BRANCH_AND_FIX(adjacency_matrix, $\mathcal{U}$)
  simp_matrix, status ← Degree_Simplification(adjacency_matrix)
  if status ≠ feasible **then**
    return False, $\emptyset$
  status, $\boldsymbol{x_2}$, $F(\boldsymbol{x_2})$ ← Second_LP(simp_matrix, $\mathcal{U}$)
  if status ≠ feasible or $F(\boldsymbol{x_2})$> bound **then**
    return False, $\emptyset$
  candidate_vertices ← Inspect_Subcycles($\boldsymbol{x_2}$)
  if |candidate_vertices|> 0 **then**
    if candidate_vertices $\subset \mathcal{U}$ **then**
      return False, $\emptyset$
  status, $\boldsymbol{x}$, $F(\boldsymbol{x})$ ← First_LP(simp_matrix, $\mathcal{U}$)
  if status ≠ feasible **then**
    return False, $\emptyset$
  splitting_node ← Identify_Splitting_Node($\boldsymbol{x}$)
  if splitting node = False **then**
    HC ← Identify_HC($\boldsymbol{x}$)
    return True,HC
  else
    $d \leftarrow |\mathcal{A}$(splitting_node)|
    found ←False
    $i \leftarrow 0$
    while (found = False) and $(i < d)$ **do**
      fixed_arc ← Get_ith_Outgoing_Arc(splitting_node, i)
      new_adj_matrix ← Update_Adjacency_Matrix(fixed_arc)
      $\mathcal{U}$ ← Update_Fixed_Arcs(new_adj_matrix)
      last_fixed_arcs ← Refined_Fixed_Arcs($\mathcal{U}$)
      while |last_fixed_arcs|> 0 **do**
        for arc ∈ last_fixed_arcs **do**
          new_adj_matrix ←Update_Adjacency_Matrix(arc)
        $\mathcal{U}$ ← Update_Fixed_Arcs(new_adj_matrix)
        last_fixed_arcs ← Refined_Fixed_Arcs($\mathcal{U}$)
      found, HC ← ENHANCED_BRANCH_AND_FIX(new_adj_matrix, $\mathcal{U}$)
      $i \leftarrow i + 1$
    if found=True **then**
      return True, HC
    else
      return False, $\emptyset$

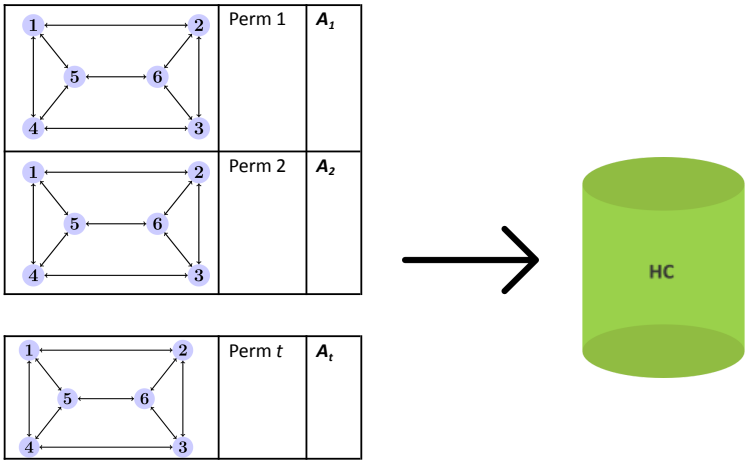


Fig. 7.4: Permutation-based BF, where t permutations are used to obtain t adjacency matrices of the same graph and contribute to the HCP.

presents the benchmark used in the experiments, the metrics employed for the evaluation of the experiments and compares the algorithm's performance in those graphs with and without applying the proposed enhancements. Finally, an analysis with the permutation-based BF is presented.

7.5.1 Benchmark definition

Two main benchmarks of Hamiltonian graphs are used throughout the dissertation. The first set comprises 500 random Hamiltonian graphs. To study the behavior of the methods on a diverse set of graphs, we create a benchmark of instances by generating graphs defined by two parameters, the number of vertices, $|V|$ and the graph density, D. We have selected the parameter $|V|$ because in NP-complete problems, the size of the problem is a critical parameter. The density was selected as the sparsity of a graph plays an important role when solving the HCP [60]. The values for both parameters are presented below. Twenty five graphs are created for all possible combinations of both parameters.

- $|V|$: 60, 70, 80, 90 and 100.
- D: 0.15, 0.25, 0.44 and 0.64.

First, to generate the adjacency matrices, a random permutation of length N is created, this is because an HC can be represented by a permutation. Secondly, a random symmetric matrix of 0s and 1s of dimension $N \times N$ is created. This matrix represents the adjacency matrix of a graph. We use

probabilities p and $1-p$ to distribute 1s and 0s in the matrix respectively. In this case, $D = d \times 0.5$. Finally, the permutation (by definition it is a cycle) is used to connect the arcs that are part of it in the adjacency matrix. In this way, we can guarantee the existence of at least one HC. Algorithm 6 shows the pseudocode describing the steps in the generation of the adjacency matrices.

Algorithm 6 Random Adjacency Matrix

Input: N, p
Output: $\boldsymbol{A}$

function RANDOM_ADJACENCY_MATRIX(N, p)
  $\sigma \leftarrow$ Random_perm(N) ▷ Random permutation of length N
  $M \leftarrow$ Random_Matrix($N, p, 1-p$) ▷ Random matrix $N \times N$
  for row $\in \{1, ..., N\}$ **do**
    for column $\in \{1, ..., N\}$ **do**
      $\boldsymbol{M}[row, column] = \boldsymbol{M}[column, row]$ ▷ Symmetric matrix
  for $i \in \{1, ..., N\}$ **do** ▷ At least one HC using the permutation
    $\boldsymbol{M}[i, \sigma[i]] = 1$
    $\boldsymbol{M}[\sigma[i], i] = 1$ ▷ Symmetric matrix
  $\boldsymbol{A} \leftarrow$ Fill_Diagonal_Zeros($\boldsymbol{M}$)
return $\boldsymbol{A}$

The second set of Hamiltonian graphs includes graphs that belong to a collection called the *challenge set*. These Hamiltonian graphs are designed to be difficult to solve using standard HCP heuristics [79]. There are several sets of HCP instances available in the literature. However, none of the instances are designed to be difficult to solve with algorithms such as Concorde [6], Lin-Kernighan heuristic [80] and snakes and ladders heuristic [11]. This set includes four type of instances: 1) graphs from the literature; 2) modified graphs from the literature; 3) graphs obtained by converting from other discrete problems; 4) combinations of the graphs in 1), 2) and 3).

The main characteristics of the first 30 graphs are summarized in Table 7.2. As can be observed in Table 7.2, the first 30 graphs have between 66 and 234 vertices and 198 to 702 arcs. More vertices in the graph implies more arcs. The densities of the graphs are also indicated; D slightly decreases when the number of vertices increases. The values of D are lower than those considered for random graphs. In this particular chapter, Graph 59 and Graph 188 are also used, which both present $D = 0.5$. Graph 59 has 400 vertices and 80002 arcs and Graph 188 has 1123 vertices and 630566 arcs.

7.5.2 Metrics for the comparison of the approaches

In this section, we introduce the metrics employed in the chapter to evaluate the algorithms. First, the number of unsolved instances (UIs) is used to measure the ability of a method to solve the instances. An instance is considered

Table 7.2: Characteristics of the first 30 graphs of the *challenge set*, indicating the number of vertices and arcs and the density.

G	$\|V\|$	$\|A\|$	D
1	66	198	0.09
2	70	212	0.09
3	78	234	0.08
4	84	254	0.07
5	90	270	0.07
6	94	284	0.06
7	102	306	0.06
8	108	326	0.06
9	114	342	0.05
10	118	356	0.05
11	126	378	0.05
12	132	398	0.05
13	138	414	0.04
14	142	428	0.04
15	150	450	0.04
16	156	470	0.04
17	162	486	0.04
18	166	500	0.04
19	170	780	0.05
20	174	522	0.03
21	180	542	0.03
22	186	558	0.03
23	190	572	0.03
24	198	594	0.03
25	204	614	0.03
26	210	640	0.03
27	214	644	0.03
28	222	666	0.03
29	228	686	0.03
30	234	702	0.03

unsolved if the approach is not able to find an HC in a determined period of time. Second, the number of *calls* to the algorithms are compared to analyze which of them is more efficient. The approaches employed here are recursive algorithms, and the number of recursive calls is the metric we compute for the comparison. The algorithm's efficiency is related to the number of calls; the algorithm is more efficient when there are fewer calls. From both metrics, the first one is more critical, and the number of calls has a secondary importance.

7.5.3 Evaluation of Branch-and-Fix enhancements

In this section, we carry out the experiments to evaluate the performance of the two proposed enhancements. Three variants of the BF algorithm are executed:

1. Original BF
2. Original BF with the subcycle detection step (BF_SD)
3. Original BF with both the subcycle detection step and the graph simplification step (BF_SD_GS).

In the first part of the experiments, we compare the three approaches in two sets of Hamiltonian graphs. In the second part, we carry out additional experiments to analyze the effect of the degree-based simplification on the BF. Each experiment is executed for 24 hours.

Regarding the first part of the experiments, Figure 7.5 shows the distributions of the number of calls performed by each algorithm. The comparison is performed by differentiating the graphs by density; the original BF is compared to BF_SD and BF_SD_GS. Higher numbers of calls are shown for the original BF than for BF_SD, this is more noticeable for $D = 0.44$. It should be noted that when $D = 0.64$, the number of calls decreases significantly for both. Similar distributions are observed for BF_SD and BF_SD_GS, in all the

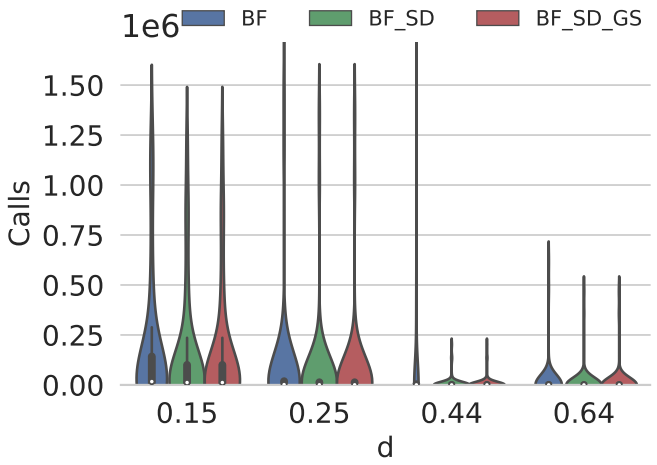


Fig. 7.5: Number of calls required by different enhancements to the BF for random graphs of different D.

cases. When $D = 0.44$ and $D = 0.64$, the number of calls decreases significantly for both. The difference in the number of calls is lower if we compare BF_SD and BF_SD_GS than for the original BF and BF_SD. This indicates that as expected, when the density of the graphs increase it is easier to the BF finding an HC.

Table 7.3 shows the number of UIs and the mean value (μ) and standard deviation (σ) of the calls for the instances that were solved by the three variants. They are compared in graphs of a different number of vertices and densities, with the lowest value of each of the performance parameters considered indicated in bold. The lowest number of UIs is achieved by BF_SD_GS. The number of UIs decreases significantly from the original BF to BF_SD and, to a lesser extent, from BF_SD to BF_SD_GS. The number of UIs decreases when the density increases. The lowest mean values of the number of calls are achieved with BF_SD, with the exception of $D = 0.64$.

To determine whether there are significant differences between the approaches in the number of UIs and the number of calls, we apply a statistical test, the Friedman test, which is a non-parametric test. This test can be applied to paired samples, making it suitable here, as the instances are the same. It tests whether repeated measurements of the same observation have the same distribution. Thus, the null hypothesis is that the median difference between pairs of observations is zero. To measure the differences between the number of UIs, the solved instances are represented by 1 and those that are unsolved by 0. To measure the differences between the number of calls, the instances that have all the methods in common are considered. The signifi-

Table 7.3: Comparison of original BF, BF_SD and BF_SD_GS, showing the number of UIs and the mean and standard values of the algorithm's calls to the common solved instances by the three variants.

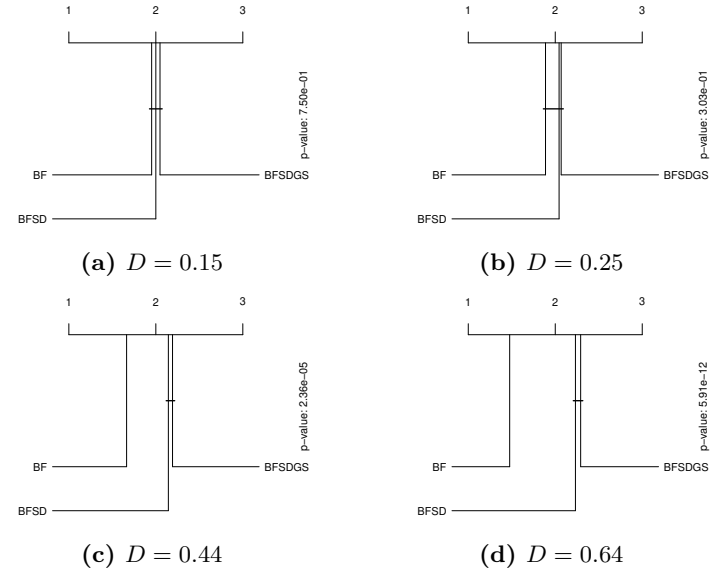
D	BF variants	UI	μ	σ
	Original BF	89	178297.83	342316.93
0.15	BF_SD	85	**150872.46**	**299329.45**
	BF_SD_GS	**81**	150894.91	299375.96
	Original BF	82	90421.41	287970.08
0.25	BF_SD	69	**75592.80**	**261357.22**
	BF_SD_GS	**67**	75599.93	261378.37
	Original BF	57	79506.02	260109.99
0.44	BF_SD	17	**8776.53**	**33905.13**
	BF_SD_GS	**13**	8776.67	33905.35
	Original BF	73	23306.59	107704.87
0.64	BF_SD	10	16368.60	80900.22
	BF_SD_GS	**5**	**16368.56**	**80900.22**

cance threshold is $p = 0.05$. A correction function is applied to the p-values for multiple testing, in this case Shaffer's procedure [61].

The Friedman test is applied to the random Hamiltonian graphs with different densities. Figure 7.6 shows the critical difference diagrams after applying the Friedman test to the distributions that represent solved and UIs in graphs of different densities. It can be observed that significant differences were achieved for $D = 0.44$ and $D = 0.64$ with p-value$=2.36e-05$ and p-value$=5.91e-12$, respectively. In both cases, the highest number of UIs were for the BF continued by BF_SD and BF_SD_GS. The horizontal bold line joining two or more methods represent groups of methods that are not significantly different, in this case BF_SD and BF_SD_GS.

Figure 7.7 shows the critical difference diagrams after applying the Friedman test to the number of calls in graphs of different densities. It can be observed that significant differences were achieved for all the densities, where the highest number of calls was for the BF in all the cases. Also in this case, significant differences were not found between BF_SD and BF_SD_GS (they are joined by a bold line).

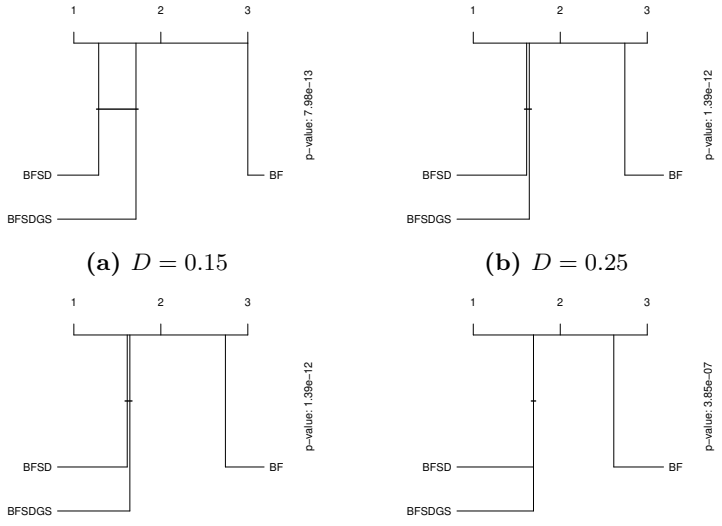
Table 7.4 shows the UIs, μ and σ for the three algorithms in graphs from the *challenge set* of 66-234 vertices. In this case, the permutation-based BF was used (also for the variants) with $t = 30$ (t is the number of permutations employed in the permutation-based BF). From the 30 computed graphs, there were 22 graphs that were solved at least once. For nine graphs, Graphs 7, 13, 17, 20, 22, 24, 27, 28 and 30, the results were identical for the three variants. Therefore, Table 7.4 shows the information for the remaining 13 graphs with the lowest values indicated in bold.



(a) $D = 0.15$ **(b)** $D = 0.25$

(c) $D = 0.44$ **(d)** $D = 0.64$

Fig. 7.6: Critical difference diagram of the Friedman test for the UIs with the graphs with different D.



(a) $D = 0.15$ **(b)** $D = 0.25$

(c) $D = 0.44$ **(d)** $D = 0.64$

Fig. 7.7: Critical difference diagram of the Friedman test for the number of calls with the graphs with different D.

Table 7.4: Comparison of the original BF, BF_SD and BF_SD_GS showing the number of calls on some graphs of the *challenge set* with 66-234 vertices.

Graph	BF variants	UI	μ	σ
	Original BF	**0**	247856.70	221704.57
1	BF_SD	**0**	**115160.60**	**108984.01**
	BF_SD_GS	**0**	115160.83	108984.87
	Original BF	**0**	303654.77	383504.28
2	BF_SD	**0**	131587.57	162707.84
	BF_SD_GS	**0**	**131586.40**	**162704.88**
	Original BF	16	1167130.36	961860.93
3	BF_SD	**4**	924064.00	688305.75
	BF_SD_GS	**4**	**885305.11**	**672574.57**
	Original BF	24	1111592.33	520706.40
4	BF_SD	24	745149.83	349902.09
	BF_SD_GS	23	894914.42	489405.62
	Original BF	27	239861.67	316787.71
5	BF_SD	**21**	652495.56	385162.23
	BF_SD_GS	**21**	**652485.33**	**385158.92**
	Original BF	29	81706.00	0
6	BF_SD	**25**	**828703.00**	**428252.19**
	BF_SD_GS	**25**	828707.60	428256.93
	Original BF	**25**	16410.40	32702.84
9	BF_SD	**25**	**16330.20**	**32542.44**
	BF_SD_GS	**25**	**16330.20**	**32542.44**
	Original BF	29	**55**	**0**
10	BF_SD	28	311661.50	311606.50
	BF_SD_GS	**27**	632900.33	520699.66
	Original BF	**27**	7658.00	10827.22
11	BF_SD	**27**	**7129.00**	**10079.10**
	BF_SD_GS	**27**	**7129.00**	**10079.10**
	Original BF	**28**	6337.50	4825.50
14	BF_SD	**28**	**5416.50**	**3904.50**
	BF_SD_GS	**28**	**5416.50**	**3904.50**
	Original BF	**28**	1436.50	1434.50
15	BF_SD	**28**	**1434.50**	**1432.50**
	BF_SD_GS	**28**	**1434.50**	**1432.50**
	Original BF	**26**	4980.75	3163.07
18	BF_SD	**26**	**3951.75**	**2068.12**
	BF_SD_GS	**26**	**3951.75**	**2068.12**
	Original BF	**26**	8690.75	4659.57
23	BF_SD	**26**	**7620.75**	**3397.02**
	BF_SD_GS	**26**	**7620.75**	**3397.02**

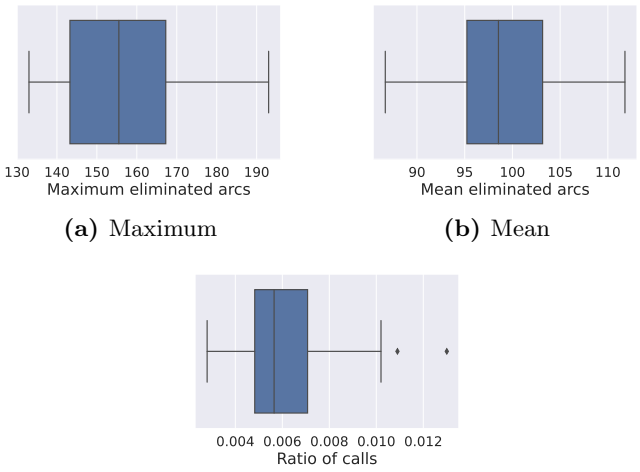
It can be observed that for four graphs, the number of UIs is higher for the BF than for the other two variants (Graph 3, 5, 6 and 10). The number of calls to the BF is higher than for the BF_SD and BF_SD_GS for the same number of UIs (Graphs 1, 2, 4, 9, 11, 14, 15, 18 and 23). Regarding the BF_SD compared to the BF_SD_GS, very similar results are found in terms of UIs or μ (Graphs 1-6 and Graph 10). This is expected, as when the density decreases the number of arcs also decreases, hence arc removal might not be necessary. Identical results are obtained for the BF_SD compared to BF_SD_GS (Graphs 9, 11, 14, 15, 18 and 23).

The three variants were also tested in two specific graphs (Graph 59 and 188) of the *challenge set*. These graphs were selected due to their high density ($D = 0.5$) and large number of vertices. On the one hand, Graph 59 has 400 vertices and 80002 arcs and it was solved by the BF_SD_GS in 2657.32 seconds for all the $t = 30$ permutations. The degree-based simplification function removed iteratively 79202 arcs in a single call. On the other hand, Graph 188 has 1123 vertices and 630566 arcs and it was solved by the BF_SD_GS in 330504.56 seconds for all the $t = 30$ permutations. The degree-based simplification function removed iteratively 628,320 arcs in a single call. The BF and BF_SD were not able to solve these graphs with none of the $t = 30$ permutations.

In the second part of the experiments, we investigate the simplification process from a different perspective. The aim of these experiments is to analyze how the degree-based simplification contributes to eliminate arcs from the graph. For this purpose, we employ Graph 2 of the *challenge set* and permutation-based BF with $t = 30$ and variant BF_SD_GS.

First, the metrics of the number of eliminated arcs are analyzed. Figure 7.8 shows three metrics related to the degree-based simplification method for the 30 permutations. The first consideration is the number of calls to the simplification function where at least one arc is eliminated. For each execution related to a permutation, the maximum and mean numbers of eliminated arcs are shown in Figures 7.8a and 7.8b, respectively. Figure 7.8c shows the ratio of the calls where at least one arc is eliminated out of the total number of calls to the simplification function. The figure indicates that the maximum values (Figure 7.8a) are much higher than the mean values (Figure 7.8b). In general, the number of eliminated arcs is quite low, but, for some calls, the number of eliminated arcs increases considerably. For some executions, the maximum number of eliminated arcs is much lower than for others. The ratio of the number of calls where at least one arc is eliminated is considerably low (mean value of 0.06). This means that the arcs are eliminated in few calls to the algorithm (taking into account the total amount of calls), but when the arcs are eliminated the number of eliminated arcs is considerably high.

Second, the number of arcs eliminated using the degree-based simplification method is analyzed taking into account the levels in the logical tree. This



(a) Maximum **(b)** Mean

(c) Ratio

Fig. 7.8: Metrics related to the degree-based simplification method for the permutations employed.

provides us with a more detailed view of the impact of the simplification and the nature of the BF algorithm. As the same level of a tree can be revisited more than once, the numbers of eliminated arcs are added per level. The deepest level explored among all the permutations used by BF_SD_GS is 50. As not all the executions reach this level, the number of eliminated arcs is indicated by $\alpha = -10000$. Figure 7.9 shows the heatmap of the number of arcs eliminated by the degree-based simplification method for each level of the tree. The black boxes of the heatmap indicate that those levels are not reached by the specific execution of the BF_SD_GS. As can be observed, the highest number of eliminated arcs is achieved in the central levels of the search tree. This is more noticeable for permutation 13. This effect of a higher number of eliminated arcs in the central levels is related to the fact that those levels are more often revisited.

7.5.4 Effect of the labeling of the vertices

In this section, some insights into the effect of the labeling of the vertices is given. In the previous section (Section 7.5.3), we have already seen that there is a difference when using different representations of the adjacency matrix when solving a graph. If we observe Table 7.4, there is a difference in the number of UIs and also in the number of calls to the BF. Moreover, in the second part of the experiments, it was seen that the simplification has a different effect when using a different matrix representation by the BF_SD_GS.

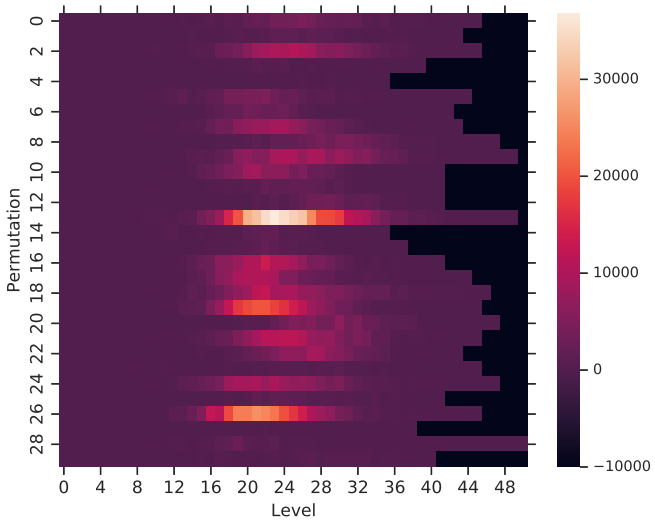


Fig. 7.9: Heatmap of the number of eliminated arcs using the degree-based simplification method for the levels in the logical tree.

For some permutations employed by the algorithm, the number of removed arcs was much higher than for other permutations.

Figure 7.10 shows the probability density function of the execution time by the permutation-based BF. The BF_SD_GS algorithm was used in this case. The Graph 2 of the *challenge set* and 30 permutations were employed. In this case, all the instances were solved, however it is possible to appreciate a variation in the execution time of the 30 permutations employed by the BF.

7.6 Conclusions

In this chapter, we have proposed two enhancements to the BF algorithm to solve the HCP. The first enhancement is related to early subcycle detection and addresses a limitation that, when fixing arcs, presents the BF. We have proved mathematically that subcycles can be generated when an arc is added to the set of fixed arcs and that will not be detected by the second LP. We have introduced a conjecture related to this issue to detect the generated subcycles. The experiments with random Hamiltonian graphs clearly show that our approach decreases the number of calls to the BF when solving an instance, and it solves more instances than the original BF. In more challenging graphs with lower densities and up to 234 vertices, it was observed that for some of the graphs there was also a gain in terms of the number of solved instances and calls to the BF.

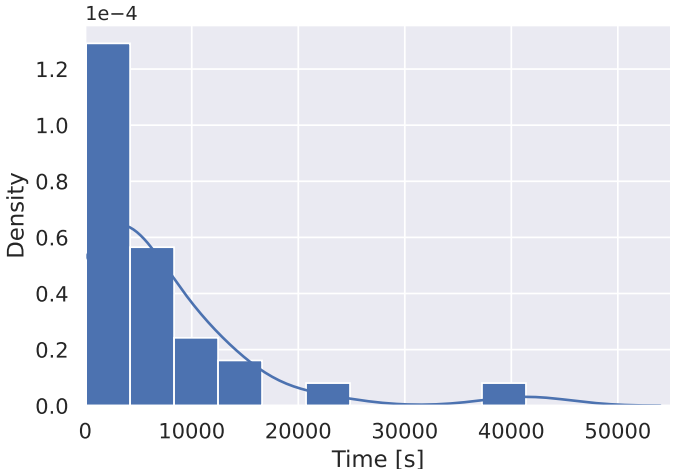


Fig. 7.10: Density plot of the execution times of the permutation-based BF for $t = 30$ and Graph 2 of the *challenge set*.

The second proposed enhancement builds on a previous proposal for undirected graphs [29] and simplifies the graph by eliminating some arc connections using an iterative method. This enhancement is not related to a limitation of the BF, but to a characteristic that the graph to be solved may present. It was found that a simplification can result in a reduction of a substantial number of arcs, even if the percentage of calls where at least one arc can be removed is relatively low. The experiments show an improvement in the number of solved instances for the BF with the simplification step compared to the BF with subcycle detection step in random graphs, although significant differences were not found. To a lesser degree, there was an improvement in some challenging graphs with lower densities, as when the graph's density decreases so does the number of arcs, hence hindering the simplification step. In two graphs of the *challenge set* with $D = 0.5$ and 400 and 1123 vertices, this method succeeded in solving them, in contrast to the original method and the one with the subcycle detection step.

Finally, it was seen in the experiments the advantage of employing the permutation-based BF. Thanks to this variant, there are more choices to solve a challenging instance and it allows to find, from a set of permutations, the most efficient matrix representation of a given instance.

8

Branching methods for the Branch-and-Fix

8.1 Introduction

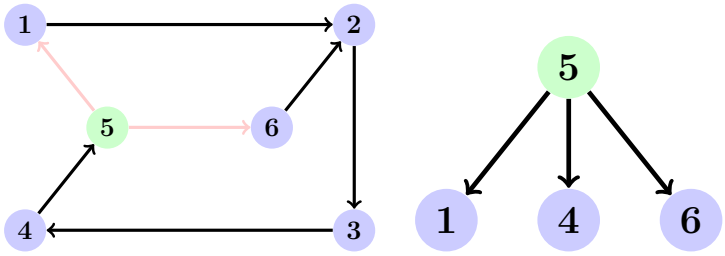
BB methods play an essential role in the solution of a variety of (mixed) integer programs and other constraint satisfaction problems. Recent research [10, 95, 107] has highlighted the convenience of devising strategies for learning to branch instead of using fixed branching schedules. However, the conception of such adaptive branching schemes requires the study of the particular domain where the BB method is going to be applied. It is also important to understand the benefits and limitations of branching methods presently used in that domain [118].

Branching is one of the main operations of BB algorithms, the best known exact algorithms for discrete optimization problems [107]. Branching consists of recursively splitting the search space into a smaller space and minimizing the objective function in the subspace [10]. In these operations, it is crucial the branching rule employed to determine which branch of the tree is used to generate the children [82]. An efficient branching rule can considerably reduce (several orders of magnitude) the size of the search tree.

The most important decisions associated with BB are variable selection and node selection [107]. The first involves selecting one of two variables related to a node to branch on; the second involves selecting a node to proceed. In the context of BF, the node selection is determined by the solution of the first LP, as the selected node will be the splitting node. However, the variable selection depends on the number of arcs emanating from a vertex called the splitting node, as d arcs will emanate from vertex i, in contrast to the two variables in BB algorithms.

To have a better understanding of the branching step in the BF, Figure 8.1 shows an example for the Hamiltonian graph shown in Figure 3.2. Figure 8.1a shows the solution of the first LP, which in this case corresponds to a 1-randomized policy. The splitting node is the vertex 5, the vertex in which the

randomization occurs, and it branches on arcs (5,1) and (5,6). The branching options are shown in Figure 8.1b, where the arcs that emanate from vertex 5 ($d = |\mathcal{A}(5)|$), are three: (5,1), (5,4) and (5,6). The branching step consists on determining the order in which the algorithm will proceed.



(a) 1-randomized solution from the first LP. The splitting node 5 branches on (5,1) and (5,6).

(b) The splitting node 5, where $d = 3$ arcs emanate from it.

Fig. 8.1: Example of a branching in the Hamiltonian graph of six vertices introduced in Figure 3.2.

One of the most efficient branching strategies is strong branching (SB) [3], despite its huge computational cost. SB evaluates the dual bound for every candidate variable by computing the LP relaxation resulting from the branching of that variable. Some branching approaches [69, 94] collect information on the BB performance. Lookahead branching consists of measuring the impact of the candidate variable on LP gains [69]. This approach offers a new branching rule, as well as tools to fix the bounds.

Fathom-based branching methods are also proposed motivated by the need to minimize the number of explored nodes and the fact that branching decisions are more important in the first levels of the tree. Entropy branching was proposed by Gilpin and Sandholm [68], based on the idea of removing uncertainty from the search tree by measuring uncertainty/certainty of a node using entropy. Lodi and Zarpellon [107] summarized the ideal properties of a branching method as follows: 1) node-efficiency; 2) increased importance of the top levels of the search tree; 3) time-efficiency; 4) adaptiveness within the tree evolution.

In this chapter, we investigate the effectiveness of various branching methods in the solution of the HCP using the BF. We compare several branching strategies from the literature and propose a novel one that consists of performing the branching by maximizing the quantity of fixed arcs in the third step of the BF algorithm.

The remainder of the chapter is organized as follows. Section 8.2 provides the context of the analyzed branching methods and presents the proposed

one. Section 8.3 is devoted to the evaluation of the methods in a number of experiments. Finally, Section 8.4 draws some conclusions.

8.2 The global branching method

Branching is one of the main operations for BB algorithms and the BF. As mentioned, it is very important the branching rule employed, as it can considerably reduce the size of the search tree. This section reviews some branching methods for the BF reported in the literature and proposes the global branching method.

8.2.1 Branching methods for the Branch-and-Fix reported in the literature

The authors of BF [78], proposed five branching methods based on the solution of the first LP. We will denote them local methods, as they take into account only the arcs where the branching has occurred. If vertex i is the splitting node that branches on (i, j) and (i, k), where $x_{ij} \leq x_{ik}$, these are the methods considered:

1. **Uninformed**: The branching is carried out in the numerical order of the labels of the vertices.
2. **Local-1**: First branch on arcs (i, j) and (i, k), and then the rest of the arcs in vertex order.
3. **Local-2**: First branch on arcs (i, k) and (i, j), and then the rest of the arcs in vertex order.
4. **Local-3**: First branch on arcs in vertex order. The last arcs are (i, j) and (i, k).
5. **Local-4**: First branch on arc (i, k), then the rest of arcs in vertex order and finally arc (i, j).

These methods are explained with an example in Figure 8.2. For the Uninformed branching method (Figure 8.2a), the arcs that emanate from the splitting node follow the numerical order of the labels of the vertices: (5,1), (5,4) and (5,6). For the rest of them, the solution of the first LP is taken into account. The components of the solution involved are x_{51} and x_{56} (the splitting node branches on (5,1) and (5,6)), where in this case $x_{56} \leq x_{51}$. In the following lines the branching order for the rest of the methods is indicated:

- **Local-1**: (5,6), (5,1) and (5,4) (Figure 8.2b).
- **Local-2**: (5,1), (5,6) and (5,4) (Figure 8.2c).
- **Local-3**: (5,4), (5,6) and (5,1) (Figure 8.2d).
- **Local-4**: (5,1), (5,4) and (5,6) (Figure 8.2e).

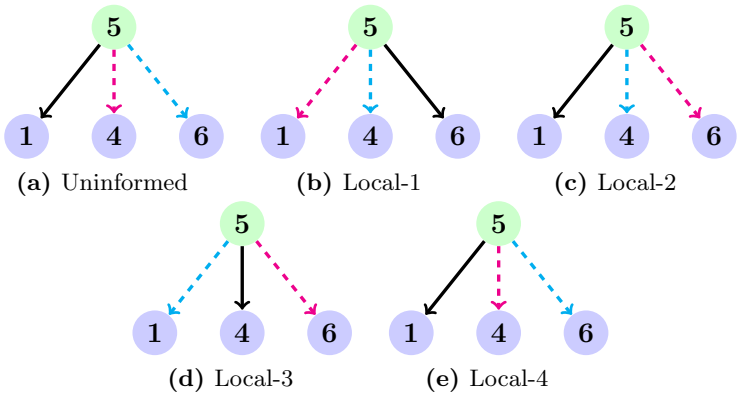


Fig. 8.2: Different orders to perform the branching depending of the branching method. The arrow in black refers to the first branching arc, the dash in pink to the second, and the dash in blue to the last.

Some preliminary research has analyzed the performance of these methods. In one investigation [19], the methods were evaluated in 50 randomly generated Hamiltonian cubic N-node graphs, where $N \in \{20, 30, 40, 50\}$. A specialized computer program (GENREG [115]) was used to create the cubic graphs that were used as a benchmark to evaluate different branching strategies. For the evaluation, the authors considered the number of branches explored by the BF among the five branching methods. In this benchmark, the Local-2 method performed best, with the lowest average number of explored branches. The same graphs and branching methods were employed to compute the BF using wedge constraints. The Local-4 performed well when wedge constraints were included, in contrast when the original BF was used. Local-2 and Local-4 were the most efficient; they both first branched on arc (i, k). For cubic graphs generated by this computer program, the branching arc should be (i, k) when using wedge constraints.

8.2.2 The global branching method

In this section, we propose the global branching method that takes into account the number of fixed arcs in the third step of the BF algorithm. The aim of this branching method is to select first the branch that leads to a more simplified subgraph. The pseudocode of the algorithm is shown in Algorithm 7. It takes as an input the splitting node and returns a set of branching candidates ordered using the mentioned criterion. In the following lines, its main steps are summarized:

- Each branching candidate has associated an empty set $\mathcal{L}_i$ (line 5 of the pseudocode).

- The arc that emanates from the splitting node is computed and the adjacency matrix is updated according to it. The set of fixed arcs of the updated adjacency matrix is computed and the arcs are added to $\mathcal{L}_i$. The `Update_Adjacency_Matrix` and `Update_Fixed_Arcs` follow the same rules as the ones described in Section 3.4. This is performed in lines 6-9 of the pseudocode.
- The function `Refined_Fixed_Arcs` checks whether there are or not new elements in $\mathcal{U}$. These elements are saved in the variable `last_fixed_arcs` (line 10 of the pseudocode). Afterwards, a loop is performed to iteratively update the adjacency matrix and fixed arcs (lines 11-15 of the pseudocode). The new fixed arcs computed in each step are added to $\mathcal{L}_i$ (line 16 of the pseudocode).
- The fixed arcs associated to the i-th emanating arc from the splitting node, are added to $\mathcal{L}$ (line 17). The function `Sort_Arcs` orders the emanating arcs from the splitting node according to the cardinality of the subset $\mathcal{L}_i \subset \mathcal{L}$ (line 18 of the pseudocode).

Algorithm 7 Global branching method

```
   Input: splitting_node
   Output: sorted_branching_candidates
 1: function GLOBAL(splitting_node)
 2:     L = {}
 3:     d ← |A(splitting_node)|
 4:     for i in d do
 5:         L_i = {}
 6:         arc ← Get_ith_Outgoing_Arc(splitting_node,i)
 7:         new_adjacency_matrix ← Update_Adjacency_Matrix(arc)
 8:         U ← Update_Fixed_Arcs(new_adjacency_matrix)
 9:         L_i ∪ U
10:         last_fixed_arcs ← Refined_Fixed_Arcs(U)
11:         while |last_fixed_arcs|> 0 do
12:             for a ∈ last_fixed_arcs do
13:                 new_adjacency_matrix ←Update_Adjacency_Matrix(a)
14:             U ← Update_Fixed_Arcs(new_adjacency_matrix)
15:             last_fixed_arcs ← Refined_Fixed_Arcs(U)
16:             L_i ∪ last_fixed_arcs
17:         L ∪ L_i
18:     sorted_branching_candidates ← Sort_Arcs(L,A(splitting_node))
   return sorted_branching_candidates
```

The global branching method is illustrated in Figure 8.3. In this case, the splitting node is vertex 3, where three arcs emanate from it, $d = |\mathcal{A}(3)| = 3$. The cardinality of $\mathcal{L}_1$ associated to the first arc (3,2) emanating from the

splitting node is $|\mathcal{L}_1|$=4. This means that fixing the arc (3,2) implies fixing other four more arcs. The cardinality of $\mathcal{L}_2$ associated to (3,4) is $|\mathcal{L}_2| = 6$ and the cardinality of $\mathcal{L}_3$ associated to (3,6) is $|\mathcal{L}_3| = 6$. The branching order in this case is the following: (3,4), (3,6) and (3,2) (in the case of a tie the numerical order of the labels of the vertices is followed.)

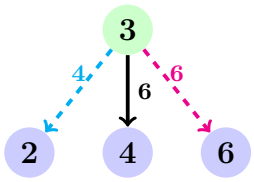


Fig. 8.3: The splitting node where the three branching candidates and the cardinality of the $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_3$ are indicated. The arrow in black refers to the first branching arc, the dash in pink to the second, and the dash in blue to the last.

Section 8.1 mentions the properties that a branching method should accomplish. The methods proposed in the literature (except for the uninformed method) and the global are adaptive within the tree evolution. They depend on the splitting node that comes from the solution of the first LP, and this is related to the subgraph built on the specific level of the tree. The main difference between the methods in the literature and the global one is that the nodes in the top levels have more importance in the latter. The global method prioritizes the arcs that will lead to a simplified subgraph of G. In the top levels, the graph can be more simplified by eliminating arcs that in deeper levels.

There are some issues that may have an influence on the behaviour of the branching strategies.

- The interaction between the parameters of the BF and the branching method.
- The particular characteristics of a graph (density and degree of the vertices).
- The permutation employed in the permutation-based BF.

Regarding the first item, there are some components of the BF such as the initial probability distribution or the discounted factor that may influence on the branching. Also, other components that were included in the dissertation, such as the simplification may have an impact on the branching strategy. Moreover, the impact may be different to one strategy to other. It could be useful to analyze the effect of these parameters, however the computational cost of the experiments would be high.

With respect to the second item, the particular characteristic of a graph can influence on the branching strategy. In a denser graph, more branching candidates will be available for the splitting node than in a sparser graph. Also, if there are some vertices in the graph with higher degree, and as a result of the first LP lead to splitting nodes, more branching candidates will be available. These characteristics of the graph will influence more on the global branching method, in comparison to the rest of the methods, as the computational cost of the global one will be higher. The number of $\mathcal{L}_i$s to be computed increases with the available branching candidates.

For the third item and in regard to the permutation-based BF, the employed permutation will have a clear effect on the branching strategy. For the uninformed branching method, that uses the numerical order of the labels, using a different labeling indicated by the permutation, will affect to the result. But even for the rest of the strategies, there is an influence of the permutation as they use at any time the numerical order of the labels. The local methods use to order the branching candidates, except for the two arcs where the randomization occurs ((i, j) and (i, k) where i is the splitting node). In the case of the global method, the numerical order of the labels is used when there is a tie among the candidates.

To summarize, there could be several factors that have influence on the branching, but it is very difficult to measure their effect without doing an exhaustive evaluation of each of the factors.

8.3 Experiments

The main goal of our experiments is to analyze the performance of different branching methods. In particular, we would like to assess the efficiency of the global method introduced in this dissertation. To this end, we use a benchmark of Hamiltonian graphs of different complexity; Graphs 1-10 from the *challenge set* and the 500 random Hamiltonian graphs introduced in Section 7.5.1.

This section describes the experiments to analyze the differences between the branching methods. For the evaluation of the experiments, the metrics introduced in Section 7.5.2 are used. The analysis includes the five branching methods proposed (uninformed method and four local methods) and the proposed global one. For the graphs in the *challenge set*, the permutation-based BF is used with $t = 30$. Each run of the BF is executed for a maximum time of 24 hours. The BF algorithm used to solve the problem incorporates the two enhancements mentioned in Section 7 (BF_SD_GS).

8.3.1 Results and discussion

In this section, the analysis of the results of the experiments and a discussion are carried out. First, the number of UIs through different branching strategies

is analyzed. Then, the distributions of the number of calls to the algorithm are analyzed.

8.3.1.1 Analysis of the number of unsolved instances

The distributions of the number of UIs are compared for the six branching methods. Table 8.1 shows the number of UIs in the graphs from the *challenge set* for the six branching methods. The lowest values are indicated in bold. For the first two graphs of the set, all instances are solved for all branching methods. The BF_SD_GS has more difficulty solving the rest of the graphs. The Local-2 method solves more instances; the lest efficient method is the uninformed method.

Table 8.1: Number of UIs for the 10 graphs from the *challenge set* and six branching methods.

Graph	Uninformed	Local-1	Local-2	Local-3	Local-4	Global
G_1	0	0	0	0	0	0
G_2	0	0	0	0	0	0
G_3	9	2	**0**	7	3	8
G_4	23	**17**	20	22	22	24
G_5	21	25	**18**	21	24	20
G_6	25	**11**	19	21	12	26
G_7	26	30	**22**	28	29	24
G_8	30	30	**29**	**29**	30	**29**
G_9	25	29	**24**	26	29	**24**
G_{10}	27	25	**24**	27	**24**	27

Table 8.2 shows the number of UIs for the random Hamiltonian graphs with different numbers of vertices and density. The number of UIs decreases when the density of the graph increases. This change in the number of UIs is more noticeable between $D = 0.25$ and $D = 0.44$. The global method solves more instances, and the least efficient method is Local-4.

To determine whether there are significant differences between the methods in the number of UIs, we apply the Friedman test [38], with $p = 0.05$ and using the Shaffer's procedure [61] as a correction function. The solved instances are represented by 1 and the unsolved ones by 0. There are no significant differences between the methods for any of the instances in the set of random Hamiltonian graphs and the *challenge set* except for Graph 6. In this particular case, the best method was the Local-1 and the worst one the global method.

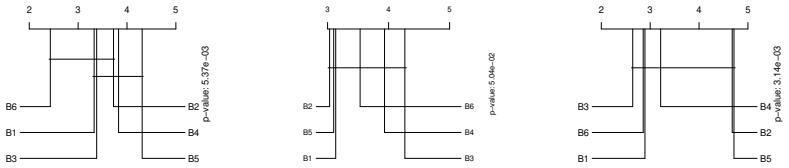
Table 8.2: Number of UIs for each of the branching methods for graphs with different D and $|V|$.

D	$\|V\|$	Uninformed	Local-1	Local-2	Local-3	Local-4	Global
	60	10	9	11	9	9	**6**
	70	**13**	18	17	15	18	17
0.15	80	19	22	19	20	21	**15**
	90	**17**	20	20	20	22	19
	100	22	23	**20**	24	24	22
	60	11	**6**	7	8	6	11
	70	14	12	11	14	11	**9**
0.25	80	15	13	13	14	17	**12**
	90	15	12	13	12	12	**10**
	100	12	15	**11**	13	18	15
	60	**1**	2	**1**	**1**	2	2
	70	3	**2**	7	**2**	2	4
0.44	80	**2**	4	3	6	8	3
	90	**2**	5	3	**2**	7	4
	100	5	**2**	6	5	3	5
	60	**0**	2	1	**0**	2	**0**
	70	**1**	**1**	**1**	**1**	**1**	**1**
0.64	80	1	2	1	1	**0**	1
	90	2	2	2	**0**	2	2
	100	1	3	1	**0**	2	**0**

8.3.1.2 Analysis of the number of calls

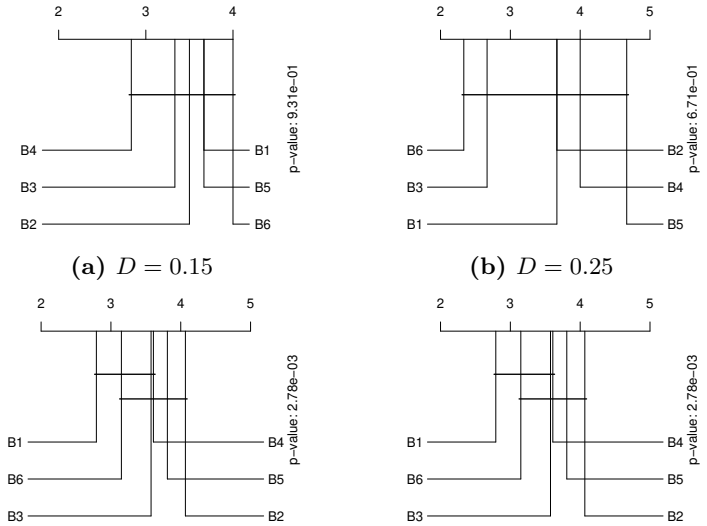
The distributions of the number of calls are compared for a subset of graphs solved for all the branching methods. In the *challenge set*, only Graphs 1, 2 and 3 have a subset of permutations for which BF has been able to solve the problem with all branching methods. This means that only some BF executions are solved by the six branching methods. The Friedman test is applied in this case as well, and significant differences appear for Graph 1 (p-value=7.05e-03) and Graph 3 (p-value=3.35e-01). The critical difference diagrams for these graphs are shown in Figure 8.4. In the figure, B_1 corresponds to the uninformed method, $B_2 - B_5$ to Local-1–Local-4, and B_6 to the global method. In both cases, the best method was the global one and the worst ones the Local-1 and Local-3.

The Friedman test is applied to the random Hamiltonian graphs with different densities. Significant differences are only found for the highest densities, $D = 0.44$ and $D = 0.64$ with p-value=2.78e-03 and p-value=5.44e-03 respectively. The critical difference diagrams for these graphs are shown in Figure 8.5. In both cases, the worst method is the Local-1, and the best one is the uninformed for $D = 0.44$ and Local-2 for $D = 0.64$.



(a) Graph 1 **(b)** Graph 2 **(c)** Graph 3

Fig. 8.4: Critical difference diagrams of the distributions of the number of calls for Graphs 1, 2, and 3 from the *challenge set*. The p-values obtained from the Friedman test are indicated.



(a) $D = 0.15$ **(b)** $D = 0.25$

(c) $D = 0.44$ **(d)** $D = 0.64$

Fig. 8.5: Critical difference diagrams of the distributions of the number of calls for random Hamiltonian graphs. The p-values obtained when applying the Friedman test are indicated.

For eight graphs from the *challenge set*, there are no executions of the permutation-based BF solved by all the branching methods for all the instances, making comparison impossible. For some random Hamiltonian graphs, these subsets are small (few graphs are solved by all the branching methods). For that reason, the branching methods are compared (in terms of number of calls) in pairs, taking the graphs solved in both cases. The Wilcoxon signed-rank test [38] is used to find significant differences between methods. This statistical test is a non-parametric test that can be applied to paired samples. This test is applied to Graphs 3-10 from the *challenge set* (for Graphs 1

and 2, all instances are solved by all methods) and the random Hamiltonian graphs, considering them in groups differentiated by density. This means that to compare two branching methods in random graphs the subset of common solved instances is considered for a given density. The number of these common instances is shown in Table 8.3. It can be observed that the number of common instances solved by two branching methods increases when the density gets higher.

Table 8.3: Number of common instances for the branching methods considered in pairs for the four densities.

		Common instances			
Method 1	**Method 2**	$D = 0.15$	$D = 0.25$	$D = 0.44$	$D = 0.64$
Uninformed	Local-1	18	27	98	109
Uninformed	Local-2	16	33	92	113
Uninformed	Local-3	20	27	98	117
Uninformed	Local-4	18	30	91	112
Uninformed	Global	27	42	102	119
Local-1	Local-2	14	39	90	110
Local-1	Local-3	12	34	96	112
Local-1	Local-4	24	49	98	110
Local-1	Global	18	37	93	110
Local-2	Local-3	13	31	89	116
Local-2	Local-4	14	32	85	111
Local-2	Global	16	34	87	114
Local-3	Local-4	13	34	90	115
Local-3	Global	24	33	94	118
Local-4	Global	15	36	88	113

Table 8.4 shows how many times there are significant differences between two methods in all explored graphs. The numbers above the principal diagonal indicate that the methods in the rows are significantly better than those in the columns. The numbers below the principal diagonal indicate that the methods in the rows were significantly worse than those in the columns. The uninformed branching method is frequently found to be significantly better.

8.4 Conclusions

In this chapter, we have analyzed the branching methods in the literature and introduced the global branching method. This latter method takes into account other specific aspects of the BF than those proposed in the literature. We have considered two aspects to evaluate the performance: the number of UIs and the number of calls to the algorithm.

Table 8.4: Number of times when one method is significantly different from other method (in terms of the number of calls) when applying the Wilcoxon signed-rank test for the graphs from the *challenge set* and random Hamiltonian graphs.

	Uninformed	**Local-1**	**Local-2**	**Local-3**	**Local-4**	**Global**
Uninformed	–	3	1	0	3	1
Local-1	1	–	0	1	1	0
Local-2	0	2	–	0	2	0
Local-3	0	3	0	–	3	0
Local-4	1	1	1	0	–	0
Global	0	2	0	0	1	–

In the case of the number of UIs, the global method was the method solving most instances for the random Hamiltonian graphs, specially for $D = 0.15$ and $D = 0.25$. However, for the graphs from the *challenge set*, that have smaller densities than the random graphs, the Local-2 was the method able to solve more instances. There were not found significant differences in the number UIs except for one graph, for which the global method was the one that solved the smallest number of instances. There was not found a method outperforming others, so depending on the density of the graph to be solved the global or the Local-2 could be selected.

Regarding the number of calls to the algorithm, for two graphs from the *challenge set*, the global method was found to be significantly faster. In the case of random Hamiltonian graphs and densities $D = 0.44$ and $D = 0.64$ the uninformed and the Local-2 were the best. As occurred with the number of UIs, there was not found a method clearly outperforming others.

To summarize, adding specific characteristics of the BF algorithm to a branching rule has not brought any clear advantage compared to the methods reported in the literature. Moreover, there was not found any significant gain when using a branching rule instead of using the numerical order of the labels of the vertices. This indicates that much more work is required in the design of branching strategies for the BF.

9

Branch-and-Fix collapse algorithm

9.1 Introduction

Linear programming based methods present some limitations when applied to discrete optimization problems. One of them is that the number of constraints increases with the size of the problem, making it computationally expensive to the LP solver [63]. Another limitation is that these methods consume time exploring solution spaces that lead to infeasible solutions [109]. In Chapters 7 and 8 we have focused on the second limitation and proposed strategies to fathom the branches of the logical tree constructed by the BF, thus reducing the size of the logical tree [68]. In this chapter, we will focus on reducing the size of the problem (number of vertices of the graph) to decrease the computational cost when solving the LPs.

To face the aforementioned limitations, we build our proposal on the BF algorithm. In BF [51], two main components can be identified: one related to the solution of the LPs, where the main role is played by the LP solver. Another component is the way branching is implemented and how the recursive calls to the LPs are invoked. The efficiency of this second component is closely related to the choice of the data structure and other design decisions of the algorithm.

In order to make the BF a more efficient approach, it is not only important to improve the performance of the LP solver, but also to make the second component more agile, decreasing its time and memory requirements [97]. In this chapter, a BF collapse algorithm is proposed that addresses some of the limitations of the BF by using the following strategies: 1) degree-based simplification; 2) a more efficient matrix representation; 3) the global branching method; 3) a matrix collapse step by finding paths. Some of the components, such as the degree-based simplification and the global branching method, were introduced in previous chapters. Here, we will introduce a more efficient matrix representation and the matrix collapse step related to decreasing the number of constraints in the LPs.

One possible approach to solve the HCP is to convert an instance into a TSP instance [84]. Nevertheless, most of the methods to solve the TSP are originally conceived for symmetric instances [114], including the very well-known Concorde TSP solver [5]. Solving asymmetric TSP instances requires transforming them into symmetric TSP instances by doubling the number of the vertices [92].

This chapter proposes a new algorithm called BF collapse which incorporates the four components. It evaluates the behaviour of its components and makes a comparison to the original BF. Moreover, the algorithm is compared to the Concorde solver in some challenging instances. For that purpose, a particular benchmark is built to evaluate the efficiency of the BF collapse algorithm.

The remainder of this chapter is organized as follows. In Section 9.2 the BF collapse algorithm is introduced and details about the implementation are given. Afterwards, in Section 9.3 an experimental validation of the method is carried out. Finally, Section 9.4 draws some conclusions.

9.2 Branch-and-Fix collapse algorithm

In this section, we present a new algorithm that addresses some of the limitations of linear programming based methods. Although the introduced algorithm shares the embedding of the HCP in an MDP [56] with the BF, it has fundamental differences.

1. Degree-based simplification.
2. A more efficient matrix representation.
3. The global branching method.
4. The matrix collapsing step by finding fixed paths.

Regarding the first component, the degree-based simplification was explained in Section 7.3. The second component will be presented in Section 9.2.1 of this chapter. In relation to the third component, the global branching method was introduced in Section 8.2. Although in the previous chapter no clear conclusion was drawn about the branching methods, the global branching method prioritizes fixing arcs, which is convenient for the matrix collapsing step. Finally, the matrix collapsing step will be presented in Section 9.2.2 of this chapter.

Before introducing the algorithm, we present the two components that have not been previously discussed in the dissertation, a more efficient matrix representation and a strategy to reduce the problem by collapsing the adjacency matrix.

9.2.1 Efficient matrix representation

Challenging HCP instances usually have low density, and consequently have associated sparse adjacency matrices. This means that the adjacency matrices of dimension $N \times N$, have a lot of zero entries. For high values of N, it is not efficient to have big data structures with redundant information; above all, taking into account the recursive nature of the BF algorithm. For this reason, we consider a more efficient matrix representation, $\boldsymbol{S}(G)$, instead of using the adjacency matrix as input of the BF algorithm.

$\boldsymbol{S}(G) = [s_{ij}]$ is an $N \times M$ matrix, where $N = |V(G)|$ and $M = max(d^+G(i))$, $i \in V(G)$. The columns of the matrix are filled by the outgoing vertices of each vertex i. As the out-degree of each vertex might be different, the rest of the positions are filled with zeros. Each row is represented in a way that the non-null values appear in the first positions.

As $M \leq N$, the number of columns of $\boldsymbol{S}(G)$ will be less than or equal to the number of columns of $\boldsymbol{A}(G)$. For large graphs, this difference can be noticeable in terms of execution time and the required memory of the algorithm. An example of the difference between the matrices $\boldsymbol{A}(G)$ and $\boldsymbol{S}(G)$ is shown in Example 2.

Example 2. Consider the directed graph G of 6 vertices shown in Fig. 3.2. The adjacency matrix $\boldsymbol{A}(G)$ and matrix $\boldsymbol{S}(G)$ are shown in Equation (9.1).

$$\boldsymbol{A}(G) = \begin{pmatrix} 0\,1\,0\,1\,1\,0 \\ 1\,0\,1\,0\,0\,1 \\ 0\,1\,0\,1\,0\,1 \\ 1\,0\,1\,0\,1\,0 \\ 1\,0\,0\,1\,0\,1 \\ 0\,1\,1\,0\,1\,0 \end{pmatrix} \qquad \boldsymbol{S}(G) = \begin{pmatrix} 2\,4\,5 \\ 1\,3\,6 \\ 2\,4\,6 \\ 1\,3\,5 \\ 1\,4\,6 \\ 2\,3\,5 \end{pmatrix} \tag{9.1}$$

One of the limitations of the presented representation is that the presence of a vertex i with $d^+(i) = N - 1$ destroys the efficiency of $\boldsymbol{S}$. However, it is not common to have that casuistry in challenging instances of the HCP.

9.2.2 Matrix collapsing by fixing arcs

In this section, the employed approach for matrix collapsing is presented. In the construction of an HC, the BF algorithm often fixes arcs that constitute a path. In these cases, it is clear that new arcs can not reach or leave the internal vertices of that path. However, in the constraints of the LP, those vertices are considered by keeping $N+1$ constraints (first LP) throughout the recursive calls. The collapse strategy allows the original number of vertices of a graph to be iteratively reduced. As the number of constraints is related to

the number of vertices, a decrease in the number of vertices would reduce the computational cost.

We say that an arc $(i,j) \in \boldsymbol{A}(G)$ is a *fixed arc* if the following three conditions are simultaneously fulfilled:

1. (i,j) is the only arc that leaves vertex i
2. (i,j) is the only arc that enters vertex j
3. There is no an arc (j,i)

A *fixed path* is a path $P = \{i_0 a_1 i_1 a_2 ... a_k i_k\}$, where $a_1, a_2, ..., a_k$ are fixed arcs. i_0 and i_k are called *extreme* vertices, whereas the rest of the vertices are called *intermediate* vertices. $k-1$ is said to be the length of the fixed path.

A *deflated graph* (B) is a graph created from G, where, for each fixed path, the intermediate vertices are eliminated. Suppose that there are l fixed paths $P_1, ..., P_l$ of length $k_1 - 1, ..., k_l - 1$. The number of vertices in B is of $N - ((k_1 - 1) + ... + (k_l - 1)) = m$. In the deflated graph, the vertices are re-labelled from 1 to m using a bijection. The *reduced matrix* is the $\boldsymbol{S}$ matrix of B, $\boldsymbol{S}(\boldsymbol{D})$. The extreme vertices of a fixed path constitute a fixed arc in B.

A *reduced cycle* is an HC that belongs to B. An *inflated cycle* is a cycle in G that is constructed from the reduced cycle using a bijection of the labels of the vertices and by considering the intermediate vertices of the fixed arcs in B. The way of creating an inflated cycle from the reduced cycle is explained in the following lines.

$h = (v_0, v_1, ..., v_{m-1}, v_0)$ is a reduced cycle in B. If there are l fixed paths in G, $|V(B)| = N - ((k_1 - 1) + ... + (k_l - 1)) = m$. We build a bijection between the vertices in B and the vertices in G, such that $v_0 = v'_0, v_1 = v'_1, ..., v_{m-1} = v'_{m-1}$. It is possible to construct $h'' = (v'_0, v'_1, ..., v'_{m-1}, v'_0)$. If there are l fixed paths in G, there will be $2 \times l$ extreme vertices in h'', where $(k_1 - 1) + ... + (k_l - 1)$ intermediate vertices can be added. Thus, the inflated cycle is built with length N.

Theorem 2. *If h is an HC in B, there exists another HC h' in G.*

Proof. Let $h = (v_0, v_1, ..., v_{m-1}, v_0)$ be an HC in B. As B is a reduced graph of another graph G, there exists at least one fixed path P_1. Without loss of generality, suppose that there is one fixed path of vertices $i_0, ..., i_k$ of length $k-1$, so that $m = N - (k-1)$.

Using the bijection between the vertices of B and G, we construct $h'' = (v'_0, v'_1, ..., v'_{m-1}, v'_0)$. Suppose that (v'_t, v'_{t+1}) are the extreme vertices of the fixed path P_1, where $v'_t = i_0$, $v'_{t+1} = i_k$ and $t \in \{0, ..., m-1\}$. Thus, $h' = (v'_0, ...v'_t, i_1, ..., i_{k-1}, v'_{t+1}, ..., v'_{m-1}, v'_0)$ is an HC if the following three conditions are fulfilled:

1. h' has N vertices. This is accomplished, because $m + (k-1) = N$.

2. All the vertices of h' are distinct.

   h contains different vertices because it is an HC in B. h'' also contains different vertices, as it is constructed using a bijection (by definition it is one-to-one correspondence). $i_1, ..., i_{k-1}$ are distinct as they constitute a path by definition. Additionally, they are distinct to the rest of the vertices, as B is constructed by eliminating those vertices.

3. All the arcs of h' belong to G.

   (i_0, i_1), (i_1, i_2),...,(i_{k-1}, i_k) belong to G as they were fixed arcs in G. (v'_0, v'_1), (v'_1, v'_2) ,.., (v'_{m-1}, v'_0) belong to B. They are not extreme vertices (there were not eliminated to construct B), thus they also belong to G.

□

Corollary 1. *Given a graph G, if its deflated graph B exists and B is a Hamiltonian graph, G is also a Hamiltonian graph.*

Proof. If B is a Hamiltonian graph, there exists at least one HC, h. From Theorem 2, its inflated cycle h' exists in G, so that G is a Hamiltonian graph.

□

Example 3. In this example, the matrix collapsing method is illustrated. Let $\mathcal{U} = \{[1, 2], [2, 3]\}$ be the set of fixed arcs of a directed graph G of 12 vertices shown in Fig. 9.1. This graph was constructed based on the graph shown in Fig. 3.2. The $\boldsymbol{S}(G)$ is shown on the left-hand side of Equation (9.2). In this case, there is one fixed path.

- $P_1 = \{v_0 a_1 v_1 a_2 v_2\}$, where $v_0 = 1, v_1 = 2, v_2 = 3, a_1 = (1, 2), a_2 = (2, 3)$. The length of the fixed path is 1.

In the deflated graph (Fig. 9.2), the intermediate vertex of the fixed path is eliminated and the vertices are re-labelled. $|V(B)| = N - (1)$=11. The bijection between the vertices of G and vertices of B is shown in Table 9.1. The reduced matrix constructed taking into account the bijection is shown in the right-hand side of Equation (9.2).

$V(G)$	1	2	3	4	5	6	7	8	9	10	11	12
$V(B)$	1	-	2	3	4	5	6	7	8	9	10	11

Table 9.1: Vertices of G and vertices of B after applying the bijection. The eliminated vertices are indicated with $-$.

A reduced cycle of B is $h = (1, 2, 6, 7, 3, 11, 10, 9, 5, 4, 8, 1)$. After using the bijection described in Table 9.1, $h'' = (1, 3, 7, 8, 4, 12, 11, 10, 6, 5, 9, 1)$. Its inflate cycle in G is $h' = (1, 2, 3, 7, 8, 4, 12, 11, 10, 6, 5, 9, 1)$.

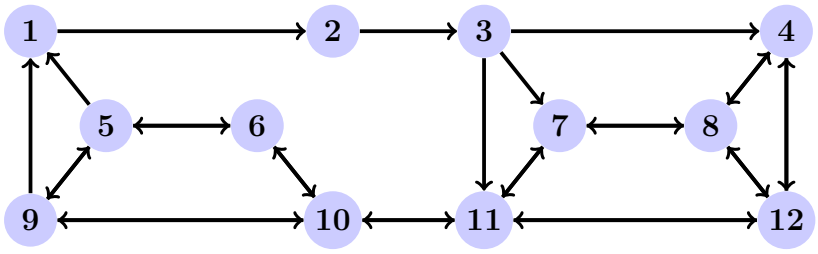


Fig. 9.1: Original graph.

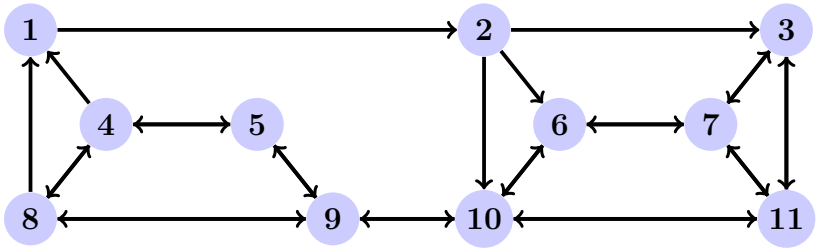


Fig. 9.2: Deflated graph.

$$\boldsymbol{S}(G) = \begin{pmatrix} 2 & 0 & 0 \\ 3 & 0 & 0 \\ 4 & 7 & 11 \\ 8 & 12 & 0 \\ 1 & 6 & 9 \\ 5 & 10 & 0 \\ 8 & 11 & 0 \\ 4 & 7 & 12 \\ 1 & 5 & 10 \\ 6 & 11 & 0 \\ 7 & 10 & 12 \\ 4 & 8 & 11 \end{pmatrix} \qquad \boldsymbol{S}(B) \begin{pmatrix} 2 & 0 & 0 \\ 3 & 6 & 10 \\ 7 & 11 & 0 \\ 1 & 5 & 8 \\ 4 & 9 & 0 \\ 7 & 10 & 0 \\ 3 & 6 & 11 \\ 1 & 4 & 9 \\ 5 & 10 & 0 \\ 6 & 9 & 11 \\ 3 & 7 & 10 \end{pmatrix} \tag{9.2}$$

The pseudocode of the implemented BF collapse algorithm is shown in Algorithm 8. It takes as an input an adjacency matrix and an empty set of fixed arcs ($\emptyset$). It returns value *True* of the boolean variable `found` and the HC, or value *False* of the boolean variable `found` and $\emptyset$, when all the branches have been explored without finding an HC.

The functions that this algorithm uses were described in Section 3.4, Section 7.3.2 and Section 8.2. Some of the functions were adapted to take as an input the $\boldsymbol{S}(G)$ instead of the adjacency matrix. The function `Get_Arc`

computes the arc that emanates from the splitting node to the i-th branching candidate. The functions `Deflate_Graph` and `Inflate_Graph` implements the matrix collapsing method presented in this section.

9.3 Experiments

The main goal of the experiments is to evaluate the different components of the BF collapse algorithm and to compare it with another HCP approach in directed graphs. The analysis of the results is performed using two criteria: 1) the number of UIs by each method; 2) the time needed to solve the instances by each method. Each method will comprise different components of the BF collapse, and we will refer to them as BF variants. We also apply the BF collapse to a manufacturing problem for illustrative purposes.

This section first presents the application of the BF collapse algorithm to a manufacturing problem. Next, it introduces the benchmark used to perform the experiments. Later, the effect of the simplification, matrix collapsing and the branching method are analyzed. Finally, our algorithm is compared to the TSP Concorde solver.

9.3.1 Application of the HCP to a manufacturing problem

Digital 3D objects are widely used in many different industries including virtual reality, CAD and AM. This is a fast-growing area of research that applies concepts of Applied Mathematics, Computer Science and Engineering. In 3D computer graphics, a polygon mesh is a collection of vertices, edges and faces that defines the shape of a polyhedral object. The faces can be triangles (triangle mesh), quadrilaterals or other simple convex polygons.

A *cubic graph* is a graph in which every vertex has degree three. These graphs are of special interest in the generation of 3D computer images, as they are built from triangle meshes. One of the main drawbacks of such representations is the difficulty that they present to feed the hardware [143]. Combining the triangles into a single triangle strip would significantly accelerate the process of feeding the hardware. Finding a single strip that incorporates all triangles in the mesh is equivalent to finding the HCP in the corresponding dual graph in which every triangle is a vertex. A *dual graph* of a planar graph G is a graph that has a vertex for each face in G.

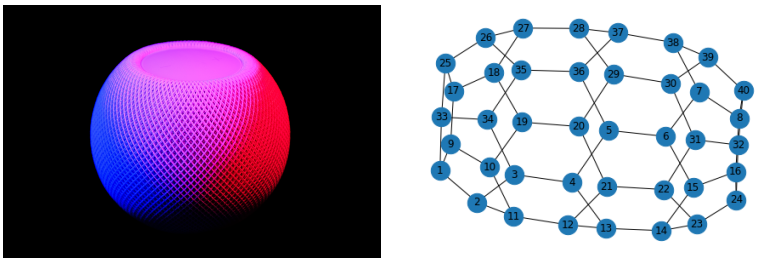
Figure 9.3 shows a sphere and its dual graph after applying a triangle mesh. The graph has 40 vertices, 120 arcs and $D = 0.15$. We apply the BF collapse to find an HC in the graph shown in Figure 9.3b using the global branching method, the degree-based simplification and the matrix collapsing. The execution time of the algorithm was 0.70 seconds. The HC with the set of arcs that constitute the cycle is given in Equation (9.3).

Algorithm 8 Branch-and-Fix collapse

Input: $\boldsymbol{S}(G)$, $\emptyset$
Output: True/False, HC/$\emptyset$

function BRANCH_AND_FIX_COLLAPSE($\boldsymbol{S}(G)$, $\mathcal{U}$)
  updated_ $\boldsymbol{S}(G)$, status ← Degree_Simplification($\boldsymbol{S}(G)$)
  if status ≠ feasible **then**
    return False,$\emptyset$
  status, $\boldsymbol{x_2}$, $F(\boldsymbol{x_2})$ ← Second_LP(updated_ $\boldsymbol{S}(G)$, $\mathcal{U}$)
  if status ≠ feasible or $F(\boldsymbol{x_2})$> bound **then**
    return False,$\emptyset$
  candidate_vertices ← Inspect_Subcycles($\boldsymbol{x_2}$)
  if |candidate_vertices|> 0 **then**
    if candidate_vertices ⊂ $\mathcal{U}$ **then**
      return False,$\emptyset$
  status, $\boldsymbol{x}$, $F(\boldsymbol{x})$ ← First_LP(updated_ $\boldsymbol{S}(G)$, $\mathcal{U}$)
  if status ≠ feasible **then**
    return False,$\emptyset$
  splitting_node ← Identify_Splitting_Node($\boldsymbol{x}$)
  if splitting node = $\emptyset$ **then**
    HC ← Identify_HC($\boldsymbol{x}$)
    return True,HC
  else
    d ← |$\mathcal{A}$(splitting_node)|
    found ←False
    i ← 0
    sorted_candidates ← GLOBAL(splitting_node)
    while (found = False) and (i < |sorted_candidates|) **do**
      fixed_arc ← Get_Arc(sorted_candidates,i)
      new_ $\boldsymbol{S}(G)$ ← Update_ $\boldsymbol{S}(G)$(fixed_arc)
      $\mathcal{U}$ ← Update_Fixed_Arcs(new_$\boldsymbol{S}(G)$)
      last_fixed_arcs ← Refined_Fixed_Arcs($\mathcal{U}$)
      while |last_fixed_arcs|> 0 **do**
        for arc ∈ last_fixed_arcs **do**
          new_ $\boldsymbol{S}(G)$ ←Update_ $\boldsymbol{S}(G)$(arc)
        $\mathcal{U}$ ← Update_Fixed_Arcs(new_ $\boldsymbol{S}(G)$)
        last_fixed_arcs ← Refined_Fixed_Arcs($\mathcal{U}$)
      circuit,red_$\boldsymbol{S}(G)$ ← Deflate_Graph(Update_ $\boldsymbol{S}(G)$)
      if circuit=True **then**
        HC ← Get_HC(Update_ $\boldsymbol{S}(G)$)
        return True, HC
      if |red_$\boldsymbol{S}(G)$| > 0 **then**
        found,red_HC ← BRANCH_AND_FIX_COLLAPSE(red_$\boldsymbol{S}(G)$, $\mathcal{U}$)
        if found=True **then**
          HC ← Inflate_Graph(red_HC)
      else
        pass
      found, HC ← BRANCH_AND_FIX_COLLAPSE(new_ $\boldsymbol{S}(G)$, $\mathcal{U}$)
      i ← i + 1
    if found=True **then**
      return True, HC
    else
      return False, $\emptyset$

$$\begin{aligned}\mathrm{HC} = \{&[1,33],[33,25],[25,17],[17,18],[18,19],[19,20],[20,21],[21,12],[12,13],[13,4],\\ &[4,5],[5,36],[36,37],[37,38],[38,39],[39,40],[40,32],[32,24],[24,16],[16,8],\\ &[8,7],[7,6],[6,15],[15,14],[14,23],[23,22],[22,31],[31,30],[30,29],[29,28],\\ &[28,27],[27,26],[26,35],[35,34],[34,3],[3,2],[2,11],[11,10],[10,9],[9,1]\}\end{aligned} \tag{9.3}$$



(a) Sphere **(b)** Dual graph

Fig. 9.3: A sphere and its dual graph obtained after applying a triangle mesh.

9.3.2 Benchmark definition

The employed benchmark is compound by two different sets of Hamiltonian graphs. Here, in comparison to Chapters 7 and 8, we use directed graphs that are not necessarily doubly connected. The first set includes graphs from the Knight's tour problem of dimension 8×8 introduced in Section 4.1.2.

The second set includes challenging directed Hamiltonian instances, which were generated in two steps. First, the Hamiltonian directed graphs are created randomly. Then, some modifications are made to the adjacency matrices with the aim of being challenging for the Concorde to solve them. Concorde is currently considered among the state-of-the-art software for solving the TSP. Our objective is to compare the BF collapse algorithm with the Concorde solver in instances that are challenging for the Concorde.

To study the behavior of the algorithm on a diverse set of graphs, we create a benchmark by generating graphs of $N = 100$ vertices and different densities, $D = 0.15, 0.25, 0.44$ and 0.64. Four instances were created for each different value of the parameter.

To generate the adjacency matrices, a random permutation of length N is created because an HC can be represented by a permutation. Next, a random asymmetric matrix of zeros and ones of dimension $N \times N$ is created. We

use probability p and $1 - p$ to distribute ones and zeros in the adjacency matrix respectively. In this case, $p = D \times 0.5$. Finally, the permutation is used to connect the arcs in the adjacency matrix that are part of the HC. This procedure is similar to the one described in Algorithm 6, with the difference that the matrix is not filled symmetrically (lines 4-6).

Regarding the creation of challenging instances, we use a method that tries to construct graphs that maximizes the time spent by Concorde to solve them. It is a way of identifying HCP instances that are challenging for Concorde. The following steps were repeated until satisfying a termination criterion.

1. Execute Concorde for the generated random adjacency matrix ($\boldsymbol{M_0}$) and save the execution time T_0.
2. Apply u random arc connection changes to $\boldsymbol{M_0}$ to obtain $\boldsymbol{M}$. If an arc already exists, it is eliminated, otherwise it is added to the graph.
3. Execute Concorde for $\boldsymbol{M}$ and save the execution time T.
4. If $T > T_0$, then $T_0 = T$ and $\boldsymbol{M_0} = \boldsymbol{M}$. Otherwise, return to Step 2.

In this case, as termination criterion, we considered 24 hours execution and $u = 5$.

9.3.3 Evaluation of the components of the BF collapse algorithm

The effect of the degree-based simplification (simp), the global branching method and the matrix collapsing (collapse) are evaluated, as they are the main proposed components of the BF collapse algorithm to address the limitations of the previous methods. For that purpose, we replace different components of the BF collapse algorithm, independently and in combination. We also considered a combination that does not include any of the four components. This combination would correspond to the basic BF setting.

The experiments also allow us to evaluate the impact of the branching method (global versus uninformed) in the introduced BF collapse method. These experiments are carried out in the Knight's tour of 64 vertices, using the permutation-based BF collapse ($t = 30$). The experiments were executed for a maximum time of 48 hours.

First, the number of UIs are analyzed between the two branching methods that include different combinations of simp and collapse. Table 9.2 shows the number of UIs and the mean and standard deviation of the execution time for all the considered BF variants. It can be observed that the lower number of UIs was achieved by the global method when applying collapse. The lowest mean and standard deviation values were achieved with the uninformed method and collapse. On the contrary, the worst results were for the original BF and uninformed method. It can be observed that, rather than applying them together, it is better to apply simplification or collapse separately.

Branch	Simp	Collapse	UI	μ	σ
Uninformed			17	36063	44900
Uninformed	✓		9	9074	16934
Uninformed		✓	8	**8323**	**16892**
Uninformed	✓	✓	9	15267	28844
Global			20	31605	47713
Global	✓		8	9525	18616
Global		✓	**5**	13726	22802
Global	✓	✓	15	18611	40377

Table 9.2: The number of UIs, the execution mean time (μ) and standard deviation (σ) for all the considered BF variants. The lowest values are indicated in bold.

A statistical test is used to determine whether there are significant differences between the global branching method and the uninformed branching method, for equal configurations of simp and collapse. The Wilcoxon signed-rank test is used with a significance threshold of $p = 0.05$. The solved instances are represented by 1 and the unsolved ones by 0, and we call them $\{0, 1\}$ distributions. Only the instances that have been solved by the methods to be compared are taken into account.

Table 9.3 shows the p-values after applying the statistical tests to know whether or not there are significant differences between the branching methods in terms of the number of UIs among different methods. Significant differences were not found in any of the cases. That is, when using different variants of the algorithm regarding the simplification or collapse, it does not matter which branching rule to use to have more or less solved instances.

M1			**M2**			**p-value**
branch	simp	collapse	branch	simp	collapse	
Uninformed			global			0.43
Uninformed	✓		global	✓		0.70
Uninformed		✓	global		✓	0.32
Uninformed	✓	✓	global	✓	✓	0.10

Table 9.3: p-values obtained from the Wilcoxon signed-rank test to measure significant differences between $\{0, 1\}$ distributions of the global and uninformed branching methods.

Similarly, Table 9.4 shows the p-values to know whether or not there are significant differences between the different methods for the uninformed branching method and the global branching method. It can be observed that for the uninformed branching method, there are significant differences be-

tween the original BF and the components applied by us in favour of our proposals. For the global one, significant differences were found between the original BF and the simplification and collapse in favour of these last two. However, applying the simplification and the collapse together compared to applying only collapse significantly reduces the number of solved instances for the global method.

Uninformed					**Global**				
M1		M2		p-value	M1		M2		p-value
simp	collapse	simp	collapse		simp	collapse	simp	collapse	
		✓		**0.01***			✓		**0.00***
			✓	**0.00***				✓	**0.00***
		✓	✓	**0.03***			✓	✓	0.17
✓			✓	0.32	✓			✓	0.18
✓		✓	✓	1.00	✓		✓	✓	0.05
	✓	✓	✓	0.74		✓	✓	✓	**0.01**$^+$

Table 9.4: p-values obtained from the Wilcoxon signed-rank test to measure significant differences between $\{0, 1\}$ distributions of the components for the global and uninformed branching methods. The p-values lower than 0.05 are indicated in bold. The symbols $*$ and $+$ indicate that the differences are in favor of Method 1 and Method 2, respectively.

It can be concluded that the proposed components improved the original BF in terms of the number of solved instances, with the exception of using simplification and collapse together for the global method. As the number of solved instances for the original BF are lower compared to the other methods and we need a common subset of instances to compare the time; therefore, in the following analysis the original BF has not been considered.

Secondly, the analysis of the execution times is carried out. Table 9.5 shows the p-values obtained from the statistical tests to measure significant differences between uninformed and global branching methods for different configurations of simp and collapse. It can be observed that significant differences were found when applying simplification and collapse together in favor of the global branching method. Note, that although not significantly, for the global branching method less instances were solved with this method than for the uninformed method.

Table 9.6 shows the p-values obtained from the statistical tests to measure significant differences between distributions of time executions for different combinations of simp and collapse. This comparison is carried out independently for global and uninformed branching methods. It can be observed that for the uninformed branching method there are significant differences in terms of time when applying only simplification, than both simplification and collapse, in favour of the first one. For the global method, it is better in terms of time considering only simplification or collapse instead of combining both.

M1			M2			p-value
branch	simp	collapse	branch	simp	collapse	
Uninformed	✓		global	✓		0.24
Uninformed		✓	global		✓	0.17
Uninformed	✓	✓	global	✓	✓	**0.00***

Table 9.5: p-values obtained from the Wilcoxon signed-rank test to measure significant differences between distributions of time executions between global and uninformed branching methods. The p-values lower than 0.05 are indicated in bold. The symbols $*$ and $+$ indicate that the differences are in favor of Method 1 and Method 2, respectively.

Uninformed					Global				
M1		M2		p-value	M1		M2		p-value
simp	collapse	simp	collapse		simp	collapse	simp	collapse	
✓			✓	0.61	✓			✓	0.96
✓		✓	✓	**0.04**$^{+}$	✓		✓	✓	**0.01**$^{+}$
	✓	✓	✓	0.06		✓	✓	✓	**0.01**$^{+}$

Table 9.6: p-values obtained from the Wilcoxon signed-rank test to measure significant differences between distributions of time executions between different components for the global and uninformed branching methods. The p-values lower than 0.05 are indicated in bold. The symbols $*$ and $+$ indicate that the differences are in favor of the right side method and left side method, respectively.

It can be concluded that the proposed main components have a positive effect both in terms of the number of solved instances and execution time. It was found that it is more efficient to apply collapse or simplification separately than in a combined way, above all for the global branching method.

9.3.4 Comparison to Concorde solver

In this section, the results of the experiments carried out with the Concorde and BF collapse algorithm in the defined challenging directed Hamiltonian instances are presented. In order to apply the Concorde to them, the directed HCP instances are converted into symmetric TSP instances, as explained in Section 4.1.1.1. These instances have $2N = 200$ vertices and the distance matrices are built as described in Equations (4.2) and (4.3).

Also in this case, the BF collapse algorithm is executed using the permutation-based variant with $t = 30$. The experiments were executed for a maximum time of 24 hours. The BF collapse is considered with the uninformed branching method, without degree-based simplification and with the matrix collapsing as it was found to be the fastest method in the previous experiments. Note, that the global branching method was the variant that solved more instances; however the uninformed branching method was faster. The resulting minimum

time of the 30 parallel executions of the BF collapse is considered the best time for each graph.

D	**0.15**				**0.25**			
Graph	1	2	3	4	1	2	3	4
BF	32.00	12.99	15.12	13.86	19.05	10.69	20.15	19.77
Concorde	**10.40**	**9.22**	**11.44**	**11.48**	**12.15**	**8.86**	**11.54**	**10.21**
D	**0.44**				**0.64**			
Graph	1	2	3	4	1	2	3	4
BF	**17.51**	17.22	27.74	28.30	23.96	22.97	39.85	40.79
Concorde	21.83	**13.57**	**19.40**	**11.03**	**12.26**	**22.40**	**15.93**	**14.16**

Table 9.7: Execution time in seconds of the Concorde and BF collapse algorithm for random challenging Hamiltonian graphs. The lowest values are indicated in bold.

Table 9.7 shows the execution time for the Concorde and the BF collapse algorithm (minimum time of the 30 parallel executions) in the challenging graphs with the best results indicated in bold. All the challenging instances were found to be Hamiltonian, as the cost of the routes was $N = 100$ when executing Concorde and the BF collapse succeeded in finding the HCs. In this case, all the instances were solved by both Concorde and BF collapse. It can be observed that the time required by the Concorde to solve the instances was smaller than the time required by the BF collapse with the exception of $D = 0.44$ and Graph 1. It should be noted that an implementation in C programming language can be 45 times faster than the same program implemented with the Python programming language.

9.4 Conclusions

In this chapter, we have proposed a new approach to deal with the HCP in directed graphs by addressing some limitations that present linear programming based algorithms. Specifically, we have focused on reducing the number of constraints and variables in the LPs to decrease the computational cost and fathoming solution spaces that lead to infeasible solutions. For that purpose, we have employed four components: 1) degree-based simplification; 2) more efficient matrix representation; 3) the global branching method; 4) matrix collapsing step by fixing arcs. The degree-based simplification and the global branching methods presented in previous chapters focuses on rejecting the branches that lead to non-HCs, thus reducing the number of variables. The matrix collapsing step by fixing arcs focuses on reducing the number of constraints of the LPs.

In the case of the matrix collapsing, we have proved mathematically that it is possible to solve the problem in a deflated graph with smaller number of

vertices than the original graph, hence decreasing computational cost. This was corroborated in the conducted experiments, as it was observed that the proposed components have a positive effect in the execution time. We have also observed that it was possible to apply the BF collapse method to a manufacturing problem where the HCP arises. In the experiments carried out to compare our approach to the Concorde TSP solver, promising results were obtained. Our algorithm, with an implementation in C programming language, could be an efficient alternative for the solution of both directed and undirected instances of the HCP.

10

An extension to the Branch-and-Fix to solve the multi-objetive Hamiltonian cycle problem

10.1 Introduction

In many real-world problems, various criteria should be considered simultaneously, turning optimization problems into MO optimization problems. The TSP is by definition a single-objective optimization problem, but there is some research [65, 18] that investigates the MO TSP. George and Amudha [65] defined the MO TSP as an expanded instance of the TSP by considering more than one objective function and solved the problem using a GA [70].

Lianshuan and Zegyan [104] formulated the MO TSP as a bi-objective optimization problem. The problem consisted of searching for the best route by balancing between the distance and the cost, and it was solved using a GA. Similarly, Bock and Klamroth [18] addressed a bi-objective optimization problem by balancing cost minimization (TSP) and arrival time minimization (traveling repairman problem) [42]. The traveling repairman problem consists of visiting all nodes in a graph in order to collect time-dependent profits. The objective is to minimize the sum of arrival times. The authors combined the optimization problems in an MO scenario and solved it using a dynamic programming approach.

The TSP has been solved by both exact and metaheuristic methods, but in the case of MO TSP, the latter method predominates [26, 119]. MO combinatorial optimization problems have commonly been addressed using metaheuristic techniques because of their computational complexity [85]. The main advantages of metaheuristics are that they are computationally efficient, general and simple to implement.

To the best of our knowledge, there is no other approximation for the MO HCP in the literature and, although the HCP is closely related to the TSP, often it is not trivial to find HCs in a graph. Minimizing the total weight of a tour constitutes a simple, linear objective function and some authors have argued [11] that much of the difficulty of the TSP is embedded in the HCP.

There are several specific methods for the TSP that take into account the sparsity of a graph [147] [15], hence it could be argued that those methods could serve to address the HCP. However, there is a gap in the literature related to the MO variant of the TSP focused on sparse graphs, as there are no straightforward methods to address this MO problem, which has specific constraints. The most common alternatives for MO problems with constraints are to *repair* the solution [108] or to *enforce* the solution space to be feasible [137]. A solution (tour) of the MO TSP is feasible for the MO HCP if it is an HC. As previously mentioned, in the literature there are no ways to *repair* non-HC solutions or to *enforce* the tours to be always HCs.

Therefore, in this chapter, we present a framework for the MO HCP and a heuristic approach to solve it. In the experimental section, we focus on MO problems defined on undirected graphs. However, the introduced algorithm could also be applied to optimize multiple objectives related to HCs defined on directed graphs. This is an important difference over strategies that present the HCP as a special case of the TSP on undirected graphs, and apply efficient TSP solvers on the transformed problem. This was explained in more detail in Section 4.1.1.1. Therefore, it is important to introduce the MO scenario, and in particular propose methods to solve it.

Simply stated, the MO HCP is the problem of finding HCs that minimize determined multiple criteria. The problem encapsulates two other problems: 1) finding an HC (solving the HCP); 2) finding an HC that minimizes a given objective (a single-objective HCP). This idea is explained in the diagram shown in Figure 10.1.

Let G be a Hamiltonian graph of N vertices with an associated set of matrices $\boldsymbol{W}^1, ..., \boldsymbol{W}^m$ where the entry $w^l_{i,j}$ $l \in \{1, ..., m\}$ represents the l^{th} weight associated with the arc (i, j). Each matrix represents a different way to evaluate HCs in G. The approximate PS will comprise HCs that minimize the sum of the weights for the different matrices. We use the term of approximation as it is not practical to determine the whole PS [134]. Notice that we simultaneously address the problems of finding an HC and minimizing the weights associated with the arcs in the HC.

Our heuristic approach consists of extending the BF algorithm [51] to deal with the MO HCP. The problem addressed in this chapter consists of finding the set of non-dominated HCs (HCs with associated non-dominated objective functions) using the BF algorithm. A number of modifications are made to the algorithm to avoid exploring dominated solutions and promote the exploration of different search areas.

The remainder of this chapter is organized as follows. Section 10.2 presents the MO HCP and explains the modifications performed to the BF algorithm. Section 10.3 gives the context of the experiments by describing the benchmark instances, methods and measurements employed to evaluate the performance

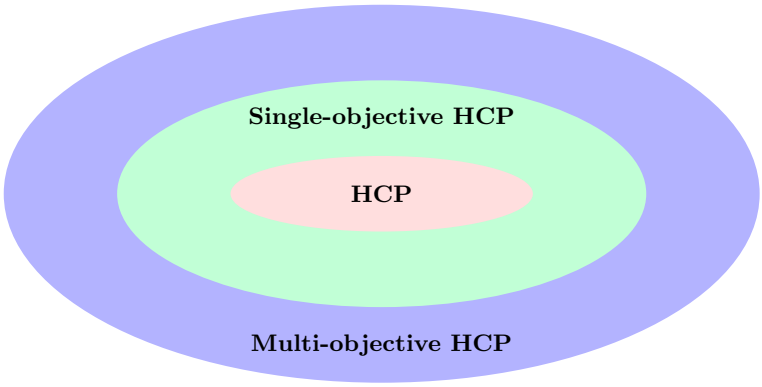


Fig. 10.1: Explanatory diagram of the proposed approach. The single-objective HCP encapsulates the HCP, and the multi-objective HCP, the HCP and single-objective HCP.

of the approach. Section 10.3.3 is devoted to the experimental results and discussion. Finally, Section 10.4 draws some conclusions.

10.2 Multi-objective Hamiltonian cycle problem

Let $G = (V(G), A(G))$ be a graph, where $V(G)$ is the set of nodes $|V(G)| = N$, and $A(G)$ is the set of arcs. This scenario can be defined for both directed and undirected graphs. We define m asymmetric weight-matrices, $\boldsymbol{W}^l$, $l \in \{1, ..., m\}$. We use asymmetric weight-matrices because the BF internally considers arcs with direction. Every arc in the graph has associated a positive weight w_{ij}^l in each l matrix. The objective function associated with a matrix $\boldsymbol{W}^l$ and a specific HC is denoted as δ_l^{HC} and is defined in Equation (10.1).

$$\delta_l^{HC} = \sum_{k=1}^{|A|} s_k w_k^l \qquad s_k \in \{0, 1\} \tag{10.1}$$

where $s_k = 1$ if the k^{th} arc belongs to the HC and $s_k = 0$ otherwise. w_k^l represents the entry of $\boldsymbol{W}^l$ corresponding to arc k. Finally, $\boldsymbol{\delta^{HC}}$ is the multiple-objective function associated with the HC, as defined in Equation (10.2).

$$\boldsymbol{\delta^{HC}} = \{\delta_1^{HC}, ..., \delta_m^{HC}\} \tag{10.2}$$

An important constraint in this framework is that G is not a complete graph; therefore, finding an HC is not trivial. A straightforward approach for finding the non-dominated HCs is to enumerate all possible solutions and select the optimal ones. This is illustrated in Example 4. However, complete

enumeration is not a feasible approach, even for small values of N. The BF algorithm searches the space of the HCs by pruning those branches guaranteed not to lead to a valid HC. We extend this rationale but add a new criterion to fathom the possible dominance of a point.

Example 4. Consider the Hamiltonian graph of 6 vertices shown in Figure 3.2 and two asymmetric weight-matrices. The graph contains the following 6 HCs that we represent them as permutations.

$$\{(4\ 6\ 2\ 3\ 1\ 5), (2\ 6\ 4\ 5\ 1\ 3), (2\ 3\ 6\ 1\ 4\ 5), (4\ 1\ 2\ 5\ 6\ 3), (5\ 3\ 4\ 1\ 6\ 2)$$

$$(5\ 1\ 6\ 3\ 4\ 2)\}$$

For instance, for the first $HC_1 = (4\ 6\ 2\ 3\ 1\ 5)$, the set of arcs belonging to the HC is $\{(1,4),(2,6),(3,2),(4,3),(5,1),(6,5)\}$. To calculate the first objective function associated to the HC, the entries of the first cost matrix $\boldsymbol{W}^1$ corresponding to each arc in the set are added.

$$\delta_1^{HC_1} = w_{14}^1 + w_{26}^1 + w_{32}^1 + w_{43}^1 + w_{51}^1 + w_{65}^1$$

In the same way, the second objective function associated to the HC is constructed using the second cost matrix $\boldsymbol{W}^2$. The bi-objective function $\boldsymbol{\delta^{HC_1}}$ is compounded by $\delta_1^{HC_1}$ and $\delta_2^{HC_1}$. Suppose that the following bi-objetive functions are considered:

$$\boldsymbol{\delta^{HC_1}} = \{5309, 613.4\}\ \boldsymbol{\delta^{HC_2}} = \{6416, 514.3\}\ \boldsymbol{\delta^{HC_3}} = \{5347, 516.98\}$$

$$\boldsymbol{\delta^{HC_4}} = \{4338, 516.98\}\ \boldsymbol{\delta^{HC_5}} = \{5256, 613.4\}\ \boldsymbol{\delta^{HC_6}} = \{5422, 514.3\}$$

.

The PS approximation and non-dominated HCs can be calculated using the evaluations of the two objectives for the six solutions. The non-dominated HCs are HC_4 and HC_6, as the vectors of objectives $\boldsymbol{\delta^{HC_4}}$ and $\boldsymbol{\delta^{HC_6}}$ are non-dominated.

$$\text{PS} = \{(4338, 516.98), (5422, 514.3)\}$$

10.2.1 Multi-objective Branch-and-Fix

The algorithm solves the MO HCP for a given graph. It takes as input the adjacency matrix of the graph and the corresponding weight-matrices and returns the PS. As the purpose is different from the original BF algorithm, this revised algorithm is called the MO-BF. The original BF is modified by making to stop exploring solutions which can be proved to be dominated partial HCs even before they are completed. In principle, the algorithm allows as many objectives as the user specifies. However, for problems with many objectives,

the storage and time requirements of the algorithm may increase considerably, as these problems have very large PSs. The algorithm focuses on minimization problems. A maximization problem can be transformed into a minimization problem and can be solved by the algorithm. The main modifications to the original BF are explained in the following lines:

1. **Initialization:** Save an HC whenever it is found without stopping the search and continuing to explore the tree.
2. **Initialization:** Compute the objective functions for every HC found and calculate the PS.
3. **Iteration:** Stop exploring a branch that will return an HC dominated by any solution already in the PS. A branch will be fathomed, if the $\boldsymbol{\delta^{\mathcal{U}}}$ is dominated by a $\boldsymbol{\delta^{HC}}$, $\mathcal{U}$ is the set of fixed arcs at the current stage. This check is performed in the fourth stage of the method (Iteration), after the second LP is computed.

Algorithm 9 shows the pseudocode of the MO-BF algorithm. It takes as an input an adjacency matrix and an empty set of fixed arcs ($\emptyset$). It returns the sets of HCs and the PS. Most of the functions from the algorithm were previously explained in Section 3.4. The function `Dominance_Current_Arcs` stops exploring a branch that will lead to a dominated HC. The pseudocode of the function is shown in Algorithm 10. In function `Update_Pareto_Set` the approximated PS is updated taking into account the set of all HCs.

The MO-BF can be executed as an anytime algorithm. An anytime algorithm is an algorithm whose quality of results enhances progressively as computation time increases. One of the main properties of these algorithms is interruptibility, as the algorithm can be stopped at any time and provide a solution [86]. If at least one HC has been found, the MO-BF can be understood as an anytime algorithm, as a PS approximation is returned at any time, and it improves or remains constant as computation time increases.

In the construction of the HCs, candidate arcs are selected according to the order of the labels of the vertices. Therefore, both the original and the MO-BF algorithms are sensitive to the way the graph nodes are labeled. Different label assignments to the vertices will generate different results. We create a variant of the MO-BF , a permutation-based MO-BF, by introducing a previous stage of generating t random permutations to use as different initial representations. Each permutation is used to sort the rows and columns of the original adjacency matrix in the order indicated by the permutation. Next, an MO-BF instance is initiated for each permutation. This variant of the MO-BF can be naturally implemented as a parallel algorithm, where the t instances of the MO-BF algorithm start at the same time, and the output of the individual algorithms can contribute a solution to the global PS approximation. This is similar to the variant presented in Section 7.4. The

Algorithm 9 Multi-objective-Branch-and-Fix

Input: adjacency_matrix, $\emptyset$
Global variables: All_HCs, PS
Output: All_HCs, PS

function MO_BF(adjacency_matrix, $\mathcal{U}$,Max_Time)
  while time < Max_Time **do**
    status, $\boldsymbol{x_2}$, $F(\boldsymbol{x_2}) \leftarrow$ Second_LP(adjacency_matrix, $\mathcal{U}$)
    if status $\neq$ feasible or $F(\boldsymbol{x_2})>$ bound **then return** False,$\emptyset$
    dominated $\leftarrow$ Dominance_Current_Arcs($\mathcal{U}$)
    if dominated=true **then return** False, $\emptyset$
    status, $\boldsymbol{x}$, $F(\boldsymbol{x}) \leftarrow$ First_LP(adjacency_matrix, $\mathcal{U}$)
    if status $\neq$ feasible **then return** False,$\emptyset$
    splitting_node $\leftarrow$ Identify_Splitting_Node($\boldsymbol{x}$)
    if splitting node = $\emptyset$ **then**
      HC=Identify_HC($\boldsymbol{x}$)
      All_HCs=HC $\cup$ All_HCs
      Update_Pareto_Set($\boldsymbol{\delta^{HC}}$) **return** True, $\emptyset$
    else
      $d \leftarrow |\mathcal{A}$(splitting_node)|
      $i \leftarrow 0$
      while $i < d$ **do**
        fixed_arc $\leftarrow$ Get_ith_Outgoing_Arc(splitting_node,i)
        new_adjacency_matrix $\leftarrow$ Update_Adjacency_Matrix(fixed_arc)
        $\mathcal{U} \leftarrow$ Update_Fixed_Arcs(new_adjacency_matrix)
        last_fixed_arcs $\leftarrow$ Refined_Fixed_Arcs($\mathcal{U}$)
        while |last_fixed_arcs|> 0 **do**
          for arc $\in$ last_fixed_arcs **do**
            new_adjacency_matrix $\leftarrow$Update_Adjacency_Matrix(arc)
          $\mathcal{U} \leftarrow$ Update_Fixed_Arcs(new_adjacency_matrix)
          last_fixed_arcs $\leftarrow$ Refined_Fixed_Arcs($\mathcal{U}$)
        found, HC $\leftarrow$ MO_BF(new_adjacency_matrix, $\mathcal{U}$)
        $i \leftarrow i + 1$
      return False, $\emptyset$
All_HCs $\leftarrow \emptyset$
PS $\leftarrow \emptyset$
MO_BF(adjacency_matrix, $\mathcal{U}$, Max_Time)

MO-BF is a heuristic method as can be stopped at anytime and can be used with different initiations. In this chapter, we consider the MO-BF from the enhanced variant of the BF including subcycle detection and degree-based simplification, introduced in Chapter 7.

Algorithm 10 Dominance Current Arcs

Input: $\mathcal{U}$
Global variables: All_HCs, PS
Output: dominated

```
1: function DOMINANCE_CURRENT_ARCS(U)
2:     dominated ← false
3:     if U ≠ ∅ & All_HCs ≠ ∅ then
4:         for δ^HC ∈ PS do
5:             if δ_1^U ≥ δ_1^HC & δ_2^U ≥ δ_2^HC & ... & δ_m^U ≥ δ_m^HC then
6:                 dominated ← true
       return dominated
```

10.3 Experiments

The main goal of our experiments is to evaluate MO-BF's ability to solve the MO HCP problem in graphs of a different number of vertices and densities. To this end, we compare its results to those of an MO-GA. We use the MO-GA as a baseline method because EAs are the most commonly used heuristics to solve the MO combinatorial optimization problems [136]. A second goal is to evaluate the influence of the graphs' characteristics on the MO-BF performance. This section presents the benchmark instances used in the experiments, gives details on the MO-GA, and explains the metrics used to evaluate the proposed approach, as well as presents the obtained results.

10.3.1 Benchmark definition

In this chapter, we consider the 500 random Hamiltonian graphs introduced in Section 7.5.1 and the first 513 graphs of the *challenge set*, containing less than 3000 vertices. In addition, the MO problem that we consider is bi-objective. For that reason, for each graph in each set, we create two weight-matrices, $\boldsymbol{W}^1$ and $\boldsymbol{W}^2$, of dimension $N \times N$. The entries of the matrix that correspond to an arc that it is not in the graph, they will take value 0. At the time of generating the instances, we will consider a slight negative correlation between the values of both matrices to guarantee that there will be some elements in the PS.

- $\boldsymbol{W}^1$: Weights (w_{ij}^1) are sampled from a normal distribution and multiplied by a constant c. We use $\mu = 10$ and $\sigma = 3$ for the normal distribution and $c = 10$.
- $\boldsymbol{W}^2$: Weights (w_{ij}^2) are created using the following linear function $w_{ij}^2 = \alpha \times w_{ij}^1 + |\epsilon|$. The parameter ϵ is sampled from a normal distribution of $\mu = 0$ and $\sigma = 100$ and $\alpha = 0.01$.

10.3.2 Evaluation of the performance of the MO-BF

EAs are the most commonly used heuristics to solve the MO TSP [136], and, for that reason, we use an MO-GA that incorporates genetic operators specifically suited to deal with permutation-based MO problems as a baseline method of comparison to evaluate the proposed MO-BF.

10.3.2.1 Description of the MO-GA approach

An initial population of 500 individuals is used in the solution space defined by permutations (Equation (10.3)). The MO-GA uses an ordered crossover, a shuffle operator as a mutation, and a NSGA-II selection to select the solutions that are recombined to generate the new population. The employed termination criterion is the maximum execution time; this criterion creates equal conditions for the two approaches, thus facilitating comparison.

$$\Omega = \{(i_1, i_2, ..., i_N) \mid k, l \in \{1, 2, ..., N\} \text{ and } i_k \neq i_l \ \forall \ k \neq l\} \tag{10.3}$$

Regarding the fitness function that evaluates the quality of the HC candidates, we should take into account the constraints that feasible HCP solutions should fulfill. Therefore, we propose a penalty approach.

A strong restriction of the HCP problem is that not every permutation is valid as an individual, as it must be an HC in the given graph. In a valid permutation, the pair of vertices (σ_{i-1}, σ_i) and (σ_N, σ_1) are connected in the graph $i \in \{2, ..., N\}$. We address this by penalizing in the fitness function those solutions that do not satisfy the restriction. The fitness function is shown in Equation (10.4).

$$fitness(\sigma) = \begin{cases} \boldsymbol{\delta}(\sigma) \text{ if } \sigma \text{ is a HC in } G, \\ |\mathcal{A}_\sigma| \times c + \boldsymbol{\delta}(\sigma) \text{ otherwise.} \end{cases} \tag{10.4}$$

where σ is an individual, $|\mathcal{A}_\sigma|$ is the number of arcs in σ that are not in G and c is a penalty parameter $c = 1000$. $\boldsymbol{\delta}(\sigma)$ is the multiple-objective function associated to the permutation. The objective function associated to a permutation given a distance matrix is computed as in Equation (4.1).

As can be observed in Equation (10.4), the fitness function promotes feasible solutions of the HCP, and then optimizes the total value of the weights. Note that we implement the described MO-GA using DEAP library [59] implemented in Python programming language. More details about the MO-GA were given in Section 6.3.3 and Algorithm 2.

10.3.2.2 Metrics for the comparison of the approaches

The corresponding PS approximations obtained by the MO-BF and MO-GA are compared by hypervolume (HV), a metric of m-dimensional volume of the region in the objective space enclosed by the solutions in the PS approximation and a dominated reference point [132]. Also, the number of UIs is employed, which was introduced in Section 7.5.2. A set of non-dominated solutions with higher HV can be considered as a better set of solutions. The PS approximations obtained by the MO-BF and the MO-GA are denoted as PS_{BF} and PS_{GA}, respectively. The reference point for each of the graphs is determined as $(r_1, r_2) = (max(\delta_1^{HC}), max(\delta_2^{HC}) + 1)$ for every HC belonging to PS_{BF} and PS_{GA}.

Each independent run of the MO-GA and MO-BF algorithms is allowed a maximum running time of 24 hours. For the MO-BF, we save partial results (non-dominated solutions) every 4 hours, so we have information for six time periods. The partial results are used to evaluate the performance of the algorithm through time. The reference point is determined as $(r_1, r_2) = (max(\delta_1(\boldsymbol{x})),\ max(\delta_2(\boldsymbol{x})) + 1)$, where $\boldsymbol{x} \in \bigcup_{T=1}^{6} PS_T$. PS_T is the PS approximation obtained in the T-th time period.

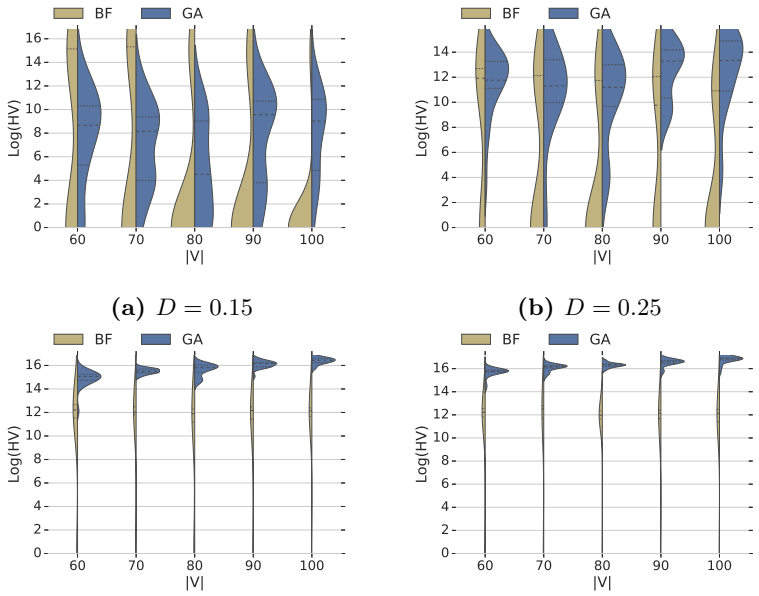
10.3.3 Results and discussion

This section presents the results of the experiments carried out with the mentioned benchmark of instances. The first part explains the results for random graphs, and the second part provides the results for the graphs in the *challenge set*.

10.3.3.1 Random graphs

The proposed MO-BF and the MO-GA are applied to the 500 random Hamiltonian graphs. Figure 10.2 shows the distributions of the HV values of the PS approximations obtained from graphs with different numbers of vertices and different densities. The logarithmic function is applied to the HV values. As can be observed, the differences between the distributions are more related to the density of the graphs than to the number of vertices. When $D = 0.15$ (Figure 10.2a), higher values of the HV are achieved for the MO-BF than the MO-GA. This pattern is consistent for all numbers of vertices, but the difference is more notable in the graphs with fewer vertices (60 and 70 vertices). When $D = 0.25$ (Figure 10.2b), both algorithms exhibit a similar performance, but when $D = 0.44$ (Figure 10.2c) and $D = 0.64$ (Figure 10.2d) the MO-GA outperforms the MO-BF .

To clarify the results, Table 10.1 shows the number of UIs and the mean HV value for the 25 instances of the graphs with different numbers of vertices and different densities. An instance is considered unsolved if no HC is found



(a) $D = 0.15$ **(b)** $D = 0.25$

(c) $D = 0.44$ **(d)** $D = 0.64$

Fig. 10.2: Comparison of the HVs obtained for PS_{BF} (in brown) and PS_{GA} (in blue) for the random graphs with different numbers of vertices and different densities.

by the algorithm. As can be observed, the number of UIs increases when D decreases. A total of 152 and 179 instances out of 500 instances are not solved for the MO-GA and the MO-BF, respectively.

To determine whether there are significant differences between the approaches in the number of UIs, we apply a statistical test, the Wilcoxon signed-rank test, with the significance threshold of $p = 0.05$.

Table 10.2 shows the p-values obtained when the statistical test is applied to the random graphs for both methods. For the p-values lower than 0.05, we find significant differences between MO-BF and MO-GA distributions. When $D = 0.15$, there are significant differences for all numbers of vertices with the exception of 100. When $D = 0.25$, there are significant differences for 80 and 100 vertices. For the highest densities 0.44 and 0.64, significant differences are only found in graphs with 80 and 70 vertices, respectively. It can be concluded that significant differences appear between the MO-BF and the MO-GA when $D = 0.15$, with the MO-BF performing better (indicated by the $*$ symbol). For the remainder of the densities, differences appear as the number of vertices increases, this time in favor of the MO-GA (indicated by the $+$ symbol). In graphs with higher density, it is more probable to find an

Table 10.1: The number of UIs and mean HV for MO-BF and MO-GA for each graph with different numbers of vertices and different densities.

		UI		Mean HV	
D	$\|V\|$	GA	BF	GA	BF
0.15	60	24	13	84007	**2282395**
	70	25	15	26354	**3487455**
	80	25	15	8746	**742745**
	90	25	19	**50015**	**50015**
	100	25	20	56903	**266940**
0.25	60	5	9	384084	**487353**
	70	9	14	**495350**	145442
	80	5	14	**376878**	100473
	90	7	12	**1242895**	175742
	100	2	15	**2013127**	45442
0.44	60	0	4	**3464489**	241593
	70	0	3	**5566105**	231729
	80	0	3	**6668915**	161899
	90	0	3	**10132446**	191363
	100	0	3	**13537552**	195399
0.64	60	0	2	**6889402**	240116
	70	0	5	**10006568**	243567
	80	0	5	**12155360**	187568
	90	0	3	**15564132**	181360
	100	0	2	**20326602**	186357

HC, hence they are easier problems to solve. For these problems, the MO-BF is not the recommended approach.

Table 10.2: The p-values obtained from the Wilcoxon signed-rank test applied to the graphs with different numbers of vertices and different densities. The p-values lower than 0.05 are indicated in bold. The symbols $*$ and $+$ indicate that the differences are in favor of MO-BF and MO-GA , respectively.

D	$\|V\|$	p-value	D	$\|V\|$	p-value
0.15	60	**0.00***	0.44	60	0.05
	70	**0.00***		70	0.8
	80	**0.03***		80	**0.01**$^+$
	90	**0.03***		90	0.08
	100	0.16		100	0.08
0.25	60	0.21	0.64	60	0.16
	70	0.13		70	**0.03**$^+$
	80	**0.00**$^+$		80	0.3
	90	0.17		90	0.08
	100	**0.01**$^+$		100	0.08

10.3.3.2 Graphs from the challenge set

At first, the proposed MO-BF and the MO-GA are applied to 10 graphs from the *challenge set* using five runs of the permutation-based MO-BF with $t = 30$, for a total of 150 executions. For all the runs, if at least one of the 30 parallel executions finds an HC, the performance is considered a 100% success. To carry out a fair comparison, 150 executions are also considered for the MO-GA.

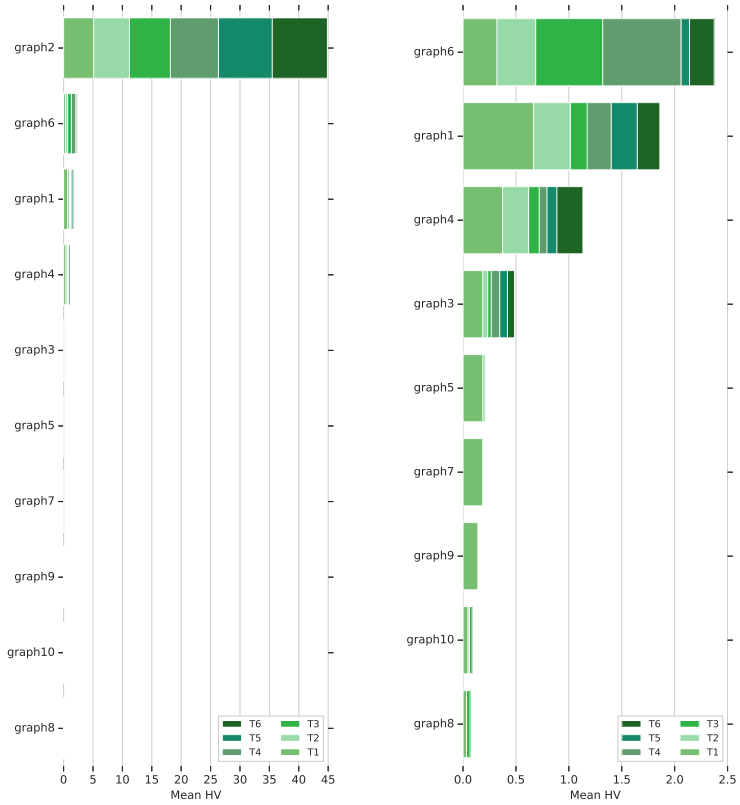
The permutation-based MO-BF achieves a 100% success rate for 8 of the 10 graphs (1,2,3,4,5,6,7,9), 80% for one graph (10) and 40% for one graph (8). From this analysis, we conclude the MO-BF has more difficulty finding HCs when the densities decrease. However, it clearly outperforms the MO-GA, which is not able to find any HC in 150 executions.

For further analysis, we consider 30 permutations of the five MO-BF runs independently as 150 different representations (labelings) of the same graph. Figure 10.3 shows the evolution of the HV mean value for the 150 representations of the graphs for the MO-BF throughout the six time periods.

As Figure 10.3a shows, the highest HV values are achieved for Graph 2. The barplot reveals that the mean HV increases as time goes by, as the number of HCs increases, thus enhancing the quality of the PS approximations. Figure 10.3b zooms in on Figure 10.3a to highlight the details of all the graphs, with the exception of Graph 2. The HV mean value also increases as time goes by for Graphs 1, 3, 4 and 6 (in all time periods) and for 5, 8 and 10 (in some time periods). The low values appear because, for some representations, the algorithm is not able to find any HC, and, for that reason, there are no elements in the corresponding PS approximations. For Graphs 7 and 9, the mean HV value remains constant; thus, there is no evolution throughout the six time periods. This means the algorithm is able to find one or more HCs in the first four hours, and no other solution improves the quality of the PS approximation in the remaining time.

Figure 10.4 shows the distributions of the HV values for the 150 representations of each of the graphs obtained for the MO-BF and the MO-GA. To represent these values in a figure, we apply a logarithmic function to the obtained HVs. As previously mentioned, for the MO-GA, since no HC is found for any of the 150 representations of the graphs, the obtained HV values are significantly lower than those obtained by the MO-BF. The results reveal that the MO-BF outperforms the MO-GA; the difference is clear in Graphs 1 and 2. It should be noted that when the MO-BF does not find any HC, the HV cannot be computed, whereas in the case of the MO-GA, there are always elements in the PS approximation whether there are HCs or not.

For the remaining 503 graphs from the *challenge set*, the MO-BF (with only one initialization) and the MO-GA were executed. The MO-GA has not found any HC in any of the graphs. In the case of the MO-BF , 276 graphs



(a) All the graphs **(b)** Zoom in on Figure 10.3a

Fig. 10.3: Evolution of the mean value of the HV of the 150 representations of the graphs for the six time periods for MO-BF.

from 503 were solved. Note that 17 executions were not terminated due to the fact that the program exceeded the allocated RAM memory (90GB), thus 486 graphs were executed for 24 hours.

Figure 10.5 shows the mean number of HCs found by the MO-BF in the six time periods for the mentioned 486 graphs. It can be observed that in the first time period (T1), which corresponds to the first six hours of execution, the lowest mean value is obtained. This arises from the fact that in the first six hours for many instances there are no HCs found. As time goes by, the number of HCs increases. Note, that in this case we did not execute the permutation-

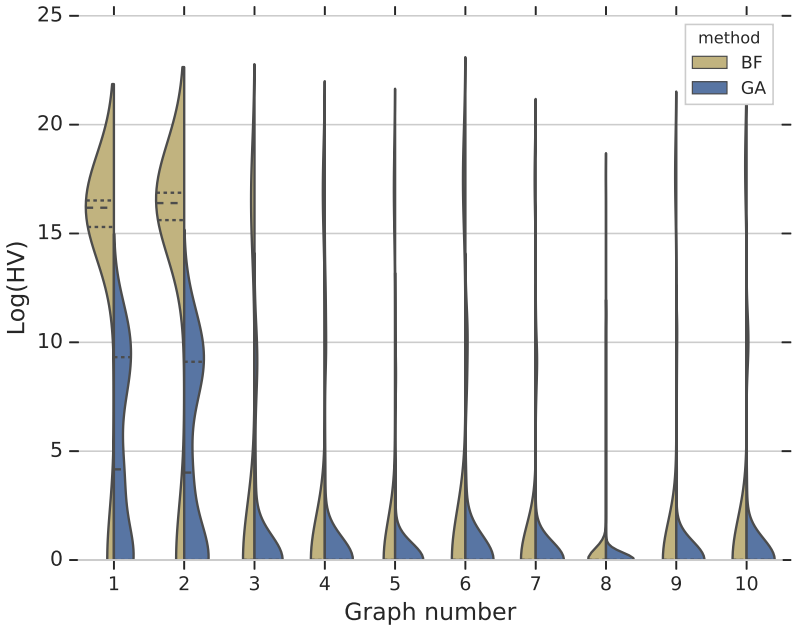


Fig. 10.4: Comparison of the HVs obtained for PS_{BF} (in brown) and PS_{GA} (in blue) for the graphs in the *challenge set*.

based MO-BF, this is because of the high number of experiments required to solve the 503 graphs. This variant of the algorithm would have increased the success rate of the experiments.

10.4 Conclusions

In this section, we introduced the MO-BF, an algorithm to address the MO HCP. To the best of our knowledge, there was not a previously proposed method to solve MO variants of the HCP. We built it on the BF algorithm originally proposed for HCP. It should be noted that in previous works [51, 19], the BF had not been tested in a large benchmark of graphs and the maximum number of vertices considered in the trials was of 64. In this chapter, the MO-BF was applied to graphs with up to 3000 vertices. In addition, we introduced a variant of the algorithm (permutation-based MO-BF) to use with graphs of low density and a high number of vertices. The proposed approach allows us to simultaneously find HCs and minimize the weights related to the arcs in the HC. As in MO optimization with conflicting objectives, there is no

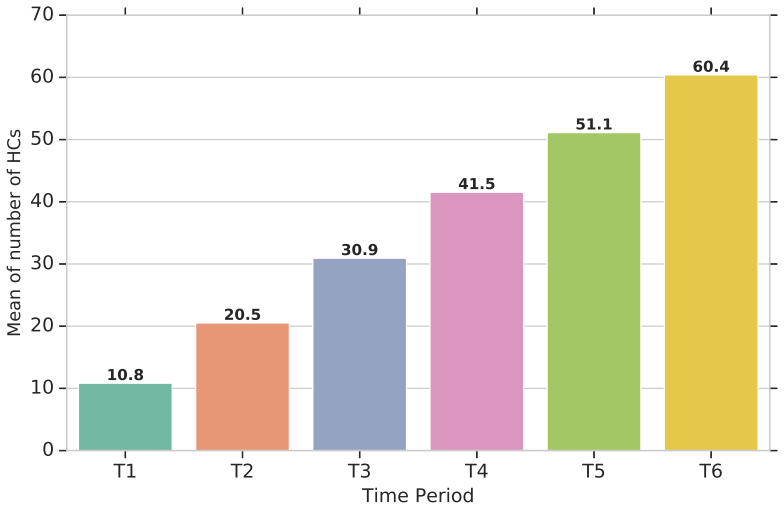


Fig. 10.5: Evolution of the mean number of HCs in the six time periods for the 486 graphs from the *challenge set*.

global optimal solution; the PS approximation was computed for each graph, comprising HCs that minimize the sum of weights for the different matrices.

To evaluate the performance of the MO-BF in graphs with a different number of nodes and densities, we compared it to an MO-GA that uses NSGA-II selection. The applied genetic operators in the design of the MO-GA were reported in the literature to be successful, but we introduced a specific fitness function that considers the constraints of the HCP.

Experiments showed that the density of the graphs was critical to the problem difficulty and the MO-BF performance. The MO-BF had more difficulty in solving the random graphs with lower density than those with higher density. The results also indicated that the number of the graph's vertices influences the effectiveness of the algorithm, but to a lesser extent than the density. Clearly, the MO-BF outperformed the MO-GA for lower densities, whereas for high densities, it is the other way around. The difficulties that the BF has to solve relatively easy graphs of high density is worth noting. One possible explanation is that it might spend a long time exploring a wrong branch (a branch that does not lead to an HC), whereas the MO-GA is able to find the HCs faster.

The effect of the density was reasserted in experiments with the graphs from the *challenge set*. These graphs represent a challenging set of Hamiltonian graphs, and the lower densities associated with them make the instances less like the TSP. It is noteworthy that 513 graphs were used in these ex-

periments with a maximum number of 3000 vertices, where the MO-BF was better than the MO-GA.

The MO-BF can simultaneously find several HCs and continuously compute a remarkably improving PS approximation when it is used as an anytime algorithm. The results show that the obtained PS approximations are enhanced as time goes by, as the HV values increase or remain constant. Our algorithm can be an efficient alternative for the solution of hard MO HCP arising in many real-world problems related with Production and Energy Industries, Finance or Agriculture and Forestry.

Part IV

Conclusions and future work

11

Conclusions and future work

This chapter condenses the main conclusions obtained throughout this dissertation. It suggests lines for future research that emerged from the dissertation and concludes by enumerating the scientific publications and submissions in the last section.

11.1 Conclusions

This dissertation has made some methodological contributions to solve the HCP based on the previously proposed BF method. The contributions are related to two main issues in linear programming based methods: 1) reducing the number of constraints of optimization problems; 2) fathoming solution spaces that lead to non-HC solutions. In particular, it has proposed two enhancements and a new branching method. It has also proposed a new algorithm, called BF collapse algorithm, taking into account the enhancements, the branching method and a matrix collapsing step. Finally, it has introduced a new problem, MO HCP, together with an extension of the BF to solve it. The starting point for these contributions was the optimization of the tool-path in an AM process.

The first contribution has been a mathematical framework to model a specific AM technology, based on an extensive literature review. Once the mathematical framework has been proposed, a new problem called sequence strategy generation has been introduced. We have concluded that it is possible to address the sequence strategy generation problem with the proposed mathematical framework. In addition, when the problem involves parts with different geometry and different numbers of beads, it can be solved with an MO-GA approach.

In this context, the dissertation has reviewed the most frequently reported strategies used to solve the HCP. We have found that most of them are focused on undirected graphs. The few that focus on directed graphs have not

been fully implemented or tested in large size graphs. One of the methods reported in the literature that can be applied to directed graphs is the BF method. In Chapter 7, we have presented two enhancements. The first has addressed a limitation in the BF when fixing arcs, and the second has exploited a characteristic that graphs may present. The first enhancement consists of detecting subcycles that can be generated when adding a fixed arc to the set of fixed arcs. This phenomenon has been proven mathematically in the chapter. We have introduced a conjecture related to this issue to detect the generated subcycles, and the conducted experiments have shown that our approach outperforms the BF. The second enhancement is a degree-based simplification step that eliminates arcs from the graph that are infeasible to be in an HC. We have found that a simplification can result in a reduction of a substantial number of arcs, and in some graphs, it makes a difference compared to the BF. In the experiments, we have used graphs with more than 192 vertices, tripling the number of vertices considered in previous investigations. In addition, we have proposed a permutation-based BF with the clear advantage of parallelizing the BF naturally.

One of the main characteristics of the BF is that it belongs to the family of branching algorithms. In Chapter 8, after analyzing the branching rules proposed in the literature, we have proposed the global branching method where the nodes in the top levels have more importance, prioritizing the arcs that will lead to a simplified subgraph of the original graph. Several experiments we have conducted to compare the methods in two sets of Hamiltonian graphs showed that adding the specific characteristic of the BF algorithm to the branching rule does not give any clear advantage over the methods reported in the literature, in terms of the number of UIs and number of calls to the algorithm. In addition, several issues can affect the branching strategies: 1) the interaction between components of the BF and the branching; 2) particular characteristics of a graph; 3) the effect of the permutation used to rearrange the matrix.

These contributions, together with a matrix collapsing step, have led to the BF collapse algorithm. In Chapter 9, we have proposed four components to address the limitations of previously reported linear programming based methods: 1) degree-based simplification; 2) more efficient matrix representation; 3) the global branching method; 4) matrix collapsing step by fixing arcs. In our experiments, we have observed that the proposed components have a positive effect in the execution time in comparison to the original BF, especially the matrix collapsing step. Moreover, when compared to the time required for the Concorde TSP solver, and considering the characteristics of the implementation, the results could be considered competitive in terms of time.

Since most of the real-world problems are multi-objective optimization problems, in Chapter 10 we have proposed the MO HCP. We have also introduced the MO-BF algorithm and built on the BF to address the MO HCP.

We have compared the performance of the algorithm to an MO-GA algorithm in two sets of Hamiltonian graphs. For the first set of random Hamiltonian graphs, the MO-BF had more difficulty solving the random graphs with lower density than those with higher density. The MO-BF outperformed the MO-GA for lower densities, but for higher densities, it was the other way around. In the second set, where more challenging graphs are included, the MO-BF was better than the MO-GA. We have used graphs with up to 3000 vertices in the experiments.

In the following lines, we answer the RQs in Section 5.1. With respect to **RQ1**, Chapter 7 has shown it was possible to improve the efficiency of the BF by adding an early step of subcycle detection. By incorporating this modification, the BF was able to solve more instances and was faster than the original BF. Related to the efficiency of the algorithm, Chapter 9 has reported that the BF collapse method, especially the matrix collapsing component, solved more instances and was faster than the original BF. Chapter 7 has shown that the BF together with the subcycle detection step and degree-based simplification step was able to solve challenging instances with more than 192 vertices. It also solved two specific instances of 400 and 1123 vertices. With respect to **RQ2**, Chapter 8 has shown no branching method outperformed the others. It has compared the five methods reported in the literature and the one proposed in this dissertation. With respect to **RQ3**, Chapter 10 has shown that it was possible to extend the BF into an MO HCP scenario. Our approach outperformed the MO-GA in random graphs of lower densities, but for higher densities it was the other way around. In a benchmark of challenging graphs, our approach clearly outperformed the MO-GA.

To summarize, the dissertation has started with the tool-path problem for a specific AM technology, making a contribution by proposing a mathematical framework and a novel problem. Then, it has made several methodological contributions to the HCP: enhancements to the BF method, a new branching method that takes into account specific BF characteristics, a new BF collapse algorithm to solve the HCP and an extension to the BF to solve the MO HCP.

11.2 Future work

This section presents the future research lines derived from this dissertation.

- As discussed in Chapter 6, there is no automatic way to link a CAD model to the generation of welding paths in WAAM. This requires the automatic generation of the PDGs from a CAD and posterior automatic generation of the manufacturing scheme. In addition, the proposed framework can be extended to other DED technologies. The contributions performed in this chapter, can lead to a way to generate CAM packages to AM technologies. It would be interesting to use a preference-based EA to solve the

sequence strategy generation problem, thus allowing the decision-maker to give preferences in terms of optimality.

- Chapter 8 has observed that the proposed branching method did not improve the previously reported branching methods. There are a number of issues that can affect the branching strategies that were not considered in this analysis. An interesting future research line would be to study the effect that the mentioned issues have (other components of the BF, the characteristics of the graphs and the effect of the permutation used to rearrange the matrix) in the branching strategies, especially in the global branching method. More sophisticated methods can be created that make use of learning to branch strategies. For instance, there are models, such as the Mallow's model, that can be used to learn the distribution of permutations. This could allow a specific permutation to be used to follow a branching order for a given graph G. This permutation would be sampled from a model learnt from other permutations with graphs that share some characteristics with G.
- Chapter 9 has introduced a new linear programming based algorithm that builds on previous BF. The experiments carried out show promising results; however, the algorithm could be tested in challenging larger graphs. An interesting research line could be to apply the BF collapse algorithm to larger graphs and to compare it to other methods reported in the literature, such as the Concorde solver or the snakes and ladders heuristic.
- Eshragh et al. [53] improved the polytope $\boldsymbol{X}_\beta$ used in this investigation and proposed a hybrid approach to solve the HCP, called Cross-entropy/optimization hybrid approach. Then, Eshragh and Filar [52] and Eshragh et al. [54] analyzed different aspects related to the extreme points of the polytope proposed by Eshragh et al. [53]. This dissertation reveals the need to go deeper into the construction of new algorithmic approaches that exploit the embedding of the HCP in an MDP and to compare their performance to the proposed BF collapse algorithm. Those approaches could use as a basis the polytope defined by Eshragh et al. [53] and exploit the characteristics discovered for the extreme points.
- Chapter 10 has concluded that the MO-BF outperforms the MO-GA in challenging graphs. Nevertheless, there are specific methods reported in the literature that deal with the sparse TSP instead of considering the TSP in a complete graph. Our method could be compared to a specific method for sparse TSP extended into a MO scenario. The results presented in the dissertation could raise the interest in the MO HCP and lead to methods specifically tailored for MO on sparse graphs.
- The BF could be combined with a metaheuristic method such as a GA. This hybrid approach could be generated in two different ways: 1) first apply the GA and then the BF; 2) first apply the BF and then the GA. In the case of variant 1), the GA can be used to generate a population

of partial HCs. A partial HC is a sequence of arcs that may include arcs that do not exist in the graph. Taking from the sequence only the arcs belonging to the graph, the BF can be applied from a deeper level of the tree using those arcs as an initial set of $\mathcal{U}$. The permutation-based BF can be computed for each individual (partial HC) of the population. In the case of variant 2), the permutation-based BF can be computed by a certain time, and the set of $\mathcal{U}$s obtained in each run can be used as the individuals of the initial population of the GA.

11.3 Publications

The research work carried out during this period has associated the following publications and submissions:

11.3.1 Publications and submissions derived from this dissertation

- **M. Murua**, A. Suárez, D. Galar & R. Santana (2020). Tool-path problem in direct energy deposition metal-additive manufacturing: sequence strategy generation. *IEEE Access*, 8: 91574-91585.
- **M. Murua**, D. Galar & R. Santana (2020). Adaptation of a Branching Algorithm to Solve the Multi-Objective Hamiltonian Cycle Problem. *Operations Research Proceedings 2019*. Springer. Pp. 231-237.
- **M. Murua**, D. Galar & R. Santana (2021). Solving the multi-objective Hamiltonian cycle problem using a Branch-and-Fix based algorithm. *Journal of Computational Science*. Accepted for publication.
- **M. Murua**, D. Galar & R. Santana (2021). An enhanced Branch-and-Fix method to Hamiltonian cycle problem on directed graphs. *Operations Research Letters*. Submitted.
- **M. Murua**, D. Galar & R. Santana (2022). A Branch-and-Fix collapse algorithm for solving the Hamiltonian cycle problem in directed graphs. *European Journal of Operational Research*. In preparation.

11.3.2 Publications not directly related to this dissertation

- **M. Murua**, A. Suárez, L.N. López de Lacalle, R. Santana & A. Wretland (2018). Feature extraction-based prediction of tool wear of Inconel 718 in face turning. *Insight-Non-Destructive Testing and Condition Monitoring*, 8: 443-450.

11.3.3 Conference and workshop communications

- **M. Murua**, R. Santana, D. Galar, A Suárez (2017). Tool routing problem based on the Hamiltonian cycle problem for Wire Arc Additive Manufacturing. Poster presented at: *2nd Bilbao Data Science Workshop, Bilbao, Spain, 16-17 November.*

- **M. Murua**, D. Galar, R. Santana (2019). Adaptation of a Branching Algorithm to Solve Discrete Optimization Problems. *Operations Research 2019 (OR2019), Dresden, Germany, 3-6 September.*

Part V

Appendices

A

FindHC: A Python package to solve the HCP

The different algorithms implemented throughout this dissertation have been included in a software repository called FindHC. This repository is publicly available at `https://github.com/maialenmurua/FindHC` which is written using Python programming language [144]. A large part of the code was inspired by the BF algorithm. We used an implementation proposed by Ejov et al. [51] and we were provided a Matlab code by the authors. In the following lines some details of the code are given.

- Auxiliary functions: `aux_bf.py` and `aux_bf_mo.py`.
- Branching methods: `aux_branching6.py` and `branching_methods.py`. The five branching methods reported in [19] (page 129) have been implemented as well as the global branching method shown in Algorithm 7.
- Solvers: two solvers are available CPLEX (`cplex_solver.py`) and Gurobi (`gurobi.py`).
- Linear programs: `constraints_class.py` and `linear_programs.py`.
- Structure of the graph: `graph_class.py`.
- Degree-based simplification: `degree_based_simp.py`. This was explained in a deeper extent in Section 7.3 and Algorithm 4.
- Enhanced BF: `BF_classes_branching.py`. This was explained in further detail in Section 7.3.2 and Algorithm 5.
- Collapse BF: `simplification.py` and `BF_classes_collapse.py`. This was developed in Section 9.2 and Algorithm 8.
- MO-BF: `BF_multi_objective_HCP.py` and `pareto_set.py`. This was explained in Chapter 10 and Algorithm 9.
- Main functions: `main.py` and `main_mo.py`.

Table A.1 shows the functionalities that are available for the BF and MO-BF. For both cases, it is possible to use a permutation-based variant by indicating the name of the file with the permutation. The six branching methods are available for BF and MO-BF, as well as the degree-based simplification. The solvers of CPLEX [31] and Gurobi [73] are available for both cases. The

	Permutation	Branch	Simp	Solver	Collapse	Max. time
BF	✓	{1,2,3,4,5,6}	✓	CPLEX/Gurobi	✓	
MO-BF	✓	{1,2,3,4,5,6}	✓	CPLEX/Gurobi		✓

Table A.1: Functionalities that are available for BF and MO-BF.

matrix collapsing can be activated for the BF, and in the case of MO-BF, a maximum time can be given. In this last case, partial results of the PS are returned divided in six time periods, so that every max.time/6.

A.1 Installation

This program has been tested in the operating system Ubuntu 20.04.3. The first step consists of downloading the code by cloning the repository.

```
git clone https://github.com/maialenmurua/FindHC
cd FindHC
```

In order to install the dependencies, the Anaconda distribution should be installed which is available at `https://www.anaconda.com/products/individual`.

```
bash ~/Downloads/Anaconda3-[edition]-Linux-x86_64.sh
```

Create a conda environment for the software. Note, that the Python programming language version must be compatible with the solver to be installed.

```
conda create -n FindHC python=3.5 anaconda
conda activate FindHC
```

Once in the folder where the software was downloaded, and after creating and activating the conda environment, install the dependencies.

```
pip install -r FindHC/requirements.txt
```

The last step is to install the `IBM ILOG CPLEX` solver. Our software also allows the Gurobi Optimizer as solver, however, in the following lines the instructions to install `IBM ILOG CPLEX` are given. It is available at: `https://www.ibm.com/support/pages/downloading-ibm-ilog-cplex-optimization-studio-v1290`.

```
chmod +x cplex_studio[edition].linux-x86-64.bin
./cplex_studio[edition].linux-x86-64.bin
```

The installation path in Ubuntu should be `/opt/ibm/ILOG/CPLEX_Studio[edition]`.

A.2 Usage

In the following lines, some examples of how to execute the software for different configurations is given. The adjacency matrix is given in a `.txt` file, as well as the permutation and the weight matrices. An example can be found in the repository.

Example 5. Execution of the BF without using a permutation, with the branching method 4, using the degree-based simplification and collapse and the CPLEX solver.

```
cd FindHC
python3 main.py graph1 No 4 Yes cplex Yes
```

It will generate a file `solution_graph1.txt` with the HC and the time required to find the HC.

Example 6. Execution of the BF with a permutation, with the branching method 6, using the degree-based simplification but not collapse and the Gurobi solver.

```
cd FindHC
python3 main.py graph2 perm_graph2 6 Yes gurobi No
```

It will generate a file `solution_graph2_perm_graph2.txt` with the HC and the time required to find the HC.

Example 7. Execution of the MO-BF without using a permutation, with the branching method 1, without using the degree-based simplification, the CPLEX solver and 1 hour execution. The files with weight matrices for the bi-objective functions must be indicated.

```
cd FindHC
python3 main_mo.py graph3 No 1 c1 c2 No cplex 3600
```

It will generate six files, one every ten minutes with the number of HCs found and the elements of the PS. For instance, for the first time period it will generate the file `partial_output_graph3_1.txt`.

References

[1] Ahsan, A., Habib, M., and Khoda, B. (2015). Resource based process planning for additive manufacturing. *Computer-Aided Design*, 69:112–125.

[2] AI-wswasi, M., Ivanov, A., and Makatsoris, H. (2018). A survey on smart automated computer-aided process planning (ACAPP) techniques. *The International Journal of Advanced Manufacturing Technology*, 97:809–832.

[3] Alvarez, A., Louveaux, Q., and Wehenkel, L. (2017). A machine learning-based approximation of strong branching. *INFORMS Journal of Computing*, 29:185–195.

[4] Antonysamy, A. (2012). *Microstructure, Texture and Mechanical Property Evolution during Additive Manufacturing of Ti6Al4V Alloy for Aerospace Applications*. PhD thesis, University of Manchester.

[5] Applegate, D., Bixby, R., Chvatal, R., and Cook, W. (2007). *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton Univeristy Press.

[6] Applegate, D., Bixby, R., Chvátal, V., and Cook, W. (2006). Concorde TSP solver. http://www.math.uwaterloo.ca/tsp/concorde.html. Accessed: 2021-05-05.

[7] Artaza, T., Alberdi, A., Murua, M., Gorrotxategi, J., Frías, J., Puertas, G., Melchor, M. A., Múgica, D., and Suárez, A. (2017). Design and integration of WAAM technology and in situ monitoring system in a gantry machine. *Procedia Manufacturing*, 13:778–785.

[8] Ayodele, M., McCall, J., Regnier-Coudert, O., and Bowie, L. (2017). A random key based estimation of distribution algorithm for the permutation flowshop scheduling problem. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 2364–2371.

[9] Back, T. (1996). *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford university press.

[10] Balcan, M.-F., Dick, T., Sandholm, T., and Vitercik, E. (2018). Learning to branch. In *Proceedings of the 35th International Conference on Machine Learning*, pages 344–353.

[11] Baniasadi, P., Ejov, V., Filar, J., Haythorpe, M., and Rossomakhine, S. (2014). Deterministic "Snakes and Ladders" heuristic for the Hamiltonian cycle problem. *Mathematical Programming Computation*, 6:55–75.

[12] Banzhaf, W., Nordin, P., Keller, R., and Francone, F. (1998). *Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann Publishers Inc.

[13] Bartels, R. and Golub, G. (1969). The simplex method of linear programming using LU decomposition. *Communications of the ACM*, 12:266–268.

[14] Baryannis, G., Validi, S., Dani, S., and Antoniou, G. (2019). Supply chain risk management and artificial intelligence: state of the art and future research directions. *International Journal of Production Research*, 57:2179–2202.

[15] Basu, S., Gajulapalli, R., and Ghosh, D. (2013). A fast tabu search implementation for large asymmetric traveling salesman problems defined on sparse graphs. *OPSEARCH*, 50:75–88.

[16] Behun, M., Gavurova, B., Tkacova, A., and Kotaskova, A. (2018). The impact of the manufacturing industry on the economic cycle of European Union countries. *Journal of Competitiveness*, 10:23–39.

[17] Beloborodov, D., Foerster, A., Whiteson, J., and Lvovsky, A. (2020). Reinforcement learning enhanced quantum-inpsired algorithm for combinatorial optimization. *Mach. Learn.: Sci. Technol*, 2:025009.

[18] Bock, S. and Klamroth, K. (2019). Combining traveling salesman and traveling repairman problems: a multi-objective approach based on multiple scenarios. *Computers and Operations Research*, 112:104766.

[19] Borkar, V., Ejov, V., Filar, J., and Nguyen, G. (2012). *Hamiltonian Cycle Problem and Markov Chains*. Spriger Science & Business Media.

[20] Bosman, P., Luong, N., and Thierens, D. (2016). Expanding from discrete Cartesian to permutation gene-pool optimal mixing evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 637–644.

[21] Boyd, S., Pulleyblank, W., and Cornuéjols, G. (1987). TRAVEL-An interactive travelling salesman problem package for IBM-personal computer. *Operations Research Letters*, 6:141–143.

[22] Brusco, M. and Stahl, S. (2005). *Branch-and-Bound Applications in Combinatorial Data Analysis*. New York: Springer.

[23] Carroll, B., Palmer, T., and Beese, A. (2015). Anisotropic tensile behavior of ti-6al-4v components fabricated with directed energy deposition additive manufacturing. *Acta Materialia*, 87:309–320.

[24] Castelino, K., D'Souza, R., and Wright, P. (2002). Toolpath optimization for minimizing airtime during machining. *Journal of Manufacturing Systems*, 22:173–180.

[25] Chan, G. and Na, S. (1999). A study on torch path planning in laser cutting processes part 2: cutting path optimization using simulated annealing. *Journal of Manufacturing Processes*, 1:62–70.

[26] Chen, X., Liu, Y., Li, X., Wang, Z., Wang, S., and Gao, C. (2019). A new evolutionary multiobjective model for traveling salesman problem. *IEEE Access*, 7:66964–66979.

[27] Chen, Y., Zhou, B., Zhang, M., and Chen, C. (2020). Using IoT technology for computer-integrated manufacturing systems in the semiconductor industry. *Applied Soft Computing*, 89:106065.

[28] Chih-Hsing, C. and Jang-Ting, C. (2006). Tool path planning for five-axis flank milling with developable surface approximation. *The International Journal of Advanced Manufacturing Technology*, 29:707.

[29] Cohen, N. and Coudert, D. (2017). Le défi des 1001 graphes. Technical report, Interstices, INRIA.

[30] Conner, B., Manogharan, G., Martof, A., Rodomsky, L., Rodomsky, C., Jordan, D., and Limperos, J. (2014). Making sense of 3-d printing: Creating a map of additive manufacturing products and services. *Additive Manufacturing*, 1:64–76.

[31] Cplex-IBM-ILOG (2009). V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157.

[32] Croce, F., Ghirardi, M., and Tadei, R. (2004). Recovering beam search: Enhancing the beam search approach for combinatorial optimization problems. *Journal of Heuristics*, 10:89–104.

[33] Davendra, D. (2010). *Traveling Salesman Problem: Theory and Applications*. BoD-Books on Demand.

[34] de Boer, P., Kroese, D., Mannor, S., and Rubinstein, R. (2005). A tutorial on the cross-entropy method. *Annals of Operations Research*, 134:19–67.

[35] de Gans, B., Duineveld, P., and Schubert, U. (2004). Inkjet printing of polymers: state of the art and future developments. *Advanced Materials*, 16:203–213.

[36] Deb, K. (2013). Multi-objective optimization. In *Search Methodologies*, pages 403–449.

[37] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6:182–197.

[38] Demsar, J. (2006). Statistical comparison of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30.

[39] Dewil, R., Luteyn, C., and Cattrysse, D. (2019). A critical review of multihole drilling path optimization. *Archives of Computational Methods in Engineering*, 26:449–459.

[40] Dewil, R., Vansteenwegen, P., and Cattrysse, D. (2016). A review of cutting path algorithms for laser cutters. *The International Journal of Advanced Manufacturing Technology*, 87:1865–1884.

[41] Dewil, R., Vansteenwegen, P., Cattrysse, D., Laguna, M., and Vossen, T. (2015). An improvement heuristic framework for the laser cutting tool path problem. *International Journal of Production Research*, 53:1761–1776.

[42] Dewilde, T., Cattrysse, D., Coene, S., Spieksma, F., and Vansteenwegen, P. (2013). Heuristics for the traveling repairman problem with profits. *Computers and Operations Research*, 40:1700–1707.

[43] Ding, D. (2015). *Process Planning for Robotic Wire Arc Additive Manufacturing*. PhD thesis, University of Wollongong.

[44] Ding, D., Chen, C., Pan, Z., Cuiuri, D., Li, H., and Larkin, N. (2016a). Towards an automated robotic arc-welding-based additive manufacturing system from CAD to finished part. *Computer-Aided Design*, 73:66–75.

[45] Ding, D., Pan, Z., Cuiuri, D., and Li, H. (2014). A tool-path generation strategy for wire and arc additive manufacturing. *International Journal of Advanced Manufacturing Technology*, 73:173–183.

[46] Ding, D., Pan, Z., Cuiuri, D., and Li, H. (2015). A practical path planning methodology for wire and arc additive manufacturing of thin-walled structures. *Robotics and Computer-Integrated Manufacturing*, 34:8–19.

[47] Ding, D., Pan, Z., Cuiuri, D., Li, H., Duin, S., and Larkin, N. (2016b). Bead modelling and implementation of adaptive MAT path in wire and arc additive manufacturing. *Robotics and Computer-Integrated Manufacturing*, 39:32–42.

[48] Ding, D., Pan, Z., Cuiuri, D., Li, H., and Larkin, N. (2016c). Adaptive path planning for wire-feed additive manufacturing using medial axis transformation. *Journal of Cleaner Production*, 133:942–952.

[49] Dowling, L., Kennedy, J., O'Shaughnessy, S., and Trimble, D. (2020). A review of critical repeatability and reproducibility issues in powder bed fusion. *Materials & Design*, 186:108346.

[50] Eiben, A. and Smith, J. (2003). *Introduction to Evolutionary Computing*. Berlin: Springer.

[51] Ejov, V., Filar, J., Haythorpe, M., and Nguyen, G. (2009). Refined MDP-based branch-and-fix algorithm for the Hamiltonian Cycle Problem. *Mathematics of Operations Research*, 34(3):758–768.

[52] Eshragh, A. and Filar, J. (2011). Hamiltonian cycles, random walks, and discounted occupational measures. *Mathematics of Operations Research*, 36:258–270.

[53] Eshragh, A., Filar, J., and Haythorpe, M. (2011). A hybrid simulation-optimization algorithm for the Hamiltonian cycle problem. *Annals of Operations Research*, 189:103–125.

[54] Eshragh, A., Filar, J., Kalinowski, T., and Mohammadian, S. (2020). Hamiltonian cycles and subsets of discounted occupational measures. *Mathematics of Operations Research*, 45:713–721.

[55] Fayzrakhmanov, R., Murzakaev, R., and Polyakov, A. (2018). Cutting time optimization using technology for CNC machines. In *Proceedings of the 6th International Conference on Applied Innovations in IT*, pages 38–44.

[56] Feinberg, E. (2000). Constrained discounted Markov decision processes and Hamiltonian cycles. *Mathematics of Operations Research*, 25(1):130–140.

[57] Filar, J., Gondzio, J., and Ejov, V. (2004). An interior point heuristic for the Hamiltonian cycle problem via Markov decision process. *Journal of Global Optimization*, 29:315–334.

[58] Filar, J. and Krass, D. (1994). Hamiltonian cycles and Markov chains. *Mathematics of Operations Research*, 19:223–237.

[59] Fortin, F.-A., Rainville, F.-M. D., Gardner, M.-A., Parizeau, M., and Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175.

[60] Gamarnik, D. and Sviridenko, M. (2005). Hamiltonian completions of sparse random graphs. *Discrete applied mathematics*, 152:139–158.

[61] Garcia, S. and Herrera, F. (2008). An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*, 9:2677–2694.

[62] Gardan, J. (2016). Additive manufacturing technologies: state of the art and trends. *International Journal of Production Research*, 54:3118–3132.

[63] Gass, S. (2003). *Linear Programming: Methods and Applications*. Courier Corporation.

[64] Gen, M., Lin, L., and Zhang, H. (2009). Evolutionary techniques for optimization problems in integrated manufacturing systems: state-of-the-art-survey. *Computers & Industrial Engineering*, 56:779–808.

[65] George, T. and Amudha, T. (2020). Genetic algorithm based multi-objective optimization framework to solve traveling salesman problem. *Advances in Computing and Intelligent Systems*, pages 141–151.

[66] Ghobakhloo, M. (2018). The future of manufacturing industry: a strategic roadmap toward Industry 4.0. *Journal of Manufacturing Technology Management*, 29:910–936.

[67] Ghobakhloo, M. (2020). Industry 4.0, digitization, and opportunities for sustainability. *Journal of Cleaner Production*, 252:119869.

[68] Gilpin, A. and Sandholm, T. (2011). Information-theoretic approaches to branching in search. *Discrete Optimization*, 8:147–159.

[69] Glankwamdee, W. and Linderoth, J. (2006). Lookahead branching for mixed integer programming. Technical report, Technical Report 06T-004, Lehigh University.

[70] Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley.

[71] Gong, H., Rafi, K., Gu, H., Starr, T., and Stucker, B. (2014). Analysis of defect generation in Ti-6Al-4V parts made using powder bed fusion additive manufacturing processes. *Additive Manufacturing*, 1-4:87–98.

[72] Gross, J., Yellen, J., and Zhang, P. (2013). *Handbook of Graph Theory*. CRC press.

[73] Gurobi Optimization, LLC (2021). Gurobi Optimizer Reference Manual.

[74] Gutin, G. and Punnen, A. (2006). *The Traveling Salesman Problem and its Variations*. Springer Science & Business Media.

[75] Hashemi, S., Babic, U., Hadikhani, P., and Psaltis, D. (2020). The potentials of additive manufacturing for mass production of electrochemical energy systems. *Current Opinion in Electrochemistry*, 20:54–59.

[76] Hatem, N., Yusof, Y., Kadir, A., Latif, K., and Mohammed, M. (2021). A novel integrating between tool path optimization using an ACO algorithm and interpreter for open architecture CNC system. *Expert Systems with Applications*, 178:114988.

[77] Haythorpe, H. (2010a). *Markov Chain Based Algorithms for the Hamiltonian Cycle Problem*. PhD thesis, University of South Australia.

[78] Haythorpe, M. (2010b). Finding Hamiltonian cycles using an interior point method. *The Australian Society Gazette*, 37:170–179.

[79] Haythorpe, M. (2018). FHCP challenge set: The first set of structurally difficult instances of the Hamiltonian cycle problem. *Bulletin of the ICA*, 83:98–107.

[80] Helsgaun, K. (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126:106–130.

[81] Helsgaun, K. (2009). General k-opt submoves for the Lin-Kernighan TSP heuristic. *Mathematical Programming Computation*, 1:119–163.

[82] Hooker, J. and Vinay, V. (1995). Branching rules for satisfiability. *Journal of Automated Reasoning*, 15:359–393.

[83] Huang, S., P.Liu, Mokasdar, A., and Hou, L. (2013). Additive manufacturing and its societal impact: a literature review. *The International Journal of Advanced Manufacturing Technology*, 67:1191–1203.

[84] Jäger, G. and Zhang, W. (2010). An effective algorithm for and phase transitions of the directed Hamiltonian cycle problem. *Journal of Artificial Intelligence Research*, 39:663–687.

[85] Jaszkiewicz, A. (2002). Genetic local search for multi-objective combinatorial optimization. *European Journal of Operational Research*, 137:50–71.

[86] Jesus, A., Paquete, L., and Liefooghe, A. (2021). A model of anytime algorithm performance for bi-objective optimization. *Journal of Global Optimization*, 79:329–350.

[87] Jiang, J., Xu, X., and Stringer, J. (2019). Optimization of process planning for reducing material waste in extrusion based additive manufacturing. *Robotics and Computer-Integrated Manufacturing*, 59:317–325.

[88] Jin, Y., He, Y., and Du, J. (2017). A novel path planning methodology for extrusion-based additive manufacturing of thin-walled parts. *International Journal of Computer Integrated Manufacturing*, 30:1301–1315.

[89] Jin, Y., He, Y., and Fu, J. (2013). An adaptive tool path generation for fused deposition modeling. *Advanced Materials Research*, 819:7–12.

[90] Jin, Y., He, Y., Fu, J., Gan, W., and Lin, Z. (2014). Optimization of tool-path generation for material extrusion-based additive manufacturing technology. *Additive Manufacturing*, 1-4:32–47.

[91] Jin, Y., He, Y., Xue, G., and Fu, J. (2015). A parallel-based path generation method for fused deposition modeling. *The International Journal of Advanced Manufacturing Technology*, 77:929–939.

[92] Jonker, R. and Volgenant, A. (1983). Transforming asymmetric into symmetric traveling salesman problems. *Operations Research Letters*, 2:161–163.

[93] Karunakaran, K., Suryakumar, S., Pushpa, V., and Akula, S. (2010). Low cost integration of additive and substractive processes for hybrid layered manufacturing. *Robotics and Computer-Integrated Manufacturing*, 26:490–499.

[94] Karzan, F. K., Nemhauser, G. L., and Savelsbergh, M. W. (2009). Information-based branching schemes for binary linear mixed integer problems. *Mathematical Programming Computation*, 1:249–293.

[95] Khalil, E., Bodic, P. L., Song, L., Nemhauser, G., and Dilkina, B. (2016). Learning to branch in mixed integer programming. In *Thirtieth AAAI Conference on Artificial Intelligence*, pages 724–731.

[96] Kumar, L., Pandey, P., and Wimpenny, D. (2019). *3D Printing and Additive Manufacturing Technologies*. Springer.

[97] Lahdelma, R. and Hakonen, H. (2003). An efficient linear programming algorithm for combined heat and power production. *European Journal of Operational Research*, 148:141–151.

[98] Lam, T., Xiong, Y., Dharmawan, A., Foong, S., and Soh, G. (2020). Adaptive process control implementation of wire arc additive manufacturing for thin-walled components with overhang features. *The International Journal of Advanced Manufacturing Technology*, 108:1061–1071.

[99] Larrañaga, P. and Lozano, J. (2001). *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Springer Science & Business Media.

[100] Lawler, E. and Wood, D. (1966). Branch-and-bound methods: a survey. *Operations Research*, 14:699–719.

[101] Lazoglu, I., Manav, C., and Murtezaoglu, Y. (2009). Tool path optimization for free form surface machining. *CIRP Annals-Manufacturing Technology*, 58:101–104.

[102] Lenstra, J. and Kan, A. R. (1975). Some simple applications of the travelling salesman problem. *Journal of the Operational Research Society*, 26:717–733.

[103] Li, H., Dong, Z., and Vickers, G. (1998). Optimal tool path generation $2^{1/2}$-D milling of dies molds. In *SSM'98 Sculptured Surface Machining Conference*, pages 272–278.

[104] Lianshuan, S. and Zegyan, L. (2009). An improved Pareto genetic algorithm for multi-objective TSP. In *2009 Fifth International Conference on Natural Computation*, pages 585–588.

[105] Liao, Y., Li, H., and Chiu, Y. (2006). A study of laminated object manufacturing with separately applied heating and pressing. *The International Journal of Advanced Manufacturing Technology*, 27:703–707.

[106] Liu, M., Ding, X., Yan, Y., and Ci, X. (2011). Study on optimal path changing tools in CNC turret typing machine based on genetic algorithm. In *Computer and Computing Technologies in Agriculture IV: 4th IFIP TC 12 Conference*, pages 345–354.

[107] Lodi, A. and Zarpellon, G. (2017). On learning and branching: a survey. *Top*, 25(2):207–236.

[108] Long, Q. (2014). A constraint handling technique for constrained multi-objective genetic algorithm. *Swarm and Evolutionary Computation*, 15:66–79.

[109] Mahajan, A. and Ralphs, T. (2010). On the complexity of selecting disjunctions in integer programming. *SIAM Journal on Optimization*, 20:2181–2198.

[110] Makbul, H., Viboon, T., Chorkaew, J., and Chaiya, D. (2019). Laser cutting path optimization using simulated annealing with an adaptive large neighborhood search. *The International Journal of Advanced Manufacturing Technology*, 103:781–792.

[111] Manav, C., Bank, H., and Lazoglu, I. (2013). Intelligent toolpath selection via multi-criteria optimization in complex sculptured surface milling. *Journal of Intelligent Manufacturing*, 24:349–355.

[112] Marler, R. and Aora, J. (2004). Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26:369–395.

[113] Martínez, P. and García, J. (2021). ACOTSP-MF: A memory-friendly and high scalable ACOTSP approach. *Engineering Applications of Artificial Intelligence*, 99:104131.

[114] McMenemy, P., Veerapen, N., Adair, J., and Ochoa, G. (2019). Rigorous performance analysis of state-of-the-art tsp heuristic solvers. In *Evolutionary Computation in Combinatorial Optimization*, pages 99–114.

[115] Meringer, M. (1999). Fast generation of regular graphs and construction of cages. *Journal of Graph Theory*, 30:137–146.

[116] Meteyer, S., Xu, X., Perry, N., and Zhao, Y. (2014). Energy and material flow analysis of binder-jetting additive manufacturing process. *Procedia CIRP*, 15:19–25.

[117] Micali, M. and Dornfeld, D. (2016). Fully three-dimensional toolpath generation for point-based additive manufacturing systems. In *Solid Freeform Fabrication 2016: Proceedings of the 27 th Annual International Solid Freeform Fabrication Symposium - An Additive Manufacturing Conference Reviewed Paper*, pages 36–42.

[118] Morrison, D., Jacobson, S., Sauppe, J., and Sewell, E. (2016). Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102.

[119] Myszkowski, P., Laszczyk, M., and Dziadek, K. (2019). A non-dominated sorting tournament genetic algorithm for multi-objective travelling salesman problem. In *Proceedings of the Federated Conference on Computer Science and Information Systems*, pages 67–76.

[120] Nassehi, A., Essink, W., and Barclay, J. (2015). Evolutionary algorithms for generation and optimization of tool paths. *CIRP Annals*, 64:455–458.

[121] Nguyen, G. (2009). *Hamiltonian Cycle Problem, Markov Decision Processes and Graph Spectra*. PhD thesis, University of South Australia.

[122] Onwubolu, G. and Clerc, M. (2004). Optimal path for automated drilling operations by a new heuristic approach using particle swarm optimization. *International Journal of Production Research*, 42:473–491.

[123] Oztemel, E. and Gursev, S. (2020). Literature review of Industry 4.0 and related technologies. *Journal of Intelligent Manufacturing*, 31:127–182.

[124] Parberry, I. (1997). An efficient algorithm for the Knight's tour problem. *Discrete Applied Mathematics*, 73:251–260.

[125] Peterson, G., Schwartz, J., Zhang, D., Welss, B., Ganter, M., Storti, D., and Boydston, A. (2016). Production of materials with spatially-controlled cross-link density via vat photopolymerization. *ACS Appl. Mater. Interfaces*, 8:29037–29043.

[126] Petunin, A. and Stylios, C. (2016). Optimization models of tool path problem for CNC sheet metal cutting machines. *IFAC-PapersOnLine*, 49:23–38.

[127] Pezer, D. (2016). Efficiency of tool path optimization using genetic algorithm in relation to the optimization achieved with CAM software. In *International Conference on Manufacturing Engineering and Materials, ICMEM 2016, 6-10 June 2016, Nový Smokovec, Slovakia*, pages 374–379.

[128] Pinkerton, A. (2016). Lasers in additive manufacturing. *Optics&Laser Technology*, 78:25–32.

[129] Punnen, A. (2007). *The Traveling Salesman Problem: Applications, Formulations and Variations*, pages 1–28. Springer US.

[130] Rojko, A. (2017). Industry 4.0 concept: background and overview. *International Journal of Interactive Mobile Technologies*, 11:77–89.

[131] Rui, W., Haiou, Z., Guilan, W., Shangyong, T., and Runsheng, L. (2017). Generation of deposition paths and quadrilateral meshes in additive manufacturing. In *Proceedings of the 28 th Annual International Solid Freeform Fabrication Symposium- An Additive Manufacturing Conference*, pages 972–983.

[132] Sato, H., Aguirre, H., and Tanaka, K. (2007). Controlling dominance area of solutions and its impact on the performance of MOEAs. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 5–20. Springer.

[133] Schuh, G., Prote, J., Luckert, M., and Hünnekes, P. (2017). Knowledge discovery approach for automated process planning. *Procedia CIRP*, 63:539–544.

[134] Schütze, O., Laumanns, M., Coello, C., Dellnitz, M., and Talbi, E. (2008). Convergence of stochastic search algorithms to finite size Pareto set approximations. *Journal of Global Optimization*, 41:559–577.

[135] Serdeczny, M., Comminal, R., Pedersen, D., and Spangenberg, J. (2018). Experimental validation of a numerical model for the strand shape in material extrusion additive manufacturing. *Additive Manufacturing*, 24:145–153.

[136] Shim, V., Tan, K., Chia, J., and Chong, J. (2011). Evolutionary algorithms for solving multi-objective travelling salesman problem. *Flexible Services and Manufacturing Journal*, 23:207–241.

[137] Tessema, B., Yen, G., and Member, S. (2006). A self adaptive penalty function based algorithm for constrained optimization. In *IEEE International Conference on Evolutionary Computation*, pages 246–253.

[138] Thompson, M., Moroni, G., Vaneker, T., Fadel, G., Campbell, R., Gibson, I., Bernard, A., Schulz, J., Graf, P., Ahuja, B., and Martina, F. (2016). Design for additive manufacturing: Trends, opportunities, considerations, and constraints. *CIRP Annals-Manufacturing Technology*, 65:737–760.

[139] Thornton, A. (2015). *Additive Manufacturing (AM): Emerging Technologies, Application and Economic Implication*. Nova Science Publisher.

[140] Torralbo, M. (2021). *Implementation of the "Snakes and Ladders" Heuristic for Solving the Hamiltonian Cycle Problem*. Bachelor's thesis, University of the Basque Country.

[141] Turner, B., Strong, R., and Gold, S. (2014). A review of melt extrusion additive manufacturing processes: I. process design and modeling. *Rapid Prototyping Journal*, 20:192–204.

[142] Uğur, A. and Aydin, D. (2009). An interactive simulation and analysis software for solving TSP using ant colony optimization algorithms. *Advances in Engineering Software*, 40:341–349.

[143] van Horn, G., Olij, R., Sleegers, J., and van den Berg, D. (2018). A predictive data analytic for the hardness of hamiltonian cycle problem instances. In *Data Analytics 2018: The Seventh International Conference on Data Analytics*, pages 91–96.

[144] Van-Rossum, G. and Drake, F. (2009). *Python 3 Reference Manual*. CreateSpace.

[145] Vayre, B., Vignat, D., and Villeneuve, F. (2012). Designing for additive manufacturing. *Procedia CIRP*, 3:632–637.

[146] Vincent, T., Seipp, F., Ruzika, S., Przybylski, A., and Gandibleux, X. (2013). Multiple objective branch and bound for mixed 0-1 linear programming: Corrections and improvements for the biobjective case. *Computers & Operations Research*, 40:498–509.

[147] Wang, Y. (2018). An improved method to compute sparse graphs for traveling salesman problem. *International Journal of Industrial and Manufacturing Engineering*, 12:198–206.

[148] Will, M., Bertrand, J., and Fransoo, J. (2002). Operations management research methodologies using quantitative modeling. *International Journal of Operations & Production Management*, 22:241–264.

[149] Xia, C., Pan, Z., Zhang, S., Polden, J., Wang, L., Li, H., and Chen, S. (2020). Model predictive control of layer width in wire arc additive manufacturing. *Journal of Manufacturing Processes*, 58:179–186.

[150] Yang, R., Lee, C., Liu, Q., and Zheng, S. (2019). A carrier-shipper contract under asymmetric information in the ocean transport industry. *Annals of Operations Research*, 273:377–408.

[151] Zheng, P., Wang, H., Sang, Z., Zhong, R., Liu, Y., Liu, C., and Mubarok, K. (2018). Smart manufacturing systems for Industry 4.0: conceptual framework, scenarios and future perspectives. *Frontiers of Mechanical Engineering*, 13:137–150.

[152] Zoutendijk, G. (1960). *Methods of Feasible Directions: A Study in Linear and Non-Linear Programming.* Elsevier.