

UNIVERSITÁ DEGLI STUDI DELLA CALABRIA

Dipartimento di Elettronica Informatica e Sistemistica

Corso di dottorato in

Ricerca Operativa

MAT/09

XXII CICLO

Coordinatore Prof. Lucio Grandinetti

Tesi di Dottorato

OMEGA

OUR MULTI ETHNIC GENETIC ALGORITHM

Carmine Cerrone

Il Relatore:

Prof. Manlio Gaudioso

ANNO ACCADEMICO 2008-2009

Table of Contents

Table of Contents	i
Abstract	iii
Introduction	1
1 Our Multi Ethnic Genetic Algorithm (OMEGA)	4
1.1 Introduction	4
1.2 Our Multi Ethnic Genetic Algorithm (OMEGA)	7
1.2.1 OMEGA Approach	8
1.2.2 OMEGA's steps	10
1.2.3 Evolutionary Step Features	12
1.3 Building Block and chromosome's definition	13
1.3.1 GA and OMEGA chromosome observation	14
1.3.2 Basic ideas about the chromosome	17
1.3.3 OMEGA and Blocks	20
1.4 Minimum Labeling Spanning Tree	21
1.4.1 Problem Definition	22
1.4.2 The Algorithm	22
1.4.3 Pseudocode	28
1.4.4 Computational Results	28
2 Monochromatic Set Partitioning	31
2.1 Problem Definition and Basic Notation	31
2.2 Problem Complexity	32
2.3 Mathematical Formulations	35
2.4 Polynomial case	40
2.4.1 Solution's Optimality	44
2.5 Genetic Approach	47
2.5.1 The Chromosome	48

2.5.2	The Crossover	50
2.5.3	The Mutation	50
2.5.4	The Splitting Population Function	51
2.5.5	The Fitness Functions	51
2.5.6	Pseudocode	52
2.6	Results	52
3	Bounded Degree Spanning Tree	55
3.1	Introduction	55
3.2	Problem definition and motivations	57
3.3	Mathematical Formulation	59
3.4	Relations among MBV, MDS and ML	64
3.4.1	Notations	64
3.4.2	The problems are not equivalent	64
3.4.3	Relations among the problems	69
3.5	OMEGA Algorithm	72
3.6	Results	75
4	Multi-Period Street Scheduling and Sweeping (MPS3)	76
4.1	Introduction	76
4.1.1	Variant 1	77
4.1.2	Variant 2	81
4.2	Literature Review	82
4.3	Problem Formulation	83
4.3.1	Variant 1	83
4.3.2	Variant 2	87
4.4	Genetic Algorithm	87
4.4.1	Initialization	88
4.4.2	Schedules	89
4.4.3	Fitness	93
4.4.4	Breeding	93
4.4.5	Mutation	97
4.4.6	Genetic Algorithm Summary	98
4.5	Results	100
	Bibliography	102

Abstract

Combinatorial optimization is a branch of optimization. Its domain is optimization problems where the set of feasible solutions is discrete or can be reduced to a discrete one, the goal being that of finding the best possible solution. Two fundamental aims in optimization are finding algorithms characterized by both provably good run times and provably good or even optimal solution quality. When no method to find an optimal solution, under the given constraints (of time, space etc.) is available, heuristic approaches are typically used. A metaheuristic is a heuristic method for solving a very general class of computational problems by combining user-given black-box procedures, usually heuristics themselves, in the hope of obtaining a more efficient or more robust procedure. The genetic algorithms are one of the best metaheuristic approaches to deal with optimization problems. They are a population-based search technique that uses an ever changing neighborhood structure, based on population evolution and genetic operators, to take into account different points in the search space. The core of the thesis is to introduce a variant of the classic GA approach, which is referred to as OMEGA (Multi Ethnic Genetic Algorithm). The main feature of this new metaheuristic is the presence of different populations that evolve simultaneously, and exchange genetic material with each other. We focus our attention on four different optimization problems defined on graphs. Each one is

proved to be NP-HARD. We analyze each problem from different points of view, and for each one we define and implement both a genetic algorithm and our OMEGA.

Introduction

In the 1950s and the 1960s several computer scientists independently studied evolutionary system starting from the idea that evolution could be used as an optimization tool for engineering problems. The idea in all these systems was to let a population of candidate solutions to a given problem, evolve using operators inspired by natural genetic variation and natural selection. The approach was firstly introduced in 1954 in the work by Nils Aall Barricelli, who was with the Institute for Advanced Study in Princeton, New Jersey. [27] [2]. This publication was not widely noticed. Nevertheless, computer simulation based on biological evolution became more common in the early 1960s, and the methods were described in books by Fraser and Burnell [16, 17]. Genetic algorithms (GA) in particular became popular thanks to the work by John Holland in the early 1970s, and particularly his book *Adaptation in Natural and Artificial Systems* (1975) [21] . His work originated from studies of cellular automata, conducted by Holland and his students at the University of Michigan. Holland introduced a formalized framework for predicting the quality of the next generation, known as Holland's Schema Theorem. In the last few years there has been widespread interaction among researchers studying various evolutionary computation methods, and it is quite hard to retrace the boundaries between GAs, evolution strategy, evolution programming, and other evolutionary approaches have broken down to same

extent. Nowadays researchers use the term "genetic algorithm" to describe something very far from Holland's original concept. Among the most famous GA variants, the Lemarcking evolution and the Memetic algorithms [12] play a crucial role. Grefenstette introduced the Lemarckian operator into Genetic algorithms [20]. This theory introduces the concept of experience into the GAs, and suggest the use of a clever crossover technique. Moscato and Norman introduced the term Memetic algorithm to describe genetic algorithms in which local search plays a significant role [25]. There are several variants to the basic GA model, and in most cases these variants are combined to obtain the best solution to the specific problem. Another possible solution is the rank-based selection. We cite this technique in particular because the implicit splitting of the population in sub-populations and the concept of migration are largely used and reinterpreted in this work.

Genetic Algorithms are the logical thread that unifies all the chapters of the dissertation.

It is organized as follows. In chapter one we introduce the basic ideas of our multi ethnic genetic approach (OMEGA) and state it formally. The main feature of this new metaheuristic is the presence of different populations that evolve simultaneously, and exchange genetic material with each other. The applications are reported in chapter two three and four. In each of them a particular problem of graph optimization is studied. All studied problems are defined on a graph, and in all the cases we try to solve a combinatorial optimization problem via a GA. We introduce in the second chapter a problem defined on labelled graph, in this case we introduce two mathematical formulations and after a brief study of their complexity in general and in particular cases, we introduce a GA, able to find a solution very near to the

optimal one. In the third chapter we study a bounded degree spanning-tree problem, we present a tree variant about the same problem, we introduce some results about the relation between the problems, and we provide a branch and bound algorithm able to find the optimal solution. In the last section we use a GA. In the last chapter we focus our attention on a arch-routing problem on multigraph, for this problem we introduce a mathematical formulation and a GA. In this work we apply a GAs to three combinatorial optimization problems defined on a graph.

Chapter 1

Our Multi Ethnic Genetic Algorithm (OMEGA)

1.1 Introduction

The genetic algorithm is one of the best approaches to solving optimization problems. These algorithms are a population-based search technique that use an ever-changing neighborhood structure, based on population evolution and genetic operators, to take into account different points in the search space. Many techniques have been developed to escape from the local optimum when genetic algorithms fail to individuate the global optimum. In any case it appears clear that the intrinsic evolution scheme of genetic algorithms cannot be enforced to avoid the local optima without upsetting the "natural" evolution of the initial population.

In this work, in order to reduce the probability of remaining trapped in a local minimum we keep the basic schema of GA. The main difference is that, starting from an initial population, we produce, one by one, k different populations and we define k different evolution environments in which we left the k populations to evolve independently, Although an appropriate merging scheme, is embedded to guarantee possible

interaction.

John Holland showed in the book *Adaptation in Natural and Artificial Systems* (1975) [21], how the evolutionary process applied to solving a wide variety of problems. Many authors have in fact drawn inspiration from nature to create a highly adaptive optimization technique. In this work we take into account other characteristics of the evolutionary process, in order to improve the performance and the flexibility of the classic genetic algorithm; our approach uses the concept of Speciation to increase the amount of the analyzed solutions.

To achieve this aim we modify three different components of a GA:

- The population.
- The fitness functions.
- The chromosome representation.

In an evolutive process the aim of each individual is reproduction. Obviously in the world the resources are limited and for this reason, it isn't possible that every one of the individuals will be able to reproduce. In this way only the strongest ones can achieve their goal of producing offspring for the next generation. These processes from generation to generation have produced bodies perfectly adapted to their environment. This brief description of the Darwinian principle of reproduction and survival of the fittest is the basis of the current GA.

An important observation is in order make. If the goal of each organism is the same, why do we have many different species? There are two basic reasons: (i) There are many streets which arrive at the same place; (ii) to survive you must adapt to the environment. Usually in a GA approach we observe the evolution of a population from a basic set of individuals to a set of relatives who are better adapted to the problem. This strategy does not take into account the option of producing from the same species, different races. The idea at the basis of our approach is that, when ever many different races are involved in the same evolutionary process, the genetic difference between the populations can increase the quality of the results of all the process. In other words using this technique we can try to escape from a local minimum solution, creating new populations that evolve independently. The use of different variants of the same fitness function, is adopted to guarantee that different populations do not take the same evolutionary path.

The main idea of our technique consists of translating the concept of genetic isolation and genetic convergence from a biologic context into an algorithmic approach. This work introduces a modified GA approach that draws its inspiration from two fundamental biological concepts: Speciation and Convergent Evolution.

Speciation is the evolutionary process by which new biological species arise. There are four geographic modes of speciation in nature, based on the extent to which speciating populations are isolated from one another.

Convergent evolution describes the acquisition of the same biological trait in unrelated populations. The wing is a classic example of convergent evolution in action. Although their last common ancestor did not have wings, birds and bats are fitted

with wings, similar in construction, due to the physical constraints imposed upon wing shape.

Our main item is to design a genetic schema able to escape from a local minimum solution. The concept of Convergent Evolution suggests to us that if we simply design a multi-start genetic algorithm, there is a great probability of obtaining similar results from any instances. On the other hand, the concept of Speciation tells us that if a population is branched into two or more sub populations forced to adapt to new environments, in a short period any population can probably evolve into a new species.

1.2 Our Multi Ethnic Genetic Algorithm (OMEGA)

A main issue regarding metaheuristic approaches in general and genetic algorithms in particular is how to avoid to be trapped in local minima while exploring the largest possible feasible region. This problem has particular relevance when the objective function is characterized by a large number of local minima. In the next example we show the function $f_1 = 5 * \sin(\frac{1}{2}x) + (\frac{1}{10}x)^2$. If our objective is to find the minimum of f_1 , our genetic algorithm can be trapped in any of the many minimal solutions.

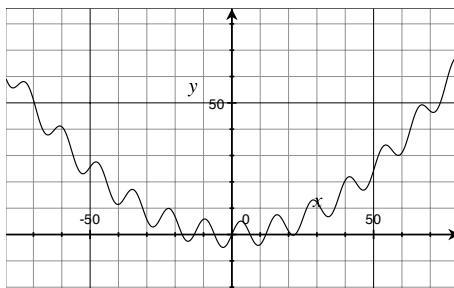


Figure 1.1: f_1

In these cases, in order to avoid to remain trapped in local minima, various techniques can be applied:

- We can work on the crossover operators. Although all crossover operators are designed to take potentially large steps, many analytical results show that the crossover tends to make its largest jumps during the first few generations because it takes advantage of the diversity in the population.
- We can work on the mutation operators. But usually, in order to ensure the convergence of this method, large steps are not desirable in this phase.
- We can use the multi-start technique, but the biological concept of convergent evolution suggests that there is a concrete probability of creating many similar populations. Moreover, a multi-start technique usually does not use information from the previous iteration in the next population.

In the rest of this chapter we will describe our genetic approach and how it is related to the above mentioned issues.

1.2.1 OMEGA Approach

OMEGA approach takes inspiration from the biological concept of genetic Isolation and Speciation; moreover it leans on the building-block hypothesis (Holland, 1975; Goldberg, 1989).

Using a classical genetic algorithm we create a population P_0 . After a sufficient number of iterations (i), we have a population P_i containing descendants of the best solutions found. In this population the rate of good features (blocks) is high (building-block hypothesis) [15]. We consider this population a new biological species and we

try to get the process of Speciation by splitting the population. The new populations are immersed in different environments in order to minimize convergent evolution. This ensures the formation of different species, compatible with each other in terms of building-blocks. After a sufficient number of iterations, this process can be iterated, creating and merging a random number of populations.

Consider for instance the following problem:

$$\min f_1 \tag{1.2.1}$$

$$x \in \mathfrak{R} \tag{1.2.2}$$

where f_1 is the previously defined objective function.

By using a classical genetic algorithm, we could get stuck in a local minimum with high probability.

Now let's consider the above presented technique. We split the population P derived from the classical genetic algorithm in P1 and P2. These two populations will evolve independently, but for P2 we consider a different fitness function $f_2 = 5 + (\frac{1}{10}x)^2$ as shown in figure 1.2.

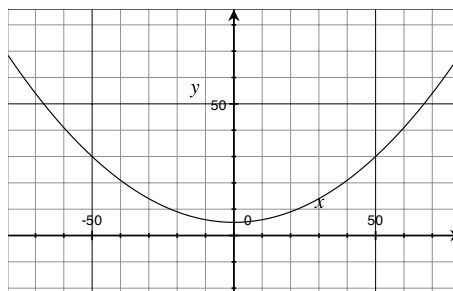


Figure 1.2: f_2

After a short number of iterations, P2 could converge to solution $x=0$. By merging

the two populations again, the new population P subject to function f_1 can take advantage of the building-blocks that led P_2 to the solution 0 to expand its genetic diversity and possibly improve the solution for the original problem.

1.2.2 OMEGA's steps

In this section we present the basic scheme of our OMEGA approach. It is easy to understand that all the concepts introduced in this approach can to be combined in several ways. In the following we introduce a very easy pattern, useful to understand the approach. In the other chapters of this dissertation we apply this algorithm to different problems and in some cases we introduce different patterns to mix the population.

Let $G(x, P, f)$ be an evolutionary scheme, where x is the input instance, P is a population of solutions and f is the fitness function. Let f_0 be the main fitness function of our problem, and let $F = \{f_1, f_2, \dots, f_n\}$ be a set of n fitness functions related with f_0 .

Input: problem instance x . **Output:** a feasible solution of the given problem.

1. $t = 0$.
2. *Creation Operation:* Create P_t , the starting population for the problem.
3. Repeat until a given stopping criterion is satisfied:
 - (a) Execute $G(x, P_t, f_0)$ to obtain the evolved population \hat{P}_t .
 - (b) *Split Operation:* Split \hat{P}_t in $P_{(t,1)}, \dots, P_{(t,k)}$ populations
 - (c) For each population $P_{(t,j)}$

- Randomly select a fitness function $f_r \in F$
- Execute $G(x, P_{(t,j)}, f_r)$ to obtain the evolved population $\hat{P}_{(t,j)}$
- (d) *Merge Operation*: Merge populations $\hat{P}_{(t,1)}, \hat{P}_{(t,2)}, \dots, \hat{P}_{(t,k)}$ to obtain $P_{(t+1)}$.
- (e) $t := t + 1$

The basic version of the algorithm starts at time $t = 0$ from a unique population P_t that evolves according to a given genetic scheme G and to the main fitness function f_0 . After this evolutionary step, the obtained evolved population \hat{P}_t is split into k different populations. Each obtained population evolves according to a given evolutionary scheme and a fitness function f_r randomly selected among those in the set F . The evolved populations are merged into a unique population P_{t+1} and the process is iterated until a given stopping criterion is reached (see figure 1.3).

In the figure, lines $S1, S2, S3, S4$ represent respectively:

1. $S1$: Evolutionary Step.
2. $S2$: Split Operation.
3. $S3$: Merge Operation.
4. $S4$: Repeat until a given stopping criterion is satisfied.

The *Split Operation* takes as input a given population P and returns a set of k populations $(P_1 \dots P_k)$. The k populations are obtained from P by partitioning it into k sub-populations where k is a random value ranging from 2 to n . The size of each P_i is chosen either equal to $\lceil \frac{|P|}{k} \rceil$ or equal to $\lfloor \frac{|P|}{k} \rfloor$ such that $\sum_{i=1}^k |P_i| = |P|$.

The *Merge Operation* takes as input a given set of populations P_i and returns a unique population $P = \bigcup_i P_i$.

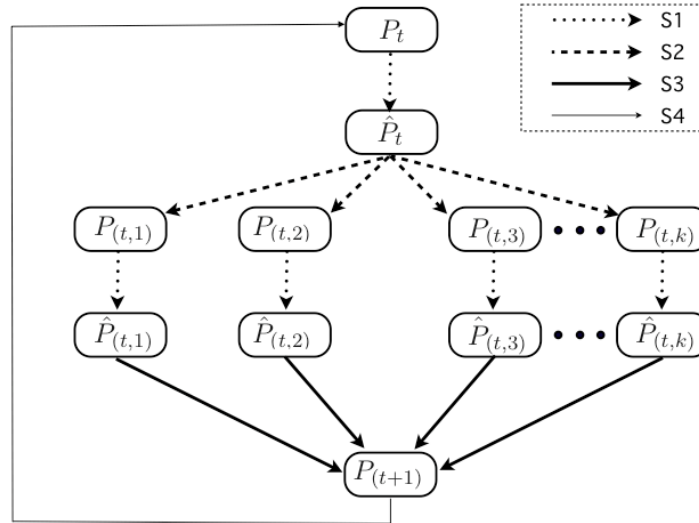


Figure 1.3: *Block diagram of OMEGA's Basic version*

1.2.3 Evolutionary Step Features

When we speak about evolutionary steps, we refer to a sequence of j iterations of a classical genetic algorithm. In particular $G(x, P, f)$ is a genetic algorithm, where x is the input instance, P is a population of solutions and f is the fitness function. The population P is an input of the algorithm; after a fixed number of iterations, the genetic algorithm evolves P into a new population \hat{P} .

In this metaheuristic structure we can use any genetic algorithm. However, to fully exploit all the characteristics of the OMEGA it is important to pay particular attention to two main aspects.

1. *The chromosome's definition :*

It is important to create a structure of chromosomes able to preserve the features

of the solution after the crossover operation. In particular, in our metaheuristic we introduce the *Merge* operator. This operator simulates the migration of different ethnic groups (the populations P) in the same environment. For this reason we want a chromosome able to preserve the features of the solution, although this solution is coupled to another of a different population.

2. *The crossover's definition :*

The observations of the previous step imply the crossover's importance. We can assert that the crossover and the chromosome definition are two of the most relevant items of a genetic algorithm and in particular these two aspects are fundamental for the creation of a multi ethnic genetic algorithm.

1.3 Building Block and chromosome's definition

The Genetic algorithms are relatively simple to implement, but their behavior is difficult to understand. In particular it isn't easy understand why they often produce high quality solutions. One hypothesis is that a genetic algorithm performs adaptation by implicitly and efficiently implementing the building block hypothesis (BBH): a description of an abstract adaptive mechanism that performs adaptation by recombining "building blocks". A description of this mechanism is given by (Goldberg 1989:41).

Anyway there are several results which contradict this hypothesis. The debate over the building block hypothesis demonstrates that the issue of how GAs "work" is currently far from settled. Nevertheless most solutions are designed taking into account the building block hypothesis.

1.3.1 GA and OMEGA chromosome observation

The main characteristic of OMEGA is the presence of different populations that evolve together. This characteristics does not imply any difference between this approach and a classic GA during the lifecycle of a population. On the other hand, when elements of different species are mixed in a unique population, we have to take into account this new situation. It is very important to create a strategy that ensures compatibility between individuals of different populations.

For example if our problem is to look for a Hamiltonian cycle in the graph $G(V, E)$ (fig. 1.4) , we can use as a chromosome for the GA a $\{0,1\}$ vector associated with the edges.

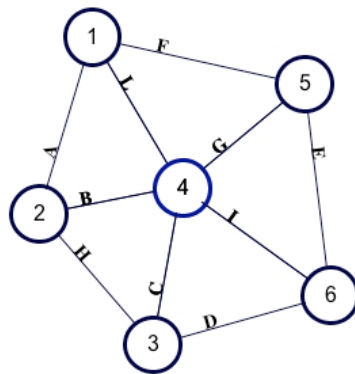
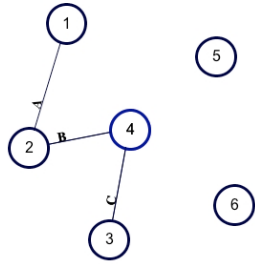


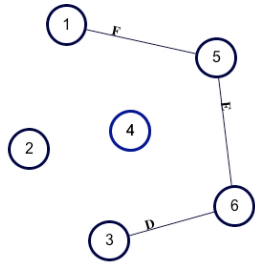
Figure 1.4: $G(V, E)$

In fig. 1.5 we have a population of four elements. The elements are very different from each other. If we define a basic crossover function using the $+$ operator we can produce six different new chromosomes. Figs. 1.6,1.7,1.8,1.9 show four of these solutions, where figs. 1.6,1.7 show two feasible solutions and figs. 1.8,1.9 show two



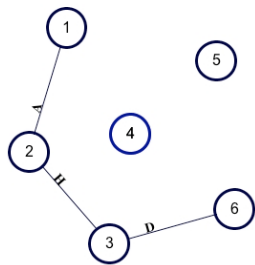
chromosome 1

A	B	C	D	E	F	G	H	I	L
1	1	1	0	0	0	0	0	0	0



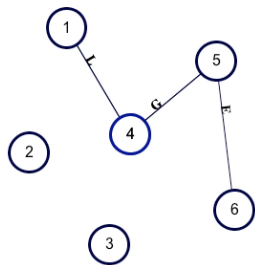
chromosome 2

A	B	C	D	E	F	G	H	I	L
0	0	0	1	1	1	0	0	0	0



chromosome 3

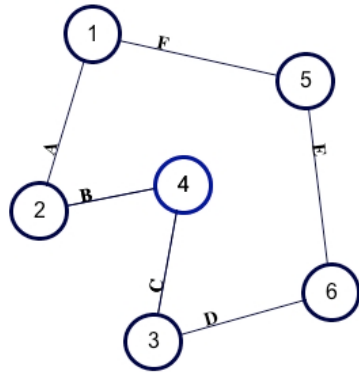
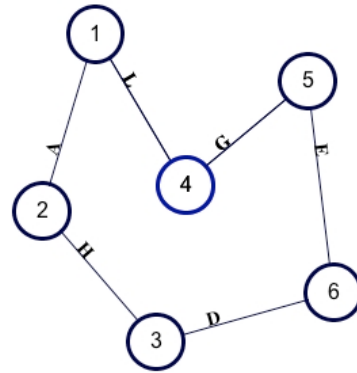
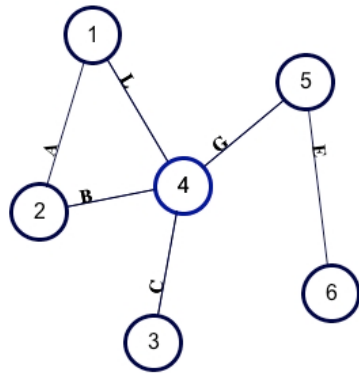
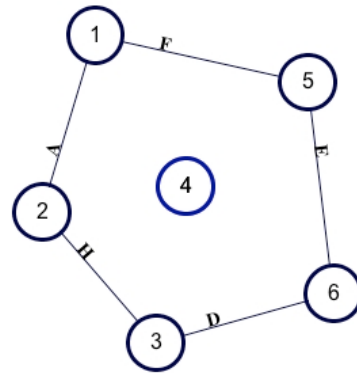
A	B	C	D	E	F	G	H	I	L
1	0	0	1	0	0	0	1	0	0



chromosome 4

A	B	C	D	E	F	G	H	I	L
0	0	0	0	1	0	1	0	0	1

Figure 1.5: chromosomes

Figure 1.6: *solution ch1 + ch2*Figure 1.7: *solution ch3 + ch4*Figure 1.8: *solution ch1 + ch4*Figure 1.9: *solution ch2 + ch3*

unfeasible solutions. It is easy to understand that if we have a population of such different elements it is very unlikely that we will produce feasible solutions.

For this reason if we have a GA that uses an unintelligent crossover it is necessary to balance two aspects: (i) we need a population composed of different elements to allow an evolution process and escape from the local minima; (ii) we need elements that are not very far from each other, in order to produce a stable and convergent search procedure.

These two aspects of a GA are very relevant when we define the structure of the chromosome. When we define the chromosome for the OMEGA this problem is accentuated. In a GA there is a single population and we can assume that the difference between two elements isn't very big. The reason is that the two elements are evolved in the same population and are descendants of the same relatives. In an OMEGA we can merge different populations and it is possible to have a crossover between very different chromosomes.

To exploit this situation it is important to design a chromosome very accurately. In the following a basic idea to design a chromosome for the OMEGA is shown, as well as a possible application to the *SpanningTree* problem.

1.3.2 Basic ideas about the chromosome

The aim of the OMEGA chromosome is to improve the crossover compatibility between two individuals. A possible approach to achieve it is to define a structure that automatically focuses attention on some characteristics of the chromosome and tries to import it into the new generation. Using this approach the chromosome isn't a representation of a solution but a data set that contains information about the creation

of new elements.

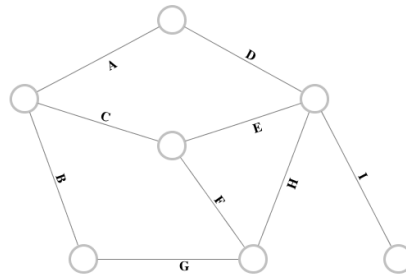


Figure 1.10: $G(V, E)$

It is easy to understand the idea by looking at an example. If our problem is to look for a particular (for example a bounded degree) spanning tree of a graph $G(V, E)$ (e.g. the graph in fig. 1.10) , with $|V| = k$, we can use as chromosome C an integer vector with size greater than $(k - 1)$. Each element of the vector represents an edge of G . We can have repetitions in C . Obviously C does not represent a spanning tree of G but it is possible to produce it using an easy procedure. We can associate a value to each edge that represents the number of occurrences that the edge has in C (fig. 1.11) .

Now, we can use an algorithm such as Kruskal or Prim to compute the minimum weight spanning tree of G . To obtain a feasible solution to our original problem.

Using this procedure the chromosome represents a suggestion for spanning tree procedure. Obviously by mixing different populations with elements very far from each other, we are sure that the "suggestions" arriving from both the parents are taken into account in the child.

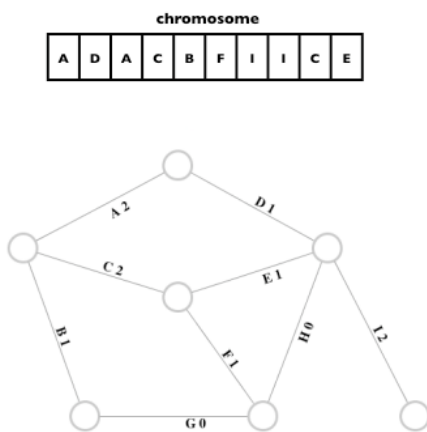
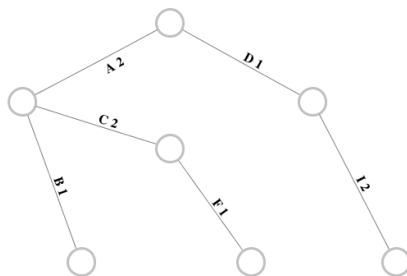
Figure 1.11: $G(V, E)$ weighted graph

Figure 1.12: Spanning Tree

1.3.3 OMEGA and Blocks

In the previous section, we introduced a possible definition of the OMEGA chromosome. The structure is motivated by the necessity of improving the crossover compatibility between chromosomes.

The aim of this section is to add to the previous approach another idea. Essentially taking inspiration from the building block theory [21], we want to modify the chromosome's structure, in order to preserve substructure information after the crossover. Our solution is to split the chromosome into sub-components, called blocks.

- Each block has a static or dynamic dimension, depending on the problem and the approach.
- The crossover isn't able to modify a block, it can just recombine parents' blocks.
- The mutation is the only function able to modify a block, by adding or removing elements, changing elements, or deleting from or adding to a chromosome's blocks.

In the figure 1.13 we can see three examples of OMEGA chromosomes. Each one of the three examples is related to the figure 1.10 of the previous page. The meaning for this chromosome is exactly the same one that we have used in the previous section.

In this last step, we want to introduce a partition of the chromosome into sub-components.

Our intent is to create logical blocks able to describe little slices of a solution. In our opinion there are three consequences to using this approach.

- If a block describes part of a good solution, there is some chance that its introduction in another chromosome increases the fitness function, and for this

reason we can have more copies of the same block in the population.

- Iteration by iteration we can obtain an automatic partition of the instance into sub-problems, each one described by one or more blocks.
- The recombination of these blocks can increase the crossover's compatibility between elements, in particular if the elements are very different from each other.

All the previous points are conjectures, but they represent an important part of the intuition that led us to design this technique. In the last section of this chapter, in order to better explain the OMEGA approach, we will apply the technique to a well known problem, the Minimum Labeling Spanning Tree (MLST).

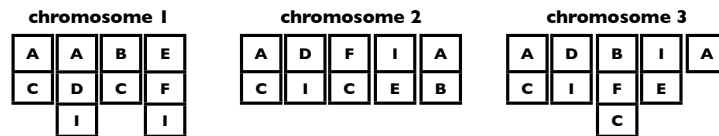


Figure 1.13: *OMEGA's chromosome examples*

1.4 Minimum Labeling Spanning Tree

Given a connected, undirected graph whose edges are colored (or labeled), the minimum labeling spanning tree (MLST) problem seeks a spanning tree with the minimum number of distinct colors (or labels). In this section we will use the OMEGA algorithm to solve the MLST problem. We chosen this problem because it is easy to understand and describe and because there are several papers that compare many

optimization techniques. Our goal is not to produce the best metaheuristic for the MLST but to compare our technique with other approaches.

1.4.1 Problem Definition

In the minimum labeling spanning tree (MLST) problem, we are given an undirected labeled graph $G = (V, E, L)$, where V is the set of nodes, E is the set of edges, and L is the set of labels. We seek a spanning tree with the minimum number of distinct labels. This problem was introduced by Chang and Leu [9], who proved its NP-hardness by reducing it to a minimum cover problem. Since then, other researchers have studied and presented other heuristics for the MLST. Some references on this topic are [23, 7, 29, 11, 26, 6, 30]. In particular there are three works that introduce Genetic Algorithms for MLST [29, 26, 30].

1.4.2 The Algorithm

The structure of the algorithm is exactly the same as the one introduced in the section 1.2.2. To describe OMEGA as applied to the MLST it is sufficient to introduce the definition of the chromosome and explain how the crossover, the mutation and splitting population function work, and finally, to show the set of fitness functions.

The Chromosome

The first step to describe the chromosome is the introduction of two parameters \mathcal{W} and \mathcal{H} . They are used to describe the maximum dimensions of the chromosome. \mathcal{W} represents the width of the chromosome, in particular the maximum number of blocks. \mathcal{H} represents the height of the chromosome, corresponding to the maximum number

<i>TreeCreator</i>
<i>INPUT: $G(V, E, L)$, chromosome c</i>
<i>OUTPUT: $Sol \in E$ Sapnning tree of G</i>
<pre> 1: $Cost \leftarrow \emptyset$ 2: for all edge $e \in c$ do 3: $l \leftarrow L(e)$ 4: if $\exists (l, i) \in Cost$ then {Case 1} 5: $Cost \leftarrow \{(l, i - 1)\} \cup Cost \setminus \{(l, i)\}$ 6: else {Case 2} 7: $Cost \leftarrow \{(l, -1)\} \cup Cost$ 8: end if 9: end for 10: $Sol \leftarrow MST(G(V, E, L, Cost))$ 11: return Sol </pre>

Figure 1.15: Spanning Tree Generator

$$L = \{0, 1, 2\} ,$$

and the chromosome

$$c = \{\{A, C, F\}, \{B, C\}, \{F, H\}\} \text{ of figure 1.17,}$$

the procedure TreeCreator builds

$$Sol = \{(A, -1), (B, -1), (C, -2), (D, 0), (E, 0), (F, -2), (G, 0), (H, -1), (I, 0)\} .$$

In figure 1.19 the resulting graph is shown. Using a MST procedure, we are able to produce the spanning tree of figure 1.20, using just two different labels.

The Crossover

The crossover is the easiest function of OMEGA's architecture. Basically the function takes as input two chromosomes $c1, c2$ and it produces as output the chromosome $c3$. The procedure randomly selects 50% of the blocks of $c1$ and $c2$, and it inserts the blocks into the new chromosome $c3$. In figure 1.21 a graphical example is shown.

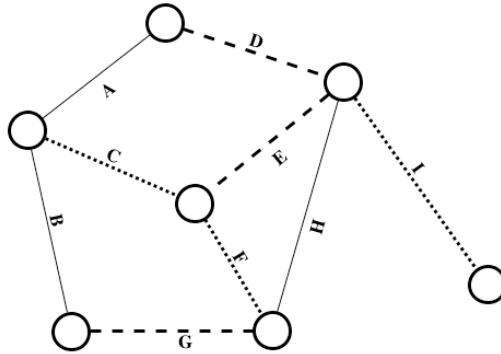


Figure 1.16: *Input Graph*

chromosome

A	B	F
C	C	H
F		

Figure 1.17: chromosome

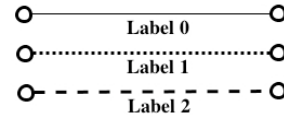


Figure 1.18: Labels

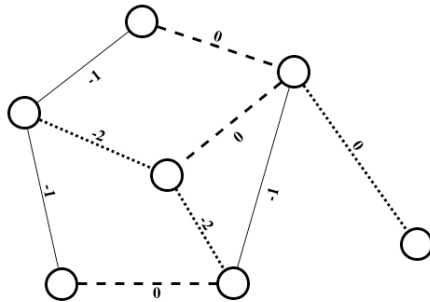


Figure 1.19: MST Graph

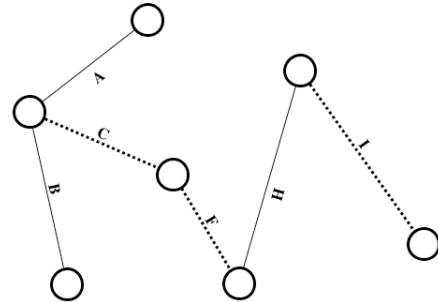
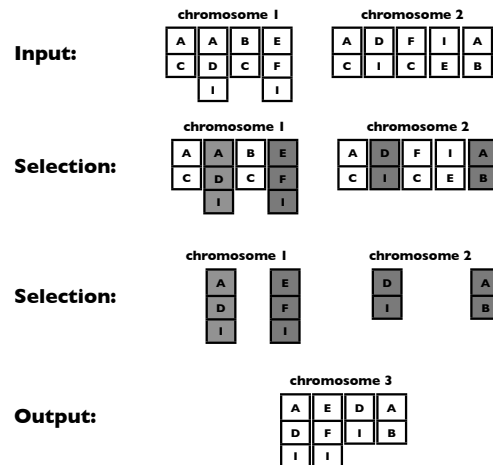


Figure 1.20: Spanning Tree

Figure 1.21: *Crossover*

The Mutation

There are three different kind of mutation: MUT-ADD, MUT-DEL and MUT-CHANGE.

The mutation operator randomly runs one of these three; if it isn't possible to use MUT-ADD or MUT-DEL, MUT-CHANGE is used.

Respectively, the chances of selecting the operators are: 50%, 30%,20%.

- MUT-ADD randomly selects a block in the chromosome and if it isn't full, it adds another edge in the last position. The new edge is selected from the set of edges near the connected structure, present in the same block.
- MUT-DEL randomly selects a block in the chromosome and if its dimension is greater than one, it removes a random edge from the block.
- MUT-CHANGE randomly selects a block in the chromosome, randomly selects an edge from the block, and replaces it with a new edge. The new edge is

selected from the set of edges near the connected structure, present in the same block.

The Splitting Population Function

The Splitting Population Function (SPF) is used to partition the original population into k sub-populations. This procedure is totally random, and if h is the dimension of the original population, it produces k new populations, each one of cardinality $\frac{h}{k}$. The aim of this procedure is to move the chromosomes of a big population into k different new populations.

The Fitness Functions

The fitness function is very easy to understand. It takes as input a chromosome c , using the procedure 2.6 to produce a spanning tree SP that is composed of $|V| - 1$ edges of the original graph $G(V, E, L)$. It produces a set C including all the labels identified in SP , and the return value is equal to $|C|$. Now we want to identify other fitness functions related to the first one. This step is very easy. The procedure builds the function $CT : L \rightarrow \mathbb{R}$ used to associate a numerical value to each label. The new fitness function is: $f : \sum_{l \in C} CT(l)$. Obviously if for each label $l \in L$ $CT(l) = 1$ we produce the original fitness function. When a new fitness function is randomly generated, it produces a function CT that produces one for most of the labels in L , but is able to produce a value greater than one for some labels.

Usually the procedure identifies some good solutions in the current population and tries to forbid the selection of one or more used labels.

1.4.3 Pseudocode

<i>BASIC-GA</i>
<i>INPUT</i> : $G(V, E, L)$ and Population P and Fitness function ft
<i>OUTPUT</i> : Population P
<pre> 1: $Cost \leftarrow \emptyset$ 2: for it iterations do 3: for all c in P les the best tree do 4: select randomly $c1 \in P$ such that $ft(c) > ft(c1)$ 5: $P \leftarrow (P \setminus \{c\}) \cup \{crossover(c1, c)\}$ 6: end for 7: for all c in P les the best one do 8: $P \leftarrow (P \setminus \{c\}) \cup \{mutation(c)\}$ 9: compute $ft(c)$ 10: end for 11: end for 12: return P </pre>

Figure 1.22: Basic Genetic Algorithm

1.4.4 Computational Results

This is a preliminary version of this section. We compared the OMEGA's results with the results proposed in [7]. In fig.1.24 the first three columns show the parameters characterizing the different scenarios (n, l and d) n: number of nodes of the graph, l: total number of labels assigned to the graph, d: measure of density of the graph. The remaining columns give the results of the MVCA heuristic, Variable Neighborhood Search, Simulated Annealing, Reactive Tabu Search and the Pilot Method, respectively. All the values are average values over 10 different instances. The instances set is composed of 150 instances. In each row of the table in the figure we wrote in bold face the best results obtained by the different tested procedures. We can see that our approach OMEGA, except for one instance, is able to always find the best solution.

<i>OMEGA</i>
<i>INPUT: G(V, E, L)</i>
<i>OUTPUT: Sol ∈ E Spanning tree of G</i>
<pre> 1: POP = {P₀, P₁, ... P₁₀} set of populations 2: P₀ = h 3: P₁ = P₂ = ... = P₁₀ = 0 4: while Not End Conditions do 5: BASIC-GA(G, P₀, f₀) 6: if new best solution in P₀ then 7: set new best solution 8: end if 9: SPF(P₀) ⇒ P₀ ≈ P₁ ≈ ... ≈ P₁₀ ≈ $\frac{h}{10}$ 10: for all P ∈ POP do 11: BASIC-GA(G, P, f_{random}) 12: end for 13: P₀ ← P₁ ∪ P₂ ∪ ... P₁₀ 14: P₁ ← P₂ ← ... ← P₁₀ ← ∅ 15: end while </pre>

Figure 1.23: OMEGA

n	l	d		MVCA	VNS	SA	RTS	PILOT	OMEGA
20	20	0,8		2,6	2,4	2,4	2,4	2,4	2,4
20	20	0,5		3,5	3,2	3,1	3,1	3,2	3,1
20	20	0,2		7,1	6,9	6,7	6,7	6,7	6,7
30	30	0,8		2,8	2,8	2,8	2,8	2,8	2,8
30	30	0,5		3,7	3,7	3,7	3,7	3,7	3,7
30	30	0,2		8,0	7,8	7,4	7,4	7,5	7,4
40	40	0,8		2,9	2,9	2,9	2,9	2,9	2,9
40	40	0,5		3,9	3,9	3,9	4,0	3,7	3,7
40	40	0,2		8,6	8,3	7,4	7,9	7,7	7,4
50	50	0,8		3,0	3,0	3,0	3,0	3,0	3,0
50	50	0,5		3,9	3,9	3,9	4,0	3,7	4,1
50	50	0,2		9,2	9,1	8,7	8,8	8,6	8,6
100	100	0,8		3,3	3,1	4,0	3,4	3,0	3,0
100	100	0,5		5,1	5,0	5,2	5,1	4,7	4,6
100	100	0,2		11,0	10,7	10,7	10,9	10,3	10,1

Figure 1.24: *OMEGA's computational results*

Chapter 2

Monochromatic Set Partitioning

2.1 Problem Definition and Basic Notation

In this chapter we focalize our attention on a new problem, the "monochromatic set partitioning" (*MMP*) problem. Given a graph G with labels (colors) associated with the edges, a feasible solution for the *MMP* is a set of connected components of G such that each component is composed exclusively of monochromatic edges. We are looking for a feasible solution with the minimum number of monochromatic components. In the following we prove that *MMP* is NP-hard and we provide some mathematical formulations for this problem. Moreover we show a polynomial case and in the last part of this chapter we present an OMEGA approach to the problem.

Let $G = (V, E, C)$ be an undirected and edges labeled graph where V is the vertex set, E the edge set and C a set of labels. A label $\mathcal{C}(i, j) \in C$ is associated with each edge $(i, j) \in E$. A feasible solution for the *MMP* problem is a set of edges $SOL \subseteq E$ such that for each two adjacent edges $(i, j), (j, k) \in SOL$, we have that $\mathcal{C}(i, j) = \mathcal{C}(j, k)$. The value of the objective function is equal to the number of connected components in the sub-graph of G induced by the edges set SOL .

Given a subset $S \subseteq E$ of edges, the set $\mathcal{C}(S) = \bigcup_{(i,j) \in S} \mathcal{C}(i,j)$ is the corresponding set of labels. The subgraph induced by a given label $c \in C$ is denoted by G^c , i.e. $G^c = (V, E^c, c)$ with $E^c = \{(i,j) \in E : \mathcal{C}(i,j) = c\}$. Similarly, we define $G^{\bar{c}}$ as the subgraph of G without edges whose label is c , i.e. $G^{\bar{c}} = (V, E \setminus E^c, C \setminus \{c\})$. Let us define \mathcal{P}_c as the set of connected components in G^c , i.e. $\mathcal{P}_c = \{P_c^1, \dots, P_c^r\}$ where P_c^j , $1 \leq j \leq r$, is a monochromatic component whose color is c . The set of edges incident to a vertex $i \in V$ is denoted by $\delta(i) = \{j : (i,j) \in E\}$. If $|\mathcal{C}(\delta(i))| = 1$ ($|\mathcal{C}(\delta(i))| = 2$) then i is called a monochromatic (bi-chromatic) vertex. Given a set of vertices $X \subseteq V$, we denote by $G[X] = (X, E[X], \mathcal{C}(E[X]))$ the subgraph of G induced by X where $E[X] = \{(i,j) : i,j \in X\}$.

2.2 Problem Complexity

In this section we prove the addressed problem is NP-complete. Let us define the decision version of our problem, namely, the *Bounded Minimum Monochromatic Partitioning Problem (BMMP)*:

Bounded Minimum Monochromatic Partitioning Problem: *Given an undirected and edge labeled graph $G = (V, E, C)$ and a positive integer k : is there a monochromatic partitioning of G composed of less than or equal to k components, i.e. $|\mathcal{P}(G)| \leq k$?*

Theorem 2.2.1. *The Bounded Monochromatic Partitioning Problem is NP-Complete.*

Proof. We prove the theorem by reduction from the well known Minimum Set

Covering Problem (MSC). Let $S = \{s_1, s_2, \dots, s_p\}$ be a set of p elements, $\mathcal{F} = \{F_1, F_2, \dots, F_q\}$ be a family of q subsets of S , i.e. $F_i \subseteq S, i = 1, 2, \dots, q$, and k be a positive integer. The decisional version of MSC consists in selecting no more than k subsets in \mathcal{F} that cover all the elements in S . We now define from the generic instance of MSC a graph $G = (V, E)$, a labeling function of the edges, and show that there exists a covering of S with at most k subsets if and only if there exists a monochromatic partitioning of G using at most k components.

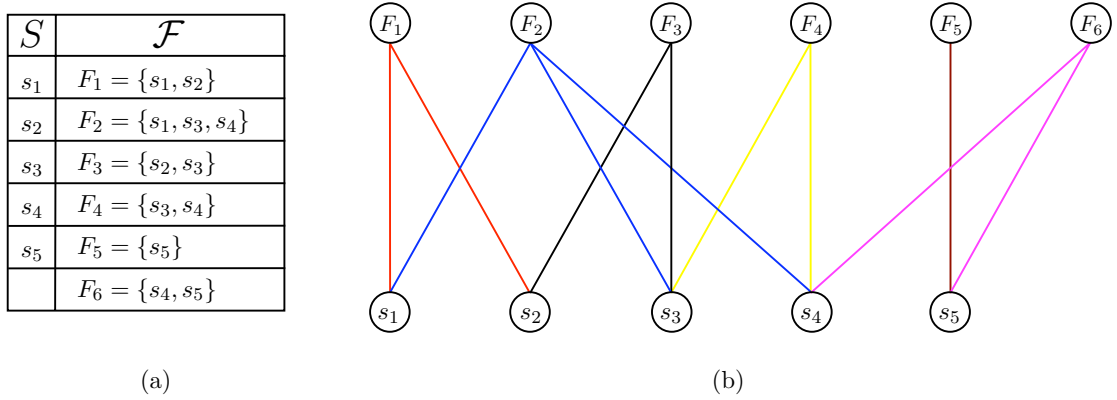


Figure 2.1: (a) A generic instance of the Minimum Set Covering Problem, and, (b) the corresponding instance of the Bounded Labeled Maximum Matching Problem.

Let us denote by $S(F_i) = \{s_j \in S : s_j \in F_i\}$ the collection of elements of S covered by F_i . For each subset $F_i \in \mathcal{F}$ and element $s_j \in S$ we define in G a corresponding vertex F_i and s_j , respectively. We define an edge in G between each vertex F_i and the vertices $s_j \in S(F_i)$ with associated label i (Figure 2.1). This construction can be accomplished in polynomial time.

Since G is bipartite and the color of edges incident to each vertex F_i is different, it is easy to see that into any monochromatic partitioning $\mathcal{P}(G) = \{P_1, P_2, \dots, P_k\}$

of G , each component P_i either is a singleton $\{s_j\}$ or contains exactly one vertex F_h . Each component P_i is univocally identified by a vertex, denoted by $\mathcal{I}(P_i)$, that is equal to s_j if $P_i = \{s_j\}$, to $F_h \in P_i$ otherwise.

Let $\mathcal{C}(S) = \{F_{i_1}, F_{i_2}, \dots, F_{i_k}\}$ be a covering of S with k size. We can define a monochromatic partitioning with at most k components as follows. Let G' be the subgraph of G induced by $\mathcal{C}(S)$. It is easy to see that in G' to each element s_j is incident at least one edge, because $\mathcal{C}(S)$ is a cover of S . Here we distinguish two cases: if there is only one edge incident to s_j , for instance (F_{i_h}, s_j) then put this element into the same component of F_{i_h} . Otherwise, if there are more edges, select one of them randomly and put s_j into the corresponding component. At the end of process at most k monochromatic components are produced.

On the contrary, let $\mathcal{P}(G) = \{P_1, P_2, \dots, P_k\}$ be a monochromatic partitioning of G with k components. Since, by construction, there are no edges among the vertices s_j , each vertex s_j either belongs to component P_h with $\mathcal{I}(P_h) = F_i$ or it is alone, i.e $P_h = \{s_j\}$. W.l.o.g. let $\{P_1, P_2, \dots, P_r\}$ be the components composed by a single vertex $s_j \in S$ and $\{P_{r+1}, \dots, P_k\}$ be the sets of components containing a vertex $F_i \in \mathcal{F}$. The set of vertices $\bigcup_{h=r+1}^k \mathcal{I}(P_h)$ cover all the elements in $\bigcup_{h=r+1}^k S(\mathcal{I}(P_h))$. If this set includes all the elements in S then we found a covering of S whose size is equal to $k - r - 1$. Otherwise there are some components of $\{P_1, \dots, P_r\}$ which vertex s_j is not yet covered. For each of these vertices we select a vertex F_i adjacent to it. In this way, the set of vertices F_i joined to $\bigcup_{h=r+1}^k S(\mathcal{I}(P_h))$ produces a covering of S with size at most equal to k . \square

2.3 Mathematical Formulations

In this section we introduce two formulations for the *MMP* problem. The main difference between the two formulations is the technique used to obtain and count the connected substructure. In the former we use a set of flow constraints, in the latter we use the Miller-Tucker-Zemlin constraints [24].

The first mathematical model (**M1**) is modeled like a network flow problem. Given the colored and undirected graph $G = (V, E, C)$, we build a new oriented and colored graph $G' = (V', E', C)$ derived from G such that $V' = V \cup \{s\}$ and $E' = \{(i, j), (j, i) \mid (i, j) \in E\} \cup \{(s, i) \mid i \in V\}$, where s is a new source node connected to each node in V . Since G' is directed, we denote by $\delta^+(i)$ and $\delta^-(i)$ the forward and backward star of node i in G' . To each node of $i \in V' \setminus \{s\}$ is associated a request $d_i = -1$ (destination node) while to the source node s is associated an offer equal to n , $d_s = n$, in order to satisfy the requests of destination nodes. No capacity to the edges inside the graph is associated. Notice that every time an edge of $\delta^+(s)$ is used, a new component of G is generated. Whereas our aim corresponds to minimizing the number of connected component, we associate to each edge $(s, i) \in \delta^+(s)$ a cost equal to 1 if the flow crosses it and zero otherwise. For the remaining edges in E' the cost is equal to zero. Minimizing the cost needed to satisfy the requests of destination nodes means to minimize the number of used edges in $\delta^+(s)$ and consequently the number of connected components of G . In order to guarantee that all connected components are monochromatic, we insert a constraint to ensure that in any solution all incident edges to the same node i have the same label.

Let x_{ij} be a variable representing the flow along the edge $(i, j) \in E'$, and let y_i be a binary variable associated with the edge (s, i) whose value is equal to 1 if (s, i)

is used and 0 otherwise. Finally, let us define the boolean variable $\ell_{i,c}$ whose value is equal to 1 if exist at least ane edge with label c incident to node i and 0 otherwise

$$(PLF)z = \min \sum_{i \in V' \setminus \{s\}} y_i \quad (2.3.1)$$

with constraints:

$$\sum_{(i,j) \in \delta^+(i)} x_{ij} - \sum_{(k,i) \in \delta^-(i)} x_{ki} = d_i \quad \forall i \in V' \setminus \{s\} \quad (2.3.2)$$

$$M_i y_i \geq x_{si} \quad \forall i \in V' \setminus \{s\} \quad (2.3.3)$$

$$M_{c,i} \ell_{i,c} \geq \sum_{(i,j) \in \delta^+(i): L(i,j)=c} x_{ij} \quad \forall c \in L, \forall i \in V' \setminus \{s\} \quad (2.3.4)$$

$$M_{c,i} \ell_{i,c} \geq \sum_{(k,i) \in \delta^-(i): L(k,i)=c} x_{ki} \quad \forall c \in L, \forall i \in V' \setminus \{s\} \quad (2.3.5)$$

$$\sum_{c \in L} \ell_{i,c} \leq 1 \quad \forall v \in V \quad (2.3.6)$$

$$x_{ij} \geq 0 \quad \forall (i,j) \in E' \quad (2.3.7)$$

$$y_i, \ell_{i,c} \in \{0, 1\} \quad \forall i \in V' \quad (2.3.8)$$

M_i and $M_{c,i}$ represent the size of the maximum monochromatic connected component containing the node i in G and the size of the maximum connected component whose color is c containing i respectively. Constraints (2.3.2) are the classical conservation flow constraints. The constraints (2.3.3) guarantee that y_i assume value equal to 1 if the edge (s, i) is selected; the constraints (2.3.4) and (2.3.5) guarantee that the variable $\ell_{i,c}$ assume value equal to 1 if at least one edge (k, i) whose label is c is selected and zero otherwise. Finally, constraints (2.3.6) ensures that all the edges incident to a node i have the same label.

M2. The next mathematical formulation M2, is very compact, and very easy to understand. This model selects the edges composing the solution. The boolean variable x_{ij} is equal to one if and only if the model selects the edge (i, j) . To count the number of connected components present in the solution, in order to describe the objective function an easy subtraction is used. Obviously if each connected component included in the solution is acyclic the number of components is equal to $n - \sum_{(i,j) \in E} x_{ij}$ where $n = |V|$ (2.3.9). To ensure the subcycle elimination, the model uses the Miller-Tucker-Zemlin family constraints [24]. We use an integer variable T_i associated with each nodes $i \in V$. The model is able to selects the edge x_{ij} if and only if $T_i > T_j$ (2.3.14). Now to finish this model, it is just necessary to ensure that no edges with different colors and incidents on the same vertex are selected at same time. We use the variable ℓ_{ik} to guarantee this property. ℓ_{ik} is equal to one if the model selects an edge, incident on $i \in V$ of color $k \in C$ (2.3.10) (2.3.11). Using the variable ℓ_{ik} it is easy to apply a limitation on the number of used color for edge (2.3.13).

$$z = \min \left(n - \sum_{(i,j) \in E} x_{ij} \right) \quad (2.3.9)$$

with the constraints:

$$\ell_{ik} \geq x_{ij} \quad \forall (i, j) \in E : k = L(i, j) \quad (2.3.10)$$

$$\ell_{ik} \geq x_{ji} \quad \forall (i, j) \in E : k = L(i, j) \quad (2.3.11)$$

$$\sum_{k \in C} \ell_{ik} \leq 1 \quad \forall i \in V \quad (2.3.12)$$

$$\sum_{(i, j) \in E} x_{ij} \leq 1 \quad \forall i \in V \quad (2.3.13)$$

$$n * x_{ij} - (n - 1) \leq T_i - T_j \quad \forall (i, j) \in E \quad (2.3.14)$$

$$x_{ij} \in \{0, 1\} \quad \forall (i, j) \in E \quad (2.3.15)$$

$$c(i, k) \in \{0, 1\} \quad \forall i \in V, k \in C \quad (2.3.16)$$

$$0 \leq T_i < n \quad \forall i \in V \quad (2.3.17)$$

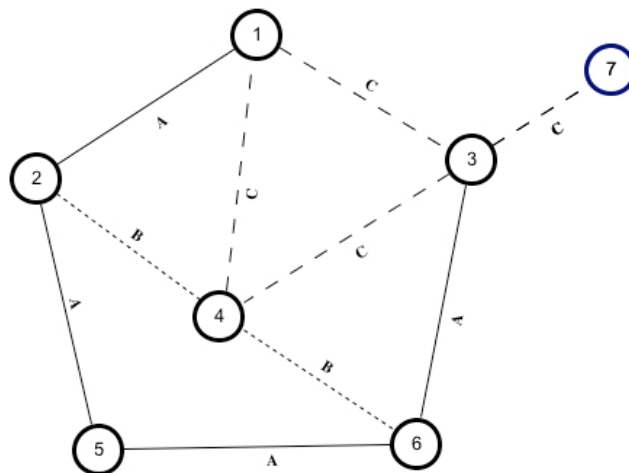


Figure 2.2: Input Graph.

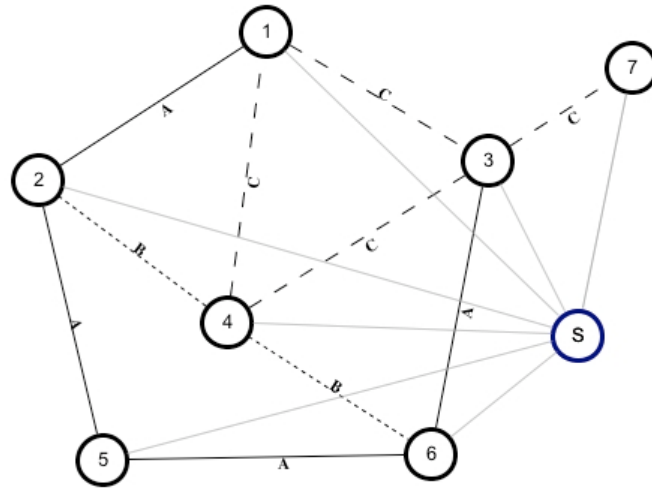


Figure 2.3: M1 resulting Graph.

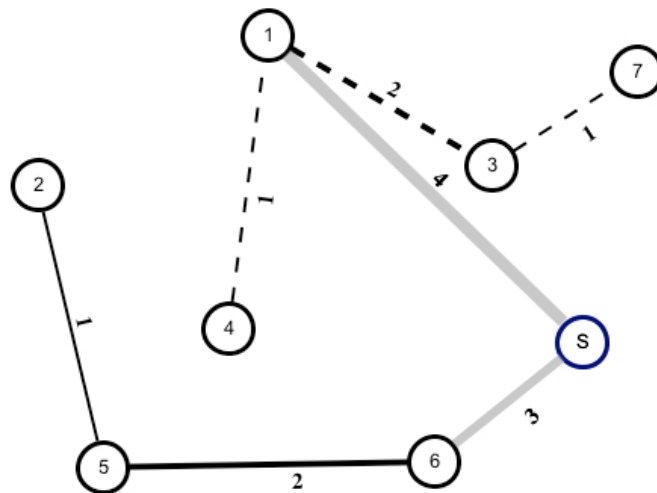


Figure 2.4: M1 Solution.

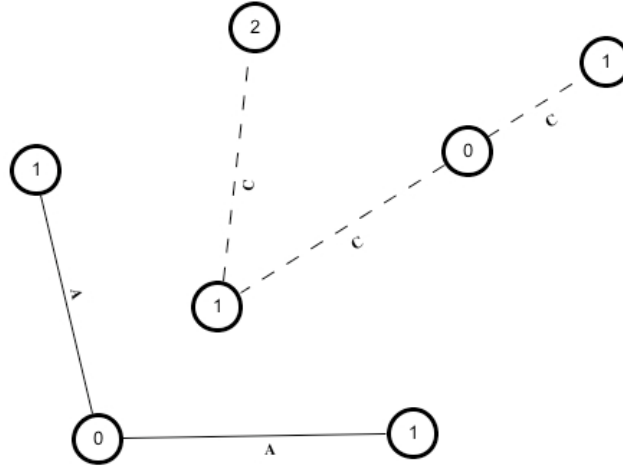


Figure 2.5: M2 Solution.

2.4 Polynomial case

In this section we will study the relation between the complexity of our problem (*MMP*) and the characteristics of the input instances. In particular we prove that is possible solve *MMP* in polynomial time on acyclic graphs. Obviously each connected component of an acyclic graph is a tree. The solution is created by combining the optimal solution of each connected component, still optimal for the original instance. For this reason in the following section we introduce a polynomial time algorithm (*TreeSolver*) able to produce the optimal solution if the input graph is a tree.

Before describing the algorithm, we introduce some further definitions:

Input Graph $G(V, E, C)$.

- Let $F(v)$ be the set of nodes adjacent to v with degree equal to 1, i.e. $F(v) = \{v' \in V : (v, v') \in E, \delta(v') = 1\}$
- Let $F(v, c)$ be the set of nodes adjacent to v , by an edge of color c and degree

equal to 1 $F(v, c) = \{v' \in F(v) : \mathcal{C}(v, v') = c\}$

- Let F_0 be the set of nodes having only adjacent node with degree equal to 1, i.e. $F_0 = \{v \in V : |F(v)| = |\delta(v)|\}$
- Finally, let F_1 be the set of nodes having one adjacent node with degree greater than 1, the father of v that is denoted by $\mathcal{R}(v)$, and all the remaining adjacent nodes with degree equal to one $F_1 = \{v \in V : |F(v)| + 1 = |\delta(v)| > 1\}$

TreeSolver

The idea at base of this algorithm is that we can select, iteration after iteration, edges surely included in an optimal solution. After a brief description of *TreeSolver* we present its pseudocode (2.6). Fixing a vertex randomly as root of this tree, we place the vertices position sorted by level, accord to the distance to the root. In level n we identify the leaves. Step by step, this algorithm selects a vertex at the level $(n - 1)$, and for this vertex selects a subset of incident edges X . It inserts all the edges of X in the solution after it removes them from the input tree.

The first step of the procedure is to initialize the set Sol . In the second step start the main loop, now iteration by iteration a vertex $v \in F_1$ is selected. It identifies $c \in C$, the color associated with the maximum number of edges in $\delta(v)$. Now we identify two different cases.

- Case 1. There is a predominant color.

<i>SolveTree</i>
<pre> 1: $Sol \leftarrow \emptyset$ 2: while $\exists v \in F_1$ do 3: $c \leftarrow \arg \max_{l \in C} F(v, l)$ 4: $p \leftarrow \mathcal{R}(v)$ 5: if $F(v, c) > F(v, \mathcal{C}(v, p))$ then {Case 1} 6: $Sol \leftarrow Sol \cup \{(v, v') \in E : v' \in F(v, c)\}$ 7: $V \leftarrow V \setminus (F(v) \cup \{v\})$ 8: $E \leftarrow E \setminus \delta(v)$ 9: else {Case 2} 10: $Sol \leftarrow Sol \cup \{(v, v') \in E : v' \in F(v, \mathcal{C}(v, p))\}$ 11: $V \leftarrow V \setminus F(v)$ 12: $E \leftarrow E \setminus \{(v, v') \in E : v' \in F(v)\}$ 13: end if 14: end while 15: if $\exists v \in F_0$ then 16: $c \leftarrow \arg \max_{l \in L} C(v, l)$ 17: $Sol \leftarrow Sol \cup \{(v, v') \in E : L(v, v') = c\}$ 18: end if 19: return Sol </pre>

Figure 2.6: TreeSolver

i.e. $|F(v, c)| \geq |F(v, l)| \forall l \in C$ and $|F(v, c)| > |F(v, \mathcal{C}(v, \mathcal{R}(v)))|$. In this case all the edges (v, v') with $v' \in F(v, c)$ are introduced in Sol and v and the nodes in $F(v)$ are removed from the tree. In figure 2.7 the result of this step is shown. There is exactly one predominant color (figure 2.7a). Consequently, *TreeSolver* inserts in S the edges (v, v_1) , (v, v_3) and (v, v_4) and it removes the nodes $v, v_1, v_2, v_3, v_4, v_5$ from the tree (figure 2.7b).

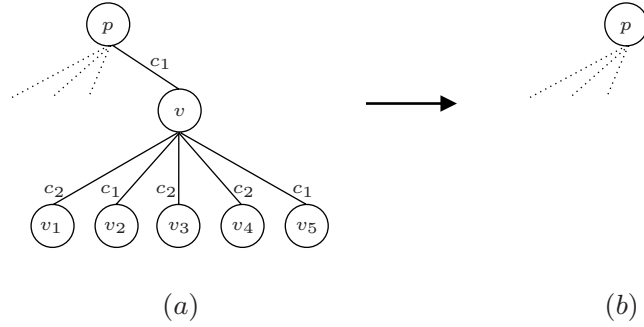


Figure 2.7: Case 1 of while loop.

- Case 2. There is no predominant color.

In this case it selects the color equal to $\mathcal{C}(v, \mathcal{R}(v))$. All the edges (v, v') with $v' \in F(v, c)$ are introduced in Sol and the nodes in $F(v)$ are removed from the tree. In figure 2.8 the result of this step is shown. There are two predominant colors, c_1 and c_2 (figure 2.7a), but since $\mathcal{C}(v, p) = c_1$ the color c_1 is selected. Consequently, *TreeSolver* inserts in S the edges (v, v_2) and (v, v_4) and it removes the nodes v_1, v_2, v_3, v_4, v_5 (figure 2.8b).

At the end of the while loop, $F_1 = \emptyset$. Now $F_0 = \emptyset$ means that we have removed

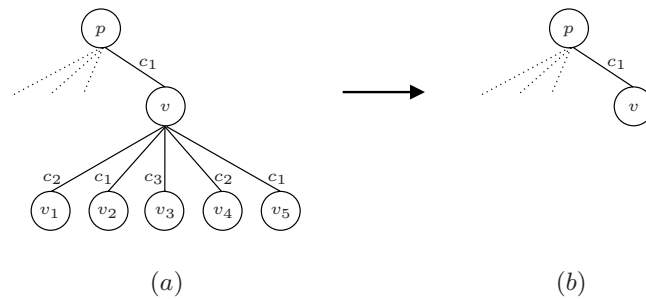


Figure 2.8: Case 2 of while loop.

all the edges. Otherwise there is just a vertex $v \in V_0$, in which case the algorithm computes the predominant color in v and it inserts in S the corresponding edges.

2.4.1 Solution's Optimality

Using the induction, we will prove the solution's optimality. The first step is to introduce a lemma. It will be used to ensure the termination of the algorithm:

Lemma 2.4.1. *The node v is an internal node if and only if it isn't a leaf.*

After each iteration of TreeSolver the number of internal nodes in the tree is decreased by one or two units.

Proof Let v be the internal node selected at the generic iteration k of the algorithm. If we are in *Case 1*, we remove v and all the leaf nodes connected to v from the tree. In the resulting tree we have at least one internal node less, exactly v (Figure 2.7a), but if $|\delta(\mathcal{R}(v))| = 1$ we have removed two internal nodes. If we are instead in *Case 2*, the procedure removes all the leaf nodes connected to v from the tree. Now v is a leaf, and we have just removed an internal node (Figure 2.8b). \square

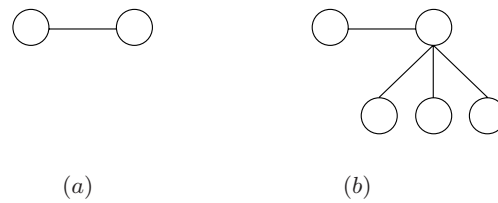


Figure 2.9: Base of the Induction

Theorem 2.4.2. *Using a connected acyclic graph $G(V, E, C)$, as input to the *TreeSolver* algorithm, produces a set of monochromatic connected components of cardinality equal to the optimal solution to the MMP problem.*

Proof This proof is based on the induction method. The induction is defined on the number of internal nodes in G . The base of the induction is defined below, it is composed of two different cases.

1. Zero internal nodes

In this case we can have a graph composed by one node, or two nodes connected by an edge (Figura 2.9a). In both cases the optimal solution is equal to one. Step 15 of *TreeSolver* is able to find it.

2. One internal node (Figura 2.9b)

In this case the optimal solution is to select the most frequent color. Step 15 of *TreeSolver* is able to find it.

Using the inductive hypothesis we can assume that *TreeSolver* is able to produce the optimal solution in each graph with a number less than or equal to k of internal nodes. Now we try to prove that *TreeSolver* found the optimal solution in a graph with

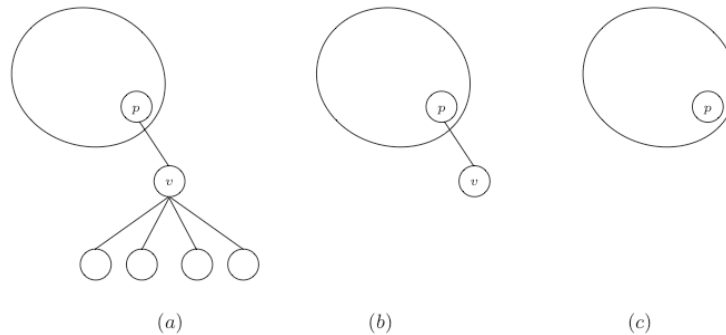


Figure 2.10: Inductive step

$k+1$ internal nodes. Let v be the first node in G selected by *TreeSolver*(Figure 2.10a). After the first iteration, using the lemma 2.4.1 we can say that in the resulting graph G' , the number of internal nodes is decreased by one or two units. We can have two cases:

C1: v is present in the resulting graph (Figure 2.10b).

C2: v isn't present in the resulting graph (Figure 2.10c).

We can analyze the two cases *C1* and *C2* separately.

C1: Let Sol^* be an optimal solution for the input graph G . We can write: $|Sol^*| = r + h - 1$ where h is the number of components present in the optimal solution in the sub-tree rooted in v , and r is the number of components present in the optimal solution in the residual tree G' . Let $S(G')$ and $S(G_v)$ be the solution computed by the procedure respectively in residual tree G' and in the sub-tree rooted in v . We can define $p = |S(G')|$ and $q = |S(G_v)|$. Using the inductive hypothesis we can say that $p \leq r$ and taking into account the greedy choice of the procedure we can say that $h \leq q$. Combining $S(G')$ and $S(G_v)$ in a feasible solution of G we have $S(G) = S(G') \cup S(G_v)$ and obviously $|S(G)| = pq - 1$.

The -1 depends on the color incident on v . In both solutions the color incident in v is the same and for this reason we can merge the two components including v in $S(G')$ and $S(G_v)$.

$C2$ is in turn divided in two sub-case $C2-a$ and $C2-b$.

- There exists an optimal solution on G that don't use the edge (p, v) .

Using the inductive hypothesis and taking into account the greedy choice of the procedure we can say that this procedure generates an optimal solution for the input graph G .

- All the optimal solutions on G use the edge (p, v) .

$|Sol^*| = r + h - 1$, $p = |S(G')|$ and $q = |S(G_v)|$. Obviously $h \leq q$ and $p \leq r$ but since the procedure doesn't use the edge (p, v) then $q < h$. The solution produced by the procedure is $S(G) = S(G') \cup S(G_v)$ and $|S(G)| = p + q < r + h \Rightarrow p + q \leq r + h - 1$. \square

2.5 Genetic Approach

In this section we use a genetic algorithm to produce an heuristic solution for the NP-Hard instances of the problem. In particular we adapt the OMEGA (1) GA structure to the *MMP* problem. This algorithm is very similar to the version proposed in the first chapter moreover, the two problems are strictly related. Despite this similarity, for this particular problem the structure of the chromosome and the mutation function try to take advantage of the specific features of the problem.

between one and four. Now we need an algorithm that takes a chromosome as input and produces a spanning tree as output. In figure 2.12 is presented an algorithm able to produce a spanning tree using a chromosome as input.

The algorithm builds the set $Cost$, associating to each edge a negative integer representing the occurrences of the label in the chromosome. In addition, for all the edges not presents in the chromosome, if they connect two vertices that are monochromatic in the chromosome, or are monochromatic with a vertex that is not present, it insert the edge as it is one time present in the chromosome.

After using a Minimum Weight Spanning Tree procedure (MST) it is possible to build the solution. The MST takes into account as cost for each edge $e : Sol(L(e))$.

<i>TreeCreator</i>
<i>INPUT: $G(V, E, L)$, chromosome c</i>
<i>OUTPUT: $Sol \in E$ Sapnning tree of G</i>
<pre> 1: $Cost \leftarrow \emptyset$ 2: for all edge $e \in c$ do 3: if $\exists (e, i) \in Cost$ then {Case 1} 4: $Cost \leftarrow \{(e, i - 1)\} \cup Cost \setminus \{(e, i)\}$ 5: else {Case 2} 6: $Cost \leftarrow \{(e, -2)\} \cup Cost$ 7: end if 8: end for 9: while is possible add edge to $Cost$ do 10: select $(v_1, v_2) \in E$ such that $\forall ((v_1, v), i) \in CostL((v_1, v)) = l$ for any $v \in V i < 0$ 11: if $\forall ((v_2, v), i) \in CostL((v_2, v)) = l$ for any $v \in V i < 0$ then 12: $Cost \leftarrow \{((v_1, v_2), -1)\} \cup Cost$ 13: end if 14: if $\exists ((v_2, v), i) \in Cost$ for any $v \in V i < 0$ then 15: $Cost \leftarrow \{((v_1, v_2), -1)\} \cup Cost$ 16: end if 17: end while 18: $Sol \leftarrow MST(G(V, E, L, Cost))$ 19: return Sol </pre>

Figure 2.12: Spanning Tree Generator

2.5.2 The Crossover

The crossover is the easiest function of OMEGA's architecture. Basically the function takes as input two chromosomes $c1, c2$ and it produces as output the chromosome $c3$. The procedure randomly selects 50% of the blocks of $c1$ and $c2$, and it inserts the blocks into the new chromosome $c3$. In figure 2.13 is shown a graphical example.

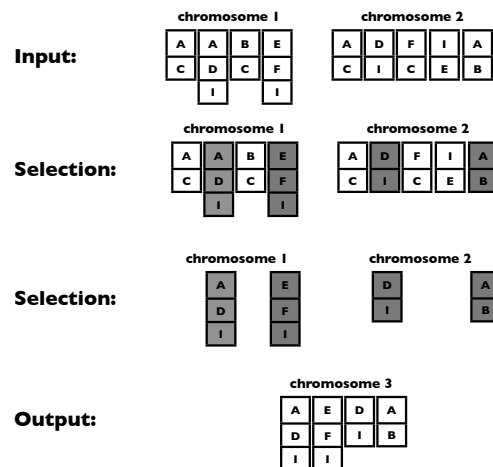


Figure 2.13: *Crossover*

2.5.3 The Mutation

There are three different kind of mutation MUT-ADD, MUT-DEL and MUT-CHANGE. The mutation operator randomly runs one of these three, if isn't possible to use MUT-ADD or MUT-DEL it uses MUT-CHANGE.

Respectively the chances of selecting the operators are: 50%, 30%,20%.

- MUT-ADD randomly select a block in the chromosome and if it isn't full, adds

another edge in the last position. Let (v_1, v_2) be the first edge of the block; the new edge is selected from the set of edges $(v_1, v) \in E$.

- MUT-DEL randomly selects a block in the chromosome, and if its dimension is greater than one, it removes a random edge from the block.
- MUT-CHANGE randomly selects a block in the chromosome, randomly selects an edge from the block and replace it with a new edge. Let (v_1, v_2) be the first edge of the block; the new edge is selected from the set of edges $(v_1, v) \in E$.

2.5.4 The Splitting Population Function

The Splitting Population Function (SPF) is used to partition the original population into k sub-populations. This procedure is totally random, and if h is the dimension of the original population, it produces k new populations, each one with a dimension equal to $\frac{h}{k}$. The aim of this procedure is to move the chromosomes present in a unique big population, into k different new populations.

2.5.5 The Fitness Functions

The fitness function takes in input a chromosome c , using the procedure 2.12 produce a spanning tree SP that is composed of $|V| - 1$ edges of the original graph $G(V, E, L)$. For each non-monochromatic vertex, it selects the most frequent color and it removes all the edges of different colors. The procedure returns the number of connected components produced by this process. Now we want to identify other fitness functions related to the first one. The technique is very easy. The procedure associates a penalty cost to a defined color on a defined vertex. If the dominant color of this

vertex is the forbidden color, an extra cost is added to the function. Usually the procedure identifies some good solutions in the current population and tries to forbid the selection of one or more used labels for particular vertices. The penalty cost is usually greater than 1 and less than 3.

2.5.6 Pseudocode

<i>BASIC-GA</i>
<i>INPUT: $G(V, E, L)$ and Population P and Fitness function ft</i>
<i>OUTPUT: Population P</i>
<pre> 1: $Cost \leftarrow \emptyset$ 2: for it iterations do 3: for all c in P les the best tree do 4: select randomly $c1 \in P$ such that $ft(c) > ft(c1)$ 5: $P \leftarrow (P \setminus \{c\}) \cup \{crossover(c1, c)\}$ 6: end for 7: for all c in P les the best one do 8: $P \leftarrow (P \setminus \{c\}) \cup \{mutation(c)\}$ 9: compute $ft(c)$ 10: end for 11: end for 12: return P </pre>

Figure 2.14: Basic Genetic Algorithm

2.6 Results

This section presents a preliminary test phase on small instances (15 nodes, number of colors between 3 and 10 and graph density between 0.25 and 1). We solved the problem to optimality by means of the CPLEX solver using both the Single Commodity flow and the MTZ formulations and compared the obtained results with our OMEGA algorithm. While the complexity of the Single Commodity formulation

Instances	Flusso		MTZ		Genetic	
	Value	Time	Value	Time	Value	Time
15_25_3_1101	4	0,71	4	0,03	4	1,30417
15_25_3_1109	3	5.31	3	0,04	3	1,34739
15_25_3_1117	4	11.16	4	0,05	4	1,36868
15_25_3_1125	4	0,67	4	0,04	4	1,30625
15_25_3_1133	5	1.26	5	0,1	5	1,24156
15_25_10_1221	7	2,78	7	0,04	7	1,0552
15_25_10_1229	7	8.24	7	0,03	7	1,04701
15_25_10_1237	7	3.07	7	0,06	7	1,05988
15_25_10_1245	6	8.38	6	0,06	6	1,12753
15_25_10_1253	6	10.04	6	0,06	6	1,14058
15_1_3_1581	1	21.36	1	3,59	1	8,4705
15_1_3_1589	1	1.26	1	0,8	1	8,4547
15_1_3_1597	2	4320.14.24	1	2,95	1	8,51108
15_1_3_1605	2	4320.14.24	1	1,17	1	8,55474
15_1_3_1613	2	4320.14.24	1	2,23	1	8,45683
15_1_10_1701	2	2211.21.36	2	74,26	3	8,28533
15_1_10_1709	2	147,68	2	70,03	3	8,40188
15_1_10_1717	2	4320.43.12	2	71,91	2	8,45794
15_1_10_1725	2	4320.14.24	2	15,2	2	8,40143
15_1_10_1733	2	918.57.36	2	20,72	2	9,4352

<i>OMEGA</i>
<i>INPUT: $G(V, E, L)$</i>
<i>OUTPUT: $Sol \in E$ Spanning tree of G</i>
<pre> 1: $POP = \{P_0, P_1, \dots, P_{10}\}$ set of populations 2: $P_0 = h$ 3: $P_1 = P_2 = \dots = P_{10} = 0$ 4: while Not End Conditions do 5: <i>BASIC-GA</i>(G, P_0, f_0) 6: if new best solution in P_0 then 7: set new best solution 8: end if 9: $SPF(P_0) \implies P_0 \simeq P_1 \simeq \dots \simeq P_{10} \simeq \frac{h}{10}$ 10: for all $P \in POP$ do 11: <i>BASIC-GA</i>(G, P, f_{random}) 12: end for 13: $P_0 \leftarrow P_1 \cup P_2 \cup \dots \cup P_{10}$ 14: $P_1 \leftarrow P_2 \leftarrow \dots \leftarrow P_{10} \leftarrow \emptyset$ 15: end while </pre>

Figure 2.15: *MMP OMEGA*

grows significantly even on these small instances, the MTZ formulation has much better performances. Our OMEGA approach reaches the optimal solution value in most of the instances and differs at most of a value of 1 on the others, while still providing fast computational times.

Chapter 3

Bounded Degree Spanning Tree

3.1 Introduction

In this chapter we focus our attention on telecommunication network problems, with emphasis on optical networks. These networks require specific constraints to be modeled in order to take into account their particular physical characteristics, such as the propagation of the light in the optical fiber. In particular, in an optical network, the wave division multiplexing technology allows to propagate different light beams on the same optical fiber, as long as they use a different fixed wavelength. In this kind of networks multicast technology permits to replicate the optical signal from one source to many destination nodes by means of a network device (switch) that permits to replicate a signal, splitting light. Many applications, such as world wide web browsing, video conferences etc., require such a technology for efficiency purposes. Such application often require the individuation of connected sub-networks such as the spanning trees (ST). There are several variants of the ST problem that are useful to model problems arising in communication networks. For example, the network may be required to connect a specified subset of nodes (Steiner Tree Problem

[22]); if a measure is assigned with each link, one could be interested in looking for homogeneous subgraphs of the network (Minimum Labelling Spanning Tree Problem [9, 7]); in optical networks it is useful to connect the nodes in a way such that the number of connections of each node is limited (Bounded Degree Spanning Tree) or the number of connection of each node influence the objective function (Spanning Tree with Minimum Number of Branch Nodes [18, 19]). In this chapter we focus our attention on the *spanning tree with Minimum number of Branch Vertices* problem or MBV and on two variants. In the optical networks the MBV problem is used to minimize the number of required light splitting device (switch). This is very important both to minimize the costs and to preserve the quality of the signals. In the following we illustrate the three problems: Given a connected graph G a vertex is said to be branch if its degree is greater than 2. We consider three problems arising in the context of optical networks: (i) finding a spanning tree of G with the minimum number of branch vertices (ii) finding a spanning tree of G such that the degree sum of the branch vertices is minimized. (iii) Let d the degree sum of the branch vertices let b the number of branch vertices, finding a spanning tree of G such that $(d - (2 * b))$ is minimized. In this work for these NP-hard problems we analyze the relation between each other, provide a single commodity flow formulation to solve the problems by means of a solver and a genetic metaheuristic. We compare the solutions of the solver with the solutions of the genetic metaheuristic and of other heuristics available in literature [8].

3.2 Problem definition and motivations

A vertex of a graph is a *branch vertex* if and only if its degree is greater than two, in the fig 3.1 the vertices labeled B.

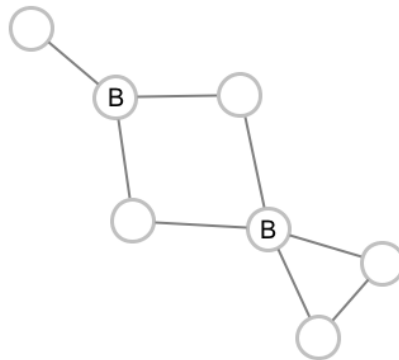


Figure 3.1: *branch vertex* example

Given a connected graph G . We can define the **minimum branch vertices problem** (*MBV*) as following.

(*MBV*): finding a spanning tree of G with the minimum number of *branch vertices*.

The light splitting device (switch), are hardware components used to propagate light beams from a single source to two different destinations. In a real network the number of this kind of devices is imitated, and for this reason is important to minimize the number of used device to allow the maximum number of possible connections. The nodes of the tree whose degree is greater than 2 are our branch nodes. Switches are

located on such branch nodes of the tree.

Such a problem has been addressed in Gargano et al. [19] where the computational complexity of the problem is studied and shown to be NP-complete on general graphs and cubic graphs.

Moreover, we introduce a related problem, that is more suitable to model real costs of such location problem on optical networks.

The definition of the **Minimum Degree Sum Problem** (*MDS*) is:

(*MDS*): finding a spanning tree of G such that the degree sum of the *branch vertices* is minimized.

This problem was introduced by Cerulli et al. in [8]. The reason of the introduction of this new problem are motivated by the necessity of create a model closest to the real problem defined on optical networks. Indeed, many devices can only duplicate laser beams, and the effective number of devices to be located on a branch node, in order to replicate lights, is directly related to the number of edges incident to the node. In a branch vertices the number of optical switch needed to duplicate the signal is strictly related to the degree of the vertex and in particular, if the degree of the vertex v is equal to x , we need exactly $x - 2$ light splits Fig. 3.2.

Let a spanning tree T of a graph G we define $sd(T)$ of T as the sum of the degree of the branch vertices. We define $nb(T)$ as the number of branch vertices of T . Now we define the third problem introduced in the previous chapter as the identification of the spanning tree ST of G that minimizes $sd(T) - 2 * nb(T)$ (ML). The motivation behind this objective function is that each branch vertex v of degree x requires exactly $x - 2$ switches. This problem was well approached in the paper [28]. In this chapter

two approximate algorithms are presented. In the general case a 2-OPT algorithm is presented. In the following we prove that it is equivalent to look for the spanning tree with the minimum number of leaves (ML); this one is a well known NP-HARD problem.

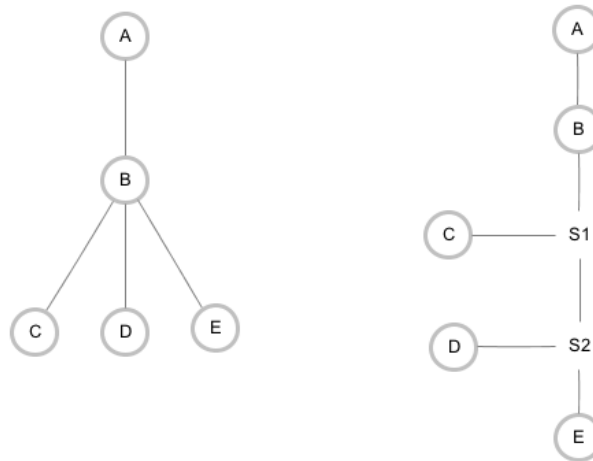


Figure 3.2: *switch* example

3.3 Mathematical Formulation

Let us consider an undirected network $G = (V, E)$, where N denotes the set of n vertices and E the set of m edges. We consider the oriented version of the graph where two oriented arcs (u, v) and (v, u) are associated to each edge $(u, v) \in E$. We denote by E' this new set of edges. The set of variables for the model are the following:

- binary variable x_e for each $e \in E$ that assumes value equal to 1 if arc e is selected and value equal to 0 otherwise;

- binary variable y_v for each $v \in V$ that assumes value equal to 1 if vertex v is of the branch type (that is its degree, as vertex of the tree, is greater than two), and is equal to 0 otherwise.

We denote by $A(v)$ the set of incident edges to vertex v and by δ_v its size, i.e. $\delta_v = |A(v)|$. Let us denote by $A^+(v)$ and $A^-(v)$, the set of edges in E' outgoing from v and incoming into v , respectively. A spanning tree T of G can be found by sending from a source vertex $s \in V$ one unit of flow to every other vertex $v \in V \setminus \{s\}$ of the graph [1]. We introduce both the binary variable x_{uv} for each $(u, v) \in E'$ that assumes value equal to 1 if edge (u, v) is selected and value equal to 0 otherwise and the flow variable f_{uv} for each $(u, v) \in E'$ representing the flow going from vertex u to vertex v .

The single commodity flow formulation (SC) of MBV is then the following [8]:

$$\min \sum_{v \in V} y_v \quad (3.3.1)$$

s.t. :

$$\sum_{(u,v) \in A^-(v)} x_{uv} = 1 \quad \forall v \in V \setminus \{s\} \quad (3.3.2)$$

$$\sum_{(s,v) \in A^+(s)} f_{sv} - \sum_{(v,s) \in A^-(s)} f_{vs} = n - 1 \quad (3.3.3)$$

$$\sum_{(v,u) \in A^+(v)} f_{vu} - \sum_{(u,v) \in A^-(v)} f_{uv} = -1 \quad \forall v \in V \setminus \{s\} \quad (3.3.4)$$

$$x_{uv} \leq f_{uv} \leq (n-1)x_{uv} \quad \forall \{u,v\} \in E' \quad (3.3.5)$$

$$\sum_{(v,u) \in A^+(v)} x_{vu} + \sum_{(u,v) \in A^-(v)} x_{uv} - 2 \leq \delta_v y_v \quad \forall v \in V \quad (3.3.6)$$

$$y_v \in \{0, 1\} \quad \forall v \in V \quad (3.3.7)$$

$$x_{uv} \in \{0, 1\} \quad \forall (u,v) \in E' \quad (3.3.8)$$

$$f_{uv} \geq 0 \quad \forall (u,v) \in E' \quad (3.3.9)$$

The objective function (3.3.1) is the minimization of the total number of branch vertices. Constraints (3.3.2) ensure that each vertex in the optimal spanning tree has exactly one incoming edge. Equations (3.3.3) and (3.3.4) balance the flow at each vertex and ensure the connectivity of any feasible solution. Constraints (3.3.5) are coupling constraints linking the flow variable f with the binary variables x . The coupling constraints (3.3.6) ensure each variable y_v to be equal to 1, whenever v has more than two adjacent edges belonging to the optimum spanning tree.

The mathematical formulation for MDS requires the additional integer decisional variables counting the degree of the branch vertices of the solution:

$$z_v = \begin{cases} 0, & \text{if } v \text{ is not branch;} \\ \delta_T(v), & \text{otherwise.} \end{cases} \quad (3.3.10)$$

The mathematical model for MDS requires to minimize the objective function

$$\min \sum_{v \in V} z_v \quad (3.3.11)$$

subject to constraints (3.3.2) - (3.3.9) and the additional constraints

$$\sum_{(u,v) \in A^-(v)} x_{uv} + \sum_{(v,u) \in A^+(v)} x_{vu} - 2 + 2y_v \leq z_v, \quad \forall v \in V \quad (3.3.12)$$

Using the same mathematical model that define the MBV problem, we can model the ML problem. It is important change the constraints 3.3.6 and modify the sense of the variables y .

- binary variable y_v for each $v \in V$ that assumes value equal to 0 if its degree, as vertex of the tree, is greater than one, and is equal to 1 otherwise.

$$\min \sum_{v \in V} y_v \quad (3.3.13)$$

s.t. :

$$\sum_{(u,v) \in A^-(v)} x_{uv} = 1 \quad \forall v \in V \setminus \{s\} \quad (3.3.14)$$

$$\sum_{(s,v) \in A^+(s)} f_{sv} - \sum_{(v,s) \in A^-(s)} f_{vs} = n - 1 \quad (3.3.15)$$

$$\sum_{(v,u) \in A^+(v)} f_{vu} - \sum_{(u,v) \in A^-(v)} f_{uv} = -1 \quad \forall v \in V \setminus \{s\} \quad (3.3.16)$$

$$x_{uv} \leq f_{uv} \leq (n-1)x_{uv} \quad \forall \{u, v\} \in E' \quad (3.3.17)$$

$$\sum_{(v,u) \in A^+(v)} x_{vu} + \sum_{(u,v) \in A^-(v)} x_{uv} - 1 \geq 1 - y_v \quad \forall v \in V \quad (3.3.18)$$

$$y_v \in \{0, 1\} \quad \forall v \in V \quad (3.3.19)$$

$$x_{uv} \in \{0, 1\} \quad \forall (u, v) \in E' \quad (3.3.20)$$

$$f_{uv} \geq 0 \quad \forall (u, v) \in E' \quad (3.3.21)$$

The objective function (3.3.13) is the minimization of the total number of degree one vertices. Constraints (3.3.14) ensure that each vertex in the optimal spanning tree has exactly one incoming edge. Equations (3.3.3) and (3.3.16) balance the flow at each vertex and ensure the connectivity of any feasible solution. Constraints (??) are coupling constraints linking the flow variable f with the binary variables x . The coupling constraints (3.3.18) ensure each variable y_v to be equal to 1, whenever v has less than two adjacent edges belonging to the optimum spanning tree.

3.4 Relations among MBV, MDS and ML

The three problems are strictly related and in many cases a optimal solution for one of the three is optimal for one or both the other two problems. Our first intention is prove that there are graphs, having optimal solution for one of the problems that is not optimal for one or both the other two.

3.4.1 Notations

We define the following variables:

- V_i : subset of the main set of vertex V containing all nodes having degree i . As a consequence, as the greatest degree of a node is $n - 1$, we have that
$$\sum_{i=0}^{n-1} |V_i| = |V|$$
- V_B : subset of the main set of vertex V containing all nodes having degree greater than 2, so $V_B = \bigcup_{i=3}^{n-1} V_i$. Given the above definition, we have that
$$\sum_{i=3}^{n-1} |V_i| = |V_B|$$
- For each subset X of V , so that $X \subseteq V$, we define the function $S(X)$ as the sum of the degrees of the nodes belonging to the set X . From this definition we can see that $S(V_y) = y |V_y|$, for $0 \leq y \leq n - 1$, because all nodes in the subset V_y have degree y . We will see that the cases $y = 1$ and $y = 2$ (the subsets of nodes with degree 1 and 2 respectively) will have a particular importance.

3.4.2 The problems are not equivalent

Theorem 1. There is a subset of graphs whose optimal solution for MBV,MDS does not coincide with the optimal solution for ML.

Proof. In Figure 3.3 we can see an example of graph.

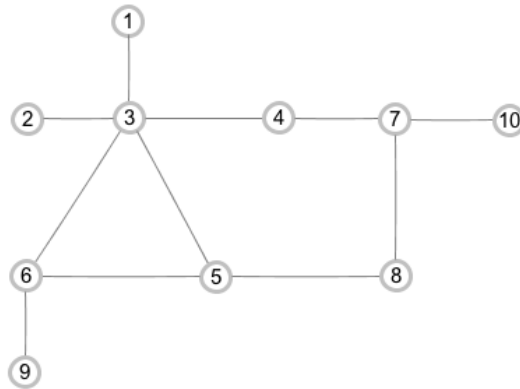


Figure 3.3: Example graph

For this graph we compute the optimal solution for the MBV and MDS problems Figure 3.4 and the solution for the ML problem Figure 3.5.

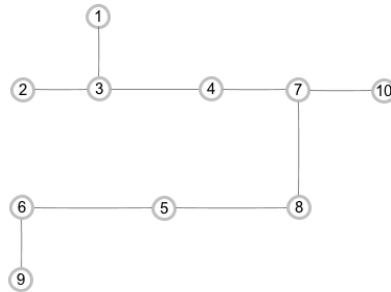


Figure 3.4: MBV MDS solution

The spanning tree ST_1 , optimal solution for the MBV and MDS is the same. In this tree we identify 1 branch vertex, with degree equal to 5. In the spanning tree

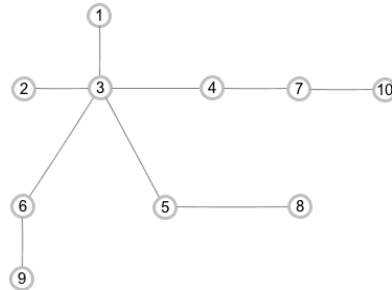


Figure 3.5: ML solution

ST2 solution for the ML we identify 2 branch vertices, with degree sum equal to 6. Is easy to see that the number of required switch for *ST1* is equal to 3 but the number of switch required for *ST2* is 2.

Theorem 2. There is a subset of graphs whose optimal solution for MBV does not coincide with the optimal solution for MDS.

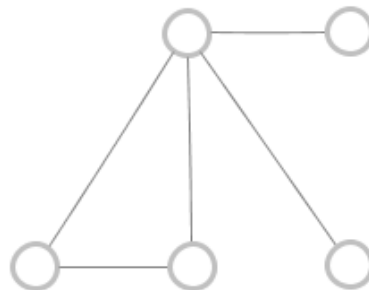


Figure 3.6: Example graph

Proof. Given the graph in figure 3.6, the spanning tree shown in picture 3.7 represents an optimal solution for the MBV problem, with value one. Anyway, it's

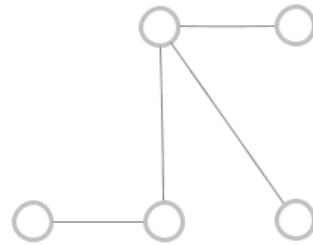
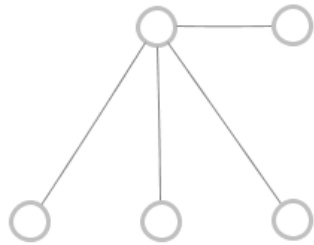


Figure 3.7: MBV optimal solution

Figure 3.8: MBV, MDS optimal solution

not an optimal solution for MDS, that is 4 in this spanning tree. The optimal solution for MDS is shown in picture 3.8 and its value is 3.

Theorem 3. There is a subset of graphs whose optimal solution for MDS does not coincide with the optimal solution for MBV.

Proof. Given the graph in figure 3.9, the spanning tree shown in figure 3.10 represents an optimal solution for the MDS problem (6). Anyway, it's not an optimal solution for MBV (2). The optimal solution for MBV is shown in figure 3.11 and his value is 1.

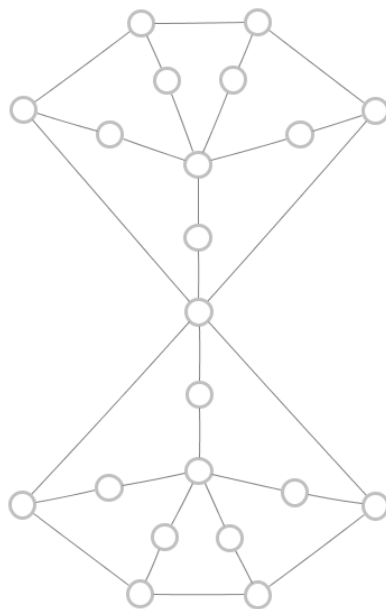


Figure 3.9: Example graph

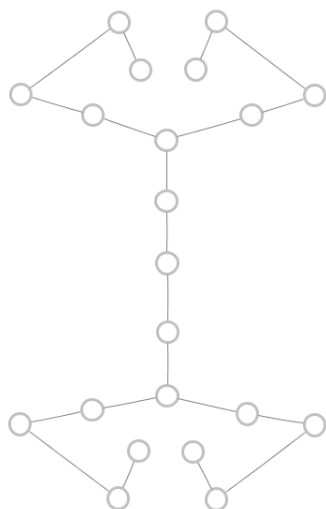


Figure 3.10: MDS optimal solution

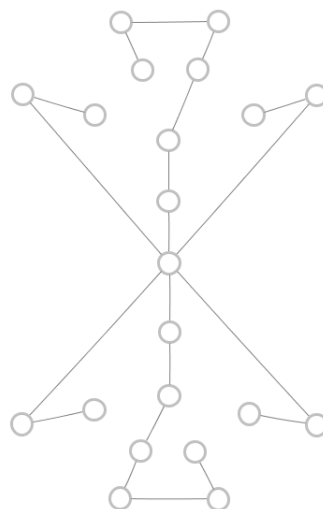


Figure 3.11: MBV, MDS optimal solution

3.4.3 Relations among the problems

Property 1. $S(V) = 2|V| - 2$

Proof. As a n -nodes' spanning tree has, as definition, exactly $n - 1$ edges, and each edge weighs upon two nodes, the sum of the degrees of all the nodes is $2(n - 1)$.

So we have that:

$$\begin{aligned} S(V) &= 2|E'| && \text{(From the spanning tree definition)} \\ &= 2(|V| - 1) && \text{(number of edges = number of nodes minus 1)} \\ &= 2|V| - 2 \end{aligned}$$

□

Property 2. $S(V_B) = 2|V| - 2 - |V_1| - 2|V_2|$

Proof. We have that:

$$\begin{aligned} S(V) &= S(V_1) + S(V_2) + S(V_B) && \text{(Because } V = V_1 \cup V_2 \cup V_B) \\ &= |V_1| + 2|V_2| + S(V_B) && \text{(Definitions)} \end{aligned}$$

Besides, from property 1:

$$S(V_B) = 2|V| - 2 - |V_1| - 2|V_2|$$

So we see that $S(V) = 2|V| - 2$ is constant, while $|V_1|$ and $|V_2|$ change depending on the morphology of the spanning tree: while they augment, $S(V_B)$ diminishes.

□

Property 3. $|V_1| = S(V_B) - 2|V_B| + 2$

Proof.

$$\begin{aligned}
S(V_B) &= 2|V| - 2 - |V_1| - 2|V_2| && \text{(by Property 2)} \\
|V_1| &= 2|V| - 2 - S(V_B) - 2|V_2| \\
&= 2(|V| - |V_2|) - 2 - S(V_B) \\
&= 2(|V_B| + |V_1|) - 2 - S(V_B) \\
&= 2|V_B| + 2|V_1| - 2 - S(V_B) \\
&= S(V_B) + 2 - 2|V_B|
\end{aligned}$$

This property shows how there is a strict relation between the degree of branch nodes and the number of degree 1 nodes.

□

Property 4. $S(V_B) = |V_B| - |V_2| - 2 + |V|$

Proof.

$$\begin{aligned}
S(V_B) &= 2|V| - 2 - |V_1| - 2|V_2| && \text{(by Property 2)} \\
&= 2|V| - 2 - S(V_B) - 2 + 2|V_B| - 2|V_2| && \text{(by Property 3)} \\
2S(V_B) &= 2|V| - 4 + 2|V_B| - 2|V_2| \\
S(V_B) &= |V| - 2 + |V_B| - |V_2|
\end{aligned}$$

This property shows the relation between the sum of the degree of branch nodes and the cardinality of branch and grade 2 nodes.

□

Now using these theorems and the previous notation, we are able to give another formulation of the objective function for all the analyzed problems.

The objective function of the MBV problem is:

$$\min |V_B| \quad (|V| = |V_1| + |V_2| + |V_B|) \quad (3.4.1)$$

$$\min |V| - |V_1| - |V_2| \quad (3.4.2)$$

$$(|V|) + \min - |V_1| - |V_2| \quad (3.4.3)$$

$$(|V|) - \max |V_1| + |V_2| \quad (3.4.4)$$

The objective function of the MDS problem is:

$$\min S(V_B) \quad (\text{by theorem 2}) \quad (3.4.5)$$

$$\min 2|V| - 2 - |V_1| - 2|V_2| \quad (3.4.6)$$

$$(2|V| - 2) + \min - |V_1| - 2|V_2| \quad (3.4.7)$$

$$(2|V| - 2) - \max |V_1| + 2|V_2| \quad (3.4.8)$$

The objective function of the ML problem is:

$$\min S(V_B - 2|V_B|) \quad (\text{by theorem 3}) \quad (3.4.9)$$

$$\min |V_1| - 2 \quad (3.4.10)$$

$$(-2) + \min |V_1| \quad (3.4.11)$$

$$(-2) - \max - |V_1| \quad (3.4.12)$$

Using the notation:

$$fo(\alpha, \beta) = \max \alpha |V_1| + \beta |V_1| \quad (3.4.13)$$

We are able to identify the optimization function to produce the optimal solution for all the three problems.

- *MBV*
 $fo(1, 1)$
- *MDS*
 $fo(1, 2)$
- *MBV*
 $fo(-1, 0)$

3.5 OMEGA Algorithm

In this section we introduce a GA that can be used for all the variants of the optical splits problem. The main characteristic of this problem that suggests us to use an OMEGA algorithm is that all the three variants are strictly related and a good solution for one of the variants is a good solution for the other two. In other words we can use the three different fitness functions (defined for the previously problems) for the different populations of OMEGA.

The largest part one of the structures and solutions used in this section are identical to the standard OMEGA algorithm presented in the previous chapter.

The Chromosome

The crossover is exactly the same used in the previous chapter. Obviously there is a big difference in the chromosome evaluation, because the MBV problem is not defined on a colored graph. To produce a spanning tree from a chromosome we use a minimum weight spanning tree algorithm. This algorithm counts the repetition of the colors

associated to the edges. Now we use the same technique but to compute the weight associated to each edge we count the repetitions of the edge in the chromosome.

The Crossover

Basically the function takes as input two chromosomes $c1, c2$ and produce as output the chromosome $c3$. The procedure randomly selects 50% of the blocks of $c1$ and $c2$, and inserts the blocks into the new chromosome $c3$.

The Mutation

There are three different kinds of mutation: MUT-ADD, MUT-DEL and MUT-CHANGE. The mutation operator randomly runs one of these three, if it isn't possible to use MUT-ADD or MUT-DEL it uses MUT-CHANGE.

Respectively, the chances of selecting the operators are: 50%, 30%,20%. This three operators are the most important difference between the previous OMEGA and this algorithm. The characteristic is that each blocks represents a little connected components of the input graph.

- MUT-ADD randomly selects a block in the chromosome and if it isn't full, adds another edge in the last position. The new edge is selected from the set of edges near the connected structure, present in the same block.
- MUT-DEL randomly selects a block in the chromosome and if its dimension is greater than one, it remove a random edge from the block.
- MUT-CHANGE randomly selects a block in the chromosome, randomly selects an edge from the block, and replaces it with a new edge. The new edge is

selected from the set of edges near the connected structure, present in the same block.

The Fitness Functions

The fitness function used in this section depends on the specific problem that we try to solve. If we count the number of vertices of degree one in the spanning tree V_1 and the number of vertices of degree two V_2 , we can define:

(MBV): $\text{MAX } V_1 + V_2$

(MDS): $\text{MAX } V_1 + 2 * V_2$

(Minimum Leaves): $\text{MAX } V_1 + 2 * V_2$

A generic problem is:

$\text{MAX } \alpha * V_1 + \beta * V_2$

modifying this two parameters α and β we are able to produce different fitness functions.

Instances	MBV		MDS		Genetic			
	Value	Time	Value	Time	MDS		MBV	
					Value	Time	Value	Time
istanza_n=30_d=130_a=39_1.txt	4	0.02000	17	0.00001	17	13,86	5	7,00
istanza_n=30_d=130_a=39_2.txt	2	0.03000	15	0.00001	16	7,34	2	5,30
istanza_n=30_d=130_a=39_3.txt	3	0.02000	16	0.00001	16	7,49	3	8,40
istanza_n=30_d=130_a=39_4.txt	4	0.01000	18	0.00001	19	8,32	5	5,36
istanza_n=30_d=130_a=39_5.txt	5	0.07000	19	0.00001	19	9,11	5	5,30
istanza_n=30_d=150_a=45_1.txt	3	0.03000	15	0.00001	17	6,96	3	7,81
istanza_n=30_d=150_a=45_2.txt	2	0.08000	10	0.00001	12	6,42	2	0,03
istanza_n=30_d=150_a=45_3.txt	2	0.04000	13	0.00001	13	7,16	2	3,68
istanza_n=30_d=150_a=45_4.txt	2	0.02000	12	0.00001	12	7,01	3	5,25
istanza_n=30_d=150_a=45_5.txt	2	0.28000	11	0.00001	12	7,12	2	5,81
istanza_n=30_d=200_a=60_1.txt	1	0.19000	3	0.00001	3	7,57	1	5,64
istanza_n=30_d=200_a=60_2.txt	1	0.10000	4	0.00001	5	6,95	1	11,07
istanza_n=30_d=200_a=60_3.txt	1	0.26000	4	0.00001	4	0,00	1	0,16
istanza_n=30_d=200_a=60_4.txt	1	0.43000	4	0.00001	4	7,08	1	6,91
istanza_n=30_d=200_a=60_5.txt	1	1.16000	4	0.00001	4	0,00	2	0,05
istanza_n=30_d=400_a=120_1.txt	0	0.09000	0	0.00001	0	0,00	0	0,02
istanza_n=30_d=396_a=119_2.txt	0	0.05000	0	0.00001	0	0,00	0	0,02
istanza_n=30_d=393_a=118_3.txt	0	0.07000	0	0.00001	0	0,00	0	0,03
istanza_n=30_d=406_a=122_4.txt	0	0.11000	0	0.00001	0	0,00	0	0,02
istanza_n=30_d=406_a=122_5.txt	0	0.07000	0	0.00001	0	0,00	0	0,02

3.6 Results

Previous table shows a set of preliminary results comparing the MBV and MDS genetic algorithm, with the optimal solution produced by the CPLEX solver. We used the mathematical model described in Section 3.3. These results point out the effectiveness of the OMEGA technique, even if it still needs further optimizations.

Chapter 4

Multi-Period Street Scheduling and Sweeping (MPS3)

4.1 Introduction

Consider a city which seeks to sweep a subset of its streets over two days, an even day and an odd day, with a single street sweeper. The street sweeper is restricted, however, in that he cannot sweep a side of a street if there are cars parked on that side. Because the city requires parking to be available in certain areas, it has parking constraints associated with each street, dictating parking availability and hence sweeping ability.

The city puts up signs to enforce these parking constraints. It is important to note that the collection of these signs need not be unique to be feasible. For example, a common parking constraint (and a central motivator for this problem) is the requirement that parking be available on at least one side of a street at all times. Assuming both sides must be swept, on each day one side of this street must be available for parking and the other side must be clear for sweeping. There are two feasible choices for the enforcing signs: Sweep the left side of the street on even days and the right on odd days (with parking available on the other side of the street), or sweep the right on even days and the left on odd days.

With the signs in place, we have the following key problem, which we call Variant 0: How does the city route the street sweeper while obeying the signs (and hence the parking constraints) to minimize total distance traveled? It is important to note this problem is not simply a Directed Rural Postman Problem¹. For example, on streets that must be swept but never have available parking, there is a choice of which day to sweep each side of the street. In particular, both sides of the street could be swept on the same day or different days. We define a schedule to assign a sweeping day, even or odd, to each street-side that must be swept. Given a schedule, the city can obtain an optimal sweeping route by solving the Directed Rural Postman Problem for each day. Thus, the problem can be stated as: solve for the optimal schedule given existing street signs.

We extend this problem to the following two extensions:

4.1.1 Variant 1

Suppose the city decides to redo all the parking signs for the entire city. How should the city schedule the closing and non-closing of each side of each street and route the street sweeper so that the length of the optimal sweeping path is minimized while satisfying the parking constraints of each street?

It is clear that the relaxing of the schedule can only improve the objective. Consider the following small city graph which all street sides must be swept and all streets require available parking at all times:

Suppose the existing city schedule is to sweep (1, 2), (4, 2), (4, 3), and (3, 1) on even days (Figure 4.2) and (2, 1), (2, 4), (3, 4), and (1, 3) on odd days (Figure 4.3).

¹The Directed Rural Postman Problem can be stated as follows: Given a directed graph $G = (V, E)$, subset $E_R \subseteq E$ of required edges, and non-negative costs associated with each edge of G , determine a closed path with minimum total cost traversing the links E_R at least once.

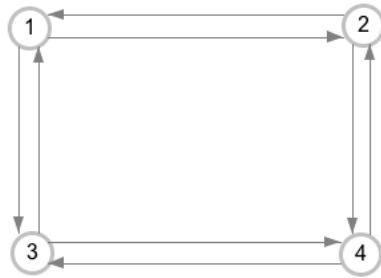


Figure 4.1: All streets must be swept and require available parking

Then, for even days, the optimal sweeping path that begins and ends at node 1 is $1 \rightarrow 2 \rightarrow 4 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$. Similarly, the optimal sweeping path for odd days is $1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 4 \rightarrow 2 \rightarrow 1$. If each edge has a cost of 1, then the total deadhead (non-productive travel) cost is 4.

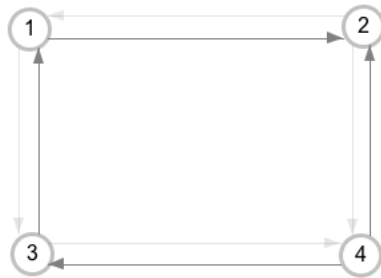


Figure 4.2: Bolded edges are street-sides that must be swept on day 0

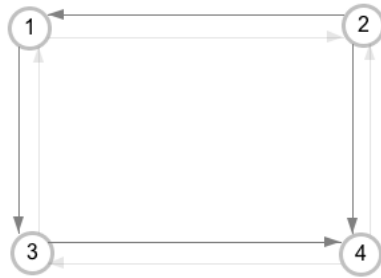


Figure 4.3: Bolded edges are street-sides that must be swept on day 1

Day	1	2
Schedule	$\left\{ \begin{array}{l} (1, 2) \\ (4, 2) \\ (4, 3) \\ (3, 1) \end{array} \right.$	$\left\{ \begin{array}{l} (1, 3) \\ (3, 4) \\ (2, 4) \\ (2, 1) \end{array} \right.$
Solution	$\left\{ \begin{array}{l} (1, 2) \\ (2, 4) \\ (4, 2) \\ (2, 4) \\ (4, 3) \\ (3, 1) \end{array} \right.$	$\left\{ \begin{array}{l} (1, 3) \\ (3, 4) \\ (4, 2) \\ (2, 4) \\ (4, 2) \\ (2, 1) \end{array} \right.$
Sub Cost	6	6
Cost	12	

However, if the city exchanges (4, 2) and (2, 4), the schedule (1, 2), (2, 4), (4, 3), and (3, 1) on even days (Figure 4.4) and (2, 1), (4, 2), (3, 4), and (1, 3) on odd days (Figure 4.5) results. The optimal solution is $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$ and $1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1$ for even and odd days, respectively, which results in a total deadhead cost of 0.

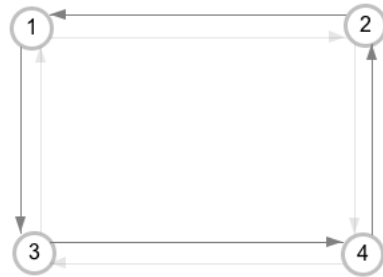


Figure 4.4: Bolded edges are street-sides that must be swept on day 0

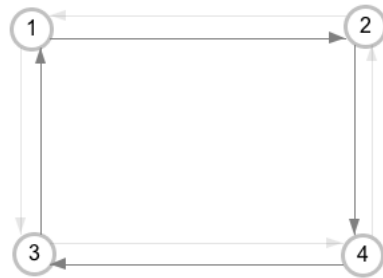
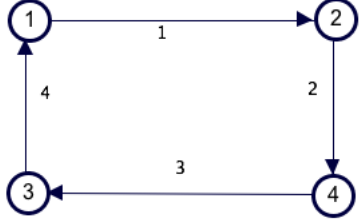
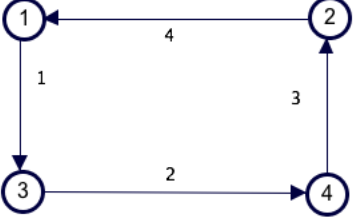


Figure 4.5: Bolded edges are street-sides that must be swept on day 1

Day	1	2
		
Solution	$\left\{ \begin{array}{l} (1, 2) \\ (2, 4) \\ (4, 3) \\ (3, 1) \end{array} \right.$	$\left\{ \begin{array}{l} (1, 3) \\ (3, 4) \\ (4, 2) \\ (2, 1) \end{array} \right.$
Schedule	$\left\{ \begin{array}{l} (1, 2) \\ (2, 4) \\ (4, 3) \\ (3, 1) \end{array} \right.$	$\left\{ \begin{array}{l} (1, 3) \\ (3, 4) \\ (4, 2) \\ (2, 1) \end{array} \right.$
Sub Cost	4	4
Cost	8	

4.1.2 Variant 2

Suppose a city has an existing set of street signs. How can the city minimally change these street signs to allow for a schedule with a maximum decrease in distance traveled by the street sweeper?

This problem is a natural extension of Variant 1. There is a cost for changing a street sign because it confuses those residents who are familiar with the existing one. Thus, it could be desirable to achieve a significant fraction of the benefit of redoing all the signs at a reduced cost of inconvenience. We note that Variant 2 will solve

Variant 0 by simply assigning a sufficiently high penalty cost to changing a street sign.

The rest of the work will be organized as follows: Section 2 will review the relevant literature. In Section 3, we formulate the problem as an integer program. In Section 4, we introduce our genetic algorithm. In section 5, we discuss the results of the genetic algorithm and compare it to the CPLEX implementation of the integer program formulation. Finally, we offer concluding remarks and directions for future research.

4.2 Literature Review

The generic street sweeping problem can be described as a Directed Rural Postman Problem, for which Christofides et al. [10] give a heuristic and mathematical programming formulation. Using their heuristic, the authors solved twenty-three instances within 1.3% of optimality.

Bodin and Kursh [4], [5], describe a computer-assisted system for scheduling and routing of multiple street sweepers. Their model, like ours, deals with urban settings that involve one-way streets and parking constraints. The required edges are directed, a subset of the entire considered graph, and not necessarily connected. Unlike our work, Bodin and Kursh do not have multi-period parking constraints and instead they regard parking constraints on a street as time-window constraints during a single day. Their algorithm seeks to assign streets to sweepers and route the sweepers, while obeying parking regulations, balancing workload, and minimizing deadhead. The authors apply the algorithms to pilot studies in New York City and Washington, D.C.

Eglese and Murdock [14] describe their street sweeping application in Lancashire

County Council in England. Their work differs from [4] and [5] in that there are no parking considerations to be made and streets can be regarded as bidirectional because in rural areas street sweepers are allowed to traverse a street against traffic.

4.3 Problem Formulation

4.3.1 Variant 1

We represent the city as a directed graph $G = (V, E)$ where $E = \{(i, j, k) | i, j \in V, k = 0, 1\}$. Each street is represented by exactly two edges representing the two sides of the street that must be swept. They are either $(i, j, 0)$ and $(i, j, 1)$ if the street is a 1-way street from intersection i to intersection j , or $(i, j, 0)$ and $(j, i, 0)$ if the street is a 2-way street. We assume that the directed graph G is strongly connected because we are considering a city. It is logical to assume that one would be able to reach any intersection in the city from any other intersection. Additionally, the street sweeper is responsible for sweeping some subset of G determined ahead of time. Thus, for each street (the associated pair of edges), we have the four possible states for an allowed schedule:

1. Both sides never need to be swept. Travel along either side will result in deadheading.
2. One side does not need to be swept and the other does. Travel along the former will result in deadheading, and the latter may be swept on either day.
3. Both sides need to be swept, but cannot be swept on the same day. This is the situation where parking is required to be available on one side of the street at all times.

4. Both sides need to be swept. This allows for both streets to be swept on the same day. This could occur if a street needs to be swept but there is no parking to consider.

These states are determined ahead of time and are requirements for a feasible schedule. If an edge (i, j, k) (and its associated pair) is subject to the constraint given by state i (enumerated as above), we say that $(i, j, k) \in S_i$. The problem is to determine a feasible schedule that yields the Eulerian tour with the smallest deadhead. We note that the orientation of the street signs is completely determined by a schedule, and so we only concern ourselves with obtaining the optimal schedule.

We define:

1. $\alpha \in \{0 = \text{even day}, 1 = \text{odd day}\}$.
2. M to be the number of vertices.
3. $x_{i,j,k}^\alpha$ as the number of times the solution traverses along edge (i, j, k) on day α .
4. $u_{i,j,k}^\alpha = \begin{cases} 1 & : (i, j, k) \text{ traversed on day } \alpha \\ 0 & : (i, j, k) \text{ otherwise.} \end{cases}$
5. c_{ij} as the cost from node i to node j .
6. $r_{i,j,k}^\alpha = \begin{cases} 1 & : (i, j, k) \text{ swept on day } \alpha \\ 0 & : (i, j, k) \text{ otherwise.} \end{cases}$
7. $y_i^\alpha = \begin{cases} 1 & : \text{node } i \text{ visited on day } \alpha \\ 0 & : \text{otherwise.} \end{cases}$

The formulation is then as follows:

$$\text{Minimize } \sum_{i,j,k,\alpha} c_{ij} x_{i,j,k}^\alpha \tag{4.3.1}$$

such that

$$\sum_{j,k} x_{0,j,k}^\alpha \geq 1 \quad \forall \alpha \quad (4.3.2)$$

$$\sum_{i,k} x_{i,j,k}^\alpha = \sum_{i,k} x_{j,i,k}^\alpha \quad \forall j, \alpha \quad (4.3.3)$$

$$x_{i,j,k}^\alpha \geq r_{i,j,k}^\alpha \quad \forall i, j, k, \alpha \quad (4.3.4)$$

$$x_{i,j,k}^\alpha \geq u_{i,j,k}^\alpha \quad \forall i, j, k, \alpha \quad (4.3.5)$$

$$\sum_{j,k} x_{i,j,k}^\alpha \leq M y_i^\alpha \quad \forall i, j, k, \alpha \quad (4.3.6)$$

$$r_{i,j,0}^\alpha + r_{j,i,0}^\alpha = 1 \quad \forall i, j, \alpha \text{ where } (i, j, 0), (j, i, 0) \in S_3 \quad (4.3.7)$$

$$r_{i,j,0}^\alpha + r_{i,j,1}^\alpha = 1 \quad \forall i, j, \alpha \text{ where } (i, j, 0), (i, j, 1) \in S_3 \quad (4.3.8)$$

$$\sum_{\alpha} r_{i,j,k}^\alpha = 1 \quad \forall i, j, k, \alpha \text{ where } (i, j, k) \in S_2 \text{ or } S_4 \quad (4.3.9)$$

$$\sum_i y_i^\alpha = 1 + \sum_{i,j,k} u_{i,j,k}^\alpha \quad \forall \alpha \quad (4.3.10)$$

$$v_j^\alpha \geq (v_i^\alpha + 1) - (M - 1)(1 - u_{i,j,k}^\alpha) \quad \forall i, j, i \neq j, \alpha \quad (4.3.11)$$

$$1 \leq v_j^\alpha \leq M - 1 \quad \forall 1 \leq j \leq M, \alpha \quad (4.3.12)$$

$$\sum_{i,k} u_{i,j,k}^\alpha \leq 1 \quad \forall j, \alpha \quad (4.3.13)$$

$$0 \leq r_{i,j,k}^\alpha, u_{i,j,k}^\alpha \leq 1, 0 \leq x_{i,j,k}^\alpha \quad \forall i, j, k, \alpha \quad (4.3.14)$$

We are essentially solving the Rural Postman Problem over two days subject to the pre-determined constraints while allowing freedom in the schedule. In (4.3.1) we simply add the costs of travel on all days. Equations (4.3.2) require the street sweeper to leave from the depot and (4.3.3) enforce flow balance on every node. Equations (4.3.4) enforce coverage of a required edge. Equations (4.3.5) and (4.3.6) partially force the binary behavior of $u_{i,j,k}^\alpha$ and y_i^α respectively (the objective function handles the rest). Equations (4.3.7) and (4.3.8) require that exactly one side of the street is swept each day when necessary and equations (4.3.9) require that if a street is to be swept, it must be swept on exactly one day. (4.3.10) forces a Hamiltonian cycle. Finally, equations (4.3.11), (4.3.12), and (4.3.13) are the traditional MTZ subtour elimination constraints [24] required since the set of required arcs may no longer be connected.

It is clear that this problem is NP-hard. If one generates a graph of one-way streets with required edges that are not connected, the problem becomes a mild generalization of the Directed Rural Chinese Postman Problem which is NP-hard. Thus, we use a heuristic methodology to solve this problem.

4.3.2 Variant 2

There are two formulations that incorporate the desire to restrict the number of street reassignments, both very similar to the formulation of Variant 1. In the first, only the objective function is changed:

$$\text{Minimize } \sum_{i,j,k} c_{ij}(x_{i,j,k} + y_{i,j,k}) + C \sum_{r_{i,j,k} \text{ initialized to } 0} r_{i,j,k} \quad (4.3.15)$$

The second term in equation (4.3.15) simply penalizes in proportion to the number of street schedules changed. One can vary C until the number of changed schedules and change in objective function above optimal is as desired. The second formulation adds a hard constraint in the number of signs allowed to be switched:

$$\sum_{r_{i,j,k} \text{ initialized to } 0} r_{i,j,k} \leq n \quad (4.3.16)$$

By varying the penalty weight in the former formulation and the number of allowed changes in the latter formulation, one can generate a plot of the objective function as a function of the penalty weight and number of allowed changes, respectively.

4.4 Genetic Algorithm

We employed a genetic algorithm heuristic to generate good solutions in a short amount of time. It acts on a population of feasible schedules and uses a heuristic for the Directed Chinese Postman Problem to return a tour whose length serves as the fitness function. We describe each component of the genetic algorithm separately and then summarize the heuristic at the end.

4.4.1 Initialization

A potential problem is that the set of required edges that must be swept may be relatively small when compared to the entire graph. For example, a city sweeper may only be responsible for a small fraction of the city, but any algorithm would consider each street in the city as a street that is possibly traversed. A genetic algorithm such as ours will use a random mutation to investigate a solution that requires the traversal of a particular random edge. However, this random edge may be very far away from the required edges and is likely to be a poor choice. To remove these poor choices, we first determine all required nodes, $R = \{r_i\}$, which are nodes that are connected to at least one required edge. For each required node, we calculate a shortest path to each other required node. If there are multiple shortest paths we randomly choose one. Any edge that is not on a chosen shortest path is removed from further consideration.

It is clear that this will not remove the optimal solution. Suppose a solution contained an edge that is not on any shortest path. Call the immediately preceding required node r_α and the immediately following required node r_β . The shortest path from r_α to r_β does not contain the selected edge and the considered solution is not optimal.

Furthermore, we perform the following operation:

1. Make a collection of all shortest paths from one required node to another;
2. Remove those shortest paths that contain a third required node;
3. Condense the remaining shortest paths into a single edge with length equal to the length of the original path.

The purpose of this operation is to further prune the state space. Step 2 removes

redundant shortest paths: if we have a shortest from r_1 to r_3 containing r_2 , then this shortest path is made redundant by the paths from r_1 to r_2 and r_2 to r_3 . Step 3 reduces the remaining paths from potentially many edges to a single one. The purpose for this is that if the schedule selects one of the edges in a shortest path to be required on a particular day, then the street sweeper must travel over that entire shortest path on the same day. Collapsing each shortest path to a single edge reduces the state space even further.

For example, Figure 4.6 is a graph of a portion of Washington D.C. The bolded (required) streets are those that must be swept by the street sweeper over two days. The required streets are strongly connected within the set of bold and dim streets, the latter being the streets that the street sweeper is allowed, but not required, to travel on. The state space considering the entire graph is prohibitively large. Applying the above initialization yields the graph in Figure 4.7. The graph is considerably smaller, usually by a factor of two, in the realistic cases of subsets of Washington D.C.

From now on, the graph G will refer to the pared down graph after applying the above procedure.

4.4.2 Schedules

We call our chromosomes schedules, which is represented as a $|E|/2$ -dimensional vector. Each component represents the state of one street which is a pair of edges and explicitly states, by the element $((u_{i,j,0}^0, u_{i,j,0}^1), (u_{j,i,0}^0, u_{j,i,0}^1))$ or $((u_{i,j,0}^0, u_{i,j,0}^1), (u_{i,j,1}^0, u_{i,j,1}^1))$, when each edge must be traveled and when it does not have to (Recall that $u_{i,j,k}^\alpha$ is a binary number indicating whether edge (i, j, k) requires travel on day α). We emphasize that a schedule does not explicitly forbid traveling an un-required edge, merely does not require it.

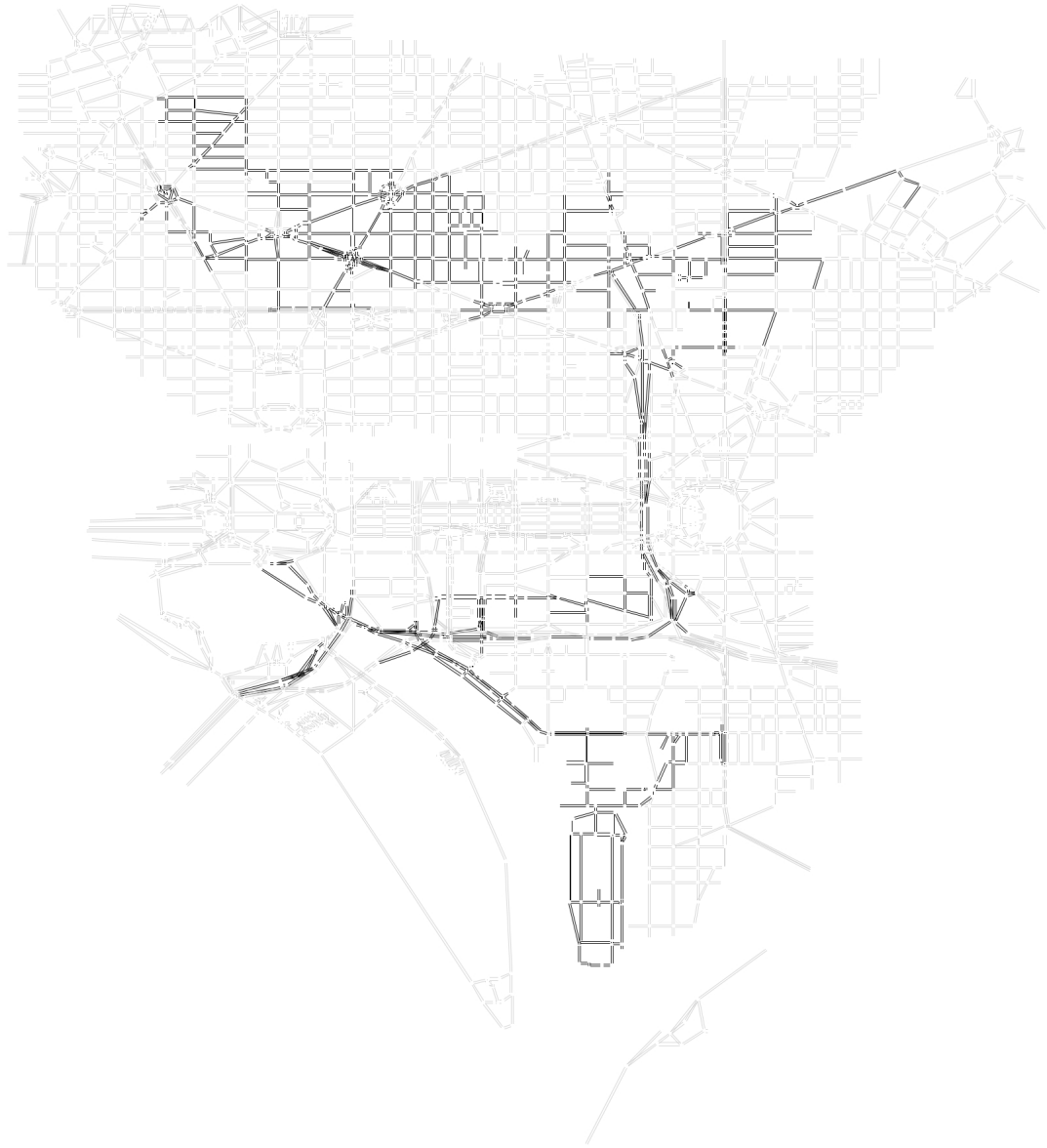


Figure 4.6: Part of Washington D.C. before initialization

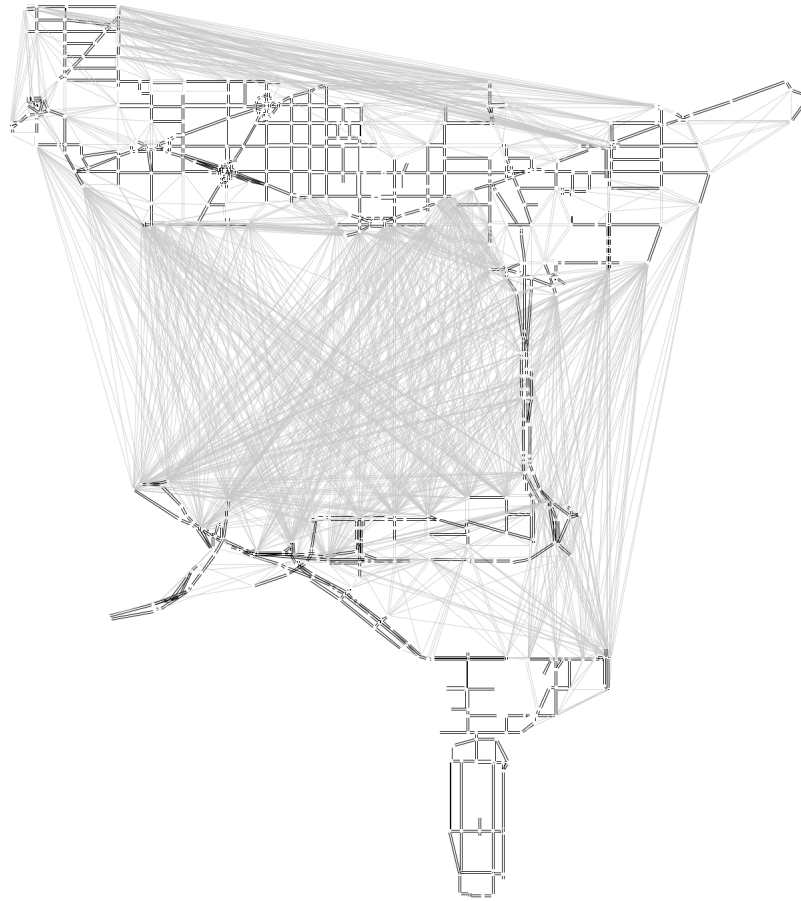


Figure 4.7: Part of Washington D.C. after initialization

Depending on what parking restrictions the component represents, the following restrictions are imposed:

S_1 : no restrictions are imposed. Either side may, or may not, be traversed.

S_2 : $u_{i,j,0}^0 + u_{i,j,0}^1 = 1$ on the required side. This forces the required side to be traversed (and hence swept).

S_3 : can, in fact, be represented as a binary element. 0, if the left side is swept on even days and the right side on odd days, 1 if the left side is swept on odd days and the right side on even days.

S_4 : $u_{i,j,0}^0 + u_{i,j,0}^1 = 1$ and $u_{j,i,0}^0 + u_{j,i,0}^1 = 1$. Similar to S_2 , but both sides are forced to be swept.

The streets are randomly assigned a component of the vector except that two-way streets in S_3 are placed at the top of the schedule and 1-way streets in S_3 are placed at the bottom of the schedule. 1-way streets in S_3 are one way streets where each side must be cleaned on different days. We place them at the end of the vector to distinguish that changing which days the different sides are swept results in equivalent schedules.

To reiterate, schedules induce non-unique daily Eulerian tours. A schedule will assign a sweeping day to every required edge, but will only assign a sweeping day to some not-required edges (possibly none). A schedule induces an optimal Eulerian tour by solving the Directed Rural Postman Problem.

4.4.3 Fitness

Given a schedule s , the next step is to construct an Eulerian tour and determine its length. This problem is simply the Directed Rural Postman Problem, which is NP-hard. Because the fitness of a schedule is called often, we employ a modification of the following simple heuristic given by Christofides et al. (1986) [10]:

1. Construct a shortest spanning arborescence connecting the connected components of required edges (see Edmonds, 1967 [13]).
2. Solve the transportation problem by adding arcs in a least-cost manner so that the number of incoming arcs is equal to the number of outgoing arcs for each node (see Beltrami and Bodin, 1974 [3]).
3. Construct an Eulerian tour on the resulting graph.

Our heuristic differs only in the first step, which constructs a spanning arborescence in a greedy manner, rather than optimal. This is done for running time considerations and was observed to not have significant impact in the quality of solution. We define the fitness $\mu(s)$ to be the length of the obtained Euler tour.

4.4.4 Breeding

A naive breeding of schedules would be to simply swap components of the schedules in a random fashion. However, one can see that the induced route of a schedule is very sensitive to small changes in the schedule (see the example in section 1.1). Requiring sweeping on an edge on an even day rather than an odd day could result in very large detours being required to satisfy the other travel requirements of the schedule. As a result, breeding two good schedules haphazardly will often destroy the good solution.

To construct a good breeding algorithm, we note that all Eulerian tours can be decomposed into cycles and thus can be defined by a set of cycles. A good Eulerian tour will have good cycles and good cycles will induce a good Eulerian tour. Our breeding method attempts to swap cycles between schedules.

A schedule does not have cycles itself, the Eulerian tour that it induces does. However, as mentioned in the previous section, such an optimal tour is difficult to find. We make the reasonable extension that a good schedule allows for good cycles. Our breeding algorithm determines a cycle on a random day allowed by one schedule and adjusts the second schedule to allow the same cycle on the same day.

A possible problem with cycle construction is the issue of feasibility. The construction of a random cycle ignores this issue in its construction. Breeding of two schedules, s_1 and s_2 , denoted as $\beta(s_1, s_2)$ is as follows:

1. Choose a random required edge in G . Call it (a_0, a_1) where $a_0, a_1 \in V$. According to the schedule s_2 , this edge is either scheduled for day 1 or day 2. Without loss of generality, assume it is day 1. We initialize a list with (a_0, a_1) as the first entry.
2. Let (a_{t-1}, a_t) be the edge added to the list in the previous step. Randomly choose an edge (other than (a_t, a_{t-1})) scheduled for day 1 that begins from vertex a_t . If no such edge exists choose a random edge (other than (a_t, a_{t-1})) regardless of schedule. If no such edge exists, choose edge (a_t, a_{t-1}) . Add the chosen edge to the end of the list and denote it as (a_t, a_{t+1}) .
3. If $a_{t+1} = a_{t'}$ where $t' \in 0, 1, 2, \dots, t$, continue to step 4. The sequence of edges $(a_{t'}, a_{t'+1}), \dots, (a_{t-1}, a_t), (a_t, a_{t+1})$ defines a “cycle”. Otherwise, return to step 2.

4. The schedule of s_1 is adjusted to allow for the cycle obtained above. It is important to note that if an edge from the list was scheduled for day 2 instead of day 1 in s_2 , it is scheduled for day 2 instead of day 1 in s_1 .

We give the following small example where we restrict our attention to the small portion of a larger graph as seen in Figure 4.8. Nodes 5 and 6 have other incident edges (not shown) that connect them in some way to the rest of the graph, but nodes 1, 2, 3, and 4 have no incident edges other than those shown. For the small part of the graph we are considering, we assume the parking restrictions are of type S_3 , where both sides must be swept but only one side may be swept on each day. Figures 4.9 and 4.10 are possible schedules, where the bolded edges indicate what edges are to be swept on day 0 (and hence the non-bold edges are to be swept on day 1).

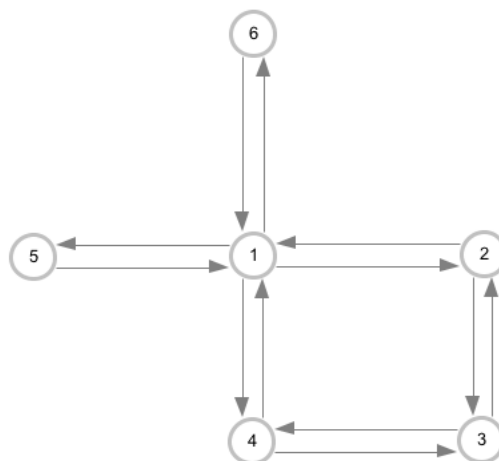


Figure 4.8: Schedule 1

For clarity, assume that on day 0, the sweeper enters from node 6 and exits at node 5 in schedule 1 and enters from node 5 and exits at node 6 in schedule 2. It is clear that schedule 1 has significant deadhead, an induced tour is $6 \rightarrow 1 \rightarrow 4 \rightarrow$

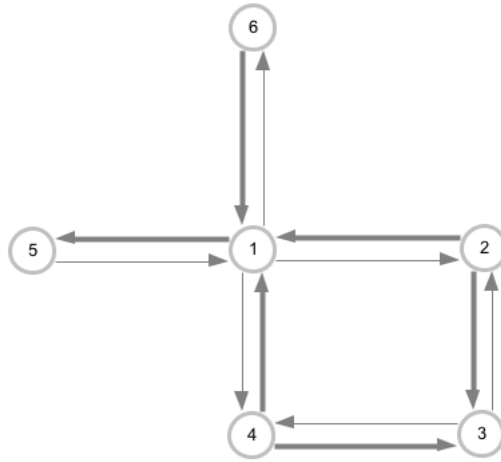


Figure 4.9: Schedule 1

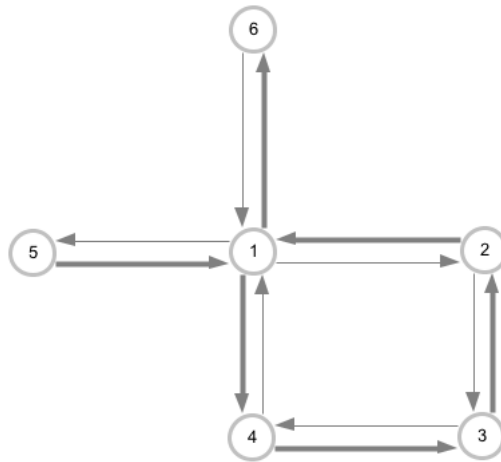


Figure 4.10: Schedule 2

$3 \rightarrow 2 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 5$. Schedule 2 has no deadhead with an induced tour of $5 \rightarrow 1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 6$. To construct the child of schedules 1 and 2, $\beta(s_1, s_2)$, we must construct a cycle in schedule 2 on day 0. Suppose that cycle is $1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$. We adjust schedule 1 to allow that cycle in day 0 yielding Figure 4.11. The result is set to be $\beta(s_1, s_2)$. Thus, the good structure of schedule 2 is imparted on schedule 1.

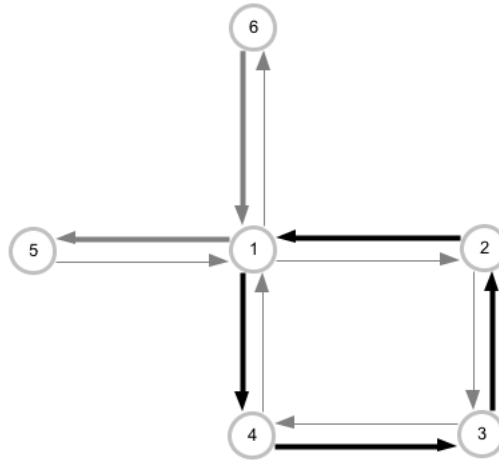


Figure 4.11: Breeding of schedule 1 and schedule 2

4.4.5 Mutation

We have several mutation operators that act on schedules:

stableVertexMutation() One characteristic of “good” schedules is that for a given day, a required node is balanced. That is, the number of required edges for that day entering the required node is equal to the number of required edges for that day exiting the required node. If a required node is not balanced, then it is

clear that deadhead will occur. This mutation function seeks to balance non-balanced required nodes by randomly choosing a required node with unbalanced degree. If it is possible, the mutation changes the day assignment of of an edge incident to the chosen node in a way decreases the difference between the in- and out-degrees.

randomPathMutation() It is possible that a simple path, not necessarily a cycle, is better on even day rather than on an odd day or vice versa. The breeding process will likely not achieve this because it deals exclusively with cycles. This mutation finds a random path of random length (between 2-5 edges long) on a random day and then adjusts the schedule to allow the same path on the other day.

randomChange(i) To achieve additional variation in our genetic algorithm, this mutation randomly changes the edge-day assignment of i streets in the schedule.

The mutation of schedule s , $m(s)$, calls `stableVertexMutation()` with 10% probability, `randomPathMutation()` with probability 20%, `randomChange(2)` with probability 20%, and `randomChange(1)` with probability 50%.

4.4.6 Genetic Algorithm Summary

We define the following notation:

- $S = \{s_i\}_i$ is set of feasible schedules
- k = size of population - input parameter
- h = number of “best” schedules (set to be about $k/10$)

- P_t = population at generation $t \in \mathbb{Z}^+$, a vector with schedules as components and
 $|P_t| = K \forall t$
- $P_t[i] = i^{th}$ schedule of population P_t
- $\mu(P_t[i]) =$ fitness of $P_t[i]$
- $\beta(P_t[i], P_t[j]) =$ child of $P_t[i]$ and $P_t[j]$
- $m(P_t[i]) =$ mutation of $P_t[i]$
- The end criterion is a fixed number of iterations without improvement

Our genetic algorithm is as follows:

1. Create a random population of k individuals
2. Sort the random population with respect to the fitness function (best at the top)
3. Set the reordered population to be P_0
4. While the end criteria is not satisfied: (indexed by t)
 - (a) For each $k \leq i \leq h$
 - i. Set $P_t[i] = \beta(P_t[i], P_t[j])$ where j is a random value in the range 1, ... $(i - 1)$. This breeds schedule i , which is not in the top h , with a random schedule better than it.
 - ii. If $P_t[i] = P_t[j]$ (after the breeding of the previous step), mutate $P_t[i]$ by setting $P_t[i] = m(P_t[i])$. Having a duplicate schedule is undesirable so, if it occurs, we change it.

- iii. Calculate the fitness $\mu(P_t[i])$
 - (b) For each $k \leq i \leq 2h$
 - i. Compute the mutation of $m(P_t[i])$
 - ii. If $\mu(m(P_t[i])) < \mu(P_t[i])$ set $(P_t[i]) = \mu(m(P_t[i]))$. This mutates a schedule and replaces it with the mutation if the mutation is better.
 - (c) Sort population with respect to the fitness function
 - (d) Set the reordered population to P_{t+1}
5. Return $P_t^{final}[1]$

4.5 Results

In this section we show the computational results obtained by our genetic algorithm applied to MPS3 problem. The input instances set is composed by 36 input graphs, each one with a number of nodes equal to 100, and a number of edges in the range between 400 and 1600. For each street we use two different edges representing both sides; the average number of streets incident on each crossroad is between 4 (for 400-edges graphs) and 16 (for 1600-edges graphs).

The computational results obtained by the GA were compared to the optimal solutions provided by the CPLEX solver. It can be easily noticed that the solution values of the GA are consistently within a 2% at most from the optimum, except for a single instance, and on bigger instances where the time limit is violated for CPLEX it even provides better values. Computational times are always extremely competitive for our GA, with a maximum solution time of 279 seconds.

Instance	Bound	Cplex			GENETIC			
		SOL	DIF LB	TIME	SOL	DIF LB	TIME	% GAP
IST100_400_5_0	40596	47997	7401	181	48098	7502	22	1,365
IST100_400_4_1	40630	46916	6286	181	47017	6387	28	1,607
IST100_400_2_2	40565	46429	5864	181	46432	5867	26	0,051
IST100_400_15_0	40587	50092	9505	181	50194	9607	36	1,073
IST100_400_12_1	40595	50435	9840	181	50838	10243	29	4,096
IST100_400_10_2	40562	50181	9619	181	50189	9627	31	0,083
IST100_400_38_0	40598	54695	14097	7	54791	14193	74	0,681
IST100_400_32_1	40607	53370	12763	45	53374	12767	60	0,031
IST100_400_32_2	40590	55875	15285	182	55876	15286	110	0,007
IST100_400_97_0	40609	63559	22950	2	63564	22955	105	0,022
IST100_400_91_1	40627	59918	19291	4	59919	19292	98	0,005
IST100_400_95_2	40569	69216	28647	1	69317	28748	208	0,353
IST100_800_7_0	81202	87172	5970	181	87181	5979	35	0,151
IST100_800_5_1	81260	86843	5583	181	86842	5582	25	0,018
IST100_800_9_2	81234	87210	5976	181	87206	5972	34	0,067
IST100_800_12_0	81229	88040	6811	181	88130	6901	60	1,321
IST100_800_26_1	81167	89054	7887	181	88955	7788	47	1,255
IST100_800_22_2	81239	88635	7396	181	88742	7503	28	1,447
IST100_800_60_0	81157	93106	11949	181	93104	11947	110	0,017
IST100_800_60_1	81172	91981	10809	181	91983	10811	71	0,019
IST100_800_52_2	81180	91317	10137	181	91303	10123	77	0,138
IST100_800_193_0	81224	106134	24910	183	106134	24910	233	0,000
IST100_800_195_1	81201	104929	23728	182	104932	23731	277	0,013
IST100_800_193_2	81134	104493	23359	182	104493	23359	279	0,000
IST100_1600_14_0	162361	168207	5846	181	168209	5848	77	0,034
IST100_1600_18_1	162415	169071	6656	181	169067	6652	74	0,060
IST100_1600_15_2	162382	168723	6341	181	168717	6335	57	0,095
IST100_1600_37_0	162408	170670	8262	181	170554	8146	72	1,404
IST100_1600_36_1	162341	171205	8864	181	171095	8754	121	1,241
IST100_1600_48_2	162471	171884	9413	181	171885	9414	126	0,011
IST100_1600_125_0	162447	177013	14566	181	177011	14564	240	0,014
IST100_1600_100_1	162367	174064	11697	181	174051	11684	139	0,111
IST100_1600_118_2	162319	177223	14904	181	177216	14897	182	0,047

Bibliography

- [1] R Ahyja, T Magnanti, and J Orlin. Network flows: theory, algorithms, and applications. *cdsweb.cern.ch*.
- [2] N Barricelli. Numerical testing of evolution theories. *Acta Biotheoretica*, Jan 1962.
- [3] EL Beltrami and LD Bodin. Networks and vehicle routing for municipal waste collection. *Networks*, 4:65–94, 1974.
- [4] LD Bodin and SJ Kursh. A computer-assisted system for the routing and scheduling of street sweepers. *Operations Research*, 26:525–537, 1978.
- [5] LD Bodin and SJ Kursh. A detailed description of a computer system for the routing and scheduling of street sweepers. *Computer and Operations Research*, 6:181–198, 1979.
- [6] M Captivo, J Clímaco, and M Pascoal. A mixed integer linear formulation for the minimum label spanning tree problem. *Computers and Operations Research*, Jan 2009.
- [7] R Cerulli, A Fink, M Gentili, and S Voß. Metaheuristics comparison for the minimum labelling spanning tree problem. *The next wave on computing*, Jan 2005.

- [8] R Cerulli, M Gentili, and A Iossa. Experimental comparison of algorithms for bounded-degree spanning tree problems. *Computational Optimization and Applications*.
- [9] R Chang and L Shing-Jiuan. The minimum labeling spanning trees. *Information Processing Letters*, Jan 1997.
- [10] N Christofides, V Campos, A Corberan, and E Mota. An algorithm for the rural postman problem on a directed graph. *Mathematical Programming Study*, 26:155–166, 1986.
- [11] S Consoli, J Moreno, and N Mladenović. Constructive heuristics for the minimum labelling spanning tree problem: a preliminary *DEIOC Documentos de Trabajo*, Jan 2006.
- [12] J Cordeau, M Gaudioso, and G Laporte. A memetic heuristic for the generalized quadratic assignment problem. *INFORMS Journal on Computing*, Jan 2006.
- [13] J Edmonds. Optimum branching. *Journal of Research of the National Bureau of Standards*, 71:233–240, 1967.
- [14] RW Eglese and H Murdock. Routing road sweepers in a rural area. *The Journal of the Operational Research Society*, 42:281–288, 1991.
- [15] S Forrest and M Mitchell. Relative building-block fitness and the building-block hypothesis. *Ann Arbor*, 1993.
- [16] Fraser and Alex. Simulation of genetic systems by automatic digital computers. i. introduction. *Aust. J. Biol. Sci.*, 10:484–491, 1957.
- [17] Fraser and Alex. Computer models in genetics. *New York: McGraw-Hill*, 1970.

- [18] L Gargano and M Hammar. There are spanning spiders in dense graphs (and we know how to find them). *Lecture notes in computer science*, pages 802–816, 2003.
- [19] L Gargano, P Hell, L Stacho, and U Vaccaro. Spanning trees with bounded number of branch vertices. *Lecture notes in computer science*, pages 355–365, 2002.
- [20] J Grefenstette and 1995. Lamarckian learning in multi-agent environments. *Cite-seer*.
- [21] John H Holland. Adaptation in natural and artificial systems. an introductory analysis with applications to biology, control and artificial intelligence. *Ann Arbor: University of Michigan Press*, Jan 1975.
- [22] P Klein and R Ravi. A nearly best-possible approximation algorithm for node-weighted steiner trees. *Journal of Algorithms*, Jan 1995.
- [23] S Krumke and H Wirth. On the minimum label spanning tree problem. *Information Processing Letters*, Jan 1998.
- [24] CE Miller, AW Tucker, and RA Zemlin. Integer programming formulations and traveling salesman problems. *J. ACM*, 7:326–329, 1960.
- [25] P Moscato and M Norman. A memetic approach for the traveling salesman problem implementation of a computational *Parallel Computing and Transputer Applications*, Jan 1992.
- [26] J Nummela and B Julstrom. An effective genetic algorithm for the minimum-label spanning tree problem. *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, Jan 2006.

- [27] J Reed, R Toombs, and NA Barricelli. Simulation of biological evolution and machine learning. i. selection of self-reproducing numeric patterns by data processing machines, effects of hereditary control, mutation type and crossing. *J. Theoret. Biol.*, 17:319–342, 1967.
- [28] G Salamon and G Wiener. On finding spanning trees with few leaves. *Information Processing Letters*, Jan 2008.
- [29] Y Xiong, B Golden, and E Wasil. Improved heuristics for the minimum label spanning tree problem. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, Jan 2006.
- [30] Yupei Xiong, Bruce Golden, and Edward Wasil. A one-parameter genetic algorithm for the minimum labeling spanning tree problem. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, 9:55–60, Feb 2005.