UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

GIOVANE ALVES FONSECA

Formulations and algorithms for the Optimum Communication Spanning Tree problem

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dr. Santiago Valdés Ravelo

Porto Alegre
May 2021

ABSTRACT

The Optimum Communication Spanning Tree problem (OCT) has applications in many fields of study such as logistics, telecommunications and bioinformatics. This problem receives as input an undirected graph with weighted edges and requirement value for each pair of nodes, and seeks for a spanning tree that minimizes the communication cost, given by the sum of requirement of each pair of nodes times the distance separating them in the tree. In this work we design a new integer formulation for OCT as well as four different strategies of evolutionary algorithms and a combined strategy with simulated annealing. We give public access to our implementations. We test our approaches on instances from the literature and from real-world data sets. The experiments show that our best strategies were able to obtain very accurate solutions, getting close to the best known value for all tested instances, improving the results of previous metaheuristics from the literature.

Keywords: Optimum Communication Spanning Tree. Linear Integer Programming. Evolutionary Algorithm. Simulated Annealing. Combinatorial Optimization.

RESUMO

O problema da árvore geradora de comunicação ótima possui aplicação em diversos campos de estudo como logística, telecomunicações e bioinformática. Esse problema recebe como entrada um grafo com pesos nas arestas e um valor de requerimento entre cada par de nodos do grafo, e procura por uma árvore geradora que minimiza o custo de comunicação que é calculado pela soma dos requerimentos de cada par de nodos vezes a distância que os separa na árvore. Neste trabalho propomos uma nova formulação inteira para o problema e desenvolvemos quatro estratégias diferentes de algoritmos evolutivos e uma combinada com o método *simulated annealing*, dando acesso público às nossas implementações. Testamos nossos algoritmos com instâncias da literatura e com outras baseadas em conjuntos de dados do mundo real. Os experimentos mostram que nossas melhores estratégias foram capazes de obter soluções muito precisas para todas as instâncias testadas, melhorando os resultados de metaheurísticas anteriores da literatura.

Palavras-chave: Árvore Geradora de Comunicação Ótima. Programação Linear Inteira. Algoritmo Evolutivo. *Simulated Annealing*. Otimização Combinatória.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

OCT	Optimum Communication Spanning Tree problem
MRCT	Minimum Routing Cost Spanning Tree problem
SROCT	Optimal Sum-Requirement Communication Spanning Tree problem
PROCT	Optimal Product-Requirement Communication Spanning Tree problem
WSDOCT	Optimum Weighted Source–Destination Communication Spanning Tree problem
p-OCT	p-source Optimum Communication Spanning Tree problem
PTAS	Polynomial-Time Approximation Scheme

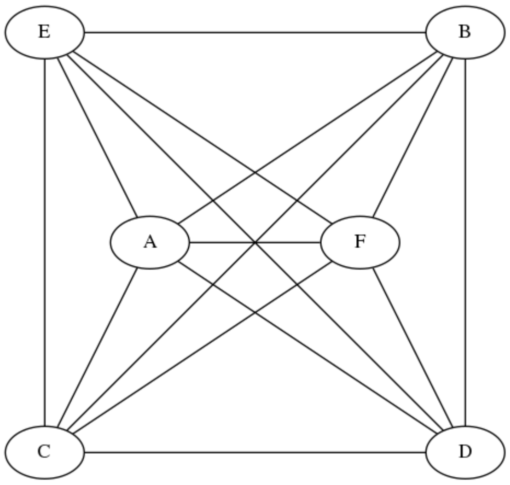
CONTENTS

1 INTRODUCTION

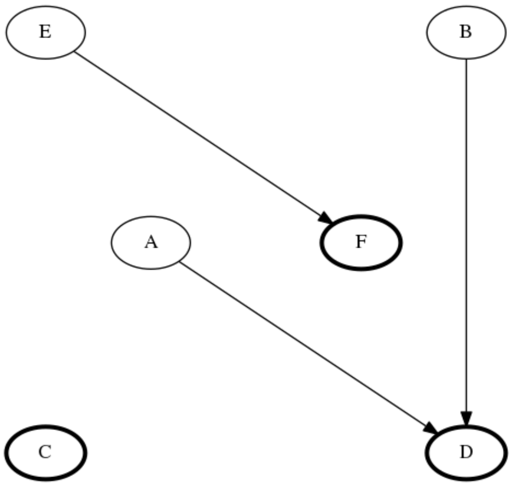
Communication networks have experienced a remarkable rise in the last decades, demanding new and efficient solutions to optimize the communication costs. In this connection many problems emerge that model a network by a weighted undirected graph and seek for a spanning tree that optimizes some cost function. One of these problems is the *Optimum Communication spanning Tree problem* (OCT). OCT receives a weighted graph where the nodes may represent cities, the edges roads, and each edge has associated a weight value (e.g., length of the road, average time to cross the road or fuel consumption to travel the road). Additionally, between each pair of nodes a requirement value is known (e.g., a value that represents how often cars travel between cities). The objective of OCT is to find a spanning tree that minimizes the communication cost, which can be computed by the sum of travel costs between each pair of cities, where the travel cost between two cities is given by the requirement between the cities times the distance separating them in the spanning tree.

OCT finds practical applications in several different areas. In logistics, OCT appears as subproblem of the Tree-of-Hubs Location problem (CONTRERAS; FERNANDEZ; MARÍN, 2010). The input of Tree-of-Hubs Location problem receives a complete weighted simple digraph $G = (V, A)$, where the nodes represent a set of locations, some of them being origins and destinations of products to be routed through other nodes called hubs. For each pair of nodes $u, v \in V$, a demand of products W_{uv} is known, and an integer $p\,(3 \leq p \leq |V|-1)$ is given, representing the number of hubs that must be selected among all nodes in V. Every non-hub node should be connected to a hub for transshipment and the hubs are connected by means of an undirected tree. The objective is to find the hub nodes, find the allocation pattern from non-hub nodes to hub nodes and find the undirected tree that connects the hub nodes and minimizes the operational cost. The operational cost per unit of flow between two nodes u and v is equal to the sum of the arc weights in the unique path of T connecting the nodes, applying a discount $\alpha\,(0 \leq \alpha \leq 1)$ when both end points of the arc are hubs. The subproblem when both the set of hubs and the node-to-hub allocation pattern are given is OCT and, which was used to prove the $\mathcal{NP}$-hardness of Tree-of-Hubs Location problem. Figure 1.1 shows an example of the reduction for $p = 3$, Figure 1.1a exposes an initial digraph that does not have arrows because it is easier to see. Then, Figure 1.1b exhibits the case when both the set of hubs represented by the nodes C, D and F and the node-to-hub allocation pattern is fixed. Since the connections between hubs are unidirectional, this subproblem is OCT because the objective is to find the spanning tree that connects the hubs with minimum objective function. Since both endpoints are hubs, the length of every selected edge has the discount factor α.

Figure 1.1: Example of the Tree-of-Hubs Location problem. Source: Author.



(a) example of an initial graph G

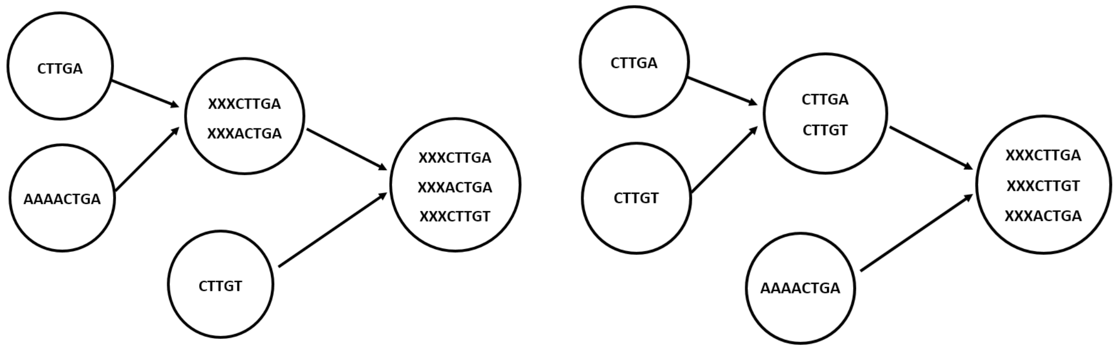


(b) Fixed set of hubs and node-to-hub allocation pattern.

OCT can be also applied in telecommunications, to minimize the total cost of link harness while satisfying the communication requirements in an embedded network as described in (Sommer, 2010). According to (Sommer, 2010), the use of Ethernet in an embedded network reduces the development cost as well as the time needed to build new components. The problem of minimizing link harness in an embedded Ethernet network has the same properties as OCT with the additional constraint that the links are organized in bundles and installed in ducts. The problem has a set of nodes which are endpoints of the network, a set of switches and a set of junction points, such that each node can only be connected to switches or junction points. Therefore, first the switches and junction points are connected by ducts using a spanning tree algorithm and the nodes are greedily connected to the nearest switch or junction point. The second part of the algorithm is a local search where an edge that connects switches and junctions is removed from the solution and a new edge that restores the connection is selected if the cost of the spanning tree is lower than the previous cost. This step tries to find an OCT with a modified cost function to take into account that using the same duct reduces the overall cost.

Another area that applies OCT is bioinformatics, for example in the multiple sequence alignment problem (WU et al., 2000); (FISCHETTI; LANCIA; SERAFINI, 2002). This problem receives n sequences of arbitrary length, a match, a mismatch and a gap value, the objective is to align the sequences with a length of at least the largest sequence by adding gaps to the sequences such that the similarity between all pairs is maximized. The similarity function is the sum over all positions and all pairs of sequences by the corresponding value: match if the letters are the same, mismatch if the letters are different and gap if a gap is introduced in at least one of the two sequences in that position. This problem is commonly used to verify how similar these sequences are. The Feng-Doolittle (FENG; DOOLITTLE, 1987) procedure is used for finding multiple sequence alignments by repeating multiple pairwise alignments and merging them. However the order that these strings are joined determines its cost, therefore the objective is to find an ordering, represented as a spanning tree, which minimizes the cost over all pairwise alignments with all requirements equal to 1. This is the special case of OCT where all requirements have the same value, and this problem is called *Minimum Routing Cost spanning Tree problem* (MRCT). Figure 1.2 shows an example of the multiple sequence alignment of the sequences $\alpha =$`CTTGA`, $\beta =$`AAAACTGA` and $\gamma =$`CTTGT` using trees. Figure 1.2a exposes one possible alignment by aligning sequences α and β first, then γ, while Figure 1.2b exhibits other possible alignment. After selecting the best alignment of two sequences then each distance is equal to an approximate pairwise evolutionary distances between them given in (FENG; DOOLITTLE, 1987). Then, a new node is created that represents the alignment of the cluster. When a new node wants to align with a cluster it will choose the best possible alignment between all sequences in the cluster. Whenever a gap is inserted in the alignment, it should be added to all sequences in that cluster, these gaps are represented by the letter X in the image.

Figure 1.2: Example of the Multiple Sequence Alignment problem. Source: Author.



(a) One possible alignment (b) Other possible alignment

Our proposal is to design a new mathematical formulation for OCT as well as five different strategies based on evolutionary algorithms to solve the problem. The main objective of this work is to develop new approaches for OCT, comparing our results with earlier works found in the literature.

The remainder of the document is organized as follows: In Section 1.1 a formal definition of the problem is given. Section 1.2 presents a literature review showing previous researches for OCT. Chapter 2 presents previous mathematical formulations for OCT and a new mixed integer linear programming formulation is proposed. Chapter 3 presents five different strategies. Those use an evolutionary approach to solve the OCT. Chapter 4 reports and analyzes the computational experiments, comparing the findings with the previous analysis found in the literature. Finally, Chapter 5 gives the conclusions and future work directions.

1.1 Definitions and Notation

OCT considers networks modeled by simple undirected graphs. A **simple graph** is a pair $G = (V, E)$, where V is a set of elements called nodes, and E is a set of pairs of different nodes (i.e., $E \subseteq V \times V \setminus \{(u, u) | u \in V\}$). If any edge $(u, v) \in E$ is considered equal to (v, u), then the graph has no orientation (or direction) and it is called **undirected graph** or simply **graph**.

The objective of OCT is related with a special type of subgraph of a given graph. The subgraph relation is defined between two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, where H is **subgraph** of G ($H \subseteq G$) if the set of nodes of H is subset of the nodes of G ($V_H \subseteq V_G$) and the set of edges of H is subset of the edges of G ($E_H \subseteq E_G$). If every edge of G with both nodes in H is also in the set of edges of H, then H is the **induced subgraph** of G over the nodes in V_H (i.e., $H \subseteq G$ is the induced subgraph of G over V_H if for every $u, v \in V_H$ with $(u, v) \in E_G$, we have $(u, v) \in E_H$).

Three other graph concepts used in this thesis are paths, cycles and spanning trees. A **path** is defined as a sequence of nodes $P = \{v_i\}_{i=1}^{n}$, such that no node is repeated ($v_i \neq v_j$ for any $1 \leq i < j \leq n$) and any two consecutive nodes define an edge ($(v_i, v_{i+1}) \in E$ for any $1 \leq i < n$). Usually we can represent a path by its sequence of nodes, or by its edges, and when the edges have length associated, the length of a path is the sum of the length of its edges.

The definition of **cycle** is very similar to paths, the only differences are that the first and last nodes are equal in a cycle ($v_1 = v_n$) and the sequence must contain at least three different nodes ($n - 1 \geq 3$).

A **tree** $T = (V_T, E_T)$ is a special graph that is **connected** (i.e., for any two nodes $u, v \in V_T$ there exists a path $p = v_1 v_2 \ldots v_n$ from $u = v_1$ to $v = v_n$) and **acyclic** (i.e., there are no cycles in T, implying that for any two nodes $u, v \in V_T$ there exist at most one path from u to v). Since there is a unique path between any pair of nodes in a tree, the length of this path is the **distance** between the nodes in the tree, the distance from any node to itself is equal to zero. Given a graph $G = (V, E)$, a **spanning tree** of G, is a tree $T = (V_T, E_T) \subseteq G$, that contains each node of G ($V_T = V$). Similarly to paths, a spanning tree of a graph G can be represented by the set of its edges.

With the graph terminology well defined, we can formally define OCT as follows:

Problem 1. *Optimum Communication spanning Tree problem (*OCT*).*

Input: *A tuple $I = \langle G = (V, E), r, \ell \rangle$, where:*

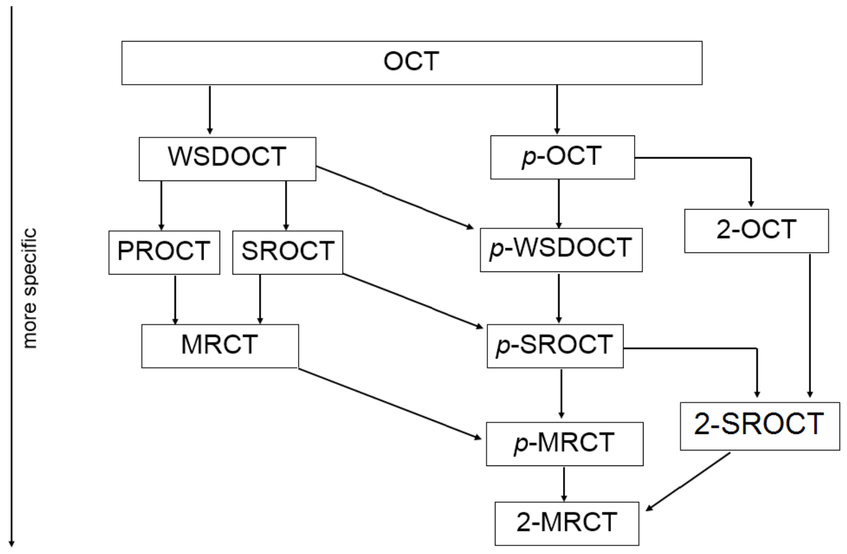
- *$G = (V, E)$ is a graph.*
- *$r : V \times V \to \mathbb{Q}^+$ is a symmetric non-negative requirement function between each pair of nodes ($r(u, v) = r(v, u)$ for all $u, v \in V$).*
- *$\ell : E \to \mathbb{Q}^+$ is a non-negative length function over the edges.*

Output: *A spanning tree $T = (V_T, E_T)$ of G that minimizes the communication cost $C(T) = \frac{1}{2} \sum\limits_{u \in V} \sum\limits_{v \in V} r(u, v)\delta_T(u, v)$, where $\delta_T(u, v)$ is the distance u and v in T.*

The above definition considers only instances of OCT where the requirements are symmetric. The reason is that any instance with asymmetric requirements can be easily transformed in an equivalent instance with symmetric requirements. If $r^a : V \times V \to \mathbb{Q}^+$ is an asymmetric requirement function, then $r(u, v) = \frac{1}{2}\left(r^a(u, v) + r^a(v, u)\right)$ defines an equivalent symmetric requirement function for OCT.

In the literature several special cases for OCT were studied. The *Minimum Routing Cost spanning Tree problem* (MRCT) is a particular case of OCT when $r(u, v) = 1$ for all $u, v \in V$. Another specific case of OCT is the *Optimal Sum-Requirement Communication spanning Tree problem* (SROCT) that receives a value $\sigma(u)$ for each $u \in V$ and $r(u, v) = \sigma(u) + \sigma(v)$. Similar as SROCT, the *Optimal Product-Requirement Communication spanning Tree problem* (PROCT) also receives $\sigma(u)$ for each $u \in V$, defining $r(u, v) = \sigma(u)\sigma(v)$. A generalization of SROCT and PROCT is the *Optimum Weighted Source–Destination Communication spanning Tree problem* (WSDOCT), which receives a $\sigma(u)$ and $\rho(u)$ values associated with each $u \in V$ and defines $r(u, v) = \sigma(u)\rho(v) + \sigma(v)\rho(u)$. There are other particular cases of OCT where only a subset of the nodes may have non-zero requirements. They are generalized by the *p-source Optimum Communication spanning Tree problem* (p-OCT), which receives and integer p and a subset $S \subset V$ with $|S| = p$, and $r(u, v) = 0$ for all $u, v \in V \setminus S$. Similarly to special cases of the OCT, special cases of p-OCT are p-MRCT, p-SROCT (also known as weighted p-MRCT), and p-WSDOCT. Figure 1.3 shows these problems relationship from more general cases at the top to specific cases at the bottom. This image generalizes the one given by (WU, 2004).

Figure 1.3: The relationship between OCT problems. Source: Author.



For all the above problems were also studied their metric cases, which consider the lengths of the edges satisfy the triangular inequality (i.e, $\ell_{uv} + \ell_{vw} \geq \ell_{uw}, \forall u, v, w \in V$).

1.2 Literature review

OCT was first introduced by (HU, 1974). In this paper Hu solved in polynomial time two special cases of OCT that consider complete graphs. The first case supposed that the lengths of all edges were equal to one. The proposed algorithm for solving this particular problem was called Gomory–Hu (GOMORY; HU, 1961), returning the cut-tree which is an optimal tree for this specific case, and it is calculated by solving $|V|-1$ maximum flow problems (e.g., Ford Fulkerson (FORD; FULKERSON, 1957), Dinic (DINITZ, 1970) and Edmonds and Karp (EDMONDS; KARP, 1972)). The second problem solved by Hu was the metric case of MRCT, where the graph also satisfied two other conditions: $|V| \geq 4$ and there exists a positive value $t \leq \frac{|V|-2}{2|V|-2}$ such that $a_{uvw} + tb_{uvw} \geq c_{uvw}$ for all $u, v, w \in V, |\{u, v, w\}| = 3$, where a_{uvw}, b_{uvw} and c_{uvw} are the distances of the triangle u, v, w in the graph with $a_{uvw} \leq b_{uvw} \leq c_{uvw}$. For that problem Hu proved the existence of a star which is an optimum spanning tree, therefore a solution consisted of finding a minimum cost star among the $|V|$ spanning stars.

The $\mathcal{NP}$-hardness for OCT was only proved four years after its introduction in (JOHNSON; LENSTRA; KAN, 1978) by showing a reduction from the 3-exact cover problem to MRCT. Other theoretical results for OCT studied the problem and its special cases approximability. In (WU et al., 1998) a PTAS for MRCT was presented, proving that the metric case of MRCT is also $\mathcal{NP}$-hard, and a $\mathcal{O}(log^2(|V|))$-approximation for OCT was given, later improved to a $\mathcal{O}(log(|V|))$-approximation in (FAKCHAROEN-PHOL; RAO; TALWAR, 2003). In (WU; CHAO; TANG, 2000a), a 2-approximation for SROCT and a 1.577-approximation for PROCT were given. In (WU, 2004), the $\mathcal{NP}$-hardness of p-OCT was proved for any $p \geq 2$, even for the metric case, and the authors also presented a 2-approximation for the metric p-OCT and a 3-approximation for 2-OCT. In (RAVELO; FERREIRA, 2015) some properties for special cases of p-OCT were proved, allowing to obtain several approximation algorithms for these particular cases. In (RAVELO; FERREIRA, 2017), a PTAS for the metric case of SROCT was shown, and it was used to obtain a more general PTAS for the metric case of WSDOCT in (RAV-ELO; FERREIRA, 2019). Also in (RAVELO; FERREIRA, 2019) the authors proposed a 2-approximation for WSDOCT.

Other researches explored exact algorithms and mathematical formulations for solving the OCT. An exact combinatorial algorithm was proposed by (AHUJA; MURTY, 1987), while the first mathematical model was presented twenty years later by (ROTH-LAUF, 2007). After that, other models were given such as the flow-based and path-based formulations of (CONTRERAS, 2009), and the rooted tree based formulation of (MOTA, 2015). These formulations will be discussed in Chapter 2.

Finally, some studies focused on metaheuristics approaches for OCT. (ROTH-LAUF, 2009) used a combined genetic algorithm that constructs solutions leading towards minimum spanning trees (KRUSKAL, 1956), and after the genetic algorithm execution a simulated annealing algorithm was used to improve the solution. In (STEITZ; ROTH-LAUF, 2011) a guided local search (i.e. a local search that avoids reaching the same local minima by increasing the weight of the edges in an already found local minima) for OCT was presented. In (SATTARI; DIDEHVAR, 2013) and (SATTARI et al., 2015) were given, respectively, a VNS and GRASP metaheuristic for MRCT. One of the most recent heuristics for MRCT was presented in (MASONE et al., 2019), where minimum path spanning trees (DIJKSTRA, 1959) were used to rank the nodes in priority layers, used to generate an initial solution, later improved by a local search.

2 MATHEMATICAL FORMULATIONS

Mathematical models have been a strategy to approach many combinatorial optimization problems (CONFORTI; CORNUEJOLS; ZAMBELLI, 2014), being used in matheuristics (MANIEZZO; STÜTZLE; VOSS, 2009), as part of polyhedral studies of the problems (SCHRIJVER, 2003), to obtain approximation results (BELLMAN; ROTH, 2012), or to produce bounds during exact algorithms executions (FOMIN; KRATSCH, 2010). Additionally, several solvers have being developed to explore these formulations in the attempt of solving difficult combinatorial optimization problems (e.g., Gurobi (OPTIMIZATION, 2021) and CPLEX (CPLEX, 2009)). In this chapter, we present a new mixed integer linear programming formulations for OCT and discuss the previous formulations from literature.

2.1 Formulations from the literature

The first mathematical formulation for OCT was presented in (ROTHLAUF, 2007). This formulation uses the binary variables x_e to represent if edge e belongs to the solution, y^e_{uv} to represent if the path between u and v in the solution contains edge e, and z^w_{uv} to represent if node w is on the path from u to v.

$$\min \sum_{u \in V} \sum_{v \in V} \sum_{e \in E} r_{uv} \ell_e y^e_{uv} \tag{2.1}$$

$s.t.:$

$$\sum_{(u,w) \in E} y^{uw}_{uv} = 1, \qquad \forall u, v \in V, u \neq v, \tag{2.2}$$

$$\sum_{(v,w) \in E} y^{vw}_{uv} = 1, \qquad \forall u, v \in V, u \neq v, \tag{2.3}$$

$$\sum_{(w,z) \in E} y^{wz}_{uv} = 2z^w_{uv}, \qquad \forall u, v, w \in V, u \neq v \neq w, \tag{2.4}$$

$$\sum_{u \in V} \sum_{v \in V} y^e_{uv} \geq x_e, \qquad \forall e \in E, \tag{2.5}$$

$$\sum_{u \in V} \sum_{v \in V} y^e_{uv} \leq M x_e, \qquad \forall e \in E, \tag{2.6}$$

$$\sum_{e \in E} x_{uv} = |V| - 1, \tag{2.7}$$

$$x_{uv}, y^{wz}_{uv}, z^w_{uv} \in \{0, 1\}, \qquad \forall u, v, w, z \in V. \tag{2.8}$$

In the above formulation, M is an upper bound on the number of times an edge can be used in a path between any two vertices, thus, M may be equal to $|V|^2$. Constraints 2.2 and 2.3 guarantee that there are only one start and one end for every pair of paths in solution. Constraints 2.4 ensure that if a node w is in the path from u to v then such a path contains two consecutive edges incident in w. Notice that, constraints 2.2, 2.3 and 2.4 guarantee the existence of an unique path between every two vertices. Constraints 2.5 and 2.6 ensure that any selected edge is used in at least one path. Finally, constraint 2.7 defines that exactly $n - 1$ edges are used in the solution, all these restrictions make sure that the final solution is a tree.

2.1.1 Flow-based Formulation

(CONTRERAS, 2009) presented a flow-based formulation. It uses the binary variables x_{uv} to represent whether the edge (u, v) is present in the spanning tree and the integer variables f^o_{uv} whose value is the total flow with origin at o traversing the arc (u,v). The set R is defined as $R = \{(u, v) | u, v \in V, r_{uv} > 0\}$.

$$\min \sum_{o \in V} \sum_{(u,v) \in A} \ell_{uv} f^o_{uv} \tag{2.9}$$

$s.t.:$

$$\sum_{(u,v) \in E} x_{uv} = |V| - 1, \tag{2.10}$$

$$\sum_{(u,o) \in A} f^o_{uo} = 0, \qquad \forall o \in V, \tag{2.11}$$

$$\sum_{(u,v) \in A} f^o_{uv} - \sum_{(v,w) \in A} f^o_{vw} = r_{ov}, \qquad \forall o \in V, \forall v \in V \setminus \{o\}, \tag{2.12}$$

$$\sum_{(o,w) \in A} f^o_{ow} = \sum_{(o,d) \in R} r_{od}, \qquad \forall o \in V, \tag{2.13}$$

$$f^o_{uv} + f^o_{vu} \leq (\sum_{(o,d) \in R} r_{od}) x_{uv}, \qquad \forall o \in V, \forall (u,v) \in E, \tag{2.14}$$

$$x_{uv} \in \{0,1\}, \qquad \forall (u,v) \in E, \tag{2.15}$$

$$f^o_{uv} \geq 0, \qquad \forall o \in V, \forall (u,v) \in A. \tag{2.16}$$

The above formulation was transcribed from (MOTA, 2015). The objective function 2.9 defines the minimization of the product between the length of each arc and the flow passing by this arc for each origin. Constraint 2.10 ensures only $n-1$ edges are selected. Constraints 2.11, 2.12 and 2.13 guarantee that the origin does not receive flow, the flow conservation and that the origin sends the sum of all its requirements as initial flow, respectively. Constraints 2.14 ensure that the flows from each origin can not be greater than the initial flow if the edge is used, otherwise the flow must be zero.

2.1.2 Path-based Formulation

Another mathematical model presented in (CONTRERAS, 2009) is the path based formulation, using the binary variables x_{uv} to represent whether the edge (u, v) is present in the spanning tree and the continuous variables y^r_{uv} to determine if the arc (u, v) is in the path used to satisfy the communication requirement r.

$$\min \sum_{s\in R}\sum_{u\in V}\sum_{v\in V\setminus\{u\}} r_s c_{uv} y^s_{uv} \tag{2.17}$$

$s.t.:$

$$\sum_{w\in V\setminus\{u\}} y^s_{uw} = 1, \qquad \forall s=(u,v)\in R, \tag{2.18}$$

$$\sum_{v\in V\setminus\{u\}} y^s_{vu} - \sum_{v\in V\setminus\{u\}} y^s_{uv} = 0, \qquad \forall s\in R, \forall u\in V\setminus\{o^s,d^s\}, \tag{2.19}$$

$$y^s_{uv} + y^s_{vu} \leq x_{uv}, \qquad \forall s\in R, \forall (u,v)\in E, \tag{2.20}$$

$$\sum_{(u,v)\in E} x_{uv} = |V|-1, \tag{2.21}$$

$$y^s_{uv} \geq 0, \qquad \forall s\in R, \forall u,v\in V, u\neq v\in E, \tag{2.22}$$

$$x_{uv}\in\{0,1\}, \qquad \forall (u,v)\in E. \tag{2.23}$$

In the above formulation, constraints 2.18 and 2.19 guarantee the connection between the vertices and the flow conservation, respectively. Constraints 2.20 ensure that only arcs in the solution can be in the path. Finally, constraint 2.21 defines that $n-1$ nodes are used in the solution.

2.1.3 Root-based Formulation

A new model was presented in (MOTA, 2015) that was named root-based formulation where $o\in V$ is the root and it uses the binary variables x_{uv} to represent whether the edge (u, v) is present in the solution, the binary variables p_{uv} to indicate if there is a directed path from u to v, and the integer variable d_{uv} whose value represents the unique distance between nodes u and v in the spanning tree. The notation for the following formulation is as follows: M is an upper bound on the length of the longest path in G, $\delta^o_v = 1$ if $v=o$, otherwise $\delta^o_v = 0$, and $(u,v,w)\in\binom{V}{3}$ means that $u,v,w\in V$ and $|\{u,v,w\}|=3$.

$$\min \sum_{(u,v)\in R} r_{uv} d_{uv} \tag{2.24}$$

$s.t.:$

$$\sum_{(u,v)\in A} x_{uv} = 1 - \delta_v^o, \tag{2.25}$$

$$x_{uv} \le p_{uv}, \qquad \forall (u,v) \in A, \tag{2.26}$$

$$p_{uv} + p_{vu} \le 1, \qquad \forall (u,v) \in A, \tag{2.27}$$

$$p_{uv} + x_{vw} \le 1 + p_{uw}, \qquad \forall (u,v,w) \in \binom{V}{3}, \tag{2.28}$$

$$p_{uw} + x_{vw} \le 1 + p_{uv}, \qquad \forall (u,v,w) \in \binom{V}{3}, \tag{2.29}$$

$$d_{uv} \ge d_{uw} + \ell_{wv} - M(2 - x_{wv} - p_{uw}), \qquad \forall (u,v,w) \in \binom{V}{3}, \tag{2.30}$$

$$d_{uv} \ge d_{uw} + \ell_{wv} - M(1 - x_{wv} + p_{uv} + p_{vu}), \qquad \forall (u,v,w) \in \binom{V}{3}, \tag{2.31}$$

$$x_{uv} \in \{0,1\}, \qquad \forall (u,v) \in A, \tag{2.32}$$

$$p_{uv} \in \{0,1\}, \qquad \forall (u,v) \in A, \tag{2.33}$$

$$d_{uv} \ge c_{uv}(x_{uv} + x_{vu}), \qquad \forall (u,v) \in E. \tag{2.34}$$

In the above formulation, constraint 2.24 defines the minimization of the objective function. Constraint 2.25 ensures that every node that is not the root o has an incoming edge, hence the solution has $n-1$ edges. Constraints from 2.26 to 2.29 ensure the variable p is correct for every pair of nodes. Constraints 2.30, 2.31 and 2.34 guarantee that the distance lower bound is equal to the distance in the tree for each pair of nodes.

2.2 New formulation

In this section, we propose a new mixed integer linear program based on common ancestors. We define the root to be node o and the binary variables are defined as:

$$x_a = \begin{cases} 1, & \text{if arc } a \text{ is used in the solution} \\ 0, & \text{otherwise.} \end{cases}$$

$$y_{uv} = \begin{cases} 1, & \text{if } u \text{ is ancestral of } v \\ 0, & \text{otherwise.} \end{cases}$$

$$z_{wuv} = \begin{cases} 1, & \text{if } w \text{ is ancestral of both } u \text{ and } v \\ 0, & \text{otherwise.} \end{cases}$$

Besides the binary variables, we use the following integer variables:

$\eta_u =$ number of nodes in the unique path from node u to o.

Also, we use the continuous non-negative variables defined below:

$\delta_u =$ distance from node u to o.

$$\rho_{wuv} = \begin{cases} \text{length of incoming arc of } w, & \text{if } w \text{ is ancestral of both } u \text{ and } v \\ 0, & \text{otherwise.} \end{cases}$$

Finally, we use a constant value to represent the sum of all edge lengths:

$$D = \sum_{e \in E} \ell(e)$$

To ensure every node except the root has an incoming arc, we use the constraints 2.35 and 2.36:

$$\sum_{a \in N^-(o)} x_a = 0, \tag{2.35}$$

$$\sum_{a \in N^-(u)} x_a = 1, \qquad \forall u \in V \setminus \{o\}, \tag{2.36}$$

Constraint 2.37 sets the distance from the root to 0, while constraints 2.38 and 2.39 ensure that the distance δ_u from any node $u \in V$ to the root is set correctly.

$$\delta_o = 0, \tag{2.37}$$

$$\delta_v >= \delta_u + x_a \ell_{uv} - (1 - x_a)D, \qquad \forall a = (u, v) \in A, \tag{2.38}$$

$$\delta_v <= \delta_u + x_a \ell_{uv} + (1 - x_a)D, \qquad \forall a = (u, v) \in A, \tag{2.39}$$

Similarly to the constraints above, constraints 2.40, 2.41 and 2.42 set the correct values for η_u for all $u \in V$.

$$\eta_o = 0, \tag{2.40}$$

$$\eta_v >= \eta_u + x_a - (1 - x_a) * |V|, \qquad \forall a = (u, v) \in A, \tag{2.41}$$

$$\eta_v <= \eta_u + x_a + (1 - x_a) * |V|, \qquad \forall a = (u, v) \in A, \tag{2.42}$$

We add the constraints 2.43 to ensure the number of ancestors of a node is equal to the number of nodes between the path from root to itself. Every node is considered ancestor of itself and this is represented by the constraints 2.44. If there is an arc between nodes u and v then u is ancestor of v and this is defined by the constraints 2.45. Constraints 2.46 guarantee that the transitive property of ancestors is satisfied. Therefore, the following set of constraints ensure the correctness of the y variables.

$$\sum_{u \in V} y_{uv} = \eta_v + 1, \qquad \forall v \in V, \tag{2.43}$$

$$y_{uu} = 1, \qquad \forall u \in V, \tag{2.44}$$

$$y_{uv} >= x_a, \qquad \forall a = (u, v) \in A, \tag{2.45}$$

$$y_{uv} + y_{vw} <= 1 + y_{uw}, \qquad \forall u \in V, \forall v \in V \setminus \{u\}, \forall w \in V, \tag{2.46}$$

To correctly assign z_{wuv} to true if both z_{wu} and z_{wv} are true, we add the constraints 2.47 and 2.48 that use the logical **AND** operator.

$$2z_{wuv} <= y_{wu} + y_{wv}, \qquad \forall u \in V, \forall v \in V \setminus \{u\}, \forall w \in V, \tag{2.47}$$

$$z_{wuv} + 1 >= y_{wu} + y_{wv}, \qquad \forall u \in V, \forall v \in V \setminus \{u\}, \forall w \in V, \tag{2.48}$$

The final constraints 2.49 and 2.50 enforce that ρ_{wuv} is equal to l_{xw} if and only if the arc a=(x,w) is used and w is ancestor of both u and v.

$$\rho_{wuv} <= \sum_{a \in N^-(w)} \ell_a x_a, \qquad \forall u \in V, \forall v \in V \setminus \{u\}, \forall w \in V, \tag{2.49}$$

$$\rho_{wuv} <= \sum_{a \in N^-(w)} \ell_a z_{wuv}, \qquad \forall u \in V, \forall v \in V \setminus \{u\}, \forall w \in V. \tag{2.50}$$

Finally, 2.51 is the objective function for our mathematical model, which must satisfy all the above constraints. Note that the distance between nodes u and v is calculated as $\delta_u + \delta_v - 2r_{uv} \sum_{w=1}^{|V|} \rho_{wuv}$, this is due to the fact that arcs incident in an ancestor of both u and v arc is counted twice, one in δ_u and another in δ_v and we subtract this value if w is ancestral of u and v.

$$\min \sum_{(u,v)\in\binom{V}{2}} r_{uv}(\delta_u + \delta_v) - 2r_{uv} \sum_{w\in V} \rho_{wuv} \tag{2.51}$$

3 EVOLUTIONARY ALGORITHMS

Evolutionary algorithms have been used to approach several combinatorial problems (YU; GEN, 2012) and (ROTHLAUF, 2006). These algorithms are based on Darwin's evolution principle and natural selection (DARWIN, 2009). They start creating an initial population of solutions, assigning to each individual a fitness value which represents the quality of the solution that is measured with its objective function. Then, some individuals in the population are selected based on their fitness to be the parents who crossover to obtain a child that shares features from different parents. Then, the parents and the offspring can mutate (i.e., apply changes to the solution). Finally, the fitness values are re-evaluated and (based on those values) the solutions with high fitness value from the parents and the offspring are selected to form the new generation. These algorithms have presented a good performance for several optimization problems, finding optimal and almost optimal solutions in a very short time. Therefore we propose four evolutionary algorithms and a combined strategy in this chapter.

3.1 Evolutionary algorithm for OCT

In this section we attempt to solve OCT by proposing four different strategies for an evolutionary algorithm. Each strategy has different procedures for the generation of the initial population and for the crossover operation. Also, a combined strategy using evolutionary algorithm and simulated annealing is presented. Algorithm 1 is common for all strategies and describes our evolutionary approach.

In addition to the instance of OCT, Algorithm 1 receives the following parameters:

- `POPULATION_SIZE`, to indicate the size of the population at the beginning of each iteration;
- `MAX_ITERATIONS` and `MAX_NO_UPDATE`, to indicate, respectively, the maximum number of generations and the maximum number of generations without updating the best solution found;
- `CROSSOVER_NUMBER`, to indicate the number of new solutions generated by crossover operations at each generation;
- `MUTATION_NUMBER`, to indicate the number of times the mutation operation is executed at each generation;

```
Input  : $\langle G, r, \ell \rangle$, instance of OCT.
Output: $A^*$, solution of OCT with instance $\langle G, r, \ell \rangle$.
1  begin
2      $A^* \leftarrow \emptyset$
3      $S_0 \leftarrow \emptyset$
4      while $|S_0| <$ POPULATION_SIZE do
5          $A \leftarrow$GENERATE($G, r, \ell$)
6          $S_0 \leftarrow S_0 \cup \{A\}$
7      end
8      $i \leftarrow 0$
9      $j \leftarrow 0$
10     while $i \leq$MAX_ITERATIONS and $j \leq$MAX_NO_UPDATE do
11         for each $A \in S_i$ do
12             $fitness[A] \leftarrow$ compute fitness of $A$ in $S_i$
13             if $C(A) < C(A^*)$ then
14                 $A^* \leftarrow A$
15                 $j \leftarrow 0$
16             end
17
18         end
19         $S' \leftarrow S_i$
20         for $l \leftarrow 1$ to CROSSOVER_NUMBER do
21             $A_1, A_2 \leftarrow$ pair of different solutions from $S_i$
22             $A \leftarrow$CROSSOVER($A_1, A_2, G, r, \ell$)
23             $S' \leftarrow S' \cup \{A\}$
24             if $C(A) < C(A^*)$ then
25                 $A^* \leftarrow A$
26                 $j \leftarrow 0$
27             end
28         end
29         for $l \leftarrow 1$ to MUTATION_NUMBER do
30             $A \leftarrow$ random solution from $S'$
31             $A \leftarrow$ MUTATION($A, G, r, \ell$)
32             if $C(A) < C(A^*)$ then
33                 $A^* \leftarrow A$
34                 $j \leftarrow 0$
35             end
36         end
37         $S_{i+1} \leftarrow$POPULATION_SIZE solutions from $S'$
38         $i \leftarrow i + 1$
39         $j \leftarrow j + 1$
40     end
41     return $A^*$
42 end
```

Algorithm 1: Evolutionary algorithm for OCT.

Algorithm 1 begins with an empty solution $\bar{A}$ at line 2, which is updated in lines 13-16 and 24-27 every time a better solution is found (i.e, a solution with lower objective value). Hence, at the end of the algorithm, the returned value $\bar{A}$ represents the best solution found. The first generation of solutions, denoted S_0, is initialized as an empty collection in line 3, and `POPULATION_SIZE` generated solutions are added between lines 4 and 7. The different generation procedures will be explained for each strategy in the following subsections.

After generating the initial population, the variables i and j are initialized (lines 8 and 9) and they reference, respectively, the current number of generations and the number of generations without updating the best solution found. The values of i and j are updated between lines 38 and 39 of the algorithm main loop (lines 10 to 40) and, whenever a new best solution is found, variable j is reset (lines 26 and 34).

At the beginning of every iteration of the main loop, the fitness function for each solution A is calculated at line 12 and defined as:

$$fitness[A] = \begin{cases} 1.2 - \frac{C(A) - minV}{maxV - minV}, & \text{if } minV \neq maxV \\ 1.0, & \text{if } minV = maxV \ . \end{cases}$$

where $minV = \min_{s \in S_i} C(s)$ and $maxV = \max_{s \in S_i} C(s)$. This fitness function ensures that solutions with low communication cost values have high fitness value associated.

Succeeding the fitness computation, the set S' of candidate solutions for the next generation is initialized as S_i (line 19), and between lines 20 and 28 the crossover operations are executed. Each crossover starts at line 21 where two different solutions A_1 and A_2 are randomly selected from S_i, being the probability of selecting a solution $s \in S_i$ equal to $\frac{fitness[s]}{\sum_{s' \in S_i} fitness[s']}$. The different crossover operations will be explained in the subsections of their their respective strategies.

Between lines 29 and 36 all mutation operations are executed. A solution from S' is selected uniformly at random at line 30, then at line 31 the mutation operation is performed. We propose, two different mutation operators for all strategies, which are randomly selected by our algorithm for each solution to be mutated. Both mutations will be explained after the algorithm explanation.

Finally, at line 37, `POPULATION_SIZE` solutions are selected from S' for the next generation. The solutions are chosen with a tournament selection algorithm, where the number of tournaments is $10 \times$ `OFFSPRING_SIZE` (`POPULATION_SIZE` $+$ `CROSSOVER_NUMBER`), the number of participants of each tournament is 2 plus an integer chosen uniformly at random in the interval $\left[0, \max\left\{0, \frac{\texttt{POPULATION\_SIZE}}{10} - 1\right\}\right]$. For each tournament, a probability multiplier parameter $p = 0.9$ is used, being $p_0 = 0.8$ the initial probability of selecting the best offspring solution. If the best solution is not selected, then the second best will be selected with probability $p_0 \times p$, and if both are not selected, then the probability of selecting the next best solution is $p_0 \times p^2$, and so on. The `POPULATION_SIZE` solutions with most wins from S' are selected for the next generation where repetitions are allowed.

3.1.1 Mutations

The first mutation operator is described by Algorithm 2, which starts by randomly selecting an edge $e \in E \setminus E_T$, to be inserted in solution T at line 2. Then, the subproblem that only considers the edges in $E_T \cup \{e\}$ is optimally solved at line 3, being its solution $\bar{T}$ the result of this mutation. The time complexity of the implemented algorithm is $\mathcal{O}(|V|^2)$.

```
Input  : a solution $T$ and a instance $\langle G, r, \ell \rangle$ of OCT.
Output: $\bar{T}$, mutation of solution $T$ for OCT with instance $\langle G, r, \ell \rangle$.
1 begin
2     $e \leftarrow$ select a random edge from $E \setminus E_T$
3     $\bar{T} \leftarrow$ optimal solution considering only edges $\in E_T \cup \{e\}$
4     return $\bar{T}$
5 end
```

Algorithm 2: Mutation by edge insertion.

The second mutation is shown in Algorithm 3, and starts with a random selection of an edge $e \in E_T$, to be removed from the solution T at line 2. The selected edge is removed from T at line 3, partitioning the tree in two sub-trees $T1$ and $T2$. Then, at line 4, we fix to be in the solution all the edges of T_1 and T_2, and we optimally solve the subproblem considering only the edges of E with one extreme in T_1 and the other in T_2. The optimal solution $\bar{T}$ for the described subproblem is returned as the result of the second mutation. The time complexity of the implemented algorithm is $\mathcal{O}(|E| \times |V|^2)$.

In the next subsections we describe the procedure for generating the initial population and the crossover operator of each strategy.

```
Input  : a solution $T$ and a instance $\langle G, r, \ell \rangle$ of OCT.
Output: $\bar{T}$, mutation of solution $T$ for OCT with instance $\langle G, r, \ell \rangle$.
1 begin
2     $e \leftarrow$ select a random edge from $E_T$
3     $T \leftarrow T \setminus \{e\}$
4     $\bar{T} \leftarrow$ select optimal solution considering the best edge $\in E$ to add to $T$
5     return $\bar{T}$
6 end
```

Algorithm 3: Mutation by edge removal.

3.2 Random strategy

The generation of the initial population with random strategy is very similar to Kruskal's algorithm for minimum spanning trees (KRUSKAL, 1956), but the list of edges is shuffled instead of being sorted in ascending length order. Algorithm 4 describes this idea, whose time complexity is $\mathcal{O}(|E| \times \log |V|)$.

```
Input  : $\langle G, r, \ell \rangle$, instance of OCT.
Output: $T$, initial solution
1  begin
2      shuffle $E$
3      $T \leftarrow \emptyset$
4      $i \leftarrow 0$
5      while $T$ is not a spanning tree of $G$ do
6          if $E_i$ does not form a cycle when added to $T$ then
7              add $E_i$ in $T$
8          end
9          $i \leftarrow i + 1$
10     end
11     return $T$
12 end
```

Algorithm 4: Initial solution generation with random strategy.

Algorithm 4 starts randomly shuffling all edges in E at line 2, initializing an empty solution T at line 3 and a variable i to iterate the shuffled list of edges at line 4. To ensure that the created solution is valid, in the main loop between lines 5 and 10, we add the current edge E_i (line 7) iff the addition of E_i to T maintains the solution T acyclic. Then, at line 9 the current edge is updated. Finally, the output of the algorithm is the constructed spanning tree T.

The crossover operator for the random strategy only considers the edges of both parents. The idea is given by Algorithm 5 and consist of shuffling the adjacency list of each node, and then to iterate over the nodes selecting those edges that do not form cycles.

```
   Input  : two parent solutions $T_1$, $T_2$ and an instance $\langle G, r, \ell \rangle$ of OCT.
   Output: $T$, new solution constructed by the crossover between $T_1$ and $T_2$.
 1 begin
 2     $G' \leftarrow T_1 \cup T_2$
 3     for $v \in V$ do
 4         shuffle adjacency list of $v$ in $V$
 5     end
 6     $T \leftarrow \emptyset$
 7     $v \leftarrow 0$
 8     while $T$ is not a spanning tree of $G'$ do
 9         $e \leftarrow$ first edge in adjacency list of node $v$
10         if $e$ exists then
11             if $T \cup \{e\}$ is acyclic then
12                 add $e$ in $T$
13             end
14         end
15         remove $e$ from adjacency list of $v$
16         $v \leftarrow$ next node that contains edges in its adjacency list
17     end
18     return $T$
19 end
```

Algorithm 5: Crossover with random strategy.

Algorithm 5 starts creating a subgraph G' with only the edges of both parents at line 2, and shuffling the adjacency lists of the nodes in G' at line 4. Then, the new empty solution T is initialized at line 6 and edges are added to the solution at line 12 only if they do not form cycles. To iterate over the nodes we use the variable v, initialized as the first node at line 7, whose value is updated at line 9. The idea is that each node will have a chance to select an edge of its adjacency list to the solution. Since only the first edge of the adjacency lists are analyzed (line 9) and after the analysis the edges are removed from the list (line 15), it follows that the edges are selected in the order they were shuffled and that the new solution T is randomly constructed. The time complexity of the implemented algorithm is $\mathcal{O}(|V| \times \log |V|)$ because the number of edges of two trees is $\mathcal{O}(|V|)$.

3.3 Greedy probabilistic strategy

The greedy probabilistic strategy is based in the fact that the communication cost $C(T)$ of a tree $T = (V_T, E_T)$ can be written as:

$$C(T) = \sum_{e \in E_T} \ell(e) \times \sum_{u \in V_{T_1^e}} \sum_{v \in V_{T_2^e}} r(u, v),$$

where $T_1^e = (V_{T_1^e}, E_{T_1^e})$ and $T_2^e = (V_{T_2^e}, E_{T_2^e})$ are the sub-trees of T obtained after removing the edge e. Since $C(T)$ depends on the length $\ell(e)$ of each edge $e \in E_T$, this strategy attempts to minimize the communication cost by selecting the shorter edges with greater probability.

Thus, the idea for the generation of the initial population is very similar to the random strategy, but the edges are selected with probability depending on their lengths instead of being randomly shuffled. Algorithm 6 shows this procedure.

```
    Input  : $\langle G, r, \ell \rangle$, instance of OCT.
    Output: $T$, initial solution
1   begin
2       $T \leftarrow \emptyset$
3       for $e \in E$ do
4           $capability[e] \leftarrow$ compute capability of $e$ in $E$
5       end
6       while $T$ is not a spanning tree of $G$ do
7           $e \leftarrow$ select an edge from $E$
8           $E \leftarrow E \setminus \{e\}$
9           if $T \cup \{e\}$ is acyclic then
10              add $e$ in $T$
11          end
12      end
13      return $T$
14  end
```

Algorithm 6: Solution generation with greedy probabilistic strategy.

Algorithm 6 starts with an empty initial solution T at line 2, and a capability value is associated to each edge at line 4. The edge capability is defined as follows:

$$capability[e] = \begin{cases} 1.4 - \frac{\ell(e) - minV}{maxV - minV}, & \text{if } minV \neq maxV \\ 1.0 & \text{, otherwise,} \end{cases}$$

where $minV = \min_{e' \in E} \ell(e')$ and $maxV = \max_{e' \in E} \ell(e')$. The above function ensures that shorter edges have high capability values. Then, the edge selection at line 7 depends on the edges capabilities, where an edge e is chosen with probability $\frac{capability[e]}{\sum_{e' \in E} capability[e']}$.

After that, the selected edge is removed from E at line 8, and included in the solution T if it does not form cycles with the previously added edges. Finally, the algorithm stops when a spanning tree T is constructed, being the time complexity $\mathcal{O}(|E|^2)$.

Similar to the random strategy, a graph considering only the edges of both parents is used for the crossover of the greedy probabilistic strategy. Therefore, given two parent solutions $T_1 = (V, E_1)$ and $T_2 = (V, E_2)$, the graph $G' = (V, E_1 \cup E_2)$ is considered as input of Algorithm 6, and the output of that algorithm is the new solution resulting from the crossover of T_1 and T_2. Since the number of edges in G' is $2 \times (|V| - 1)$, the time complexity of each crossover is $\mathcal{O}(|V|^2)$.

3.4 Minimum path strategy

This strategy emerges from theoretical results for particular classes of OCT that include some generalizations of MRCT. In (RAVELO; FERREIRA, 2019) and (WU; CHAO; TANG, 2000a), the authors proved that for some $\mathcal{NP}$-hard variants of OCT, a node u exists in the graph, such that a minimum path tree from u to the rest of nodes guarantees a 2-approximation of the optimal value. Therefore, the idea of the minimum path strategy is based on generating minimum paths trees and combining them.

The initial population is generated by picking nodes from the graph and then generating their minimum path tree with Dijkstra's algorithm for minimum paths from a source to the rest of the graph (DIJKSTRA, 1959). Algorithm 7 illustrates this idea.

```
  Input  : $\langle G, r, \ell \rangle$, instance of OCT.
  Output: $T$, initial solution
1 begin
2     $v \leftarrow$ select a node from $V$
3     $T \leftarrow$Dijkstra($v$, $G$, $\ell$)
4     return $T$
5 end
```

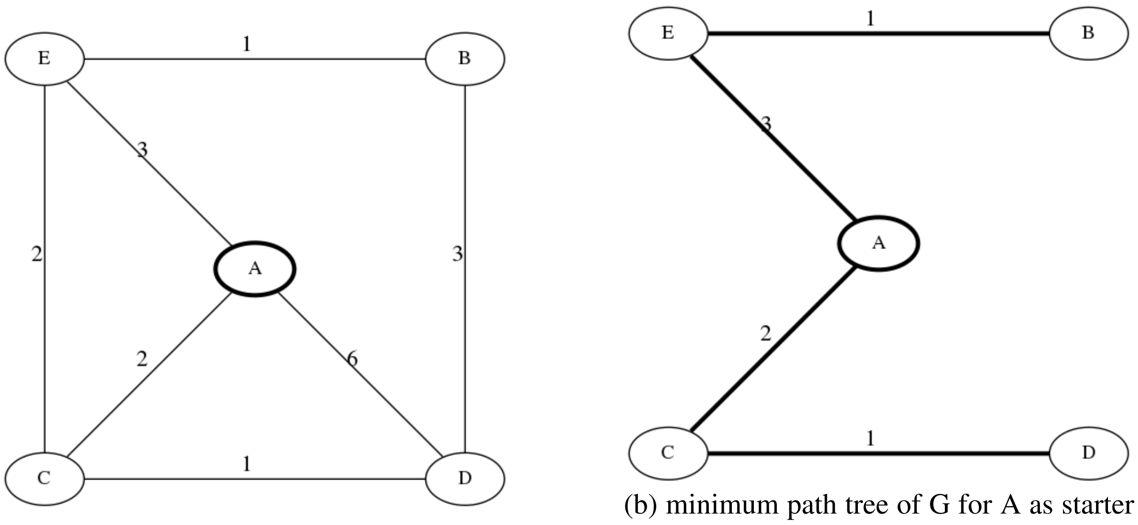
Algorithm 7: Solution generation with minimum path strategy.

In Algorithm 7 a node is selected at line 2 if it was not previously selected to generate a minimum path tree. If all nodes were already selected to generate initial solutions, a node is randomly chosen at line 2 and, before calling Dijkstra algorithm at line 3 an edge removal procedure is called, where each edge of the minimum path tree has 0.1 probability of being removed. Hence, some edges may be removed generating variability to the new constructed solution. The time complexity of the implemented algorithm is

$\mathcal{O}(|V| \log |V|)$.

Analogously to previous strategies, the crossover is executed over the subgraph G' that only considers the edges of both parents. For the crossover we call the same Algorithm 6 with the graph G' constructed from the two given parents (similar to the greedy probabilistic strategy), where at line 2 the node is always randomly selected and no edge is removed from G' before performing Dijkstra algorithm. The time complexity of the implemented algorithm is $\mathcal{O}(|V| \log |V|)$.

Figure 3.1: Example of the minimum path strategy. Source: Author.



(a) example of an initial graph G

(b) minimum path tree of G for A as starter node

3.5 Minimum k-core strategy

Similar to the previous strategy, this strategy explores ideas from theoretical results for the special cases of OCT. In (RAVELO; FERREIRA, 2015), (RAVELO; FERREIRA, 2017), (RAVELO; FERREIRA, 2019) and (WU; CHAO; TANG, 2000b), the authors proved that polynomial time approximation schemes (PTAS) can be obtained for several variants of OCT, that guarantee solutions whose values are $(1+\epsilon)$-approximations for each $\epsilon > 0$, where the time complexity is polynomial in the graph size but exponential in ϵ. The proposals select a set of k nodes named **core** (where k is a function on ϵ), and enumerate all possible trees over the core adding the other $n - k$ nodes as leaves. Although the results of the PTAS are very good, the time complexity is impractical because the required values of k are usually large. Hence, we bound k to be small and select the nodes to form the core, hybridizing this approach with minimum path trees.

To generate a new solution we randomly select a small core and disconnect it (remove the edge between its nodes). Then, we obtain a minimum path forest from the core nodes to the rest of the graph, and select the best tree connecting the core nodes to obtain the solution. Algorithm 8 describes this idea.

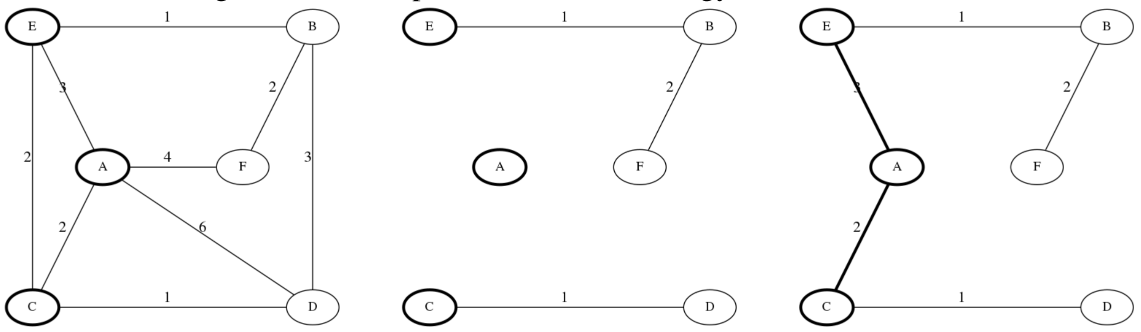
```
  Input  : $\langle G, r, \ell \rangle$, instance of OCT.
  Output: $T$, initial solution
1 begin
2     $k \leftarrow$ a random integer between $[2, 5]$
3     core$\leftarrow$ select $k$ random different and connected nodes
4     remove edges of $G$ that has both endpoints are at the core
5     $F \leftarrow$Multi-source Dijkstra(core, $G$, $\ell$)
6     $T \leftarrow F\cup$ edges of the best tree that connects the core
7     return $T$
8 end
```

Algorithm 8: Solution generation with k-core strategy.

Algorithm 8 starts by picking a random core size $k \in [2, 5]$ at line 2. Then, the core is selected at line 3 by choosing k random different nodes such that the induced subgraph over them is connected. After the core selection, all edges from G with both endpoints in the core are removed from the graph at line 4. At line 5 a version of multi-source Dijkstra (DIJKSTRA, 1959) is executed over the core nodes, so the result is a forest F with k disconnected trees, each one having one node from the core. Thus, at line 6 we select a tree on the core nodes that minimizes the communication cost of the constructed solution. Such a selection is done by enumerating all possible trees on the core nodes with Prüfer code (PRÜFER, 1918). Finally, the new solution T is obtained by adding to the forest F the tree connecting the core nodes. Since $k = \mathcal{O}(1)$, to find the best tree for the core requires $\mathcal{O}(|V|^2)$ computational time, dominating the multi-source Dijkstra algorithm whose time complexity is $\mathcal{O}(|V| \times \log |V|)$.

The crossover of this strategy randomly selects a K core, identically to lines 2 and 3 of Algorithm 8. Then, a graph G' is constructed from both parents, likewise the previous strategies, and executes Algorithm 8 considering G' from line 4 to 7. The time complexity of the implemented algorithm is $\mathcal{O}(|V|^2)$.

Figure 3.2: Example of the k-core strategy. Source: Author.



(a) example of an initial graph G

(b) minimum path tree of G when k-core is {A, C, E}

(c) one possible tree that connect the k-core

3.6 Combined strategy

We designed a combined approach with elements of both the minimum path strategy and a slightly modified simulated annealing from (ROTHLAUF, 2009). Algorithm 9 describes this idea.

In addition to the instance of OCT, Algorithm 9 receives the following extra parameters:

- `POP_SIZE`, to indicate the size of the population to calculate the standard deviation;
- `ITER_MAX` and `ITER_TERM`, to indicate, respectively, the maximum number of generations and the maximum number of generations without updating the best solution found;
- `MAX_NOT_IMPROVING`, to indicate the number of times the simulated annealing should run without improvement;

Algorithm 9 begins with a starter solution that is obtained by running Algorithm 1 for the minimum path strategy without mutations and is stored in $\bar{T}$ at line 3, which is updated in lines 15-19 every time a better solution is found (i.e, a solution with lower objective value). Hence, at the end of the algorithm, the returned value $\bar{T}$ represents the best solution found. The current solution represented by T_c is initialized at line 4 and it is updated between lines 20-24 whenever T_a has a lower objective function than T_c or with probability $\exp(\frac{C(T_c)-C(T_a)}{\mu})$ if $C(T_a) \geq C(T_c)$. Also, the current solution has a perturbation procedure at line 27 where the solution randomly keeps half of the current edges $e \in E_{T_c}$ in the tree and adds the rest of the shortest edges $e \in E$ that can be inserted in the solution with the objective to avoid local minima. The variable T_a

```
Input  : $\langle G, r, \ell \rangle$, instance of OCT.
Output: $\bar{T}$, solution of OCT with instance $\langle G, r, \ell \rangle$.
1  begin
2      $\sigma \leftarrow$ standard deviation of the objective function of POP_SIZE random
         initial solutions
3      $\bar{T} \leftarrow$ EVOLUTIONARY($G, r, \ell$)
4      $T_c \leftarrow \bar{T}$
5      $i \leftarrow 0$
6      while $i \leq$ MAX_NOT_IMPROVING do
7          $i \leftarrow i + 1$
8          $j \leftarrow 0$
9          $k \leftarrow 0$
10         $\mu \leftarrow 2 \times \sigma$
11         while $j \leq$ ITER_MAX and $k \leq$ ITER_TERM do
12             $j \leftarrow j + 1$
13             $k \leftarrow k + 1$
14             $T_a \leftarrow$ MUTATION($T_c, G, r, \ell$)
15             if $C(T_a) < C(\bar{T})$ then
16                 $\bar{T} \leftarrow T_a$
17                 $i \leftarrow 0$
18                 $k \leftarrow 0$
19             end
20             if $C(T_a) < C(T_c)$ then
21                 $T_c \leftarrow T_a$
22             else
23                 $T_c \leftarrow T_a$ with probability $\exp(\frac{C(T_c) - C(T_a)}{\mu})$
24             end
25             $\mu \leftarrow \mu \times 0.99$
26         end
27         $T_c \leftarrow$ perturbate $T_c$
28     end
29     return $\bar{T}$
30 end
```

Algorithm 9: Simulated Annealing for OCT.

represents the neighborhood of a solution and it is obtained by executing the mutation operator (Algorithm 3), however at most 100 edges were analyzed because of its time complexity.

The variables i, j and k are initialized (lines 5, 8 and 9) and they reference, respectively, the current number of simulated annealing without improvement, the current number of generations and the number of generations without updating the best solution found. The values of i, j and k are updated at lines 7, 12 and 13 and, whenever a new best solution is found, variable i and k are reseted (lines 17 and 18).

The standard deviation σ at line 2 is calculated by computing the objective function of `POP_SIZE` random initial solutions that are created by Algorithm 4. Then, the temperature μ is initialized at line 10 every time a new simulated annealing runs and, this value is updated at each iteration of the simulated annealing at line 25.

4 COMPUTATIONAL EXPERIMENTS

To test the applicability of our proposals, this section presents computational experiments over instances from the literature and real-world datasets. We also compare our results with the metaheuristics from (ROTHLAUF, 2009) and (STEITZ; ROTHLAUF, 2011). Our implementations can be accessed through *GitHub hosting platform* (FONSECA, 2021).

The rest of this section describes the computational environment for the experiments, the selected datasets, the parameters used by our algorithms, and finishes with the tests results and their analysis.

4.1 Computational environment

We implemented our algorithms in the C++ programming language using the compiler g++ 5.4.0, and executed all tests in a processor Intel Core i5-4690K with 4 cores of 3,50GHz each, and 8 GB of RAM, under Ubuntu Linux 16.04 LTS 64 bits. To generate pseudo-random float values we used the Mersenne Twister algorithm (SAITO; MATSUMOTO, 1993) implemented in C++ as std::mt19937, and for pseudo-random integer values we used the default C++ function rand().

Since our algorithms use pseudo-random generation, for each instance we executed each algorithm 10 times with the seeds given by the table 4.1 that were randomly generated by Google random number generator (GOOGLE, 2021).

Table 4.1: Seed values used in the experiments.

Number	Seed	Number	Seed
0	280192806	5	316851227
1	871237442	6	619606212
2	2540188929	7	1078082709
3	107472404	8	916212990
4	3957311442	9	698598169

4.2 Instances

The literature instances we consider were previously tested in (MASONE et al., 2019) and (Tan; Due, 2013), and consist of non-euclidean networks with edge lengths in $[0, 10]$. These instances can be found in the *OR-Library* (BEASLEY, 1990), they were originally proposed for the Steiner tree problem (GAREY; GRAHAM; JOHNSON, 1977), and were adapted to MRCT by the authors of (MASONE et al., 2019) and (Tan; Due, 2013). For OCT, we included integer requirement values randomly selected in the interval $[0, 10]$. The algorithm used to generate such values can be found in (FONSECA, 2021). The instances can be grouped by their network sizes as follows:

- **small instances**, `STEIB1-STEIB7`, up to 75 nodes and 100 edges;
- **medium instances**, `STEIC1-STEIC7`, up to 500 nodes and 1000 edges;
- **large instances**, `STEID1-STEID7`, up to 1000 nodes and 2000 edges;

For the mathematical formulation testing we consider the well known literature instances for OCT that are available in (ROTHLAUF, 2006) that are small enough to test this formulations, all these instances are complete graphs.

The real-world datasets were obtained from two different repositories, where the larger instances can be found in the *Stanford Large Dataset Collection* (LESKOVEC; KREVL, 2014) while the smaller are from the *Internet Topology Zoo* (KNIGHT et al., 2011). All instances are described in Table 4.2 where `BKV` represents the best known value for that instance. These instances can be accessed in (FONSECA, 2021).

The instances from the *Stanford Large Dataset Collection* represent temporal information exchange relations between the nodes, thus the requirement communication values were set equal to the number of times the nodes interacted, and the edge lengths equal to 1. Notice that OCT remains $\mathcal{NP}$-hard even when the edge lengths are equal to 1 (JOHNSON; LENSTRA; KAN, 1978). For the test cases `bitcoinalpha` and `bitcoinotc` we considered a different requirement function, because for these instances the network interactions had a feedback value between $[-10, 10]$. Therefore, the requirement between two nodes is calculated as follows:

$$r_{uv} = \begin{cases} 20 + feedback_{uv} + feedback_{vu}, & \text{if an edge exists} \\ 0, & \text{otherwise.} \end{cases}$$

For the *Internet Topology Zoo* instances, each node had associated longitude and latitude coordinates, hence the edge length were calculated by projecting the coordinates to the plane and computing the euclidean distance between their nodes, with precision of three decimal cases. Since for these instances no requirement information was known, we set all requirements equal to 1 turning the problem into MRCT.

Table 4.2: Instances used in the experiments.

Instance	Reference	# of nodes	# of edges	MRCT	BKV
palmer6	(ROTHLAUF, 2006)	6	15	No	693180∗
palmer12	(ROTHLAUF, 2006)	12	66	No	3428509∗
palmer24	(ROTHLAUF, 2006)	24	276	No	1086656∗
berry6	(ROTHLAUF, 2006)	6	15	No	534∗
berry35	(ROTHLAUF, 2006)	35	595	No	16915∗
berry35u	(ROTHLAUF, 2006)	35	595	No	16192
raidl10	(ROTHLAUF, 2006)	10	45	No	53674∗
raidl20	(ROTHLAUF, 2006)	20	190	No	157570∗
raidl50	(ROTHLAUF, 2006)	50	1225	No	813628
STEIB1	(BEASLEY, 1990)	50	63	No	137623
STEIB2	(BEASLEY, 1990)	50	63	No	154760
STEIB3	(BEASLEY, 1990)	50	63	No	123155
STEIB4	(BEASLEY, 1990)	50	100	No	103860
STEIB5	(BEASLEY, 1990)	50	100	No	88085
STEIB6	(BEASLEY, 1990)	50	100	No	111335
STEIB7	(BEASLEY, 1990)	75	94	No	353516
STEIC1	(BEASLEY, 1990)	500	625	No	25830411
STEIC2	(BEASLEY, 1990)	500	625	No	24941878
STEIC3	(BEASLEY, 1990)	500	625	No	30462144
STEIC4	(BEASLEY, 1990)	500	625	No	27963296
STEIC5	(BEASLEY, 1990)	500	625	No	28397460
STEIC6	(BEASLEY, 1990)	500	1000	No	18161548
STEIC7	(BEASLEY, 1990)	500	1000	No	17424693
STEID1	(BEASLEY, 1990)	1000	1250	No	104716691
STEID2	(BEASLEY, 1990)	1000	1250	No	110763595
STEID3	(BEASLEY, 1990)	1000	1250	No	114788009
STEID4	(BEASLEY, 1990)	1000	1250	No	111314217
STEID5	(BEASLEY, 1990)	1000	1250	No	117925269
STEID6	(BEASLEY, 1990)	1000	2000	No	75058079
STEID7	(BEASLEY, 1990)	1000	2000	No	78840144
Palmetto	(KNIGHT et al., 2011)	45	64	Yes	3289.568*
Tw	(KNIGHT et al., 2011)	71	115	Yes	70152.022
Deltacom	(KNIGHT et al., 2011)	113	161	Yes	302217.508
TataNld	(KNIGHT et al., 2011)	145	186	Yes	203685.234
GtsCe	(KNIGHT et al., 2011)	149	193	Yes	727643.130
Colt	(KNIGHT et al., 2011)	153	177	Yes	324569.658
UsCarrier	(KNIGHT et al., 2011)	158	189	Yes	528612.034
Cogentco	(KNIGHT et al., 2011)	197	243	Yes	1898917.778
Kdl	(KNIGHT et al., 2011)	754	895	Yes	8672954.483
email	(LESKOVEC; KREVL, 2014)	986	16064	No	767470
CollegeMsg	(LESKOVEC; KREVL, 2014)	1893	13835	No	179131
bitcoinalpha	(LESKOVEC; KREVL, 2014)	3775	14120	No	823838
bitcoinotc	(LESKOVEC; KREVL, 2014)	5875	21489	No	1222814

* indicates the optimal solution value.

4.3 Results and analysis of mathematical models

Table 4.3 shows the performance of the Gurobi solver (OPTIMIZATION, 2021) over three mathematical models for (ROTHLAUF, 2006) instances. In Table 4.3, `NF` represent our new formulation, `FB` means the flow based formulation presented in (CONTRERAS, 2009) and `RB` is the root based formulation described in (MOTA, 2015), obj means the best value found in the given time limit and t(s) means the time the algorithm took to finish in seconds. We set a time limit of 2 hours for each instance. Note that if the time to execute is less than 7200 seconds than the formulation found the optimal value for the instance.

Table 4.3: Mathematical models test using Gurobi (OPTIMIZATION, 2021).

Instance	NF		FB		RB	
	obj	t(s)	obj	t(s)	obj	t(s)
palmer6	**693180**	0	**693180**	0	**693180**	0
palmer12	3457952	7200	**3428509**	0	3457952	7200
palmer24	1542292	7200	**1086656**	0	1091506	7200
berry6	**534**	0	**534**	0	**534**	0
berry35	50636	7200	**16915**	0	20177	7200
berry35u	19084	7200	**16192**	7200	17721	7200
raidl10	**53674**	7200	**53674**	0	**53674**	795
raidl20	**157570**	7200	**157570**	4	**157570**	7200
raidl50	43517513	7200	**813628**	7200	1243664	7200

From Table 4.3 we realized that our new formulation was the worst of the three tested formulations reaching the BKV only 44.4% of the instances and only 50% of those ended before the time limit. Therefore, the new proposed formulation presented in Section 2.2 seems weaker than the previous formulations. In the future, new constraints such as families of valid inequalities may be added in order to improve our mathematical model.

4.4 Parameters

For the small and medium instances from *OR-Library* (BEASLEY, 1990) and for all instances from *Internet Topology Zoo* (KNIGHT et al., 2011), we executed our algorithms fixing the parameters as follows:

- `POPULATION_SIZE` = 75,
- `MAX_ITERATIONS` = 100,

- `MAX_NO_UPDATE` = 25,
- `CROSSOVER_NUMBER` = 150,
- `MUTATION_NUMBER` = 75.

We modified some of the above parameters for the larger instances (`STEID1-STEID7` and those from *Stanford Large Dataset Collection* (LESKOVEC; KREVL, 2014)), because for these instances the large number of edges turns much slower the crossover and mutation operators. Therefore, to increase the computational efficiency, we limited the second mutation operator (Algorithm 3) to a maximum of 100 random edges, because Algorithm 3 presented the higher time complexity $\mathcal{O}(|E| \times |V|^2)$, and only for the instances from *Stanford Large Dataset Collection* (LESKOVEC; KREVL, 2014) the probability of executing this operator was also reduced to 10% of its previous value. We also adjusted some other parameters, whose values are shown in Table 4.4.

Table 4.4: Algorithm parameters for instances for larger instances.

	STEID*	email	CollegeMsg	bitcoinalpha	bitcoinotc
POPULATION_SIZE	50	50	50	50	25
MAX_ITERATIONS	100	400	125	100	100
MAX_NO_UPDATE	25	25	25	25	25
CROSSOVERS	100	100	100	50	25
MUTATIONS	50	100	50	25	5

Finally, Algorithm 9 requires additional parameters and we executed our algorithms fixing these values for all instances as follows:

- `POP_SIZE` = 100,
- `ITER_MAX` = 200000,
- `ITER_TERM` = 10000,
- `MAX_NOT_IMPROVING` = 10.

4.5 Results and analysis of algorithms

To evaluate the quality of our proposals, first we compare the four different evolutionary strategies with the algorithms from (ROTHLAUF, 2009), (STEITZ; ROTHLAUF, 2011) as well as a combination between the minimum path strategy using a similar simulated annealing from (ROTHLAUF, 2009). Table 4.5 shows the average gap for each approach, calculated by the expression $\frac{|C(T)-BKV|}{C(T)} \times 100$, where $C(T)$ is the value obtained by the approach and BKV the best known value for the instance. The columns `RD`, GP, MP and KC represent, respectively, the results of our evolutionary algorithm with the random strategy, the greedy probabilistic strategy, the minimum path strategy, and the k-core strategy. The metaheuristic from (ROTHLAUF, 2009) is represented by GA+SA, the algorithm from (STEITZ; ROTHLAUF, 2011) is represented by GLS and the combined approach is represented by MSA.

Table 4.5: Average gap values for each instance.

Instance	gap(%)						
	RD	GP	MP	KC	GA+SA	GLS	MSA
STEIB1	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	1.1	**0.0**
STEIB2	0.2	**0.0**	**0.0**	**0.0**	**0.0**	1.3	**0.0**
STEIB3	0.1	**0.0**	**0.0**	**0.0**	0.5	2.8	**0.0**
STEIB4	4.6	**0.0**	**0.0**	**0.0**	**0.0**	2.0	**0.0**
STEIB5	2.5	**0.0**	**0.0**	**0.0**	1.6	1.8	**0.0**
STEIB6	1.1	0.4	0.1	**0.0**	1.7	4.1	**0.0**
STEIB7	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	1.8	**0.0**
STEIC1	18.1	0.7	0.2	0.2	**0.1**	6.5	**0.1**
STEIC2	20.8	0.1	0.3	0.3	**0.0**	6.0	**0.0**
STEIC3	18.0	1.4	**0.0**	**0.0**	1.8	7.3	**0.0**
STEIC4	19.0	0.2	**0.0**	**0.0**	0.2	8.4	**0.0**
STEIC5	18.4	3.3	**0.0**	**0.0**	0.3	8.7	0.3
STEIC6	47.0	7.6	**0.1**	**0.1**	3.9	7.5	0.2
STEIC7	48.1	5.6	**0.0**	**0.0**	3.9	11.5	**0.0**
STEID1	30.0	2.9	**0.0**	**0.0**	0.8	12.1	**0.0**
STEID2	29.6	3.4	**0.0**	**0.0**	3.2	9.3	**0.0**
STEID3	29.3	3.6	0.5	0.6	0.5	10.6	**0.0**
STEID4	29.9	4.4	**0.0**	**0.0**	1.1	14.7	**0.0**
STEID5	28.8	2.6	**0.0**	**0.0**	0.2	15.5	**0.0**
STEID6	60.2	23.0	0.1	0.1	15.9	19.2	**0.0**
STEID7	57.1	21.5	1.2	1.0	9.8	14.3	**0.2**
Palmetto	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	1.6	**0.0**
Tw	0.2	**0.0**	0.8	0.4	0.3	2.1	0.1
Deltacom	0.5	**0.0**	**0.0**	**0.0**	0.5	1.7	**0.0**
TataNld	2.8	**0.0**	**0.0**	**0.0**	0.1	9.9	**0.0**
GtsCe	0.5	**0.0**	**0.0**	**0.0**	0.1	1.4	**0.0**
Colt	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	7.5	**0.0**
UsCarrier	0.1	**0.0**	**0.0**	**0.0**	**0.0**	3.0	**0.0**
Cogentco	1.6	**0.0**	**0.0**	**0.0**	**0.0**	2.6	**0.0**
Kdl	22.0	0.1	**0.0**	**0.0**	**0.0**	13.9	**0.0**
email	65.2	19.9	11.6	11.7	29.4	18.5	**1.7**
CollegeMsg	67.5	47.3	9.9	9.8	23.3	32.8	**2.8**
bitcoinalpha	61.2	51.7	4.2	4.7	29.2	53.2	**3.7**
bitcoinotc	65.5	57.2	4.3	**3.8**	59.4	69.1	4.6
Average	22.1	7.6	1.0	1.0	5.5	11.3	**0.4**

From Table 4.5 we observe that the combined strategy generally obtained solutions with the lowest average gap, then the minimum path and minimum k-core strategies obtained the same average gap in second, followed by the algorithm proposed by (ROTHLAUF, 2009), then our greedy probabilistic approach. Next is the algorithms proposed by (STEITZ; ROTHLAUF, 2011). Moreover, in 88.2% of the tests the best gaps were presented by the combined strategy, in 73.5% by our minimum k-core strategy, in 67.6% by our minimum path strategy, in 41.2% by our greedy probabilistic strategy, in 32.4% by the GA+SA metaheuristic of (ROTHLAUF, 2009), in 0.0% by the GLS metaheuristic of (ROTHLAUF, 2009), and in 11.8% by our random approach. The superiority of the results obtained by the combined strategy as well as our evolutionary algorithm with the minimum k-core and minimum path strategies is also given by the average gap obtained over all instances, which was less or equal to 1.0% for those three strategies, 5.5 times smaller than the 5.5% presented by the GA+SA approach of (ROTHLAUF, 2009), and 11.3 times smaller than the 11.3% presented by the GLS approach of (STEITZ; ROTHLAUF, 2011).

A deeper comparison can be done by analysing the algorithms stability. Table 4.6 shows the stability for each instance. The σ values represents the coefficient of variation obtained by the strategy with the different seeds.

Table 4.6 shows that the coefficient of variation was very low (less than 0.4%) for the combined, the minimum path and the minimum k-core strategies, implying that our evolutionary algorithm with these strategies as well as the combined strategy were very stable and its solutions did not depended on the seeds for the random generation. The less stable strategy was the random strategy, where the average standard deviation of the mean was 4.3%, while for the greedy probabilistic strategy this value was 1.3% less than the 3.4% and 5.1% of the metaheuristics from (STEITZ; ROTHLAUF, 2011) and (ROTHLAUF, 2009). Therefore, our evolutionary algorithm with the combined, the minimum path, the minimum k-core strategies obtained higher quality solutions, and was more stable than the other proposals.

The average computational times required by each strategy are given by Table 4.7, where the three best strategies are slightly faster than the other strategies. We observe that the worst computational times were presented by the evolutionary algorithm with the random and greedy probabilistic strategies, while the rest of the approaches required very similar times for their executions.

The above discussion allows us to conclude that the combined, the minimum path strategy as well as the minimum k-core strategy outperformed our other strategies and

Table 4.6: Average standard deviation for each algorithm.

Instance	$\sigma(\%)$						
	RD	GP	MP	KC	GA+SA	GLS	MSA
STEIB1	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	1.3	**0.0**
STEIB2	0.6	**0.0**	**0.0**	**0.0**	**0.0**	1.4	**0.0**
STEIB3	0.2	**0.0**	**0.0**	**0.0**	0.4	2.7	**0.0**
STEIB4	2.4	**0.0**	**0.0**	**0.0**	**0.0**	2.5	**0.0**
STEIB5	1.9	**0.0**	**0.0**	**0.0**	3.3	1.6	**0.0**
STEIB6	1.5	0.7	**0.1**	**0.1**	2.3	4.2	**0.1**
STEIB7	0.1	**0.0**	**0.0**	**0.0**	**0.0**	1.2	**0.0**
STEIC1	6.4	0.8	**0.0**	0.1	0.1	4.5	0.1
STEIC2	5.3	**0.0**	0.1	0.2	0.1	3.3	**0.0**
STEIC3	3.1	2.4	**0.0**	**0.0**	2.8	4.5	**0.0**
STEIC4	5.6	0.1	**0.0**	**0.0**	0.5	4.8	**0.0**
STEIC5	4.6	1.7	**0.0**	**0.0**	1.0	4.8	1.0
STEIC6	6.7	2.7	**0.2**	0.3	4.0	4.8	0.3
STEIC7	9.1	1.2	**0.0**	**0.0**	17.4	6.8	**0.0**
STEID1	7.4	1.0	**0.0**	**0.0**	1.4	4.7	**0.0**
STEID2	9.3	0.4	**0.0**	**0.0**	13.2	4.4	**0.0**
STEID3	7.8	1.9	0.2	0.2	0.8	4.5	**0.0**
STEID4	7.2	3.2	**0.0**	**0.0**	1.9	21.1	**0.0**
STEID5	6.2	0.4	**0.0**	**0.0**	0.4	8.9	**0.0**
STEID6	10.2	5.4	**0.0**	0.1	21.7	4.2	**0.0**
STEID7	13.4	6.3	0.7	**0.1**	18.9	2.8	0.5
Palmetto	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	1.3	**0.0**
Tw	0.4	**0.0**	0.7	0.6	0.5	1.6	0.4
Deltacom	0.3	**0.0**	**0.0**	**0.0**	0.6	1.3	**0.0**
TataNld	1.3	**0.0**	**0.0**	**0.0**	0.2	13.9	**0.0**
GtsCe	0.4	**0.0**	**0.0**	**0.0**	0.1	2.4	**0.0**
Colt	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	6.8	**0.0**
UsCarrier	0.1	**0.0**	**0.0**	**0.0**	**0.0**	6.1	**0.0**
Cogentco	1.4	**0.0**	**0.0**	**0.0**	**0.0**	2.1	**0.0**
Kdl	6.2	0.1	**0.0**	**0.0**	**0.0**	11.3	**0.0**
email	5.7	5.0	2.5	3.1	3.3	8.4	**1.7**
CollegeMsg	7.9	4.1	**1.6**	2.2	4.3	6.5	2.0
bitcoinalpha	5.4	3.1	**0.5**	1.8	4.0	4.7	3.6
bitcoinotc	7.9	2.6	**1.6**	2.6	12.7	7.2	3.6
Average	4.3	1.3	**0.2**	0.3	3.4	5.1	0.4

Table 4.7: Average execution time in seconds for each instance.

	RD	GP	MP	KC	GA+SA	GLS	MSA
Average	370.4	351.7	**210.1**	222.2	286.2	319.0	243.8

were capable of obtaining much better solutions than the metaheuristics of (ROTHLAUF, 2009) and (STEITZ; ROTHLAUF, 2011). Thus, to propose evolutionary algorithms that explore theoretical results seems to be an adequate approach for OCT.

5 CONCLUSIONS

In this work we approached OCT, an $\mathcal{NP}$-hard problem with applications in many different areas such as logistics, telecommunications and bioinformatics. To find good solutions for the problem, we designed an evolutionary algorithm with four different strategies: the random, the greedy probabilistic, the minimum path, and the minimum k-core strategies as well as a combined strategy. We also proposed a new mixed integer linear programming formulation for OCT. To assess the applicability of our proposals we tested them over instances from the literature and from real-world datasets. All tested instances can be found in public-access datasets and we also give public access to our implementations.

Our best proposals, the combined, minimum path and the minimum k-core strategies, were able to find optimal or near optimal solutions for all the experiments, presenting a very good stability for the different seeds and requiring a reasonable amount of time for the execution. Furthermore, we compared our strategies with previous metaheuristics found in literature, obtaining much higher quality solutions.

In future work we will compare the impact of different operators (e.g., mutation and crossover) and algorithms (e.g., simulated annealing). We also intend to add new valid restrictions to our common ancestor based mathematical formulation, with the objective of reducing the search space. Another research direction will consider to use an automatic solver (e.g., irace (LÓPEZ-IBÁÑEZ et al., 2016)) to find parameters for our metaheuristic proposals.

REFERENCES

AHUJA, R. K.; MURTY, V. V. S. Exact and heuristic algorithms for the optimum communication spanning tree problem. **Transportation Science**, INFORMS, v. 21, n. 3, p. 163–170, 1987. ISSN 00411655, 15265447. Available from Internet: <https://doi.org/10.1287/trsc.21.3.163>.

BEASLEY, J. E. OR-library: Distributing test problems by electronic mail. **The Journal of the Operational Research Society**, Palgrave Macmillan Journals, v. 41, n. 11, p. 1069–1072, 1990. ISSN 01605682, 14769360. Available from Internet: <https://doi.org/10.2307/2582903>.

BELLMAN, N.; ROTH, R. S. **Methods in approximation: techniques for mathematical modelling**. Springer Science & Business Media, 2012. ISBN 9789400946002. Available from Internet: <https://doi.org/10.1007/978-94-009-4600-2>.

CONFORTI, M.; CORNUEJOLS, G.; ZAMBELLI, G. **Integer Programming**. Springer, 2014. ISBN 3319110071. Available from Internet: <https://doi.org/10.1007/978-3-319-11008-0>.

CONTRERAS, I. Network hub location: Models, algorithms, and related problems, Ph. D. thesis, Universitat politècnica de catalunya. 2009. Available from Internet: <https://dialnet.unirioja.es/servlet/tesis?codigo=21447>.

CONTRERAS, I.; FERNANDEZ, E.; MARÍN, A. The tree of hubs location problem. **European Journal of Operational Research**, v. 202, p. 390–400, 2010. Available from Internet: <https://doi.org/10.1016/j.ejor.2009.05.044>.

CPLEX, I. I. V12. 1: User's manual for CPLEX. **International Business Machines Corporation**, v. 46, n. 53, p. 157, 2009.

DARWIN, C. **The Origin of Species: By Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life**. 6. ed. Cambridge University Press, 2009. (Cambridge Library Collection - Darwin, Evolution and Genetics). Available from Internet: <https://doi.org/10.1017/CBO9780511694295>.

DIJKSTRA, E. W. A note on two problems in connexion with graphs. **Numerische Mathematik**, Springer, v. 1, n. 1, p. 269–271, 1959.

DINITZ, Y. Algorithm for solution of a problem of maximum flow in networks with power estimation. **Soviet Math. Dokl.**, v. 11, p. 1277–1280, 1970.

EDMONDS, J.; KARP, R. M. Theoretical improvements in algorithmic efficiency for network flow problems. **J. ACM**, Association for Computing Machinery, New York, NY, USA, v. 19, n. 2, p. 248–264, abr. 1972. ISSN 0004-5411. Available from Internet: <https://doi.org/10.1145/321694.321699>.

FAKCHAROENPHOL, J.; RAO, S.; TALWAR, K. A tight bound on approximating arbitrary metrics by tree metrics. In: **Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing**. New York, NY, USA: Association for Computing Machinery, 2003. (STOC '03), p. 448–455. ISBN 1581136749. Available from Internet: <https://doi.org/10.1145/780542.780608>.

FENG, D.-F.; DOOLITTLE, R. F. Progressive sequence alignment as a prerequisitetto correct phylogenetic trees. **Journal of Molecular Evolution**, v. 25, n. 4, p. 351–360, Aug 1987. ISSN 1432-1432. Available from Internet: <https://doi.org/10.1007/BF02603120>.

FISCHETTI, M.; LANCIA, G.; SERAFINI, P. Exact algorithms for minimum routing cost trees. **Networks**, v. 39, n. 3, p. 161–173, 2002. Available from Internet: <https://doi.org/10.1002/net.10022>.

FOMIN, F.; KRATSCH, D. **Exact Exponential Algorithms**. Springer Berlin Heidelberg, 2010. (Texts in Theoretical Computer Science. An EATCS Series). ISBN 9783642165337. Available from Internet: <https://doi.org/10.1007/978-3-642-16533-7>.

FONSECA, G. A. **Implementations of: Different strategies of an evolutionary algorithm for the Optimum Communication Spanning Tree**. 2021. <https://github.com/giovaneaf/OCT>. Accessed: 2021-02-20.

FORD, L. R.; FULKERSON, D. R. A simple algorithm for finding maximal network flows and an application to the hitchcock problem. **Canadian Journal of Mathematics**, Cambridge University Press, v. 9, p. 210–218, 1957. Available from Internet: <https://doi.org/10.4153/CJM-1957-024-0>.

GAREY, M. R.; GRAHAM, R. L.; JOHNSON, D. S. The complexity of computing Steiner minimal trees. **SIAM Journal on Applied Mathematics**, v. 32, n. 4, p. 835–859, 1977. Available from Internet: <https://doi.org/10.1137/0132072>.

GOMORY, R. E.; HU, T. C. Multi-terminal network flows. **Journal of the Society for Industrial and Applied Mathematics**, v. 9, n. 4, p. 551–570, 1961. Available from Internet: <https://doi.org/10.1137/0109047>.

GOOGLE. **Google Random Number Generator**. 2021. <https://www.google.com/search?q=random+number>. Accessed: 2021-01-19.

HU, T. C. Optimum communication spanning trees. **SIAM Journal on Computing**, v. 3, n. 3, p. 188–195, 1974. Available from Internet: <https://doi.org/10.1137/0203015>.

JOHNSON, D. S.; LENSTRA, J. K.; KAN, A. H. G. R. The complexity of the network design problem. **Networks**, v. 8, n. 4, p. 279–285, 1978. Available from Internet: <https://doi.org/10.1002/net.3230080402>.

KNIGHT, S. et al. The internet topology zoo. **IEEE Journal on Selected Areas in Communications**, v. 29, n. 9, p. 1765–1775, 2011. Available from Internet: <https://doi.org/10.1109/JSAC.2011.111002>.

KRUSKAL, J. B. On the shortest spanning subtree of a graph and the traveling salesman problem. **Proceedings of the American Mathematical society**, JSTOR, v. 7, n. 1, p. 48–50, 1956. Available from Internet: <https://doi.org/10.2307/2033241>.

LESKOVEC, J.; KREVL, A. **SNAP Datasets: Stanford Large Network Dataset Collection**. 2014. <http://snap.stanford.edu/data>. Accessed: 2021-01-19.

LÓPEZ-IBÁÑEZ, M. et al. The irace package: Iterated racing for automatic algorithm configuration. **Oper. Res. Perspect.**, Elsevier BV, v. 3, p. 43–58, 2016. ISSN 2214-7160. Available from Internet: <http://dx.doi.org/10.1016/j.orp.2016.09.002>.

MANIEZZO, V.; STÜTZLE, T.; VOSS, S. **Matheuristics: Hybridizing Metaheuristics and Mathematical Programming**. Springer US, 2009. (Annals of Information Systems). ISBN 9781441913050. Available from Internet: <https://doi.org/10.1007/978-1-4419-1306-7>.

MASONE, A. et al. The minimum routing cost tree problem. **Soft Computing**, v. 23, n. 9, p. 2947–2957, May 2019. ISSN 1433-7479. Available from Internet: <https://doi.org/10.1007/s00500-018-3557-3>.

MOTA, C. The optimum communication spanning tree problem : Properties, models and algorithms, Ph. D. thesis, Universitat politècnica de catalunya. 2015. Available from Internet: <https://dialnet.unirioja.es/servlet/tesis?codigo=85034>.

OPTIMIZATION, L. G. **Gurobi Optimizer Reference Manual**. 2021. Available from Internet: <http://www.gurobi.com>.

PRÜFER, H. Neuer Beweis eines Satzes über Permutationen. **Archiv der Mathematischen Physik**, v. 27, p. 742–744, 1918.

RAVELO, S. V.; FERREIRA, C. E. PTAS's for some metric p-source communication spanning tree problems. In: RAHMAN, M. S.; TOMITA, E. (Ed.). **WALCOM: Algorithms and Computation**. Cham: Springer International Publishing, 2015. p. 137–148. ISBN 978-3-319-15612-5. Available from Internet: <https://doi.org/10.1007/978-3-319-15612-5_13>.

RAVELO, S. V.; FERREIRA, C. E. A PTAS for the metric case of the minimum sum-requirement communication spanning tree problem. **Discrete Applied Mathematics**, v. 228, p. 158 – 175, 2017. ISSN 0166-218X. CALDAM 2015. Available from Internet: <https://doi.org/10.1016/j.dam.2016.09.031>.

RAVELO, S. V.; FERREIRA, C. E. A PTAS for the metric case of the optimum weighted source–destination communication spanning tree problem. **Theoretical Computer Science**, v. 771, p. 9 – 22, 2019. ISSN 0304-3975. Available from Internet: <https://doi.org/10.1016/j.tcs.2018.11.008>.

ROTHLAUF, F. **Representations for Genetic and Evolutionary Algorithms**. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN 354025059X. Available from Internet: <https://doi.org/10.1007/3-540-32444-5>.

ROTHLAUF, F. Design and applications of metaheuristics. Habilitationsschrift, Universität Mannheim. 2007.

ROTHLAUF, F. An encoding in metaheuristics for the minimum communication spanning tree problem. **INFORMS Journal on Computing**, v. 21, p. 575–584, 11 2009. Available from Internet: <https://doi.org/10.1287/ijoc.1080.0310>.

SAITO, M.; MATSUMOTO, M. SIMD-oriented fast Mersenne Twister: a 128-bit pseudorandom number generator. **In Monte Carlo and Quasi-Monte**

Carlo Methods, v. 64, n. 2, p. 607–622, 1993. Available from Internet: <https://doi.org/10.1007/978-3-540-74496-2_36>.

SATTARI, S.; DIDEHVAR, F. Variable neighborhood search approach for the minimum routing cost spanning tree problem. **Int J Oper Res**, v. 10, n. 4, p. 153–160, 2013.

SATTARI, S. et al. A metaheuristic algorithm for the minimum routing cost spanning tree problem. **Iranian Journal of Operations Research**, Iranian Journal of Operations Research, v. 6, n. 1, p. 65–78, 2015.

SCHRIJVER, A. **Combinatorial optimization: polyhedra and efficiency**. Springer Science & Business Media, 2003. ISBN 9783540443896. Available from Internet: <https://www.springer.com/gp/book/9783540443896>.

Sommer, J. On optimal communication spanning trees in embedded ethernet networks. In: **2010 IEEE International Workshop on Factory Communication Systems Proceedings**. [s.n.], 2010. p. 141–150. Available from Internet: <https://doi.org/10.1109/WFCS.2010.5548629>.

STEITZ, W.; ROTHLAUF, F. Guided local search for the optimal communication spanning tree problem. In: **Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation**. New York, NY, USA: Association for Computing Machinery, 2011. (GECCO '11), p. 51–52. ISBN 9781450306904. Available from Internet: <https://doi.org/10.1145/2001858.2001889>.

Tan, Q. P.; Due, N. N. An experimental study of minimum routing cost spanning tree algorithms. In: **2013 International Conference on Soft Computing and Pattern Recognition (SoCPaR)**. [s.n.], 2013. p. 158–165. Available from Internet: <https://doi.org/10.1109/SOCPAR.2013.7054119>.

WU, B. Y. Approximation algorithms for the optimal p-source communication spanning tree. **Discrete Appl. Math.**, Elsevier Science Publishers B. V., NLD, v. 143, n. 1–3, p. 31–42, sep. 2004. ISSN 0166-218X. Available from Internet: <https://doi.org/10.1016/j.dam.2003.10.002>.

WU, B. Y.; CHAO, K.-M.; TANG, C. Y. Approximation algorithms for some optimum communication spanning tree problems. **Discrete Applied Mathematics**, v. 102, n. 3, p. 245 – 266, 2000. ISSN 0166-218X. Available from Internet: <https://doi.org/10.1016/S0166-218X(99)00212-7>.

WU, B. Y.; CHAO, K.-M.; TANG, C. Y. A polynomial time approximation scheme for optimal product-requirement communication spanning trees. **Journal of Algorithms**, v. 36, n. 2, p. 182 – 204, 2000. ISSN 0196-6774. Available from Internet: <https://doi.org/10.1006/jagm.2000.1088>.

WU, B. Y. et al. A polynomial time approximation scheme for minimum routing cost spanning trees. In: **Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms**. USA: Society for Industrial and Applied Mathematics, 1998. (SODA '98), p. 21–32. ISBN 0898714109. Available from Internet: <https://dl.acm.org/doi/10.5555/314613.314628>.

WU, B. Y. et al. A polynomial-time approximation scheme for minimum routing cost spanning trees. **SIAM Journal on Computing**, v. 29, n. 3, p. 761–778, 2000. Available from Internet: <https://doi.org/10.1137/S009753979732253X>.

YU, X.; GEN, M. **Introduction to Evolutionary Algorithms**. Springer, 2012. ISBN 144712569X. Available from Internet: <https://dl.acm.org/doi/book/10.5555/2432058>.