UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

The Family Traveling Salesman Problem

" Documento Definitivo"

Doutoramento em Estatística e Investigação Operacional

Especialidade de Otimização

Raquel Monteiro de Nobre Costa Bernardino

Tese orientada por:

Professora Doutora Ana Maria Duarte Silve Alves Paias

Documento especialmente elaborado para a obtenção do grau de doutor

2019

UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

The Family Traveling Salesman Problem

Doutoramento em Estatística e Investigação Operacional

Especialidade de Otimização

Raquel Monteiro de Nobre Costa Bernardino

Tese orientada por:

Professora Doutora Ana Maria Duarte Silve Alves Paias

Documento especialmente elaborado para a obtenção do grau de doutor

Dissertação financiada pela FCT ao abrigo do projeto PTDC/MAT-NAN/2196/2014

2019

Para a minha mãe,
Julieta.

Agradecimentos

O meu sincero obrigada à Professora Doutora Ana Maria Duarte Silva Alves Paias, a minha orientadora. A sua dedicação a esta dissertação fez com que esta chegasse a bom porto. Obrigada pela paciência, disponibilidade e por estar sempre disposta a ensinar-me. Foi mais uma vez uma honra tê-la como minha orientadora.

Um grande obrigada à minha família. Aos meus pais, que sempre me tentaram educar da melhor forma possível e fizeram de mim aquilo que sou hoje. Um obrigada especial ao meu pai, que nestes últimos anos sempre me incentivou a aprender mais e me proporcionou esta aventura que foi o doutoramento. Aos meus avós, Babá e Avô Júlio, obrigada por ainda hoje tomarem conta de mim.

Daniel, obrigada! Obrigada pelo teu companheirismo durante esta etapa e por, mais uma vez, acreditares que isto seria possível antes de mim. Conseguimos, que venha o próximo desafio.

Gostaria também de agradecer a todos os professores que me acompanharam neste percurso. Um agradecimento especial ao professor Luís Gouveia, pois também contribuiu de forma indireta para esta dissertação.

Um obrigada especial aos meus colegas de gabinete e de almoço, Michele e Jessica pelas discussões interessantes que tivemos e pelos bons tempos que passámos juntos.

Por fim, a todos os que de uma forma ou de outra me incentivaram durante o decorrer desta dissertação nem que seja por perguntarem como está a correr, o meu mais sincero obrigada.

Abstract

Consider a depot, a partition of the set of nodes into subsets, called families, and a cost matrix. The objective of the family traveling salesman problem (FTSP) is to find the minimum cost circuit that starts and ends at the depot and visits a given number of nodes per family. The FTSP was motivated by the order picking problem in warehouses where products of the same type are stored in different places and it is a recent problem. Nevertheless, the FTSP is an extension of well-known problems, such as the traveling salesman problem.

Since the benchmark instances available are in small number we developed a generator, which given a cost matrix creates an FTSP instance with the same cost matrix. We generated several test instances that are available in a site dedicated to the FTSP.

We propose several mixed integer linear programming models for the FTSP. Additionally, we establish a theoretical and a practical comparison between them. Some of the proposed models have exponentially many constraints, therefore we developed a branch-and-cut (B&C) algorithm to solve them. With the B&C algorithm we were able to obtain the optimal value of open benchmark instances and of the majority of the generated instances.

As the FTSP is an NP-hard problem we develop three distinct heuristic methods: a genetic algorithm, an iterated local search algorithm and a hybrid algorithm. With all of them we were able to improve the best upper bounds available in the literature for the benchmark instances that still have an unknown optimal value.

We created a new variant of the FTSP, called the restricted family traveling salesman problem (RFTSP), in which nodes from the same family must be visited consecutively. We apply to the RFTSP the methods proposed for the FTSP and develop a new formulation based on the interfamily and the intrafamily relationships.

Keywords: Family traveling salesman problem; Multicommodity flow; Projections; Branch-and-cut algorithm; Metaheuristics.

Resumo

Considere-se um depósito, uma partição do conjunto de cidades em vários subconjuntos, aos quais chamamos famílias, e custos de deslocação entre o depósito e as cidades e entre as várias cidades. O objetivo do *family traveling salesman problem* (FTSP) é determinar o circuito elementar de custo mínimo que começa e acaba no depósito e visita um número predeterminado de cidades em cada família. O FTSP foi motivado pelo problema de recolha de produtos em armazéns onde os produtos do mesmo tipo estão armazenados em locais diferentes. O FTSP é um problema relativamente recente pois, além do artigo desenvolvido no âmbito desta dissertação, existe apenas um artigo na literatura sobre o mesmo. Contudo, o FTSP pode ser visto como uma generalização de problemas bem conhecidos da literatura, como o problema do caixeiro viajante e o problema do caixeiro viajante generalizado (*generalized traveling salesman problem*), daí a importância do seu estudo no âmbito de uma dissertação de doutoramento.

Como as instâncias do FTSP disponíveis na literatura são em número reduzido decidimos criar um gerador de instâncias para o FTSP. Este gerador recebe uma matriz de custos e cria quatro instâncias diferentes do FTSP, que diferem no número de visitas por família, com a mesma matriz de custos. As quatro instâncias foram geradas para terem características diferentes, nomeadamente criámos um tipo de instância para ter um número reduzido de visitas por família e outro tipo para ter um número elevado de visitas por família. Para gerarmos novas instâncias do FTSP usámos matrizes de custos, simétricas e assimétricas, de instâncias de referência do problema do caixeiro viajante e de instâncias de referência assimétricas do problema do *traveling purchaser*. Criou-se um site dedicado ao FTSP no qual se disponibilizam para toda a comunidade científica todas as instâncias existentes do FTSP, nomeadamente as instâncias de referência da literatura e as instâncias geradas.

Nesta dissertação apresentamos vários modelos de programação linear inteira mista para o FTSP. Estes modelos são comparados empiricamente e através de resultados teóricos. Alguns dos modelos propostos contêm conjuntos de restrições que são em número exponencial, pelo que temos de recorrer a um algoritmo de *branch-and-cut*, que combina um algoritmo de *branch-and-bound* com um algoritmo de planos de corte, para os resolver. Pela experiência computacional verificámos que os modelos com um número exponencial de restrições são os mais eficientes. O melhor modelo proposto, a que chamámos modelo CC+RFV, contém dois conjuntos de restrições em número exponencial, nomeadamente as desigualdades CC, que são uma adaptação das restrições de eliminação de subcircuitos propostas por Dantzig et al. (1954) para o problema do caixeiro viajante e as desigualdades RFV, criadas no âmbito desta dissertação para o FTSP. Estas últimas garantem que se num subconjunto de cidades que contém o depósito, designado por S', não existem cidades suficientes para satisfazer as visitas de uma determinada família, então teremos que visitar uma cidade no conjunto complementar de S'. Ambas as desigualdades eliminam subcircuitos contudo, como a separação das desigualdades RFV é muito demorada, usamos as desigualdades CC para eliminar subcircuitos e as desigualdades RFV são adicionadas como desigualdades válidas para melhorar o valor da relaxação linear. Foi ainda incorporado no algoritmo de *branch-and-cut* um método heurístico muito simples que permite obter uma solução admissível para o FTSP (não necessariamente a ótima). Deste modo garantimos que, com o algoritmo de *branch-and-cut*, conseguimos sempre obter uma solução admissível para o problema. Com o algoritmo de *branch-and-cut* aplicado ao modelo CC+RFV conseguimos obter o valor ótimo de instâncias de referência que tinham valor ótimo desconhecido. No que diz respeito às instâncias geradas, conseguimos obter o valor ótimo de 148 instâncias das 164 instâncias geradas, sendo que a proporção de instâncias simétricas resolvidas é 73% e a proporção de instâncias assimétricas resolvidas é de 99%.

Como o problema do caixeiro viajante pode ser visto como um caso particular do FTSP, em que temos que visitar todas as cidades de todas as famílias, podemos concluir que o FTSP pertence à classe de problemas NP-difícil, pelo que desenvolvemos métodos heurísticos para o FTSP. Nesta dissertação são propostos três métodos heurísticos, nomeadamente: um algoritmo genético, que usa permutações como cromossomas; um algoritmo de *iterated local search* (ILS), que itera entre um algoritmo de pesquisa local e um algoritmo de perturbação; e um algoritmo híbrido, que combina o modelo CC+RFV proposto com o algoritmo de ILS. Os resultados computacionais mostraram que o algoritmo genético é o algoritmo mais eficiente e que o algoritmo ILS e o algoritmo híbrido são os mais eficazes, pois nas instâncias testadas obtêm sempre uma solução de custo inferior ao da solução obtida com o algoritmo genético. Como o objetivo principal é obter as soluções com o menor custo possível decidimos usar o algoritmo ILS e o algoritmo híbrido, pois nenhum obteve o melhor resultado em todas as instâncias. Comparámos estes dois algoritmos e concluímos que, geralmente, o algoritmo ILS é mais eficiente enquanto que o algoritmo híbrido é mais eficaz. Os métodos heurísticos apenas foram aplicados a instâncias que o método exato não resolveu, isto é, instâncias que têm valor ótimo desconhecido. No que diz respeito às instâncias de referência com valor ótimo desconhecido, conseguimos obter soluções de custo inferior aos melhores limites superiores disponíveis na literatura para todas as instâncias testadas e usando ambos os algoritmos.

Relativamente às instâncias geradas, como nunca foram usadas na literatura decidimos comparar a solução obtida com os métodos heurísticos com o melhor limite superior obtido com o algoritmo de *branch-and-cut*. Das 16 instâncias geradas que têm valor ótimo desconhecido os métodos heurísticos só conseguiram obter uma solução de custo inferior à solução obtida pelo algoritmo de *branch-and-cut* em três instâncias. Contudo, a comparação entre o algoritmo de *branch-and-cut* e os métodos heurísticos não é justa pois o algoritmo de *branch-and-cut* obteve as suas soluções ao fim de três horas, enquanto que, por exemplo o algoritmo de ILS, obteve as suas soluções num tempo médio de 20 segundos.

Nesta dissertação também apresentamos uma variante do FTSP que, tanto quanto sabemos, nunca foi apresentada na literatura. A variante do FTSP chama-se *restricted family traveling salesman problem* (RFTSP) e é obtida exigindo que as cidades da mesma família sejam visitadas consecutivamente. Para resolver o RFTSP decidimos adaptar os métodos que obtiveram os melhores resultados no FTSP, nomeadamente o algoritmo de *branch-and-cut* aplicado ao modelo CC+RFV e, relativamente a métodos heurísticos, o algoritmo de ILS e o algoritmo híbrido. Com estes métodos obtivemos os valores ótimos e limites superiores, para as instâncias com valor ótimo desconhecido. Comparando os resultados obtidos para o FTSP e os obtidos para o RFTSP verificamos que, nas instâncias assimétricas, o número de instâncias com valor ótimo conhecido baixou significativamente. Das 100 instâncias geradas com custos assimétricos e considerando o RFTSP, apenas conseguimos obter o valor ótimo de 76 instâncias. Ainda relativamente ao RFTSP, propomos um novo modelo de programação linear inteira mista. Este novo modelo explora as relações dentro de cada família e as relações entre famílias como um problema de caminho mais curto e um problema do caixeiro viajante, respetivamente. Este novo modelo tem um número muito elevado de variáveis e restrições o que torna a sua resolução menos eficiente. Contudo, este novo modelo obtém valores de relaxação linear superiores aos do modelo CC+RFV adaptado, pelo pode ser usado como objeto de investigação futura como ponto de partida para novas abordagens de resolução como por exemplo a utilização de técnicas de decomposição.

Palavras-chave: Family traveling salesman problem; Fluxos multicomodidade; Projecções; Algoritmo de branch-and-cut; Meta-heurísticas.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Assume that products of the same type are scattered through different places in a warehouse and one must collect a given number of products of each type. This problem, the order picking problem in warehouses, motivated the family traveling salesman problem (FTSP), which will be addressed in this dissertation. More formally, consider a depot and a set of cities that is partitioned into several subsets, which are called families. The objective of the FTSP is to establish the minimum cost route that starts and ends at the depot and visits a given number of cities in each family.

The FTSP may be seen as a generalization of problems that have a wide variety of applications, which include the well-known traveling salesman problem and other variants. Additionally, the FTSP is a fairly recent problem and, in fact, as far as we know, there is only one article in the literature that addresses it. Therefore, for the reasons stated previously, the FTSP is a challenging problem to be studied within the scope of a Ph.D. dissertation.

The primary, and most general, objective of this dissertation is to study the FTSP in order to develop efficient methods that provide feasible solutions for this problem. The methods developed belong to one of two main categories: (i) exact methods, which guarantee that the solution obtained is the minimum cost feasible solution; and (ii) heuristic methods, which ensure that the solution obtained is feasible.

Regarding the exact methods, we will propose adaptations of methods from the literature developed for other problems, namely for the traveling salesman problem, and we will also develop new methods that take into account the specificities of the FTSP. The proposed methods will not only be compared theoretically but also empirically by using a small subset of test instances. The method that provides the best results is applied to all the test instances.

The heuristic methods are used to address the instances which the exact methods could not solve efficiently. Similarly to what was done for the exact methods, we use a small subset of test instances to do the parameter tuning of the several heuristic methods and to evaluate their behavior. The heuristic method that provides the solutions with the lowest cost will be used to solve the instances that the exact methods were unable to.

As a complement to this dissertation we decided to create a variant of the FTSP, which seems to be a natural variant and that arises by imposing the condition that cities from the same family must be visited consecutively. We denote this variant by the restricted family traveling salesman problem (RFTSP). We will adapt for the RFTSP the methods, both exact and heuristic, that provided the best results for the FTSP. Additionally, we developed an exact method that can only be applied to the RFTSP, which is compared to the adapted exact method through computational testing.

Finally, we created a website devoted to the FTSP which gathers the test instances used in this dissertation, both from the literature and the generated ones, and the best results obtained so far, which are either exact results or heuristic results. The site also contains the same type of information for the RFTSP.

This dissertation is organized as follows. In Chapter 2 we introduce definitions and theoretical results that will be used during this dissertation. Therefore, this chapter is purely expository and it is independent of the FTSP.

In Chapter 3 we provide a formal definition for the FTSP and present the notation used during the remainder of the dissertation. Additionally, we present the literature review as well as some problems that under specific circumstances can be solved as the FTSP. Some basic constructive heuristics are also described. We conclude this chapter by describing the FTSP instance generator developed and by presenting the test instances used.

Chapter 4 is devoted to the exact methods. We start by presenting several exact methods and then, by using a small subset of test instances, we establish a theoretical and an empirical comparison between the exact methods proposed.

In Chapter 5 we present the branch-and-cut algorithm, which is an algorithm used to solve some of the exact methods proposed, and the computational results obtained for the test instances with the best exact method.

In Chapter 6 we present the heuristic methods proposed for the FTSP, establish an empirical comparison between them, by using a small subset of test instances, and carry out the computational experiment, which consists in applying the best heuristic method to the instances that the exact methods could not solve efficiently.

Chapter 7 is devoted to the RFTSP, the variant of the FTSP that we proposed. We present exact methods for the RFTSP which are either adaptations of the exact methods for the FTSP or methods developed specifically for the RFTSP and we establish an empirical comparison between the exact methods proposed. We also develop heuristic methods by adapting the heuristic methods for the

FTSP. Finally, we present the computational experiment for the RFTSP.

We conclude this dissertation in Chapter 8 where we draw the main conclusions from this work and provide some bullet points of what the future work could entail.

Chapter 2

Mathematical Background

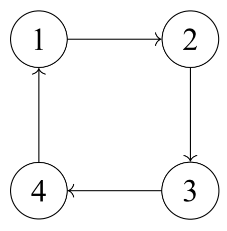
The purpose of this chapter is to present or clarify some definitions and theoretical results that are going to be used throughout this dissertation. Therefore, this chapter is purely expository as the results presented are a collection of results from the literature. This chapter is divided into five sections: graph theory, polyhedral theory, linear programming theory, complexity theory and, finally, some basic linear programming problems. More precisely, in Section 2.1 we present concepts related to graph theory, concepts related to polyhedral theory are presented in Section 2.2, in Section 2.3 we address linear programming theory, complexity theory is presented in Section 2.4 and, finally, in Section 2.5 we present some basic linear programming problems.

2.1 Graph theory

One seminal paper addressing graph theory dates back to 1739 and it was published by Euler. Since then graphs have been used to model a wide variety of problems, being routing one of the most common type. Information related to graph theory can be found in Christofides (1975) and Ahuja et al. (1993), for example.

A graph $G = (N, A)$ is defined as a pair of sets in which $N = \{1, 2, \ldots, n\}$ is a non-empty finite set called *node* (or *vertice*) set and $A = \{a_1, a_2, \ldots, a_m\}$ is either an *arc* set in which its elements are pairs of elements of N or an *edge* set, that is, a set of subsets with two elements of N. In the former case, there is an orientation assigned to the elements of A and in the latter case, there is no orientation assigned to the elements of A. If A is an arc set, then G is called a *directed* graph whereas if A is an edge set, then G is called a *nondirected* graph.

Example 1. Figure 2.1 shows examples of a directed graph (Figure 2.1a) and a nondirected graph (Figure 2.1b).


(a) Directed graph.

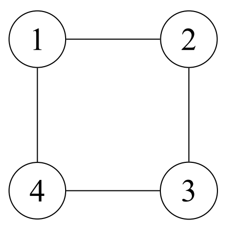

(b) Nondirected graph.

Figure 2.1: Examples of a directed and a nondirected graph.

We assume that G is a directed graph since any nondirected graph may be equivalently represented as a directed graph by assigning two arcs to each edge. We represent an arc a_i as a pair of nodes, that is, $a_i = (j, k)$ where j is the *initial* node and k is the *final* node of the arc. Additionally, if there is an arc between j and k we say that j and k are *adjacent* nodes. We also consider that G is a *simple* graph, that is, G contains a maximum of one arc between each pair of nodes and does not contain arcs which have the same initial and final nodes.

Example 2. To illustrate the definitions presented throughout this section consider the G_e graph presented in Figure 2.2.

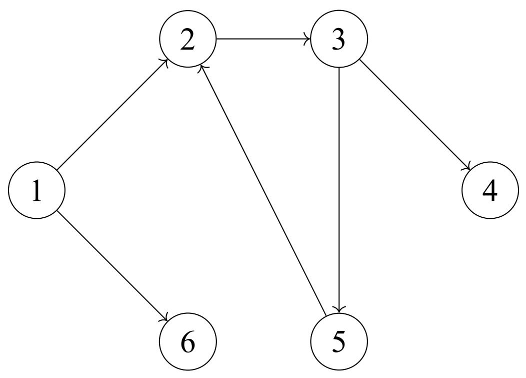


Figure 2.2: Example of a directed and simple graph.

Given a graph $G = (N, A)$, a *subgraph* $G^s = (N_s, A_s)$ is a graph such that $N_s \subseteq N$ and $A_s = \{(i, j) \in A : i, j \in N_s\}$. A *path* (*chain* in a nondirected graph) is a sequence of arcs such that the final node of one arc is the initial node of the following one, that is, $\{(i_1, i_2), (i_2, i_3), \ldots, (i_{k-1}, i_k)\}$. Node i_1 is the *initial node of the path* and node i_k is the *final node of the path*. A *circuit* (*cycle* in a nondirected graph) is a path in which the initial and the final nodes are the same, that is, $i_1 = i_k$.

Example 3. An example of a path in G_e is $\{(1, 2), (2, 3), (3, 4)\}$ and an example of a circuit is $\{(2, 3), (3, 5), (5, 2)\}$.

A *simple* path (or circuit) is a path (or circuit) which does not use the same arc more than once and an *elementary* path (or circuit) is a path (or circuit) that does not use the same node more than once.

Example 4. The path in G_e presented in Example 3 is a simple and elementary path. However, the path $\{(1,2),(2,3),(3,5),(5,2),(2,3)\}$ in G_e is neither simple (it repeats arc $(2,3)$) nor elementary (it repeats nodes 2 and 3).

An elementary circuit (or path) that goes through every node of the graph is a *Hamiltonian* circuit (or path) and a simple circuit (or path) that traverses every arc of the graph is an *Eulerian* circuit (or path). A graph is *Hamiltonian* if it contains a Hamiltonian circuit and is *Eulerian* if it contains an Eulerian circuit.

Example 5. As the circuit presented in Example 3 is the only existing circuit in G_e we can conclude that G_e is neither Hamiltonian (the circuit does not contain each node exactly once) nor Eulerian (the circuit does not contain each arc exactly once).

The number of arcs that have node i as their initial node is called *outdegree* of node i. Equivalently, the number of arcs that have node i as their final node is designated as the *indegree* of node i.

Example 6. Considering graph G_e, the outdegree of node 2 is 1 while the indegree of node 2 is 2.

A graph $G = (N, A)$ is *complete* if for every $i, j \in N,\ i \neq j$ there exists the arcs (i, j) and (j, i).

Example 7. Graph G_e is not complete since, for instance, the arc $(6,1)$ does not exist.

We say that two nodes i and j are *connected* if, ignoring the orientation of the arcs, there is at least one chain from i to j. A graph is *connected* if every pair of nodes in N is connected, otherwise the graph is *disconnected*. If there is a path between every pair of nodes then the graph is *strongly connected*. A disconnected graph is comprised of several connected subgraphs which are called *components*.

Example 8. Graph G_e is connected but it is not strongly connected as there is no path between nodes 6 and 4. If we remove the arc $(1,2)$, then the resulting graph is a disconnected graph with two components. One component is the subgraph with the node set $\{1,6\}$, and the other component is the subgraph with the node set $\{2,3,4,5\}$.

A *cut* is a partition of the node set N into two disjoint subsets S and $S' = N \setminus S$. Each cut defines a *cut-set* $[S', S]$ which is a set of arcs $(i, j) \in A$ such that $i \in S'$ and $j \in S$ or $i \in S$ and $j \in S'$. An *s-t cut* is a cut-set $[S', S]$ that is defined with respect to two distinct nodes s and t, such that: $s \in S'$ and $t \in S$.

Example 9. Considering the graph G_e and $S = \{1\}$, we have $[S', S] = \{(1, 2), (1, 6)\}$, which is an s-t cut if, for instance, $s = 1$ and $t = 2$ however, if $s = 2$ and $t = 6$ the cut $[S', S]$ is not an s-t cut.

It is possible to associate a value c_{ij} with each arc $(i, j) \in A$, which are called *costs* and the *cost of a generic path* Π is defined as $\sum_{(i,j)\in\Pi} c_{ij}$. A graph is *symmetric* if the cost matrix associated with A is symmetric, that is, if $c_{ij} = c_{ji}, \forall (i, j) \in A$, and *asymmetric* if there exists at least a pair of nodes such that $c_{ij} \neq c_{ji}$, with $(i, j) \in A$.

2.2 Polyhedral theory

In this section we present a summary of definitions and results addressing polyhedra, which are defined later on. For more on polyhedral theory see Nemhauser and Wolsey (1988), Wolsey (1998), Schrijver (1998) and Conforti et al. (2014).

Definition 1. A vector $y \in \mathbb{R}^n$ is a *linear combination* of vectors $x^1, x^2, \ldots, x^k \in \mathbb{R}^n$ if $y = \sum_{i=1}^{k} \lambda_i x^i$, with $\lambda_1, \lambda_2, \ldots, \lambda_k \in \mathbb{R}$.

Definition 2. A vector $y \in \mathbb{R}^n$ is an *affine combination* of vectors $x^1, x^2, \ldots, x^k \in \mathbb{R}^n$ if it is a linear combination and $\sum_{i=1}^{k} \lambda_i = 1$, with $\lambda_1, \lambda_2, \ldots, \lambda_k \in \mathbb{R}$.

Definition 3. A vector $y \in \mathbb{R}^n$ is a *convex combination* of vectors $x^1, x^2, \ldots, x^k \in \mathbb{R}^n$ if it is a linear combination and $\sum_{i=1}^{k} \lambda_i = 1$, with $\lambda_1, \lambda_2, \ldots, \lambda_k \geq 0$.

Definition 4. The *convex hull* of a set $X \in \mathbb{R}^n$, denoted by $conv(X)$, is the set of all vectors (or points) that are convex combinations of the vectors (or points) in X.

Definition 5. A set of vectors $x^1, x^2, \ldots, x^k \in \mathbb{R}^n$ is *linearly independent* if the only solution of the system $\sum_{i=1}^{k} \lambda_i x^i = 0^n$, with $\lambda_i \in \mathbb{R}$, is $\lambda_i = 0, \forall i = 1, \ldots, k$.

The maximum number of linearly independent points in $\mathbb{R}^n$ is n.

Definition 6. A set of vectors $x^1, x^2, \ldots, x^k \in \mathbb{R}^n$ is *affinely independent* if the only solution of the system $\sum_{i=0}^{k} \lambda_i x^i = 0^n, \sum_{i=0}^{k} \lambda_i = 0$, with $\lambda_i \in \mathbb{R}$, is $\lambda_i = 0, \forall i = 1, \ldots, k$.

Note that if vectors $x^1, x^2, \ldots, x^k \in \mathbb{R}^n$ are linearly independent then they are also affinely independent. However, the reciprocal is not valid. Consequently, the maximum number of affinely independent points in $\mathbb{R}^n$ is $n+1$ (n linearly independent points and the zero vector).

We define matrices $D \in \mathbb{R}^{m\times n}$, with m rows and n columns, and $b \in \mathbb{R}^{m\times 1}$, with m rows and 1 column. Note that the rows of D may be seen as vectors $d_R^i \in \mathbb{R}^n,\ \forall i = 1, \ldots, m$, and equivalently, the columns of D may be viewed as vectors $d_C^i \in \mathbb{R}^m,\ \forall i = 1, \ldots, n$.

Definition 7. The maximum number of linearly independent rows in D is the *rank* of D and is denoted by $rank(D)$.

Example 10. Consider the following matrix $D_e \in \mathbb{R}^{3\times 2}$:

$$D_e = \begin{bmatrix} 1 & 1 \\ 0 & 2 \\ 1 & 2 \end{bmatrix}$$

The vectors $(1,1), (0,2), (1,2)$, which correspond to the rows of matrix D_e, are not linearly independent since $(1,2) = 1 \times (1,1) + \frac{1}{2} \times (0,2)$ but vectors $(1,1)$ and $(0,2)$ are. Therefore, $rank(D_e) = 2$.

Considering the vector $x \in \mathbb{R}^{n\times 1}$ and the matrices D and b presented previously, it is possible to define a system of linear inequalities $Dx \leq b$. We define $(D^=, b^=)$ as the submatrix of (D, b) that contains the rows associated with the equalities present in the system and $(D^{\leq}, b^{\leq})$ as the submatrix associated with the inequalities.

Definition 8. A *polyhedron* $P \subseteq \mathbb{R}^n$ is the set of points that satisfies a finite number of linear inequalities, that is, $P = \{x \in \mathbb{R}^n : Dx \leq b\}$, with $D \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^{m\times 1}$. A *polytope* $P \subset \mathbb{R}^n$ is a polyhedron that is bounded, that is, $P \subseteq \{x \in \mathbb{R}^n : -\omega \leq x_j \leq \omega,\ \text{with } \omega > 0,\ \forall j \in \{1, \ldots, n\}\}$.

Throughout this dissertation we will focus our study on bounded polyhedra since it is always possible to define an $M \in \mathbb{N}$ such that $0 \leq x \leq M$. Consequently, henceforth we assume that $P = \{x \in \mathbb{R}^n\ :\ Dx \leq b\}$ is a polytope.

Example 11. Figure 2.3 shows the graphical representation of the polytope $P_e = \{x \in \mathbb{R}^2 : x_1 + x_2 \leq 4,\ 2x_2 \leq 5,\ \frac{1}{3}x_1 + x_2 = 3,\ x_1, x_2 \geq 0\}$ in light gray. Polytope P_e will be used to illustrate some of the definitions presented throughout this chapter.

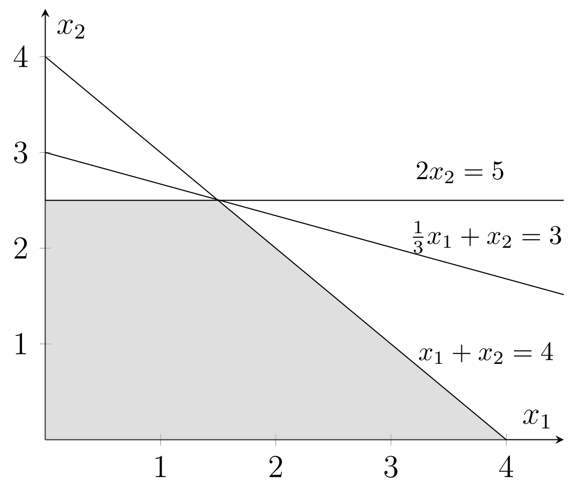


Figure 2.3: Graphical representation of the P_e polytope.

Definition 9. A polyhedron P is of *dimension* k, denoted by $dim(P) = k$, if the maximum number of affinely independent points in P is $k + 1$.

Example 12. The polytope P_e has at least three affinely independent points: $(0, 0)$, $(1, 0)$ and $(0, 1)$. Therefore, we can deduce that $dim(P_e) \geq 3 - 1 = 2$. As we saw previously, the maximum number of affinely independent points in $\mathbb{R}^2$ is 3 so, in this case, we can ensure that the equality $dim(P_e) = 2$ holds. However, usually that is not possible hence the importance of the next proposition.

Proposition 1. *If* $P \subseteq \mathbb{R}^n$*, then* $dim(P) + rank(D^=, b^=) = n$.

Example 13. Polytope $P_e \subseteq \mathbb{R}^2$ therefore $dim(P_e) = 2 - rank(D_e^=, b_e^=)$. As P_e is not defined by any equality, then $rank(D_e^=, b_e^=) = 0$. Consequently, $dim(P_e) = 2$.

Definition 10. Given $\pi \in \mathbb{R}^{1\times n}$ and $\pi_0 \in \mathbb{R}$, an inequality $\pi x \leq \pi_0$ is a *valid inequality* for $P \subseteq \mathbb{R}^n$ if is satisfied by all $x \in P$.

A given valid inequality $\pi x \leq \pi_0$ is *violated* by a point y if y does not satisfy that inequality, that is, $\pi y > \pi_0$.

Definition 11. Consider $\pi x \leq \pi_0$ and $\mu x \leq \mu_0$ two distinct valid inequalities for $P \subseteq \mathbb{R}^n$. We say that $\pi x \leq \pi_0$ *dominates* $\mu x \leq \mu_0$ if there exists $u \in \mathbb{R} : u > 0$ such that $\pi \geq u\mu$, $\pi_0 \leq u\mu_0$ and $(\pi, \pi_0) \neq (u\mu, u\mu_0)$.

Definition 12. A valid inequality $\pi x \leq \pi_0$ is *redundant* in the description of $P \subseteq \mathbb{R}^n$ if there are $k \geq 2$ valid inequalities $\mu^i x \leq \mu_0^i$, with $i = 1, \ldots, k$, for P and weights $u_i > 0$, with $i = 1, \ldots, k$, such that $(\sum_{i=1}^k u_i\mu^i)x \leq \sum_{i=1}^k u_i\mu_0^i$ dominates $\pi x \leq \pi_0$.

Example 14. Inequality $x_2 \leq 3$ is a valid inequality for P_e, while inequality $x_1 \leq 3$ is not since $(4,0) \in P_e$ and $(4,0)$ does not satisfy the previous inequality ($x_1 = 4 \nleq 3$). Additionally, inequality $x_2 \leq 3$ is dominated by inequality $2x_2 \leq 5 \Leftrightarrow x_2 \leq \frac{5}{2}$ as $\frac{5}{2} < 3$. We can also verify that inequality $\frac{1}{3}x_1 + x_2 \leq 3$, which is a valid inequality for P_e, is redundant since it can be obtained as a linear combination of inequalities $x_1 + x_2 \leq 4$ and $2x_2 \leq 5$, with $u_1 = u_2 = \frac{1}{3}$.

Definition 13. Let $\pi x \leq \pi_0$ be a valid inequality for P and $F = \{x \in P : \pi x = \pi_0\}$. Then F is called a *face* of P and we say that $\pi x \leq \pi_0$ *represents* F. A face F is a *proper face* if $F \neq \emptyset$ and $F \neq P$.

Example 15. The following sets are faces of the polytope P_e:

- $F_0 = \{(x_1, x_2) \in P_e : x_2 = 3\}$;
- $F_1 = \{(x_1, x_2) \in P_e : x_1 + x_2 = 4\}$;
- $F_2 = \{(x_1, x_2) \in P_e : 2x_2 = 5\}$, and;
- $F_3 = \{(x_1, x_2) \in P_e : \frac{1}{3}x_1 + x_2 = 3\}$.

The only face that is not a proper face of P_e is F_0, since $F_0 = \emptyset$.

Definition 14. A face F of P is a *facet* of P if $dim(F) = dim(P) - 1$.

Proposition 2. *Every inequality $d^k_R x \leq b_k$ from the system $Dx \leq b$ that represents a face of P of dimension less than $dim(P) - 1$ is irrelevant to the description of P.*

Example 16. Since F_1, F_2 and F_3 are faces of P_e we know that $dim(F_i) \leq dim(P_e) - 1 = 1$, $\forall i = 1, 2, 3$ (faces have one additional equality). Points $(4,0)$ and $(\frac{3}{2}, \frac{5}{2})$ belong to F_1 and are affinely independent, therefore $dim(F_1) \geq 2 - 1 \Rightarrow dim(F_1) = 1$, thus F_1 is a facet of P_e. By using the same argument for face F_2 and the points $(0, \frac{5}{2})$ and $(\frac{3}{2}, \frac{5}{2})$ we can conclude that F_2 is also a facet of P_e. Finally, face $F_3 = \{(\frac{3}{2}, \frac{5}{2})\}$, thus $dim(F_3) = 1 - 1 = 0$. Another way of proving that the inequality $\frac{1}{3}x_1 + x_2 = 3$ is redundant in the description of P_e is by using Proposition 2. As F_3 is not a facet, it is irrelevant in the description of P_e.

Henceforth we assume that the polytope P_e is defined by the non-redundant inequalities, that is, $P_e = \{x \in \mathbb{R}^2 : x_1 + x_2 \leq 4,\ 2x_2 \leq 5,\ x_1, x_2 \geq 0\}$, as it is shown in Figure 2.4.

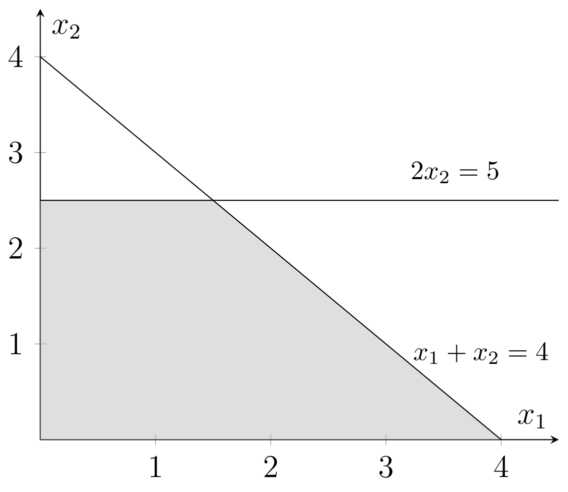


Figure 2.4: Graphical representation of the P_e polytope without the redundant inequalities.

Definition 15. A point $x \in P$ is an *extreme point* of P if it cannot be obtained as a convex combinations of other points in P.

Proposition 3. *A point $x \in P$ is an extreme point of P if and only if x is a zero-dimensional face of P.*

Example 17. As we saw previously $dim(F_3) = 0$, therefore $F_3 = \{(\frac{3}{2}, \frac{5}{2})\}$ is an extreme point of P_e.

Definition 16. Given a polyhedron $P \subseteq \mathbb{R}^{n-p} \times \mathbb{R}^p$, the *projection of P onto the subspace* $\mathbb{R}^{n-p}$, denoted $proj_x P$, is defined as:

$$proj_x P = \{x \in \mathbb{R}^{n-p} : (x, w) \in P \text{ for some } w \in \mathbb{R}^p\}$$

Example 18. The polytope $P_e \subseteq \mathbb{R}^2$ may be seen as a subset of $\mathbb{R}^1 \times \mathbb{R}^1$, thus it is possible to project P_e onto the space of x_1 and onto the space of x_2. For example, by projecting P_e onto the subspace of x_1, we obtain $proj_{x_1} P_e = \{x_1 \in \mathbb{R} : (x_1, x_2) \in P_e \text{ for some } x_2 \in \mathbb{R}\}$ and, by observing Figure 2.4 we can conclude that $proj_{x_1} P_e = [0, 4]$. Analogously, $proj_{x_2} P_e = [0, \frac{2}{5}]$.

2.3 Linear programming theory

Linear programming is a mathematical field devoted to the study of problems involving linear functions. Books that address linear programming theory are, for instance, Nemhauser and Wolsey (1988), Wolsey (1998) and Schrijver (1998).

Let $D \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^{m\times 1}$ and $c \in \mathbb{R}^{1\times n}$ be matrices with known values and $x \in \mathbb{R}^{n\times 1}$ be the vector of *variables*. A *linear programming* (LP) problem consists in determining the value of the variables, in this case the x values, that minimize (or maximize) a linear function cx, which is called *objective function*, over a polyhedron, that is,

$$\min\{cx \ : \ Dx \leq b, \ x \geq 0^{n\times 1}\}.$$

Each inequality of the system $Dx \leq b$ is called a *constraint* of the problem and the inequalities $x \geq 0^{n\times 1}$ are called the *domain* constraints of variables x.

When we impose the additional condition that some of the variables must be integer, that is, $x_1, \ldots, x_k \in \mathbb{Z}$, with $k < n$, we obtain a *mixed integer linear programming* problem. When all variables must be integer (i.e., $k = n$) we have an *integer linear programming* (ILP) problem. A particular case of ILP problems arises when the variables are binary, that is, $x \in \{0, 1\}^{n\times 1}$. This particular case is called *binary linear programming*.

Definition 17. A polyhedron $P \subseteq \mathbb{R}^n$ is a *formulation* for a set $X \subseteq \mathbb{Z}^{n-p} \times \mathbb{R}^p$ if and only if $X = P \cap (\mathbb{Z}^{n-p} \times \mathbb{R}^p)$.

Consider the following LP problem: min $\{cx \ : \ x \in P\}$. An element $x \in P$ is called a *feasible solution* for that LP problem. If $P = \emptyset$ we say that the problem is *unfeasible*. The element $x^* \in P$ such that $cx^* \leq cx, \forall x \in P$ is an *optimal solution* of the LP problem and cx^* is the *optimal value*. Note that the optimal solution of an LP problem is not necessarily unique. Additionally, recall that we are particularly interested in bounded polyhedra, that is, polytopes.

Theorem 1. *Consider a polytope $P \in \mathbb{R}^n$ and the linear programming problem* $\min\{cx : x \in P\}$*. If $P \neq \emptyset$, then there is an optimal solution of the linear programming problem that is an extreme point of P.*

According to Theorem 1, when P is a polytope, it is possible to find the optimal solution of an LP problem by enumerating all of its extreme points, however, this is not viable in practice since, usually, there are too many extreme points. Nowadays, the common approach to solve LP problems is by using commercial solvers that have specific algorithms implemented. This is how we will solve LP problems throughout this dissertation.

Example 19. Let P_e be the polytope presented in Section 2.2. Consider the following LP and ILP problems, with the same formulation P_e:

$$\max\{x_1 + 3x_2 \ : \ x \in P_e\}$$
$$\max\{x_1 + 3x_2 \ : \ x \in P_e \cap \mathbb{Z}^2\},$$

which we will designate by LP_e and P_e, respectively. Note that we called the ILP problem by the same name as the formulation since this is the notation adopted in the remainder of the dissertation.

Figure 2.4 shows the set of feasible solutions for the LP_e problem while Figure 2.5 shows the set of feasible solutions for the P_e problem. By comparing both figures we can see how adding the integrality condition changes the solution set. In fact, the solution set of P_e is not even a polyhedron.

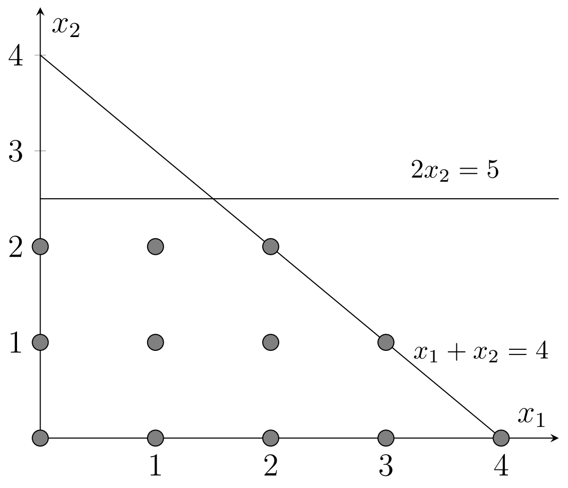


Figure 2.5: Graphical representation of the set of feasible solutions for the P_e problem.

We now analyze the optimal solutions of the LP_e and the P_e problems presented previously. In the case of the LP_e problem, the optimal solution is $(\frac{3}{2}, \frac{5}{2})$ and the optimal value is $\frac{3}{2} + 3 \times \frac{5}{2} = 9$, whilst for the P_e problem, the optimal solution is $(2, 2)$ and the optimal value is $2 + 3 \times 2 = 8$. Note that the optimal solution of the LP_e problem corresponds to an extreme point of the polytope P_e, which was determined in Section 2.2, while the optimal solution of the P_e problem does not correspond to any extreme point of the polytope P_e. Nevertheless, if we were to consider the convex hull (see Definition 4 in Section 2.2) of the points which are feasible solutions for the P_e problem, then its optimal solution would be an extreme point. This fact shows the importance of the convex hull.

Proposition 4. *For any $X \subseteq \mathbb{R}^n$ $conv(X)$ is a polyhedron.*

Proposition 5. *For any $X \subseteq \mathbb{R}^n$ the extreme points of $conv(X)$ all belong to X.*

From the results presented previously, in Propositions 4 and 5, we can conclude that the LP problem min$\{cx \ : \ x \in X\}$ is equivalent to the LP problem min$\{cx \ : \ x \in conv(X)\}$. In particular, solving the ILP problem min$\{cx \ : \ x \in X, \ x \in \mathbb{Z}^n\}$ is equivalent to solving the LP problem min$\{cx \ : \ x \in conv(X)\}$, in which $conv(X)$ is a polyhedron.

Example 20. Figure 2.6 shows, in light gray, the convex hull of the points which are feasible solutions of the P_e problem.

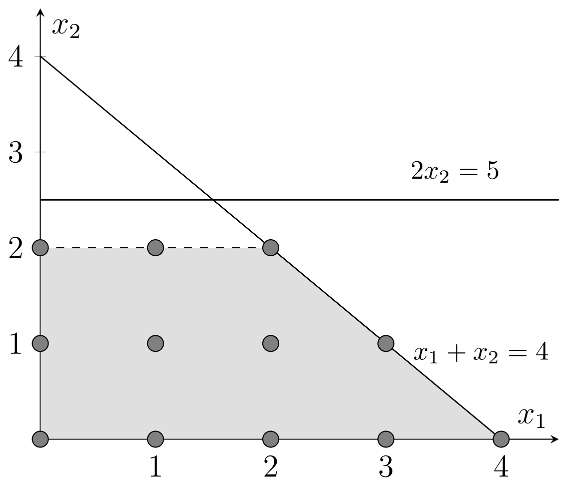


Figure 2.6: Graphical representation of the convex hull of the set of feasible solutions of the P_e problem.

When developing a formulation for an ILP problem, the aim is to create a system of inequalities that defines the convex hull of points that are the feasible solutions of the ILP problem. In most cases this is a hard task, thus, the formulation developed should give a close approximation of the convex hull.

Example 21. By observing Figure 2.6, we verify that the half-space defined by the dashed line gives a better approximation of the convex hull of the points that are feasible solutions of the P_e problem than the half-space $2x_2 \leq 5$, thus it is better to include in the formulation the inequality that originates the half-space represented by the dashed line than the inequality $2x_2 \leq 5$.

Definition 18. Given a set $X \subseteq \mathbb{R}^n$ and two formulations P_1 and P_2 for X, P_1 is a *better formulation* than P_2 if $P_1 \subseteq P_2$.

Definition 19. Given a set $X \subseteq \mathbb{R}^n$ and two formulations P_1 and P_2 for X, P_1 is *not comparable* to P_2 if $P_1 \setminus P_2 \neq \emptyset$ and $P_2 \setminus P_1 \neq \emptyset$.

In practice, in order to show that two formulations P_1 and P_2 are not comparable we consider two points $x_1 \in P_1$ and $x_2 \in P_2$ and verify that $x_1 \notin P_2$ and $x_2 \notin P_1$.

Definition 20. For the ILP min$\{cx \ : \ x \in P \cap \mathbb{Z}^n\}$ with formulation $P = \{x \in \mathbb{R}^n \ : \ Dx \leq b, \ x \geq 0^{n \times 1}\}$, the *linear programming relaxation* is the linear program min$\{cx \ : \ x \in P\}$.

Example 22. Note that the LP_e problem, presented in Example 19, is the linear programming relaxation of the P_e problem, also presented in Example 19, as the LP relaxation is obtained by relaxing the integrality conditions of the ILP problem. Consequently, the LP_e problem will be called LP relaxation of the P_e problem henceforth.

Proposition 6. *Suppose that P_1 and P_2 are two formulations for the problem* $\min\ \{cx\ :\ x \in X \subset \mathbb{Z}^n\}$ *such that P_1 is a better formulation than P_2. Let $\mathcal{V}^{LP}(P_i)$ be the linear programming relaxation value of* $\min\{cx\ :\ x \in P_i \subset \mathbb{Z}^n\}$*, with $i = 1, 2$, then $\mathcal{V}^{LP}(P_1) \geq \mathcal{V}^{LP}(P_2)$ for all c.*

If P_1 and P_2 are two not comparable formulations for the set $X \subset \mathbb{R}^n$, then it is not possible to establish a relationship between their linear programming relaxation values which is valid for every cost matrix $c \in \mathbb{R}^{1\times n}$.

Example 23. Consider a new LP problem with the formulation $P_{conv} = conv(P_e \cap \mathbb{Z}^2)$, that is,

$$\max\{x_1 + 3x_2\ :\ x \in P_{conv}\},$$

which will be designated as the LP relaxation of the P_{conv}. Note that the formulation P_{conv} is the one represented in Figure 2.6. Since the formulation P_{conv} is contained in the formulation P_e and due to the result of Proposition 6, we expect that $\mathcal{V}^{LP}(P_{conv}) \leq \mathcal{V}^{LP}(P_e)$. In fact, $\mathcal{V}^{LP}(P_{conv}) = 8$ and $\mathcal{V}^{LP}(P_e) = 9$.

Definition 21. A formulation $P \subset \mathbb{R}^n$ is an *ideal formulation* for a set $X \subset \mathbb{Z}^n$ if and only if $P = conv(X)$.

Example 24. Formulation P_{conv} is an ideal formulation as it corresponds to convex hull of the set of points that are feasible solutions of the P_e problem, as mentioned previously. A consequence of having an ideal formulation is that when we solve an LP problem over $conv(P_e)$ the optimal solution obtained is integer, and, therefore, it is the optimal solution of the ILP problem.

Considering the matrices D, b and c defined as previously, we define a formulation $P = \{x \in \mathbb{R}^n\ :\ Dx \leq b,\ x \geq 0^{n\times 1}\}$ for the ILP $\min\{cx\ :\ x \in X = P \cap \mathbb{Z}^n\}$. To simplify, we will refer to the ILP as P, which is the name of the formulation. When the formulation P is not ideal, that is, $conv(X) \neq P$, we have to resort to a *branch-and-bound* algorithm to determine the optimal solution of the ILP problem. The basic idea of a branch-and-bound (B&B) algorithm is to partition the set of feasible solutions of P into smaller and easier to search subsets. We start by solving the LP relaxation of P. Since we are considering a minimization problem, $\mathcal{V}^{LP}(P)$ is a lower bound for the optimal value of P. Similarly, the value of any feasible solution for P is an upper bound for its optimal value. Let $\overline{x_0}$ be the optimal solution of the LP relaxation of P. If $\overline{x_0}$ is integer, then it corresponds to the optimal solution of P, otherwise we must do *branching* on a variable with fractional value. Let $\overline{x_0^i}$ be the value of a variable with fractional value. We create two new

B&B subproblems, P_{k_1} and P_{k_2}, which correspond to the problem P with the additional constraints $x^i \leq \lfloor\overline{x_0^i}\rfloor$ and $x^i \geq \lceil\overline{x_0^i}\rceil$, respectively (other branching techniques may be used, see for instance Barnhart et al., 1998). The B&B subproblems P_{k_1} and P_{k_2} are added to the *list of open* subproblems. Then, we choose and remove a B&B subproblem P_k from the list of open subproblems and solve its LP relaxation. If P_k is not unfeasible, let $\mathcal{V}^{LP}(P_k)$ be the LP relaxation value of P_k and $\overline{x_k}$ be the corresponding optimal solution. If $\overline{x_k}$ is feasible for P, that is, if it is integer, we verify whether or not $\mathcal{V}^{LP}(P_k)$ is lower than the best upper bound found so far during the B&B algorithm, which we designate by UB^*, and, if it is, we update UB^* to $\mathcal{V}^{LP}(P_k)$. Otherwise, that is, if $\overline{x_k}$ is not feasible for P, we also must check whether $\mathcal{V}^{LP}(P_k) < UB^*$ or not. If $\mathcal{V}^{LP}(P_k) \geq UB^*$, then there is no need to do branching on the optimal solution of the LP relaxation of P_k as the B&B subproblems that stem from P_k would never originate a solution with a lower value than UB^*. If $\mathcal{V}^{LP}(P_k) < UB^*$, we do branching on a variable with a fractional value as we explained previously and the B&B subproblems originated go to the list of open subproblems. The B&B algorithm stops when the list of open problems is empty and one optimal solution corresponds to the integer solution with value UB^*.

Having formulations that provide a high LP relaxation value (in the case of minimization problems) is very important in order to have an efficient B&B algorithm, since we can eliminate B&B subproblems if their LP relaxation value is higher than the best upper bound found so far. We can create several formulations for the same ILP problem, namely by using different variables and, consequently, defined in different subspaces. In these cases in order to compare formulations we must use projections so that all formulations are defined in the same subspace. Additionally, we can also create formulations with an exponential number of constraints (or variables), which are called *non-compact* formulations. Formulations with a polynomial number of constraints (or variables) are designated by *compact* formulations.

Linear duality

Given matrices $D \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^{m\times 1}$ and $c \in \mathbb{R}^{1\times n}$ and the vector of variables $x \in \mathbb{R}^{n\times 1}$, we define the *primal* problem ($\mathfrak{P}$) as the following linear programming problem:

$$\max\{cx \ : \ Dx \leq b, x \geq 0^{n\times 1}\}$$

The *dual* problem ($\mathfrak{D}$) associated with ($\mathfrak{P}$) is the LP problem:

$$\min\{u^T b \ : \ D^T u \geq c^T, u \geq 0^{m\times 1}\}$$

where $u \in \mathbb{R}^{m\times 1}$ is a column vector of variables.

Note that the dual problem of the LP problem ($\mathfrak{D}$) is the LP problem ($\mathfrak{P}$). Therefore, it is equivalent to define the maximization problem as the primal problem and the minimization problem as the dual problem or vice-versa.

Proposition 7 (Weak duality)**.** *Consider the primal problem ($\mathfrak{P}$) and the dual problem ($\mathfrak{D}$) defined previously and let x and u be feasible solutions for ($\mathfrak{P}$) and ($\mathfrak{D}$), respectively. Then, $cx \leq u^T b$.*

A consequence from Proposition 7 is that any feasible solution of ($\mathfrak{D}$) provides an upper bound for the optimal value of ($\mathfrak{P}$).

Proposition 8. *Let x^* and u^* be feasible solutions for ($\mathfrak{P}$) and ($\mathfrak{D}$), respectively. Then the following statements are equivalent:*

(i) *x^* and u^* are, respectively, the optimal solutions of ($\mathfrak{P}$) and ($\mathfrak{D}$).*

(ii) *$cx^* = (u^*)^T b$.*

(iii) *If a component of u^* is positive, the corresponding inequality in $Dx \leq b$ is satisfied by x^* with equality, that is, $(u^*)^T(b - Dx^*) = 0$, and, equivalently, if a component of x^* is positive, the corresponding inequality in $D^T u \geq c^T$ is satisfied by u^* with equality, that is, $x^*(D^T u^* - c^T) = 0$.*

Example 25. Consider the formulation $P_{dual} = \{(u_1, u_2) \in \mathbb{R}^2 : u_1 + 2u_2 \geq 3,\ u_1 \geq 1,\ u_1, u_2 \geq 0\}$, which is represented in Figure 2.7, and the following LP problem, which we designate by LP_{dual}:

$$w = \min\{4u_1 + 5u_2 \ : \ (u_1, u_2) \in P_{dual}\}.$$

Notice that, the LP_{dual} problem is the dual problem associated with the LP relaxation of the P_e problem. The optimal solution of the LP_{dual} problem is $(1, 1)$ and the optimal value is $w = 4 \times 1 + 5 \times 1 = 9$, which corresponds to the optimal value of the LP relaxation of the P_e problem. Consider now the ILP problem P_e, the only relationship that we can establish between the LP_{dual} and the P_e problems is given by Proposition 7, that is, the value of any feasible solution of the LP_{dual} problem is greater than or equal to the value of any feasible solution of the P_e problem. In fact, the optimal value of the P_e problem is 8.

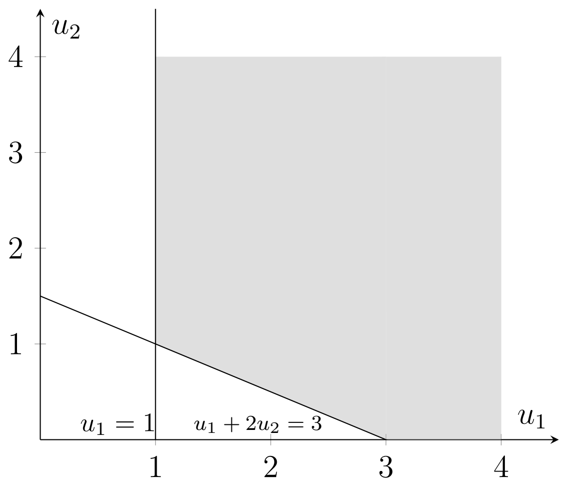


Figure 2.7: Graphical representation of the formulation P_{dual}.

Generating valid inequalities

Consider $X = P \cap \mathbb{Z}^n$, with $P = \{x \in \mathbb{R}^n \ : \ Dx \leq b, x \geq 0^{n \times 1}\}$. We know that is possible to formulate any ILP problem as an LP problem by using its convex hull. However, in order to do so we would need to have an explicit description of $conv(X)$. Therefore, we must find valid inequalities for $conv(X)$ that are violated by the points in $P \setminus conv(X)$. We present several ways of generating valid inequalities.

Chvátal-Gomory procedure to construct valid inequalities for X: Recall that d_C^i, with $i = 1, \ldots, n$, are the column vectors of D and let $\lambda \in \mathbb{R}^{m \times 1}$, $\lambda_j \geq 0$, with $j = 1, \ldots, m$:

(i) the inequality $\sum_{i=1}^n (\lambda d_C^i) x^i \leq \lambda b$ is valid for P;

(ii) the inequality $\sum_{i=1}^n \lfloor \lambda d_C^i \rfloor x_i \leq \lambda b$ is valid for P; and

(iii) the inequality $\sum_{i=1}^n \lfloor \lambda d_C^i \rfloor x_i \leq \lfloor \lambda b \rfloor$ is valid for X.

Theorem 2. *Every valid inequality for X can be obtained by applying the Chavátal-Gomory procedure a finite number of times.*

Example 26. Recall the inequality $2x_2 \leq 5$ used in formulation P_e. By using the procedure (iii) of the Chvátal-Gomory procedure to construct valid inequalities we obtain the inequality $x_2 \leq \lfloor \frac{5}{2} \rfloor = 2$ which, according to Theorem 2, is a valid inequality for the set of points which corresponds to the feasible solutions of the P_e problem. In fact, this is the inequality that defines the half-space represented by a dashed line in Figure 2.6 and describes the convex hull of the set of points which are feasible solutions of the P_e problem.

Cutting plane algorithm

Usually, when solving a non-compact formulation we do not consider explicitly the constraints that are exponential in number. Instead, and for such constraints, we iteratively identify a violated one and add it to the formulation. Next, we present an algorithm to add such constrains in a dynamic way.

Consider the LP min$\{cx \ : \ x \in P\}$, with $P = \{x \in \mathbb{R}^n \ : \ Dx \leq b, \ x \geq 0^{n\times1}\} = \{x \in \mathbb{R}^n \ : \ D^1x \leq b^1, \ D^2x \leq b^2, \ x \geq 0^{n\times1}\}$ being a polytope with a set of constraints that are in exponential number, such that matrices D, b and x are defined as previously and we assume, without loss of generality, that $D^1x \leq b^1$ is a subset of constraints of $Dx \leq b$ that is in exponential number.

Definition 22. Consider a point $y \in \mathbb{R}^n$ and a polytope $P \subset \mathbb{R}^n$. The *Separation Problem* consists in deciding whether $y \in P$ or not, and, in the latter case, finding a valid inequality for P that is violated by y.

Considering the LP problem presented previously and the polytope P, Algorithm 2.1 shows the pseudocode for the cutting plane algorithm.

Algorithm 2.1 Cutting plane algorithm.

```
1: Set $P^0 = \{x \in \mathbb{R}^n \ : \ D^2x \leq b^2, x \geq 0^{n\times1}\}$ and $t = 0$.
2: while true do
3:     Solve the LP min$\{cx \ : \ x \in P^t\}$. Let $\overline{x^t}$ be the optimal solution.
4:     if $\overline{x^t} \in P$ then
5:         $\overline{x^t}$ is an optimal solution for the LP problem. STOP.
6:     else
7:         Find a valid inequality $(d^1_R)^t x \leq (b^1)^t$ for $P$ such that $(d^1_R)^t\overline{x^t} > (b^1)^t$.
           Set $P^{t+1} = P^t \cap \{x \in \mathbb{R}^n \ : \ (d^1_R)^t x \leq (b^1)^t\}$ and $t = t + 1$.
           If there are no more violated inequalities, then $\overline{x^t} \in P$. STOP.
8:     end if
9: end while
```

The cutting plane algorithm presented in Algorithm 2.1 is designed to solve LP problems with non-compact formulations and it is the cutting plane algorithm that we will use in this dissertation. Nonetheless, there are other cutting plane algorithms designed to solve ILP problems such as the Gomory's cutting plane algorithm (see e.g. Wolsey, 1998).

Consider the following ILP problem min$\{cx : x \in P \cap \mathbb{Z}^n\}$, with the formulation P defined previously, which will be referred to as P for simplification purposes. As the polytope P has exponential many constraints, in order to obtain the optimal solution of P we have to resort to a *branch-and-cut* algorithm, which was proposed by Padberg and Rinaldi (1987, 1991). The branch-and-cut (B&C) algorithm has the same outline as a B&B algorithm but, due to the exponential many constraints, each B&C subproblem is solved using the cutting plane algorithm presented in Algorithm 2.1. The B&C algorithm is similar to the B&B algorithm, except for the case in which the optimal solution of a B&C subproblem is an integer solution, as we cannot guarantee that the integer solution is feasible for P unless we apply the separation algorithm to this integer solution. In other words, when we obtain an integer solution when solving a B&C subproblem we must apply the separation algorithm to that solution to verify whether it is feasible for P or not.

2.4 Complexity theory

Given a problem, one of the purposes of complexity theory is to determine how "difficult" it is to solve it. Throughout this section we briefly introduce some classes of problems as well as some results which will help us to identify to which class a given problem belongs. For a more extensive explanation of complexity theory see, for example, Papadimitriou and Steiglitz (1998), Schrijver (1998) and Wolsey (1998).

As we only intend to show some basic results related to complexity theory we will not be giving a mathematical definition of concepts such as *problem*, *problem instance*, *instance size*, *algorithm* and *running time function*, since only a basic idea of these concepts suffices to understand the next results.

Consider a problem min$\{cx : x \in X\}$. Each problem has a *decision problem* (or a *recognition version* of the problem) associated with it, which is the following:

Given a value $k \in \mathbb{R}$ is there an $x^* \in X$ such that $cx^* \leq k$?

The decision problem has a YES or NO answer.

Before proceeding we introduce some concepts related to an algorithm's running time. An algorithm is called *polynomial* if its running time function is bounded by a polynomial function. A problem is *solvable in polynomial time* if it can be solved by a polynomial algorithm.

Definition 23. $\mathcal{NP}$ is the class of decision problems with the following property: for any problem instance such that the answer is YES, there is a certificate that enable to verify the answer YES in polynomial time.

The certificate may be seen as a solution for the original problem. The decision problem is in $\mathcal{NP}$ if there is an algorithm that verifies if the certificate is in fact a solution for the original problem and satisfies the additional constraint ($cx \leq k$) in polynomial time.

Example 27. To illustrate the definition of $\mathcal{NP}$, consider the P_e problem presented in Section 2.3. A decision problem associated with this ILP is the following: *Is there* $(x_1^*, x_2^*) \in P_e \cap \mathbb{Z}^2$ *such that* $x_1^* + 3x_2^* \leq 10$*?*

A certificate in this case could be the point $(3, 1)$. An algorithm for the P_e problem would have to verify if $(3, 1) \in P_e \cap \mathbb{Z}^2$ and $3 + 3 \times 1 \leq 10$, which can be done in polynomial time. Therefore, the decision problem associated with the P_e problem is in $\mathcal{NP}$.

Definition 24. $\mathcal{P}$ is the class of decision problems in $\mathcal{NP}$ for which there exists a polynomial algorithm.

Definition 25. Let Q and R be decision problems. If an instance of Q can be converted into an instance of R in polynomial time, then Q is *polynomially reducible* to R.

Definition 26. *$\mathcal{NP}$-complete* is the subset of decision problems $Q \in \mathcal{NP}$ such that for all $R \in \mathcal{NP}$, R is polynomially reducible to Q.

Proposition 9. *Suppose that problems $Q, R \in \mathcal{NP}$:*

(i) If $Q \in \mathcal{P}$ and R is polynomially reducible to Q, then $R \in \mathcal{P}$.

(ii) If $Q \in \mathcal{NP}$-complete and R is polynomially reducible to Q, then $R \in \mathcal{NP}$-complete.

Definition 27. An (I)LP problem for which the decision problem belongs to the $\mathcal{NP}$-complete class is called an *NP-hard* problem.

The easiest way of showing that a problem Q is NP-hard is by using Proposition 9, that is, if we can show that Q is polynomially reducible to another problem that is NP-hard, then we can conclude that Q is also NP-hard.

When a problem is NP-hard, finding its optimal solution may not be possible in practice due to the high computational time or the limitations of the available technology, namely computational memory. Thus, we resort to *heuristic* procedures, which are algorithms that provide feasible solutions for a problem efficiently.

2.5 Basic linear programming problems

In this section we present some LP problems which are used as subroutines to solve the ILP problem which is the focus of this dissertation. All the (I)LP problems that we are going to present in this section share one property: the decision problems associated with them belongs to the class $\mathcal{P}$. This implies that there are algorithms that are able to solve them optimally in polynomial time. These problems are presented, for instance, in Ahuja et al. (1993) and Wolsey (1998).

In Section 2.5.1 we present the assignment problem, the maximum flow problem is presented in Section 2.5.2, in Section 2.5.3 we describe the minimum capacity cut problem and, finally, in Section 2.5.4 we present the shortest path problem.

2.5.1 The assignment problem

Consider that there are n people available to perform n jobs, that each person must be assigned to one job and that each job must be assigned to one person. Some individuals may be more suited to perform a given job than others, thus there is a cost c_{ij} if person i is assigned to job j. Note that assigning less suited people to perform a determined job carries a higher cost. The objective of the assignment problem is to determine the assignment with the minimum total cost.

In order to formulate the assignment problem, consider the binary variables x_{ij} which have value 1 if person $i \in \{1, \dots, n\}$ performs job $j \in \{1, \dots, n\}$ and value 0 otherwise. A formulation for the assignment problem is the following:

$$\textit{Minimize} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.1}$$

Subject to:

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i \in \{1, \dots, n\} \tag{2.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j \in \{1, \dots, n\} \tag{2.3}$$

$$x_{ij} \in \{0,1\}0 \qquad \forall i \in \{1, \dots, n\},\ \forall j \in \{1, \dots, n\}. \tag{2.4}$$

The objective (2.1) represents the minimization of the total cost. Constraints (2.2) ensure that each person performs exactly one job and the set of constraints (2.3) guarantee that each job is performed by exactly one person. Finally, constraints (2.4) define the domain of the x variables.

2.5.2 The maximum flow problem

The maximum flow (max-flow) problem consists in sending the maximum amount of flow between two nodes, which are called the source node and the target node, in a capacitated network, that is, a graph in which the arcs have a minimum and a maximum capacity.

Let $G = (N, A)$ be a directed and simple graph with minimum and maximum capacities $l_{ij} \geq 0$ and $u_{ij} \geq 0$, respectively, such that $l_{ij} \leq u_{ij}$, associated with each arc $(i, j) \in A$. The source node is denoted by s and the target node by t. Consider variables x_{ij} that represent the amount of flow that traverses the arc $(i, j) \in A$ and variable v as the value of the flow. The max-flow problem can be formulated as follows:

$$\textit{Maximize } v \tag{2.5}$$

Subject to:

$$\sum_{j:(s,j)\in A} x_{sj} - \sum_{j:(j,s)\in A} x_{js} = v \tag{2.6}$$

$$\sum_{j:(t,j)\in A} x_{tj} - \sum_{j:(j,t)\in A} x_{jt} = -v \tag{2.7}$$

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = 0 \qquad \forall i \in N \setminus \{s, t\} \tag{2.8}$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \qquad \forall (i, j) \in A. \tag{2.9}$$

The objective is to maximize the flow, which is stated in the objective (2.5). Constraints (2.6) and (2.7) state that v units of flow must leave s and enter t, respectively. The set of constraints (2.8) ensures that the amount of flow that enters and leaves each node, other than the source and the target node, is the same. Constraints (2.9) state that the flow traversing each arc must satisfy the lower capacities l and the upper capacities u.

Given a flow x which is feasible for the capacitated network (or graph) G presented previously, the *residual network* associated with x and G is the capacitated network $G' = (N, A' \cup A'')$ with $A' = \{(i, j) : (i, j) \in A \text{ and } x_{ij} < u_{ij}\}$ and $A'' = \{(j, i) : (i, j) \in A \text{ and } x_{ij} > l_{ij}\}$ in which the lower capacity of the arc $(i, j) \in A' \cup A''$ is $l'_{ij} = 0$ and the upper capacity u'_{ij} depends on whether $(i, j) \in A'$ or $(i, j) \in A''$. In the former case we have $u'_{ij} = u_{ij} - x_{ij}$ whereas in the latter case we have $u'_{ij} = x_{ji} - l_{ji}$.

2.5.3 The minimum capacity cut problem

Consider a directed and simple graph $G = (N, A)$ with capacities $l_{ij} \geq 0$ and $u_{ij} \geq 0$, such that $l_{ij} \leq u_{ij}$, associated with each arc $(i, j) \in A$. The capacity $u[S', S]$ of an s-t cut $[S', S]$ in G is defined as follows:

$$u[S', S] = \sum_{i\in S'} \sum_{j\in S} u_{ij} - \sum_{i\in S} \sum_{j\in S'} l_{ij}$$

The minimum capacity cut (min-cut) problem consists in determining the s-t cut with the lowest capacity (the s-t cut $[S', S]^*$ such that $u[S', S]^* \leq u[S', S]$, for all s-t cuts $[S', S]$).

The min-cut problem and the max-flow problem are closely related, in fact, the min-cut problem is the dual problem associated with the max-flow problem. According to Proposition 8, that states that the optimal value of the dual problem is equal to the optimal value of the primal problem, we can conclude that the capacity of the minimum s-t cut is equal to the maximum flow between s and t, which is stated in the next theorem. Additionally, we can obtain the sets S' and S which define the min-cut by considering that S' contains the nodes that are reachable from s in the residual network associated with the max-flow.

Theorem 3 (Max-flow/min-cut theorem)**.** *The maximum value of the flow from a source node s to a target node t in a capacitated network is equal to the value of the minimum s-t cut.*

There is a theorem which allow us to verify is there exists a feasible flow in a capacitated network, which may be seen as a generalization of the max-flow/min-cut theorem, presented in Theorem 3. The original theorem is due to Hoffman (see, e.g., Gondran and Minoux, 1984) and is stated for a flow which remains constant in the network, however, it can be adapted for the cases in which that does not hold. Therefore, we will present the theorem considering the latter case (see, e.g., Gouveia et al., 2013).

Theorem 4 (Theorem of compatible flow)**.** *Given a directed graph $G = (N, A)$ and capacities l_{ij} and u_{ij}, such that $l_{ij} \leq u_{ij}$, for each arc $(i, j) \in A$, a flow v such that*

$$\begin{aligned} & l_{ij} \leq v_{ij} \leq u_{ij} && \forall (i,j) \in A \\ & \sum_{j \in N} v_{ji} - \sum_{j \in N} v_{ij} = e_i && \forall i \in N \end{aligned}$$

exists if and only if

$$\sum_{i \in N \setminus S} \sum_{j \in S} u_{ij} \geq \sum_{i \in S} \sum_{j \in N \setminus S} l_{ij} + \sum_{i \in S} e_i \qquad \forall S \subset N$$

2.5.4 The shortest path problem

Given a directed graph $G = (N, A)$, two distinct nodes $s, t \in N$ and costs c_{ij} associated with each arc $(i, j) \in A$, the objective of the shortest path problem is to determined the minimum cost elementary path from s to t. In order to formulate the shortest path we define variables x_{ij} which have value 1 if the arc $(i, j) \in A$ is used in the shortest path from s to t and have value 0 otherwise.

The shortest path problem can be formulated as follows:

$$\textit{Minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{2.10}$$

Subject to:

$$\sum_{j:(s,j)\in A} x_{sj} - \sum_{j:(j,s)\in A} x_{js} = 1 \tag{2.11}$$

$$\sum_{j:(t,j)\in A} x_{tj} - \sum_{j:(j,t)\in A} x_{jt} = -1 \tag{2.12}$$

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = 0 \qquad \forall i \in N \setminus \{s,t\} \tag{2.13}$$

$$0 \le x_{ij} \le 1 \qquad \forall (i,j) \in A. \tag{2.14}$$

The objective is to minimize the cost of the path, which is stated in the objective (2.10). Constraints (2.11) and (2.12) state that one arc must leave s and enter t, respectively. Constraints (2.13) guarantee that the number of arcs that enters and leaves each node that is neither s nor t is the same. Constraints (2.14) define the domain of variables x.

Chapter 3

The Family Traveling Salesman Problem

The main focus of this dissertation is the family traveling salesman problem (FTSP) which is a variant of the well-known traveling salesman problem (TSP) (see for instance, Lawler et al., 1985; Applegate et al., 2006). Given a depot, a set of cities and a cost matrix, which represents the traveling cost between each pair of cities, the objective of the TSP is to determine a minimum cost Hamiltonian circuit (or cycle). For the FTSP consider, additionally, that the set of cities is partitioned into several subsets which are called families. The objective of the FTSP is to determine the minimum cost elementary circuit that: (i) begins and ends at the depot; and (ii) visits a given number of cities in each family.

The FTSP may be modeled by using a complete and simple directed graph $G = (\{0\} \cup N, A)$, in which 0 represents the depot and N is the set of nodes, previously called cities, that is partitioned into the several families. We will refer to the singleton subset $\{0\}$ as 0 to simplify the notation further on. The cost of using the arc $(i, j) \in A$ is denoted by c_{ij}. There are L disjoint families, represented by F_l, with $l = 1, \ldots, L$, such that $N = \cup_{l=1}^{L} F_l$. To simplify some definitions further on we define $\mathcal{L} = \{1, \ldots, L\}$. The number of members of family l is n_l and $\sum_{l=1}^{L} n_l = |N|$. We consider, without loss of generality, that the nodes that belong to family 1 are nodes $1, 2, \ldots, n_1$, the nodes that belong to family 2 are $n_1 + 1, \ldots, n_1 + n_2$, etc. In each family l we are required to visit v_l nodes and the total number of visits that we are required to make is denoted by $V = \sum_{l=1}^{L} v_l$. We assume that $n_l, v_l \geq 1, \ \forall l \in \mathcal{L}$ since if that was not the case we could define a new FTSP instance by removing the families $l \in \mathcal{L}$ such that $n_l = 0$ or $v_l = 0$. We define the set of single-visit families as $\mathcal{U} = \{l \in \mathcal{L} \ : \ v_l = 1\}$ and the set of multi-visit families as $\mathcal{M} = \mathcal{L} \setminus \mathcal{U}$. There is a special case which happens when all family members must be visited, which we define as $\mathcal{W} = \{l \in \mathcal{L} \ : \ v_l = n_l\}$. We say that family l is complete if there are v_l nodes from family l in the circuit. Clearly, in order to have a feasible solution all families must be complete. This notation and these definitions are going to be used throughout this dissertation. Figure 3.1 shows a feasible solution for an FTSP instance with two families. Family 1, which is represented with the light gray color, has two family members (nodes 1 and 2), and family 2, which is represented with the dark gray color, has three members (nodes 3, 4 and 5), and we are required to visit one node from family 1 and two nodes from family 2.

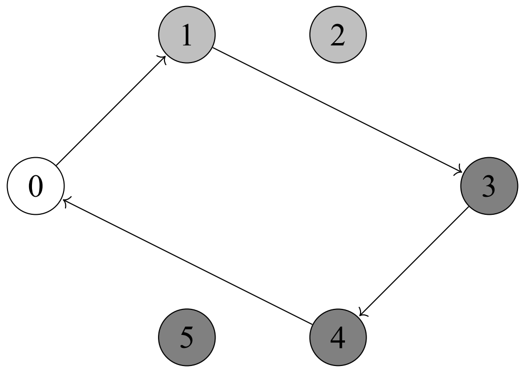


Figure 3.1: An example of a feasible solution for an FTSP instance.

The FTSP is NP-hard since it reduces to the TSP when all families belong to set $\mathcal{W}$, that is, $n_l = v_l, \forall l \in \mathcal{L}$.

The rest of this chapter is organized in the ensuing way. In Section 3.1 we present the literature review, Section 3.2 shows some ILP problems that can be solved as the FTSP under specific circumstances, in Section 3.3 we provide some basic heuristic methods to obtain feasible solutions for the FTSP and, finally, in Section 3.4 we present the test instances used in the computational experiment.

3.1 Literature review

The FTSP was firstly introduced by Morán-Mirabal et al. (2014) and, as far as we know, this is the only article that addresses it, besides the one published in the scope of this dissertation (see Bernardino and Paias, 2018a). Morán-Mirabal et al. (2014) motived the FTSP by the order picking problem in warehouses where products of the same type are stored in different warehouses or in different places in the same warehouse. Note that due to technological advances, it is possible to locate a product very easily and thus there is no need to store products of the same type in the same place. If we consider that each product is a family and the number of family members that we wish to visit is the demand of the product associated with that family, then the order picking problem in warehouses may be modeled as an FTSP.

Even though Morán-Mirabal et al. (2014) focus their work on heuristic methods, they proposed an ILP formulation for the FTSP. This formulation is identical to the one proposed for the TSP by

Dantzig et al. (1954) with the additional set of constraints ensuring the number of required visits per family. With this model Morán-Mirabal et al. (2014) were able to solve benchmark instances with up to 48 nodes (we present these benchmark instances in Section 3.4).

Regarding heuristic methods, Morán-Mirabal et al. (2014) proposed a biased random key genetic algorithm (BRKGA) and a greedy randomized adaptive search procedure (GRASP) with an evolutionary path-relinking procedure. The computational experiment shows that the GRASP method outperforms the BRKGA in the benchmark instances with higher dimension. With the referred methods, Morán-Mirabal et al. (2014) provided upper bounds for the instances that their exact method was unable to solve.

3.2 Related problems

As we said in Section 3.1, the FTSP is not widely studied in the literature, but there are several problems that may be modeled as an FTSP under specific circumstances and which have a variety of applications. The TSP is one such problem, as we have already mentioned, the FTSP in which all families belong to $\mathcal{W}$ is the TSP.

The FTSP was created as an extension of the generalized traveling salesman problem (GTSP). In the GTSP the set of cities is partitioned into clusters (which are called families in the FTSP) and one wants to find the circuit with the minimum cost that visits each cluster at least once and every node no more than once (see e.g., Srivastava et al., 1969; Gutin and Punnen, 2006). A particular case of the GTSP, which is called the equality GTSP (see, e.g., Gutin and Punnen, 2006), arises when we state that each cluster must be visited exactly once. Therefore, if we consider an FTSP where every family is a single-visit family, that is, $v_l = 1, \ \forall l \in \mathcal{L}$, then the feasible solutions for this FTSP are also feasible for the equality GTSP.

The FTSP may also be seen as a variant of the generalized covering salesman problem (GCSP), that was presented by Golden et al. (2012). In the GCSP each city $i \in N$ can cover a subset of cities D_i and it has a predefined covering demand of k_i . The objective of the GCSP is to determine a circuit in which each city i is covered at least k_i times by the cities in the circuit. Consider that we have $|N|$ families. Family i, which is associated with city i, will have as family members cities j such that $i \in D_j$ and v_i equal to k_i. Note that we must replicate the cities that belong to different families so that the families are a partition of the set of cities. The feasible solutions for the FTSP presented previously are also feasible for the GCSP.

Another problem that can be transformed into the FTSP is the capacitated traveling purchaser problem (CTPP) (see, e.g., Boctor et al., 2003). In the CTPP, one wishes to purchase several copies of items that belong to a list and each item is available in a subset of markets (cities). Consider that we have L distinct items and item l is available in the markets (cities) that belong to the set F_l and one wishes to purchase v_l units of product l. Assuming that each market only sells one unit of the product and that the cost of each item is the same in every market that sells it, solving this CTPP is equivalent to solving an FTSP.

3.3 Basic constructive heuristics and neighborhoods

In this section we present constructive heuristics which provide feasible solutions for the FTSP, as well as neighborhoods that allow us to search the solution space. The purpose is to use these procedures as subroutines in the branch-and-cut algorithm, which will be presented in Chapter 5, and in the heuristic methods, which will be presented in Chapter 6. Therefore, we will not present any computational results in this section. We start by presenting constructive heuristics in Section 3.3.1 and then the neighborhoods in Section 3.3.2.

3.3.1 Constructive heuristics

We propose three different constructive heuristics. The first one is an adaptation of a known greedy heuristic for the TSP, namely the nearest neighbor (see e.g., Bellmore and Nemhauser, 1968), the second one is a randomization of the nearest neighbor and the third, and final one, is a random constructive heuristic.

In order to adapt the nearest neighbor heuristic to the FTSP we must take into account that we must visit a fixed number of nodes per family. Thus, to apply the nearest neighbor we start at the depot and then we choose the node $i \in N$ such that $c_{0i} \leq c_{0j}, \forall j \in N$. We continue to choose the nearest node to the last node inserted in the circuit according to the cost matrix c. The nearest node i will only be added to the circuit if its family is not complete, otherwise node i is ignored and another node has to be chosen to be inserted in the circuit instead using the same criterion. The process ends when all the families are complete.

We also developed a randomization of the adaptation of the nearest neighbor, which we called random nearest neighbor, with the purpose of generating several FTSP feasible solutions using a greedy procedure. The algorithm is similar to the one described in the previous paragraph, with the difference that instead of choosing the nearest node to the depot we choose a random node $i \in N$. Thus, the feasible solution obtained will use arc $(0, i)$. Then, we apply the adapted nearest neighbor algorithm starting at node i.

Finally, the random constructive heuristic takes a permutation of N and decodes it into a feasible solution for the FTSP. We assume that the feasible solution starts at node 0, thus there is no need to include node 0 in the permutation, and then the family nodes are visited by the order in which they appear in the permutation. Since we are required to visit v_l nodes from each family $l \in \mathcal{L}$ then, before inserting the nodes in the circuit, we need to verify whether family l is complete or not. If family l is complete, we skip that node. Otherwise, we add it to the solution. This process is repeated until all the families are complete. An example of how the random constructive heuristic decodes a permutation into an FTSP feasible solution is shown in Example 28.

Example 28 (Random Constructive Heuristic)**.** Consider the FTSP instance presented in Figure 3.1 and the permutation $\pi = (4, 5, 3, 2, 1)$. The circuit starts at node 0 and then visits nodes 4 and 5. As $v_2 = 2$ and we already have two nodes from family 2 in the circuit we skip node 3. Then, we visit node 2 and obtain a feasible solution for this FTSP instance since all families are complete. The circuit obtained was $\{(0, 4), (4, 5), (5, 2), (2, 0)\}$.

3.3.2 Neighborhoods

We propose three neighborhoods for the FTSP: $\mathcal{N}_I$, $\mathcal{N}_O$ and 2-opt. Neighborhood 2-opt is commonly used for routing problems with a symmetric cost matrix (see e.g., Johnson and McGeoch, 1997).

Considering s as a feasible solution for the FTSP, neighborhoods $\mathcal{N}_I$, $\mathcal{N}_O$ and 2-opt are defined as follows:

- $\mathscr{N}_I(s) = \{s'$ feasible : s' can be obtained from s by switching a maximum of two nodes in the circuit$\}$
- $\mathscr{N}_O(s) = \{s'$ feasible : s' can be obtained from s by switching a maximum of two nodes, from the same family, such that one belongs to the circuit and the other does not$\}$
- 2-$opt(s) = \{s'$ feasible : s' can be obtained from s by inverting the order of a maximum of one path in the circuit$\}$

Since neighborhoods $\mathscr{N}_I$ and $\mathscr{N}_O$ are originated by straightforward moves, we will only explain in detail neighborhood 2-opt. Given a generic FTSP solution

$$s = \{(0, i_1), (i_1, i_2), \ldots, (i_{k-1}, i_k), (i_k, i_{k+1}), (i_{k+1}, i_{k+2}), \ldots, (i_{V-1}, i_V), (i_V, 0)\},$$

a solution $s' \in 2\text{-}opt(s)$ is, for instance,

$$s' = \{(0, i_{k-1}), (i_{k-1}, i_{k-2}), \dots, (i_2, i_1), (i_1, i_k), (i_k, i_{k+1}), (i_{k+1}, i_{k+2}), \dots, (i_{V-1}, i_V), (i_V, 0)\},$$

since s' was obtained from s by inverting the order of the path $\{(i_1, i_2), \dots, (i_{k-2}, i_{k-1})\}$ in the circuit s. This move is equivalent to, when considering nondirected graphs, switching two edges. Even though we are using a directed graph to model the FTSP, when we have a symmetric cost matrix, that is, when $c_{ij} = c_{ji}$, the cost of $s' \in 2\text{-}opt(s)$ can be easily computed through the cost of s since the cost of any path is equal to the cost of the inverted path. Therefore, neighborhood 2-opt was designed to be applied to instances with a symmetric cost matrix.

Example 29 (Neighborhoods)**.** Consider the feasible solution, for the FTSP instance presented in Figure 3.1, that is, $s = \{(0, 1), (1, 3), (3, 4), (4, 0)\}$. Then, the neighborhoods associated with s are:

$$\begin{aligned} \mathscr{N}_I(s) = \quad & \{\{(0,1),(1,3),(3,4),(4,0)\}, \{(0,3),(3,4),(4,1),(1,0)\}, \\ & \{(0,4),(4,3),(3,1),(1,0)\},\ \{(0,4),(4,1),(1,3),(3,0)\}, \\ & \{(0,1),(1,4),(4,3),(3,0)\},\ \{(0,3),(3,1),(1,4),(4,0)\}\} \end{aligned}$$

$$\begin{aligned} \mathscr{N}_O(s) = \quad & \{\{(0,1),(1,3),(3,4),(4,0)\},\ \{(0,2),(2,3),(3,4),(4,0)\}, \\ & \{(0,1),(1,5),(5,4),(4,0)\},\ \{(0,1),(1,3),(3,5),(5,0)\}\} \end{aligned}$$

$$\begin{aligned} 2\text{-}opt(s) = \quad & \{\{(0,1),(1,3),(3,4),(4,0)\}, \{(0,3),(3,4),(4,1),(1,0)\}, \\ & \{(0,3),(3,1),(1,4),(4,0)\},\ \{(0,4),(4,3),(3,1),(1,0)\}, \\ & \{(0,1),(1,4),(4,3),(3,0)\},\ \{(0,4),(4,1),(1,3),(3,0)\}\} \end{aligned}$$

In this example neighborhoods, $\mathscr{N}_I(s)$ and 2-opt are similar. Observe that this is a consequence from s having a small number of nodes, but, in general, these neighborhoods are more diverse and, thus, it is useful to use them both to search the solution space.

We define $Cost(s) = \sum_{(i,j) \in s} c_{ij}$ as the cost of a feasible solution s, which in this case is a circuit. The neighborhoods $\mathcal{N}_I$, $\mathcal{N}_O$ and 2-opt are searched using the following procedure, which can be applied to any generic neighborhood $\mathscr{N}$. We start by computing the cost of every solution s' that is in $\mathscr{N}(s)$. This process is not very time consuming since, in all three neighborhoods, the cost of s' can easily be calculated from the cost of s and the proposed neighborhoods are relatively small neighborhoods. In fact, the sizes of the neighborhoods presented have as an upper bound $\frac{(V+1)\times V}{2}+1$ for neighborhoods $\mathscr{N}_I$ and 2-opt, and $\sum_{l=1}^{L} v_l \times (n_l - v_l) + 1$ for neighborhood $\mathscr{N}_O$. To continue the search of the solution space, we choose the solution $s^* \in \mathscr{N}(s)$ such that $Cost(s^*) \leq Cost(s')$, $\forall s' \in \mathscr{N}(s)$. This process is repeated for the solution s^* until we cannot find a solution in $\mathscr{N}(s^*)$ that has a lower cost than the cost of s^*. The pseudocode for the neighborhood search is presented in Algorithm 3.1.

Algorithm 3.1 The neighborhood search procedure.

```
Require: A neighborhood $\mathscr{N}$ and a feasible solution $s$ for the FTSP.
1: while There is a solution in $\mathscr{N}(s)$ with lower cost than $Cost(s)$ do
2:     Compute the cost of every solution $s' \in \mathscr{N}(s)$.
3:     Select the solution $s'$ with the lowest cost. Let $s^*$ be that solution.
4:     $s = s^*$.
5: end while
```

3.4 Instances

The only benchmark instances available are the ones created by Morán-Mirabal et al. (2014). These instances are based on TSPLIB instances (see Reinelt, 1991, for more information on the TSPLIB) where families as well as number of visits were generated. They differ from the TSPLIB instances on how the distances are calculated, since the distances are not rounded to the nearest integer. For each TSPLIB instance there are three different FTSP instances, which have the same cost matrix and families but vary on the number of visits. There are 21 benchmark instances with the number of nodes varying between 14 and 1002. These instances will be referred to as the instance set 1. Table A.1 in appendix provides a complete description of this set.

As the number of instances in the instance set 1 is reduced, we developed an FTSP instance generator. Since there is a vast library of instances available for routing problems, namely, the TSP and the traveling purchaser problem (TPP) we decided to use some of those cost matrices. Therefore, the generator that we developed has as input a cost matrix, and consequently the number of nodes, and creates FTSP instances with that cost matrix and randomly generated families and number of visits per family. Each cost matrix generates four different FTSP instances, which only differ in the number of visits per family and, consequently, in the total number of visits. Instances of type $reference$, as the name suggests, are the instances of reference, as the number of visits is randomly generated between 1 an the number of family nodes. Instances of type low are generated to have a lower number of visits per family than the instance of type $reference$, whereas instances of type $high$ are generated to have a greater number of visits than instances of type $reference$. Finally, instances of type $mixed$ are a random combination of the number of visits of instances of type low and $high$. We now present a detailed description of how the several FTSP instance generator parameters were defined.

The number of families L. After analyzing the benchmark instances, we verified that the ratio between L and $|N| + 1$ varies between 0.04 and 0.21. Therefore, we decided that L should be a randomly generated integer in the interval $[0.10 \times (|N| + 1), 0.15 \times (|N| + 1)]$. By using this criterion, an instance with 100 nodes has between 10 and 15 families, for example.

The number of family elements n. To ensure that all families have at least one element we defined a minimum number of nodes per family, designated by min_n, which is the nearest integer to $0.25 \times \frac{|N|+1}{L}$. For family 1, n_1 is a random integer number generated between min_n and $\lfloor \frac{|N|+1}{L} \rfloor$. The number of nodes for the i-th family, with $i = 2, \ldots, L-1$, is generated between min_n and $\lfloor \frac{(|N|+1) - \sum_{j=1}^{i-1} n_i}{L-(i-1)} \rfloor$. The upper bound on the number of family members was chosen to guarantee that the number of family elements per family is balanced. Finally, to ensure that all nodes, other than the depot, belong to a family, the number of family nodes for family L is $n_L = |N| - \sum_{i=1}^{L-1} n_i$.

The number of family visits v. As mentioned previously, there are four types of instance. We denote by v^1, v^2, v^3 and v^4 the number of visits of an instance of type $reference$, low, $high$ and $mixed$, respectively. Since the instances of type $reference$ are the reference instances, the number of visits per family is a randomly generated integer between 1 and the number of family members n. That is, for a family $l \in \mathcal{L}$, $v_l^1 \in [1, n_l]$. Instances of type low were designed to have a smaller number of visits than the instances of type $reference$. Therefore, for a given family $l \in \mathcal{L}$, the number of visits of an instance of type low is a randomly generated integer between 1 and v_l^1. Regarding instances of type $high$, they were designed to have a higher number of visits than instances of type $reference$, consequently, for a family $l \in \mathcal{L}$, we have $v_l^3 \in [v_l^1, n_l]$. Finally, instances of type $mixed$ are a combination of instances of type low and $high$. For each family $l \in \mathcal{L}$ we generated, randomly, a binary variable r. If $r = 0$, then $v_l^4 = v_l^2$. Otherwise, $v_l^4 = v_l^3$.

The total number of visits V. For an instance of type i, with $i = 1, \ldots, 4$, the total number of visits is $V = \sum_{l=1}^{L} v_l^i$.

With this generator we created three different sets of instances. One set of instances, which we call instance set 2, is based on symmetric TSP instances available in the TSPLIB's website `https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`. The instance set 2 is composed of 64 instances with a number of nodes between 136 and 264 and was created as a complement to the benchmark instances. The cost matrices used were constructed by using the formulas presented by Reinelt (1991). A complete description of the instance set 2 is available in appendix, Table A.2.

The other two sets of randomly generated instances have an asymmetric cost matrix. The instance set 3 is based on asymmetric TSP instances and is comprised of all the asymmetric TSP instances available in the TSPLIB. Thus, instance set 3 contains a total of 76 instances with a number of nodes varying between 17 and 443. A complete description of the instance set 3 is available in appendix, Table A.3. Finally, the instance set 4 is based on asymmetric instances for the uncapacitated traveling purchaser problem (UTPP). These instances are available in the TPP's website `https://jriera.webs.ull.es/TPP.htm` and were created by Singh and van Oudheusden (1997). We used the cost matrix of the instances named $AsimSingh.|N| + 1.10.1.tpp$ with $|N| + 1 \in \{50, 100, 150, 200, 250, 300\}$, which originated 24 FTSP instances with a number of nodes varying between 50 and 300. A complete description of the instance set 4 is available in appendix, Table A.4.

To summarize, there are four sets of test instances: the instance set 1, which corresponds to the benchmark instances proposed by Morán-Mirabal et al. (2014); the instance set 2, which is based on symmetric TSP instances; the instance set 3, based on the asymmetric TSP instances; and, finally, the instance set 4, which is based on UTPP asymmetric instances. For the sets of instances which we generated, namely the sets 2, 3 and 4, each original instance originated four different types of FTSP instances, namely types $reference$, low, $high$ and $mixed$. All the instances presented previously are available in `http://familytsp.rd.ciencias.ulisboa.pt`, which was created by us.

Chapter 4

Mathematical Formulations

The FTSP may be seen has having two types of decision. On the one hand, we need to select which family members are visited and, on the other hand, we need to establish a single connected elementary circuit with the selected family members and the depot, being the latter the hardest to formulate. In Section 4.1 we present a generic formulation for the FTSP in which the constraints that ensure that the solution obtained is a single connected circuit are presented in a generic manner and then, in Section 4.2, we present several ways of formulating them, which is the main focus of this chapter. In Section 4.3 we establish a theoretical comparison between the several ways of formulating the constraints that ensure that the solution obtained is a single connected circuit and, finally, in Section 4.4, we present an empirical comparison between the several formulations.

When we introduced the FTSP we assumed that G was a complete graph, this assumption does not lose generality as the following exposition can easily be adapted to graphs that are not complete by simply removing the arcs (i, j) such that $(i, j) \notin A$ from the mathematical expressions present in the several formulations. Additionally, not to repeat definitions throughout this chapter, for any subset S of nodes $S \subseteq N$ we define $S' = 0 \cup (N \setminus S)$.

4.1 A generic formulation for the FTSP

Let x_{ij} be a binary variable that has value 1 if the arc $(i, j) \in A$ is in the circuit and value 0 otherwise. Variables y_i are also binary and have value 1 if the node $i \in N$ is visited in the circuit and value 0 otherwise. The FTSP may be formulated with the following generic ILP model:

$$\textit{Minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{4.1}$$

Subject to:

$$\sum_{j\in N} x_{0j} = 1 \tag{4.2}$$

$$\sum_{j\in 0\cup N} x_{ij} = y_i \qquad \forall i \in N \tag{4.3}$$

$$\sum_{j\in 0\cup N} x_{ji} - \sum_{j\in 0\cup N} x_{ij} = 0 \qquad \forall i \in 0 \cup N \tag{4.4}$$

$$\sum_{i\in F_l} y_i = v_l \qquad \forall l \in \mathcal{L} \tag{4.5}$$

$$\{(i,j) \in A : x_{ij} = 1\} \textit{ is a single connected circuit} \tag{4.6}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{4.7}$$

$$y_i \in \{0,1\} \qquad \forall i \in N. \tag{4.8}$$

The objective of the FTSP is to find the minimum cost elementary circuit, which is represented by (4.1). Constraint (4.2) ensures that there is an arc leaving the depot and, together with constraints (4.4), guarantees that there is an arc entering the depot. Constraints (4.3) state that if a node $i \in N$ is visited then there must be an arc leaving i, otherwise there is no arc leaving i. This set of constraints also shows that variables y are auxiliary, since they could be removed from the model using equality (4.3). Constraints (4.4) ensure that the indegree and the outdegree of each node are equal. The set of constraints (4.5) guarantees the number of required visits per family. Finally, constraints (4.7) and (4.8) define the domain of the x and the y variables, respectively.

There is a set of valid inequalities that can be derived for single-visit families. Consider two distinct nodes $i, j \in F_l$ such that $l \in \mathcal{U}$, which we recall is the set of single-visit families. As the number of visits of family l is one, we know that the arc $(i, j) \in A$ will never be used in a feasible solution since using it implies that two nodes from a single-visit family would be visited. Thus, we can add the ensuing inequalities as valid inequalities to the generic formulation: $x_{ij} = 0,\ \forall l \in \mathcal{U},\ \forall i, j \in F_l : (i, j) \in A$.

A solution that satisfies the equation system (4.2)–(4.5), (4.7)–(4.8) has an arc leaving and an arc entering the depot, visits the number of required nodes per family and, for each node, the indegree is equal to the outdegree, however, it may not be a single connected circuit. We designate by subtour a circuit that does not involve the depot, therefore, saying that the solution may not be a single connected circuit is equivalent to saying that the solution may have subtours. The purpose of constraints (4.6), which are written in a implicit way, is to prevent subtours and, thus, they are called subtour elimination constraints. Throughout the next sections we present several explicit ways of formulating the subtour elimination constraints, which are either adaptations of known subtour eliminations constraints from the TSP or specific subtour elimination constraints for the FTSP, which will originate several formulations for the FTSP. Before doing so we introduce some notation, considering a generic two index variable v_{ij} we define $v(S_1, S_2) = \sum_{i \in S_1} \sum_{j \in S_2} v_{ij}$ and, similarly, for a generic one index variable v_i we define $v(S_1) = \sum_{i \in S_1} v_i$.

4.2 Formulating the subtour elimination constraints

We developed compact and non-compact formulations. We start by presenting the compact models in Section 4.2.1 and then the non-compact ones in Section 4.2.2.

4.2.1 Compact formulations

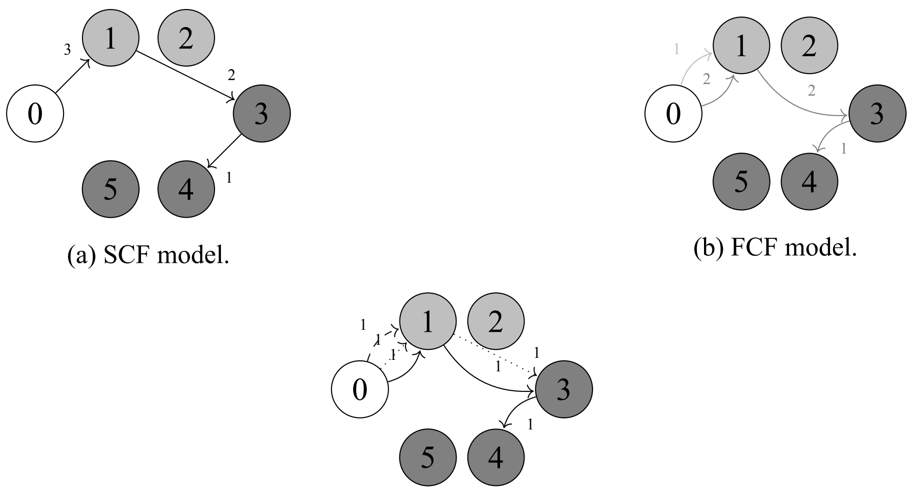
The compact formulations use flow variables to ensure that the solution obtained is connected. Some of the formulations that we propose are similar to the ones for the TSP proposed by Gavish and Graves (1978) and Wong (1980). The main difference between the FTSP and the TSP is the fact that in the FTSP we, usually, do not need to visit all the nodes.

Figure 4.1 shows a graphical representation of the flow systems associated with the flow formulations that will be presented next, by using the feasible solution for the FTSP instance introduced in Figure 3.1. We developed flow models associated with three different flow systems: a single-commodity flow (SCF) model, presented in Section 4.2.1.1, where we send one flow from the depot with V units (see Figure 4.1a); a family-commodity flow (FCF) model, available in Section 4.2.1.2, in which, as the name suggests, we send L different flows from the depot to each of the families, each one with v_l units (see Figure 4.1b); and a node-commodity flow (NCF) model, presented in Section 4.2.1.3, where we send V different one-unit flows from the depot to each of the nodes that will be visited. Additionally, we developed the node-commodity flow + (NCF$^+$) model, which we also present in Section 4.2.1.3 and uses the same flow system as the NCF model but has an additional set of constraints.

Models SCF and NCF for the FTSP are straightforward adaptations of the SCF and the multi-commodity flow models for the TSP, respectively, and may be applied, with slight modifications, to several routing problems, whereas the FCF and the NCF$^+$ models are specifically developed for the FTSP.

Figure 4.1 highlights the differences between the flow systems introduced previously in terms of the number of different flows and the amount of flow. Figures 4.1b and 4.1c have several arcs to represent the different flows associated with the corresponding flow systems. The values above each arc represent the amount of flow that traverses that arc. Observing the nodes that are not the depot, it is possible to verify how the flow conservation works in each model.

Note that the FCF and NCF are obtained by disaggregating the SCF model per family and per visited node, respectively, and the NCF model is also a disaggregation of the FCF model per node, which can also be seen in Figure 4.1. This statement will be formalized later on when establishing relationships between the LP relaxation of the proposed models.


(a) SCF model.

(b) FCF model.

(c) NCF model.

Figure 4.1: Representation of the several flow systems.

4.2.1.1 The single-commodity flow model

In order to define the single-commodity flow model we introduce variables f_{ij} which indicate the amount of flow that traverses the arc $(i, j) \in A$ and which correspond to the number of nodes that still have to be visited in the circuit. These variables can be defined as non-negative since their integrality is guaranteed by the other constraints of the model. If we replace the generic subtour elimination constraints (4.6) by the set of constraints

$$\sum_{j \in N} f_{0j} = V \tag{4.9}$$

$$\sum_{j \in 0 \cup N} f_{ji} = \sum_{j \in 0 \cup N} f_{ij} + y_i \qquad \forall i \in N \tag{4.10}$$

$$f_{ij} \leq V x_{ij} \qquad \forall (i, j) \in A \tag{4.11}$$

$$f_{ij} \geq 0 \qquad \forall (i, j) \in A, \tag{4.12}$$

we obtain a formulation for the FTSP. Constraint (4.9) guarantees that one flow with V units leaves the depot. Constraints (4.10) are the flow conservation constraints. These constraints are similar to the ones presented by Gavish and Graves (1978) for the TSP although, since we may not wish to visit every node, we will only leave one unit of flow in the nodes that are visited, which is indicated by the value of the y variables. Constraints (4.11) represent the relationship between the x and the f variables and state that flow can only traverse an arc that was chosen to be in the solution and that the flow has at most V units. Finally, constraints (4.12) define the domain of the f variables.

The equation system (4.9)–(4.12) guarantees that the solution obtained does not contain subtours since V units of flow leave the depot and are sent through a path that goes by each one of the visited nodes. If there were to be a subtour with visited nodes, then there would be no path between the depot and the nodes on that subtour.

4.2.1.2 The family-commodity flow model

As we already mentioned, the FCF model is a disaggregation of the SCF model. More precisely, the FCF model is obtained from the SCF model by disaggregating the flow variables f and the flow conservation constraints (4.10) per family. Let t^l_{ij} be the amount of flow for family $l \in \mathcal{L}$ that traverses the arc $(i, j) \in A$, which corresponds to the number of nodes from family $l \in \mathcal{L}$ that will still be visited in the circuit when we traverse the arc $(i, j) \in A$. Once again, the t variables may be defined as non-negative since their integrality is ensured by the other constraints in the model. A formulation for the FTSP is obtained by replacing the generic subtour elimination constraints (4.6) with the following set of constraints:

$$\sum_{j \in N} t^l_{0j} = v_l \qquad \forall l \in \mathcal{L} \tag{4.13}$$

$$\sum_{j \in 0 \cup N} t^l_{ji} = \sum_{j \in 0 \cup N} t^l_{ij} + y_i \qquad \forall l \in \mathcal{L},\ \forall i \in F_l \tag{4.14}$$

$$\sum_{j \in 0 \cup N} t^l_{ji} = \sum_{j \in 0 \cup N} t^l_{ij} \qquad \forall l \in \mathcal{L},\ \forall i \in N \setminus F_l \tag{4.15}$$

$$t^l_{ij} \leq v_l x_{ij}, \qquad \forall l \in \mathcal{L},\ \forall (i, j) \in A \tag{4.16}$$

$$t^l_{ij} \geq 0 \qquad \forall l \in \mathcal{L},\ \forall (i, j) \in A. \tag{4.17}$$

Constraints (4.13) guarantee that there are L different flows, each one with v_l units, leaving the depot. Constraints (4.14) and (4.15) are the flow conservation constraints which are divided in two cases: either a node belongs to the same family as the flow variable or not. In the former, constraints (4.14) are similar to constraints (4.10) presented in the SCF model, that is, we only leave one unit of flow in the visited nodes, while in the latter, the amount of flow that enters and leaves a node remains the same (see Figure 4.1b). Constraints (4.16), which model the relationship between the x and the t variables, guarantee that flow can only traverse an arc that was chosen to be in the solution and that, for each family $l \in \mathcal{L}$, the flow has at most v_l units. Finally, constraints (4.17) are the domain constraints for the t variables.

By using a similar argument to the one for the SCF model, the equation system (4.13)–(4.17) prevents subtours because it ensures that, for each family $l \in \mathcal{L}$, there is a path that starts at the depot and passes through every visited node of family l. If there were to be a subtour with visited nodes, then there would be no path between the depot and the nodes in the subtour. Besides preventing subtours, the equation system (4.13)–(4.17) also ensures that we visit the required number of nodes per family.

Due to the definition of the f and the t variables, the following relationship holds (see figure 4.1)

$$f_{ij} = \sum_{l \in \mathcal{L}} t^l_{ij}. \tag{4.18}$$

This relationship highlights the fact that the t variables are a disaggregation of the f variables per family and it is important further on to relate the LP relaxations of the two formulations.

Considering any general three-index variable v^k_{ij} we will use the following notation $v^k(S_1, S_2) = \sum_{i \in S_1} \sum_{j \in S_2} v^k_{ij}$ henceforth.

4.2.1.3 The node-commodity flow model

The NCF and the NCF$^+$ models are obtained by disaggregating the f variables and the flow conservation constraints (4.14)-(4.15) per node. Let z^k_{ij} be a binary variable which has value 1 if the arc $(i, j) \in A$ is used to send one unit of flow from the depot to node $k \in N$, with $k \neq i$, and value 0 otherwise. Even though variables z are binary, we only need to add non-negativity constraints since their integrality and bounds are ensured by the other constraints of the model. A model for the FTSP is obtained by replacing the generic subtour elimination constraint (4.6) with the following system of inequalities:

$$\sum_{j\in N} z_{0j}^k = y_k \qquad \forall k \in N \tag{4.19}$$

$$\sum_{j\in 0\cup N} z_{jk}^k = y_k \qquad \forall k \in N \tag{4.20}$$

$$\sum_{j\in (0\cup N)\setminus\{k\}} z_{ji}^k = \sum_{j\in 0\cup N} z_{ij}^k \qquad \forall k \in N,\ \forall i \in N : i \neq k \tag{4.21}$$

$$z_{ij}^k \leq y_k x_{ij}, \qquad \forall k \in N,\ \forall (i,j) \in A : i \neq k \tag{4.22}$$

$$z_{ij}^k \geq 0 \qquad \forall k \in N,\ \forall (i,j) \in A : i \neq k. \tag{4.23}$$

The first three sets of constraints are similar to the ones presented by Wong (1980) for the TSP. The main difference is that, in constraints (4.19) and (4.20), we will only use arcs to send one unit of flow from the depot to a node $k \in N$ if k is visited. Constraints (4.21) are the flow conservation constraints for the nodes that are not the destination node of the flow and, in that case, the amount of flow that enters a node is equal to the amount of flow that leaves that node (see Figure 4.1c). The set of constraints (4.22), which represent the relationship between the z variables and the x and the y variables, ensure that the flow intended for a specific node can only be sent through an arc if: (i) the node is visited; and (ii) the arc was chosen to be on the solution. We address the non-linearity of these constraints later on in this section. Finally, constraints (4.23) define the domain of the z variables.

Variables z_{ij}^k may also be seen as having value 1 if the arc $(i, j) \in A$ is used in the path from the depot to node $k \in N : k \neq i$, and value 0 otherwise. Therefore, this flow system prevents subtours since it ensures that there exists a path from the depot to every node $k \in N$ such that $y_k = 1$. Consequently, if there were to be a subtour with visited nodes (nodes k such that $y_k = 1$) then there would be no path from the depot to the nodes in the subtour.

From the definition of the t and the z variables we can derive the following relationship (see Figure 4.1)

$$t_{ij}^l = \sum_{k\in F_l} z_{ij}^k, \tag{4.24}$$

since the z variables are a disaggregation of the t variables per family node. This relationship will be used further on to compare the node-commodity flow models to the other compact models.

Linearization of constraints (4.22)

When we face non-linear constraints, the most common approach, when possible, is to linearize them. Constraints (4.22) allow the variable z to have value 1 if the corresponding x and y variables have value 1 and has value 0 if either x or y have value 0. Consequently, constraints (4.22) can be linearized by replacing them with:

$$z_{ij}^k \leq x_{ij} \qquad \forall k \in N,\ \forall (i,j) \in A : i \neq k \tag{4.25}$$

$$z_{ij}^k \leq y_k \qquad \forall k \in N,\ \forall (i,j) \in A : i \neq k. \tag{4.26}$$

Constraints (4.25) and (4.26) ensure the relationships between the z variables and the x and the y variables stated previously. When x and y have value 1 variable z can have value 1, whilst when either x or y have value 0, the value of variable z is bounded by 0. Note that constraints (4.26) are implied by constraints (4.19), (4.20) and (4.21) which makes them redundant. Therefore, in order to obtain a linear formulation for the FTSP, which is the NCF formulation, we replace constraints (4.22) by constraints (4.25).

There is another set of valid inequalities for the FTSP that can be derived from the non-linear constraints (4.22). Given an arc $(i,j) \in A$ and a family $l \in \mathcal{L}$, we aggregate constraints (4.22) for $k \in F_l$:

$$\sum_{k \in F_l} z_{ij}^k \leq \sum_{k \in F_l} y_k x_{ij} \iff \sum_{k \in F_l} z_{ij}^k \leq v_l x_{ij}$$

The previous constraints are equivalent to constraints (4.16) from the FCF model, which is easily seen through the relationship between the t and the z variables (4.24). Thus, we can define a new model, the NCF$^+$ model which is the NCF model with the additional set of constraints:

$$\sum_{k \in F_l \setminus \{i\}} z_{ij}^k \leq v_l x_{ij}, \qquad \forall l \in \mathcal{L},\ \forall (i,j) \in A. \tag{4.27}$$

Table 4.1 summarizes the proposed formulations that use the z variables.

Table 4.1: Node-commodity flow models summary.

	Constraints (4.19)-(4.21) and (4.23)
NCF	$z_{ij}^k \leq x_{ij} \quad \forall k \in N,\ \forall (i,j) \in A : i \neq k$ (4.25)
NCF$^+$	$z_{ij}^k \leq x_{ij} \quad \forall k \in N,\ \forall (i,j) \in A : i \neq k$ (4.25) $\sum_{k \in F_l \setminus \{i\}} z_{ij}^k \leq v_l x_{ij} \quad \forall l \in \mathcal{L},\ \forall (i,j) \in A$ (4.27)

4.2.2 Non-compact formulations

In this subsection, we propose three non-compact formulations for the FTSP. The connectivity cuts (CC) model, presented in Section 4.2.2.1, which is an adaptation of the well-known TSP connectivity cuts model; the rounded visits (RV) model in Section 4.2.2.2 and the rounded family visits

(RFV) model. Both the RV and the RFV models were developed specifically for the FTSP. These models are solved by using a branch-and-cut algorithm, which will be presented in Chapter 5.

4.2.2.1 Connectivity cuts model

The connectivity cuts for the TSP ensure that we must choose at least one arc from every possible cut-set in order to obtain a single connected circuit. As in the FTSP we are only required to visit V nodes, we only need to ensure that the visited nodes form a single connected circuit.

The CC model for the FTSP arises when we replace the generic subtour elimination constraints (4.6) with the following constraints:

$$x(S', S) \geq y_k \qquad \forall S \subseteq N,\ \forall k \in S. \tag{4.28}$$

Constraints (4.28) are valid for the FTSP. When $y_k = 0$, constraints (4.28) are redundant and, when $y_k = 1$, there is a node in S that is visited which implies that there must be at least one arc in the cut-set $[S', S]$ that has to be used in order to obtain a feasible circuit for the FTSP. To be easily identified, constraints (4.28) will be called CC inequalities.

4.2.2.2 Rounded visits model

Consider the flow conservation constraints (4.10) associated with the SCF model presented in Section 4.2.1.1. By adding them for $i \in S$ with $S \subseteq N$, we obtain:

$$\begin{aligned}
& f(0 \cup N, S) = f(S, N) + y(S) \\
\iff \quad & f(S', S) + f(S, S) = f(S, S) + f(S, S') + y(S) \\
\iff \quad & f(S', S) = f(S, S') + y(S) \\
\implies \quad & f(S', S) \geq y(S),
\end{aligned}$$

this results from the fact that $f(S, S') \geq 0$. By using the relationship between the f and the x variables (4.11) we obtain the following set of inequalities:

$$Vx(S', S) \geq y(S) \iff x(S', S) \geq \frac{y(S)}{V} \qquad \forall S \subseteq N, \tag{4.29}$$

which describes the projection of the system of inequalities (4.9)-(4.12) onto the subspace of the x and the y variables. This result will be presented in the next proposition, but before doing so, we define the following polytopes:

- $P_{SCF} = \{(x, y, f) \in \mathbb{R}^{|A|+|N|+|A|}$: satisfy the equation system (4.2)-(4.5), (4.9)-(4.12), $0 \leq x_{ij} \leq 1,\ \forall (i, j) \in A,$ and $0 \leq y_i \leq 1,\ \forall i \in N\}$

- $P_V = \{(x,y) \in \mathbb{R}^{|A|+|N|}$: satisfy the equation system (4.2)-(4.5), (4.29), $0 \leq x_{ij} \leq 1,\ \forall (i,j) \in A,$ and $0 \leq y_i \leq 1,\ \forall i \in N\}$

The P_{SCF} is the polytope that corresponds to the LP relaxation of the SCF model whereas the P_V is the polytope associated with the LP relaxation of the model which is obtained by replacing the generic subtour elimination constraints (4.6) with constraints (4.29). Note that due to the ensuing proposition we may conclude that P_V is a formulation for the FTSP.

Proposition 10. $P_V = proj_{x,y} P_{SCF}$

Proof. In order to prove this result, first we show that $proj_{x,y} P_{SCF} \subseteq P_V$ and, then, that $P_V \subseteq proj_{x,y} P_{SCF}$.

(i) $proj_{x,y} P_{SCF} \subseteq P_V$

This inclusion was proven when we derived constraints (4.29) at the beginning of this section, as we implicitly considered a point $(x, y, f) \in P_{SCF}$ and we verified that its projection onto the subspace of the x and the y variables satisfies constraints (4.29).

(ii) $P_V \subseteq proj_{x,y} P_{SCF}$

Consider $(x, y) \in P_V$. To show the inclusion under (ii) we must verify that is possible to construct a flow f such that $(x, y, f) \in P_{SCF}$, that is, we want to construct a flow f that satisfies the SCF constraints (4.9)-(4.12). According to the Theorem of Compatible Flow (Theorem 4) presented in Section 2.5.3, there exists a feasible circular flow f in a capacitated network, in which the lower and the upper capacities of the arc $(i, j) \in A$ are l_{ij} and u_{ij}, respectively, that satisfies the flow conservation constraints $f(0 \cup N, i) = f(i, 0 \cup N) + e_i,\ \forall i \in 0 \cup N$ if and only if for any $Q \subset 0 \cup N$ and $Q' = (0 \cup N) \setminus Q$:

$$\sum_{i \in Q'} \sum_{j \in Q} u_{ij} \geq \sum_{i \in Q} \sum_{j \in Q'} l_{ij} + \sum_{i \in Q} e_i \tag{4.30}$$

In order to apply the Theorem of Compatible Flow we must have flow conservation constraints for all the nodes in the network. Since we know that when we define a flow system there is a flow conservation constraint which is redundant in the presence of the other flow conservation constraints, when we presented the SCF model we did not define the flow conservation constraint for the depot, thus, we will deduce it immediately.

We start by adding the flow conservation constraints (4.10) of the SCF model for $i \in N$ and we obtain:

$$\begin{aligned} & f(0 \cup N, N) = f(N, 0 \cup N) + y(N) \\ \iff \quad & f(0, N) + f(N, N) = f(N, 0) + f(N, N) + y(N) \\ \iff \quad & f(0, N) - f(N, 0) = y(N) \\ \iff \quad & f(N, 0) - f(0, N) = -V, \end{aligned} \tag{4.31}$$

which is the flow conservation constraint for the depot. Note that the last equivalence holds because by adding the visits constraints (4.5) for $l \in \mathcal{L}$ we obtain $y(N) = V$.

Consequently, to show that there exists a flow satisfying (4.9)-(4.12) we need to show that the relationship (4.30) holds considering $e_i = y_i,\ \forall i \in N$, $e_0 = -V$ and, for $(i, j) \in A$, $l_{ij} = 0$ and $u_{ij} = V \times x_{ij}$.

As the e_i values differ for $i \in N$ and $i = 0$, in order to prove that the relationship (4.30) holds, we must consider two different cases depending on whether $0 \in Q$ or not.

- Case 1: $0 \notin Q$

In this case the relationship (4.30) is as follows:

$$V \times x(Q', Q) \geq y(Q) \iff x(Q', Q) \geq \frac{y(Q)}{V}$$

Since $0 \notin Q$ and $Q \subset N$, the previous inequality corresponds to constraints (4.29) which the x and the y variables satisfy.

- Case 2: $0 \in Q$

To simplify the notation during the ensuing calculations we define $Q_0 = Q \setminus 0$. The relationship (4.30) that must be satisfied in this case is the following:

$$\begin{aligned} & Vx(Q', Q) \geq y(Q_0) - V \\ \iff \quad & x(N \setminus Q_0, Q_0 \cup 0) \geq \frac{y(Q_0)}{V} - 1 \\ \iff \quad & x(Q_0 \cup 0, N \setminus Q_0) \geq \frac{y(Q_0)}{V} - 1 \\ \iff \quad & x(Q_0 \cup 0, N \setminus Q_0) \geq \frac{V - y(N \setminus Q_0)}{V} - 1 \\ \iff \quad & x(Q_0 \cup 0, N \setminus Q_0) \geq -\frac{y(N \setminus Q_0)}{V} \end{aligned}$$

The second equivalence is due to the x variables satisfying constraints (4.4) and the third one is because sets Q_0 and $N \setminus Q_0$ are a partition of the set N and we know that $y(N) = V$ by adding constraints (4.5) for $l \in \mathcal{L}$. As the x and the y variables are non-negative, the previous inequality is trivially satisfied and, consequently, there exists a feasible flow f that satisfies constraints (4.9)-(4.12). $\square$

The proof of Proposition 10 also showed that $f(N, 0) = 0$, since $f(0, N) - f(N, 0) = V$ and constraint (4.9) states that $f(0, N) = V$.

We can derive a new set of valid inequalities for the FTSP that, under specific conditions, dominate inequalities (4.29). This new set of valid inequalities is called the rounded visits (RV) inequalities. We start by presenting the RV inequalities intuitively, and then, we formalize its definition.

In the FTSP instance presented in Figure 3.1, the total number of visits is three, that is, $V = 3$. Consider the sets S' and S presented in Figure 4.2. As the set S' only contains two nodes, besides the depot, there is a node in S that has to be visited, thus there must be an arc in the cut-set $[S', S]$ that is used in order to obtain a feasible solution for this FTSP instance, which is the motivation for the RV inequalities.

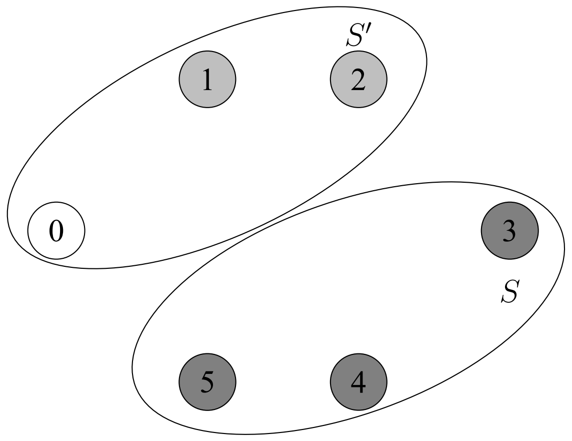


Figure 4.2: RV inequalities motivation.

Recall constraints $x(S', S) \geq \frac{y(S)}{V}$, $\forall S \subseteq N$ (4.29). Their right-hand side is less than or equal to 1, as the maximum number of visited nodes in N is V. Nonetheless, under specific circumstances we can round up the right-hand side of constraints (4.29) to 1, as we saw in Figure 4.2. Consider that $|N \cap S'| < V$ thus, since we are required to visit V nodes and the set S' contains less than V nodes we know that, in order to have a feasible solution for the FTSP, we have to use an arc in the cut-set $[S', S]$ to visit a node in S. Since $(S' \setminus 0) \cup S = N$, saying that $|S' \setminus 0| \leq V - 1$ is equivalent to saying that $|S| \geq |N| - V + 1$. Therefore, we obtain the following set of inequalities:

$$x(S', S) \geq 1 \qquad \forall S \subseteq N : |S| \geq |N| - V + 1, \tag{4.32}$$

which, for the reasons stated previously, are valid inequalities for the FTSP. Note that the RV inequalities are only defined when $|S| \geq |N| - V + 1$ and, in this case, they dominate inequalities (4.29). The following result shows that the RV inequalities may be used as subtour elimination constraints.

Proposition 11. *A formulation for the FTSP can be obtained by replacing the generic subtour eliminations constraints* (4.6) *with the RV inequalities* (4.32)*.*

Proof. Let (x^*, y^*) be an unfeasible solution for the FTSP that satisfies the equation system (4.2)–(4.5), (4.7) and (4.8). As we have already mentioned, the only way in which solution (x^*, y^*) can be unfeasible is if it contains subtours. Let $\overline{S} = \{i \in N : y_i^* = 0\}$ be the set of nodes that were not chosen to be in the solution and $\mathbb{S} = \{i_1, \ldots, i_k\}$ be the set of nodes that form a subtour. Let $S = \overline{S} \cup \mathbb{S}$. Since: (i) S' is only composed by visited nodes and the depot; (ii) S' does not contain all visited nodes due to subtour $\mathbb{S}$; and, (iii) solution (x^*, y^*) visits V nodes due to constraints (4.5); we can conclude that $|S' \setminus 0| \leq V - 1$. Equivalently, $|S| \geq |N| - V + 1$. Therefore, S satisfies the conditions in which constraints (4.32) are valid. Note that there is no arc between S' and S, as S only includes non-visited nodes and a subtour. Hence, there is a violated inequality (4.32). $\square$

The proof of Proposition 11 not only shows that if an integer solution contains subtours it is always possible to find a violated RV inequality, but also shows how to construct the set S in order to obtain such violated inequality. Moreover, the model which is obtained by replacing the generic subtour elimination constraints (4.6) with the RV inequalities (4.32) will be designed as the rounded visits (RV) model.

4.2.2.3 Rounded family visits model

The rounded family visits model can be related to the FCF model, much like the relationship of the SCF model and the RV model of the previous section, and is based on the idea that each family $l \in \mathcal{L}$ has a specific number of required visits. Therefore, consider the flow conservation constrains (4.14) and (4.15) from the FCF model, presented in Section 4.2.1.2, for a given family $l \in \mathcal{L}$. Similarly to what we did in Section 4.2.2.2, we can add those constraints for $i \in S \subseteq N$ which originates:

$$\begin{array}{ll} & t^l(0 \cup N, S) = t^l(S, N) + y(S \cap F_l) \\ \iff & t^l(S', S) + t^l(S, S) = t^l(S, S) + t^l(S, S') + y(S \cap F_l) \\ \iff & t^l(S', S) = t^l(S, S') + y(S \cap F_l) \\ \implies & t^l(S', S) \geq y(S \cap F_l) \end{array}$$

By using the relationship between the t variables and the x variables stated in constraints (4.16), we obtain:

$$v_l x(S', S) \geq y(S \cap F_l) \iff x(S', S) \geq \frac{y(S \cap F_l)}{v_l} \qquad \forall S \subseteq N. \tag{4.33}$$

Constraints (4.33) correspond to the projection of constraints (4.13)-(4.17) from the FCF model onto the subspace of the x and the y variables, as we will show in Proposition 12.

Consider the following polytopes:

- $P_{FCF} = \{(x, y, t) \in \mathbb{R}^{|A|+|N|+L\times|A|}$: satisfy the equation system (4.2)-(4.5), (4.13)-(4.17), $0 \leq x_{ij} \leq 1,\ \forall (i,j) \in A,$ and $0 \leq y_i \leq 1,\ \forall i \in N\}$
- $P_{FV} = \{(x, y) \in \mathbb{R}^{|A|+|N|}$: satisfy the equation system (4.2)-(4.5), (4.33), $0 \leq x_{ij} \leq 1,\ \forall (i,j) \in A,$ and $0 \leq y_i \leq 1,\ \forall i \in N\}$

The P_{FCF} is the polytope which corresponds to the LP relaxation of the FCF model while the P_{FV} is the polytope associated with the formulation which is obtained by replacing the generic subtour elimination constraints (4.6) with constraints (4.33). The following result besides showing that the P_{FV} polytope is the projection of the P_{FCF} polytope onto the subspace of the x and the y variables also shows that the polytope P_{FV} is a formulation for the FTSP.

Proposition 12. $P_{FV} = proj_{x,y} P_{FCF}$

Proof. This proof is very similar to the proof of Proposition 10, which means that we start by showing that $proj_{x,y} P_{FCF} \subseteq P_{FV}$ and, then, we show that $P_{FV} \subseteq proj_{x,y} P_{FCF}$.

(i) $proj_{x,y} P_{FCF} \subseteq P_{FV}$

We already proved this inclusion when we deduced constraints (4.33) since we considered a point $(x, y, t) \in P_{FCF}$ and we verified that the referred point projected onto the subspace of the x and the y variables satisfies constraints (4.33).

(ii) $P_{FV} \subseteq proj_{x,y} P_{FCF}$

Consider $(x, y) \in P_{FV}$. Similarly to what we did in the proof of Proposition 10, we must verify if it is possible to construct a flow t such that $(x, y, t) \in P_{FCF}$. For a given family $l \in \mathcal{L}$, we want to construct a flow t^l that satisfies the FCF model constraints (4.13)-(4.17). As the proof will be done using the Theorem of the Compatible Flow (Theorem 4) presented in Section 2.5.3 we must deduce the flow conservation constraint for the depot, like in the proof of Proposition 10.

For a given family $l \in \mathcal{L}$ we add the flow conservation constraints (4.14)-(4.15) for $i \in N$ and we obtain:

$$\begin{aligned}
& t^l(0 \cup N, N) = t^l(N, 0 \cup N) + y(N \cap F_l) \\
\iff \quad & t^l(0, N) + t^l(N, N) = t^l(N, 0) + t^l(N, N) + y(F_l) \\
\iff \quad & t^l(0, N) - t^l(N, 0) = y(F_l) \\
\iff \quad & t^l(N, 0) - t^l(0, N) = -v_l \qquad (4.34)
\end{aligned}$$

the flow conservation constraint for the depot. Note that the last equivalence holds due to the visits constraints (4.5). Consequently, in this case we have that $e_i = y_i, \ \forall i \in F_l$, $e_i = 0, \ \forall i \in N \setminus F_l$ and $e_0 = -v_l$. Additionally, for $(i, j) \in A$ we have $l_{ij} = 0$ and $u_{ij} = v_l \times x_{ij}$.

Now we must check whether or not the relationship $\sum_{i \in Q'} \sum_{j \in Q} u_{ij} \geq \sum_{i \in Q} \sum_{j \in Q'} l_{ij} + \sum_{i \in Q} e_i$ (4.30) is verified for any $Q \subset 0 \cup N$ and $Q' = (0 \cup N) \setminus Q$.

- Case 1: $0 \notin Q$

In this case the relationship (4.30) is as follows:

$$v_l \times x(Q', Q) \geq y(Q \cap F_l) \iff x(Q', Q) \geq \frac{y(Q \cap F_l)}{v_l}$$

Since $Q \subseteq N$, the previous inequality corresponds to a constraint (4.33) which the x and the y variables satisfy.

- Case 2: $0 \in Q$

To simplify the notation during the following calculations we define $Q_0 = Q \setminus 0$. The relationship (4.30) that must be satisfied in this case is the following:

$$\begin{aligned}
& v_l x(Q', Q) \geq y(Q_0 \cap F_l) - v_l \\
\iff \quad & x(N \setminus Q_0, Q_0 \cup 0) \geq \frac{y(Q_0 \cap F_l)}{v_l} - 1 \\
\iff \quad & x(Q_0 \cup 0, N \setminus Q_0) \geq \frac{y(Q_0 \cap F_l)}{v_l} - 1 \\
\iff \quad & x(Q_0 \cup 0, N \setminus Q_0) \geq \frac{v_l - y((N \setminus Q_0) \cap F_l)}{v_l} - 1 \\
\iff \quad & x(Q_0 \cup 0, N \setminus Q_0) \geq -\frac{y((N \setminus Q_0) \cap F_l)}{v_l}
\end{aligned}$$

The second equivalence is due to the x variables satisfying the indegree and outdegree constraints (4.4) and the third one is because sets Q_0 and $N \setminus Q_0$ are a partition of the set N and we know that $y(N \cap F_l) = v_l$ from constraints (4.5). As the x and the y variables are non-negative, the previous inequality is trivially satisfied and, consequently, for any family $l \in \mathcal{L}$ there exists a feasible flow t^l that satisfies constraints (4.13)-(4.17). Hence, we proved (ii). □

Once again, the proof of Proposition 12 showed that $t^l(N, 0) = 0, \ \forall l \in \mathcal{L}$.

Similarly to what we did in Section 4.2.2.2, we can derive a set of valid inequalities for the FTSP that under specific conditions dominate inequalities $x(S', S) \geq \frac{y(S \cap F_l)}{v_l}$ (4.33) and may be seen as a disaggregation of the RV inequalities per family. This new set of valid inequalities is called the rounded family visits (RFV) inequalities. Firstly, we present the RFV inequalities intuitively, and then, we formalize its definition.

Consider the FTSP instance presented in Figure 3.1 and the sets S' and S presented in Figure 4.3. Note that the set S does not satisfy the conditions of the RV inequalities, since $|S| = 2 < 5 - 3 + 1 = 3$, however, if we only consider the nodes in S' we could never obtain a feasible solution for this FTSP instance as the number of nodes that we are required to visit in family 2 is two and the set S' only contains one node from family 2. So, in order to complete family 2, we must visit the set S which implies that there is at least one arc in the cut-set $[S', S]$ that has to be used in any feasible circuit for this FTSP instance.

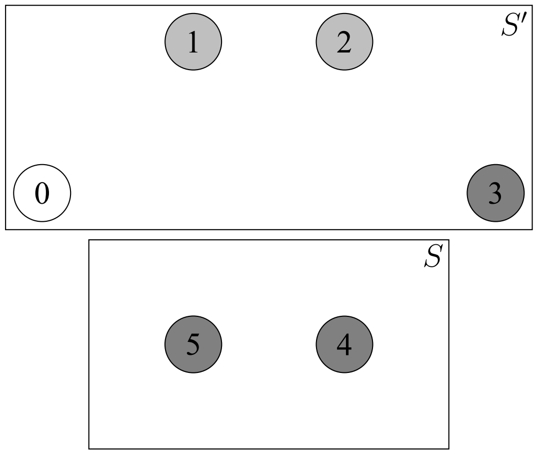


Figure 4.3: RFV inequalities motivation.

The right-hand side of constraints $x(S', S) \geq \frac{y(S \cap F_l)}{v_l}$ (4.33) is less than or equal to 1 since $y(S \cap F_l) \leq v_l$. As we saw in Figure 4.3, we can round up the right-hand side of constraints (4.33) to 1 if there are not enough nodes in S' to fulfill the family visits of, at least, one family, that is, $\exists l \in \mathcal{L} : |S' \cap F_l| \leq v_l - 1$. As $|(S' \cup S) \cap F_l| = n_l \iff |(S' \cap F_l) \cup (S \cap F_l)| = n_l$, ensuring that $|S' \cap F_l| \leq v_l - 1$ is equivalent to guaranteeing that $|S \cap F_l| \geq n_l - v_l + 1$. Consequently, we obtain the RFV inequalities:

$$x(S', S) \geq 1 \qquad \forall S \subseteq N : \exists l \in \mathcal{L} : |S \cap F_l| \geq n_l - v_l + 1. \tag{4.35}$$

These constraints are valid for the FTSP for the reasons stated previously. The next result shows that the RFV inequalities work as subtour elimination constraints.

Proposition 13. *A formulation for the FTSP is obtained by replacing the generic subtour elimination constraints* (4.6) *with the RFV inequalities* (4.35)*.*

Proof. Let (x^*, y^*) be an unfeasible FTSP solution that satisfies the equation system (4.2)–(4.5), (4.7) and (4.8) and consider $\overline{S}$ and $\mathbb{S}$ defined as in the proof of Proposition 11. Let $S = \overline{S} \cup \mathbb{S}$. Since solution (x^*, y^*) satisfies the visit requirements per family and S does not contain all the nodes that were chosen to be in the solution, we know that there is at least a family l such that $F_l \cap S \neq \emptyset$ (due to subtour $\mathbb{S}$). Consequently, for the same family l, $|S \cap F_l| \leq v_l - 1$, which implies that $|S \cap F_l| \geq n_l - v_l + 1$. Thus, S satisfies the conditions in which constraints (4.35) are valid. As there is no arc in the cut-set $[S', S]$ used in the solution, there is a constraint (4.35), more precisely, constraint $x((0 \cup N) \setminus (\overline{S} \cup \mathbb{S}), \overline{S} \cup \mathbb{S}) \geq 1$, that is violated. □

The above proof shows that if an integer solution contains subtours it is always possible to find a violated RFV inequality and, additionally, it also shows how to construct the set S in order to obtain the violated inequality. Moreover, the model which is obtained by replacing the generic subtour elimination constraints (4.6) with the RFV inequalities (4.35) is called the rounded family visits (RFV) model.

4.3 Theoretical comparison of the several formulations

This section is devoted to comparing theoretically the LP relaxation of the several proposed models for the FTSP in Section 4.2. In Section 4.3.1 we compare the compact models, in Section 4.3.2 we establish relationships between the compact and the non-compact models and, finally, in Section 4.3.3 we compare the non-compact models.

All the results presented throughout this section are stated in terms of the LP relaxation value. As we have been doing, the LP relaxation of the several proposed models is obtained by replacing the domain constraints (4.7) and (4.8) with constraints

$$0 \leq x_{ij} \leq 1 \qquad \forall (i,j) \in A \tag{4.36}$$

$$0 \leq y_i \leq 1 \qquad \forall i \in N, \tag{4.37}$$

respectively. Recall that we defined, in Section 2.3, $\mathcal{V}^{LP}(M)$ as the LP relaxation value of model M.

4.3.1 Comparing the compact models

In this section we compare the compact models, namely the SCF model, presented in Section 4.2.1.1, the FCF model, presented in Section 4.2.1.2, and the NCF and the NCF$^+$ models, presented in Section 4.2.1.3. The first result relates the compact models SCF, FCF and NCF$^+$.

Proposition 14. $\mathcal{V}^{LP}(SCF) \leq \mathcal{V}^{LP}(FCF) \leq \mathcal{V}^{LP}(NCF^+)$.

Proof. Consider the following polytopes:

- $P_{SCF} = \{(x, y, f) \in \mathbb{R}^{|A|+|N|+|A|}$: satisfy the equation system (4.2) – (4.5), (4.9) – (4.12), (4.36) and (4.37)$\}$
- $P_{FCF} = \{(x, y, f, t) \in \mathbb{R}^{|A|+|N|+|A|+L\times|A|}$: satisfy the equation system (4.2) – (4.5), (4.13) – (4.17), (4.36), (4.37) and (4.18)$\}$
- $P_{NCF^+} = \{(x, y, t, z) \in \mathbb{R}^{|A|+|N|+L\times|A|+|N|\times|A|}$: satisfy the equation system (4.2) – (4.5), (4.19) – (4.21), (4.23), (4.25), (4.27), (4.36), (4.37) and (4.24)$\}$.

More precisely, the P_{SCF} corresponds to the formulation associated with the LP relaxation of the SCF model, the P_{FCF} to the one associated to the LP relaxation of the FCF model with the additional constraints that relate the f and the t variables (4.18) and, finally, the P_{NCF^+} corresponds to the formulation associated to the LP relaxation of the NCF$^+$ model with the additional constraints that relate the t and the z variables (4.24). Consequently, proving this result is equivalent to proving that $proj_{x,y,f}FCF \subseteq P_{SCF}$ and $proj_{x,y,t}NCF^+ \subseteq P_{FCF}$. We start by showing that $proj_{x,y,f}FCF \subseteq P_{SCF}$.

Consider $(x, y, f, t) \in P_{FCF}$. We need to show that (x, y, f, t) satisfies the SCF constraints (4.9)-(4.12). Since the polyhedra P_{FCF} and P_{SCF} are defined in different subspaces, we need to use the relationship between the f and the t variables (4.18) to project the polytope P_{FCF} onto the space of the polytope P_{SCF}.

By adding constraints (4.13) for $l \in \mathcal{L}$ we obtain

$$\sum_{l \in \mathcal{L}} t^l(0, N) = \sum_{l \in \mathcal{L}} v_l \iff f(0, N) = V,$$

which shows that constraint (4.9) is satisfied.

For a fixed node $i \in F_{l^*}$ from a given family $l^* \in \mathcal{L}$, we add the flow conservation constraints (4.14)-(4.15) for $l \in \mathcal{L}$. For node i the flow conservation constraints are

$$\begin{aligned} & t^{l*}(0 \cup N, i) = t^{l*}(i, 0 \cup N) + y_i \\ & t^{l}(0 \cup N, i) = t^{l}(i, 0 \cup N) && \forall l \in \mathcal{L} \setminus \{l^*\}, \end{aligned}$$

thus, by adding them we obtain

$$\begin{aligned} & t^{l*}(0 \cup N, i) + \sum_{l \in \mathcal{L} \setminus l^*} t^{l}(0 \cup N, i) = t^{l*}(i, 0 \cup N) + \sum_{l \in \mathcal{L} \setminus l^*} t^{l}(i, 0 \cup N) + y_i \\ \iff \quad & \sum_{l \in \mathcal{L}} t^{l}(0 \cup N, i) = \sum_{l \in \mathcal{L}} t^{l}(i, 0 \cup N) + y_i \\ \iff \quad & f(0 \cup N, i) = f(i, 0 \cup N) + y_i, \end{aligned}$$

which are the flow conservation constraints (4.10) of the SCF model.

With respect to the constraints (4.11), consider an arc $(i, j) \in A$ and add constraints (4.16) for $l \in \mathcal{L}$:

$$\sum_{l \in \mathcal{L}} t^l_{ij} \leq \sum_{l \in \mathcal{L}} v_l x_{ij} \iff f_{ij} \leq V x_{ij}, \quad \forall (i, j) \in A.$$

All there is left now is to show that the constraints that define the domain of variables f are satisfied. Once again, if we consider an arc $(i, j) \in A$ and add constrains (4.17) for $l \in \mathcal{L}$ we obtain

$$\sum_{l \in \mathcal{L}} t^l_{ij} \geq \sum_{l \in \mathcal{L}} 0 \iff f_{ij} \geq 0, \quad \forall (i, j) \in A,$$

which are the domain constraints for the SCF model.

To finalize this proof, we need to show that $proj_{x,y,t} NCF^+ \subseteq P_{FCF}$. In order to do so, consider a point in P_{NCF^+}, that is, $(x, y, f, z) \in P_{NCF^+}$.

By adding constraints (4.19) for $k \in F_l$ we obtain:

$$\sum_{k \in F_l} z^k(0, N) = \sum_{k \in F_l} y_k \iff t^l(0, N) = v_l, \quad \forall l \in \mathcal{L},$$

which means that the solution (x, y, z) satisfies constraints (4.13) from the FCF model.

Concerning the satisfaction of constraints (4.14)-(4.15), consider a node $i \in N$ and a family $l \in \mathcal{L}$. There are two cases either $i \notin F_l$ or $i \in F_l$. If $i \notin F_l$, by adding constraints (4.21) for $k \in F_l$ we obtain the following expression and consequent proof:

$$\sum_{k \in F_l} z^k(0 \cup N, i) = \sum_{k \in F_l} z^k(i, 0 \cup N) \iff t^l(0 \cup N, i) = t^l(i, 0 \cup N), \quad \forall i \in N \setminus F_l.$$

Now, if $i \in F_l$ we must add constraints (4.20)-(4.21) for $k \in F_l$. This implies that when $k = i$ we must consider constraint (4.20) and when $k \in F_l$ and $k \neq i$ we must consider constraints (4.21), that is:

$$\begin{aligned} & z^i(0 \cup N, i) = y_i \\ & z^k(0 \cup N, i) = z^k(i, 0 \cup N), \quad \forall k \in F_l \setminus \{i\}. \end{aligned}$$

By adding the previous constraints we obtain the following expression:

$$\begin{aligned} & z^i(0 \cup N, i) + \sum_{k \in F_l \setminus i} z^k(0 \cup N, i) = y_i + \sum_{k \in F_l \setminus i} z^k(i, 0 \cup N) \\ \iff \quad & \sum_{k \in F_l} z^k(0 \cup N, i) = y_i + \sum_{k \in F_l \setminus i} z^k(i, 0 \cup N). \end{aligned}$$

As variables z^i_{ij} are not defined, then $\sum_{k \in F_l \setminus i} z^k(i, 0 \cup N) = \sum_{k \in F_l} z^k(i, 0 \cup N)$, therefore

$$\sum_{k \in F_l} z^k(0 \cup N, i) = y_i + \sum_{k \in F_l} z^k(i, 0 \cup N) \iff t^l(0 \cup N, i) = y_i + t^l(i, 0 \cup N), \quad \forall i \in F_l.$$

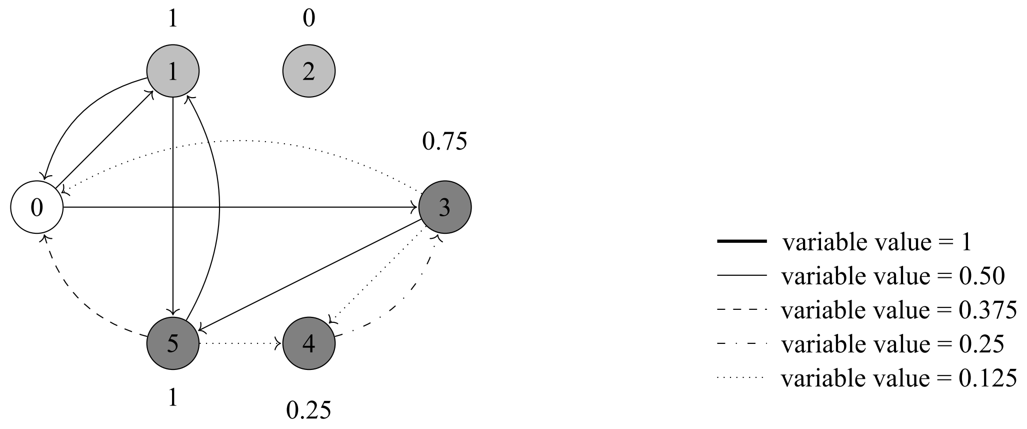
Thus, we were able to show that constraints (4.15) and (4.14) of the FCF model are satisfied by the point $(x, y, t, z) \in P_{NCF^+}$.

Consider now constraints (4.27). If we replace z with t by using the relationship (4.24) we immediately obtain that constraints (4.16) from the FCF model are satisfied.

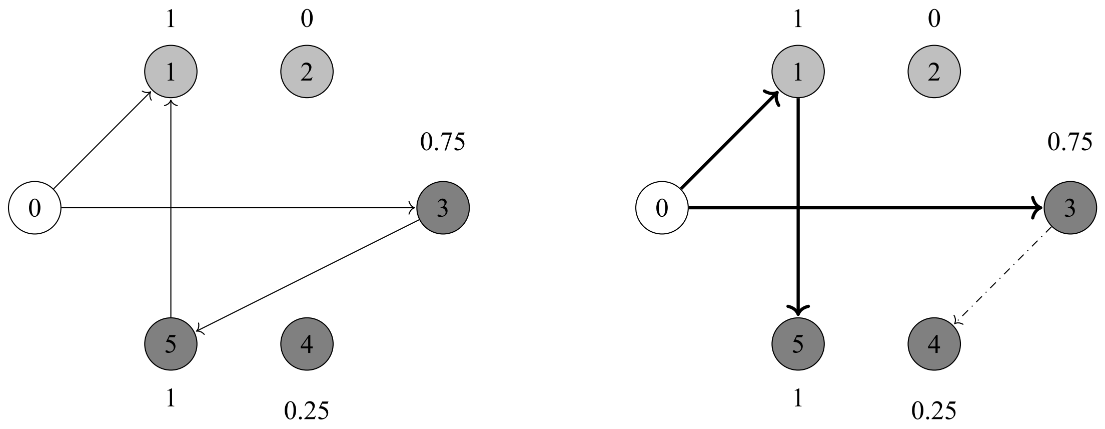
Finally, as the z variables are non-negative variables and the t variables are a summation of the z variables we can deduce that variables t are non-negative and the domain constraints for the t variables are verified. $\square$

Concerning the compact models, we still need to establish relationships with the NCF model. Intuitively, it seems that the NCF model is a better formulation than the SCF and the FCF models, since it is a disaggregation, however that may not be true. In fact, the LP relaxation of these models are not comparable. By using the FTSP instance presented in Figure 3.1, we show in Figure 4.4 a feasible solution for LP relaxation of the FCF model that is unfeasible for the LP relaxation of the NCF model and, conversely, Figure 4.5 shows a feasible solution for the LP relaxation of the NCF model that is unfeasible for the LP relaxation of both the SCF and the FCF models.

The solution presented in Figure 4.4 is feasible for the LP relaxation of the FCF model. Figure 4.4a shows the feasible solution in terms of the x variables, whereas Figures 4.4b and 4.4c show the solution in terms of the flow associated with family 1 (t^1 variables) and family 2 (t^2 variables), respectively. The figure's legend (in Figure 4.4a) relates the value of the variables to the type of arcs used. Note that the variables with value 0 were omitted from the figure. The values at the top or at the bottom of each node correspond to the value of the y variable associated with that node.



(a) Variables x.



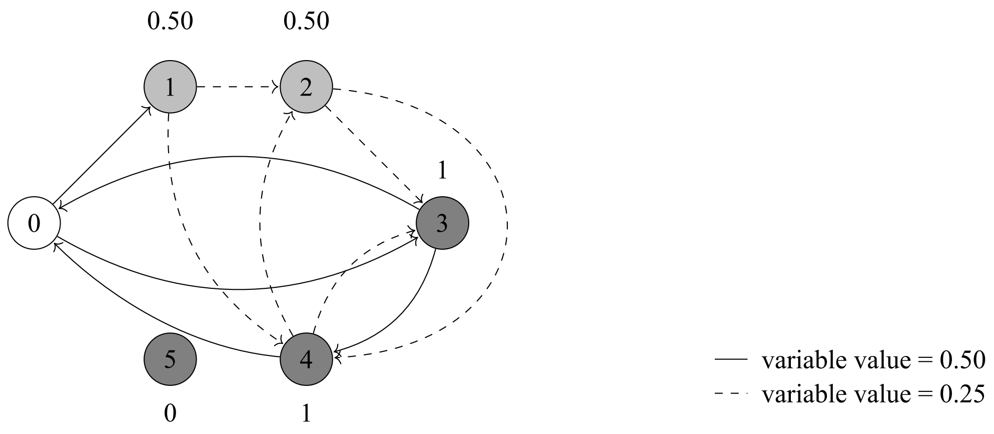
(b) Variables t^1.

(c) Variables t^2.

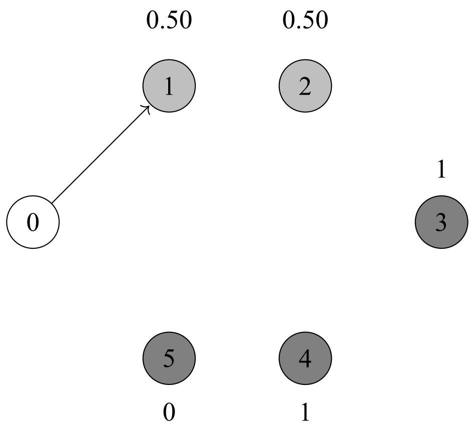
Figure 4.4: Feasible solution for the LP relaxation of the FCF model.

Consider node 3 and the disjoint fractional paths (on the arcs) from the depot to node 3 presented in Figure 4.4a, which are the paths: $\Pi_1 = \{(0,3)\}$ and $\Pi_2 = \{(0,1),(1,5),(5,4),(4,3)\}$. The NCF model ensures that the total amount of flow that traverses both paths and reaches node 3 must be $0.75 = y_3$ and the value of the z variables cannot exceed the value of the x variables. It is possible to send 0.50 units of flow through path Π_1 but it is only possible to send 0.125 units of flow through path Π_2, as $x_{43} = 0.125$. Hence, the maximum amount of flow that can reach node 3 through Π_1 and Π_2 is $0.625(= 0.50 + 0.125) < 0.75$ which implies that the solution presented does not satisfy constraint (4.20) from the NCF model that states that $\sum_{j \in 0 \cup N} z^3_{j3} = y_3$. Due to the relationship between the t and the f variables (4.18) it is possible, by using the solution presented in Figure 4.4 for the FCF model, to construct a feasible solution for the LP relaxation of the SCF model that is unfeasible for the LP relaxation of the NCF model.

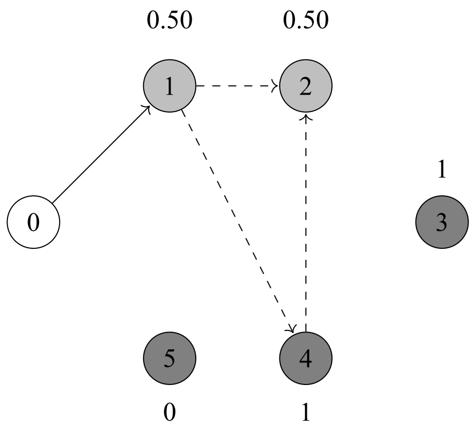
Consider now Figure 4.5, which shows a feasible solution for the LP relaxation of the NCF model. Figure 4.5a shows the feasible solution in terms of the x variables while Figures 4.5b, 4.5c, 4.5d and 4.5e show the solution in terms of the variables z^1, z^2, z^3 and z^4, respectively. The filled arcs correspond to the ones in which the variables have value 0.50 and the dashed ones correspond to a variable value of 0.25. The values at the top or at the bottom of each node correspond to the value of the y variable associated with that node, like in Figure 4.4.



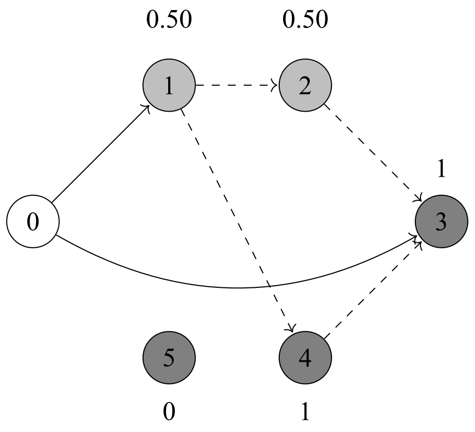
(a) Variables x.



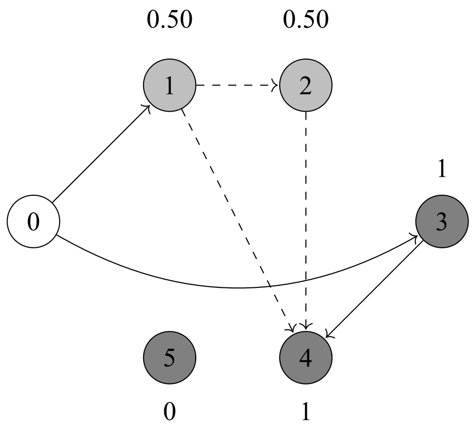
(b) Variables z^1.



(c) Variables z^2.



(d) Variables z^3.



(e) Variables z^4.

Figure 4.5: Feasible solution for the LP relaxation of the NCF model.

In order to show that the feasible solution for the LP relaxation of the NCF model presented in Figure 4.5 is not feasible for the LP relaxation of the SCF and the FCF models we will focus on the arc $(0, 1)$. Observing the several subfigures we verify that $x_{01} = z^1_{01} = z^2_{01} = z^3_{01} = z^4_{01} = 0.50$. Firstly, consider the SCF model. The value of variable f_{01} can be obtained as $\sum_{k=1}^{5} z^k_{01} = 2$, and we know that $f_{01} \leq V x_{01} = 3 \times 0.5 = 1.50$. However $2 > 1.50$, which shows that the solution presented does not satisfy constraint (4.11) associated with the arc $(0, 1)$ of the SCF model. Secondly, to show that the LP relaxation of the NCF model is not comparable to the LP relaxation of the FCF model we use a similar argument, but now we must consider variable $t^1_{01} = z^1_{01} + z^2_{01} = 1$. Constraints (4.16) state that $t^1_{01} \leq v_1 x_{01} = 1 \times 0.50 = 0.50$, therefore the solution presented violates them. Proposition 15 states these findings.

Proposition 15. *The LP relaxation of the NCF model is not comparable to the LP relaxation of the SCF model and to the LP relaxation of the FCF model.*

Finally, we relate the NCF model to the NCF$^+$ model. Recall that the NCF$^+$ model is the NCF model with the additional set of constraints (4.27), which, as we saw in Section 4.2.1.3, are equivalent, under the constraints that relate the t variables to the z variables (4.24), to constraints (4.16) from the FCF model. By using the arguments previously stated we know that constraints (4.16) are not redundant for the NCF model, since they are violated by the feasible solution for the LP relaxation of the NCF model presented in Figure 4.5, thus neither are constraints (4.27). For this reason, we have the result of Proposition 16.

Proposition 16. $\mathcal{V}^{LP}(NCF) \leq \mathcal{V}^{LP}(NCF^+)$.

As we will see in the empirical comparison of Section 4.4, there are instances for which the relationship stated in Proposition 16 is satisfied with the strict inequality.

4.3.2 Comparing the compact and the non-compact models

During this section we establish a comparison between the LP relaxation of the compact models, which are the SCF model, the FCF model and the NCF and the NCF$^+$ models presented in Sections 4.2.1.1, 4.2.1.2 and 4.2.1.3, respectively, and the non-compact models, namely the CC model, RV model and the RFV model presented in Sections 4.2.2.1, 4.2.2.2 and 4.2.2.3, respectively.

First, we compare the LP relaxation of the RV model to the LP relaxation of the compact models. This will be done by using a feasible solution for the LP relaxation of the RV model which is shown in Figure 4.6, which is not feasible for the LP relaxation of the NCF$^+$ model, and a feasible solution for the LP relaxation of the NCF$^+$ model presented in Figure 4.7 that is not feasible for the LP relaxation of the RV model. Consider a new FTSP instance in which family 1, which is represented by the light gray color, is composed by two nodes (nodes 1 and 2) and family 2, which is represented by the dark gray color, has four family members (nodes 3-6). The number of nodes that are required to be visited in family 1 is one and in family 2 is two. Figure 4.6 shows a feasible solution for the LP relaxation of the RV model considering the FTSP instance described previously. The filled arcs correspond to the ones in which the x variables have value 0.50 and the dashed ones correspond to the x values equal to 0.25. The value on the top or at the bottom of each node corresponds to the value of the y variable associated with that node.

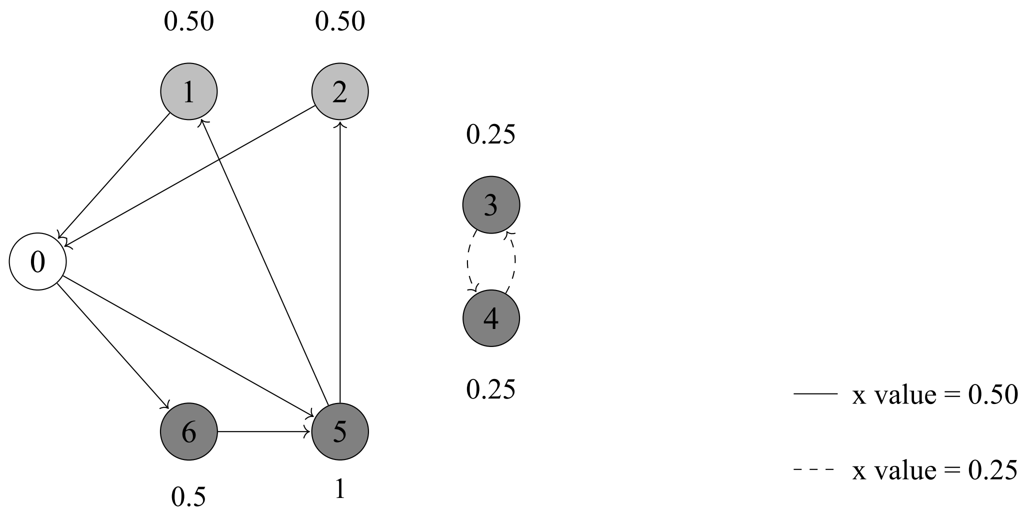
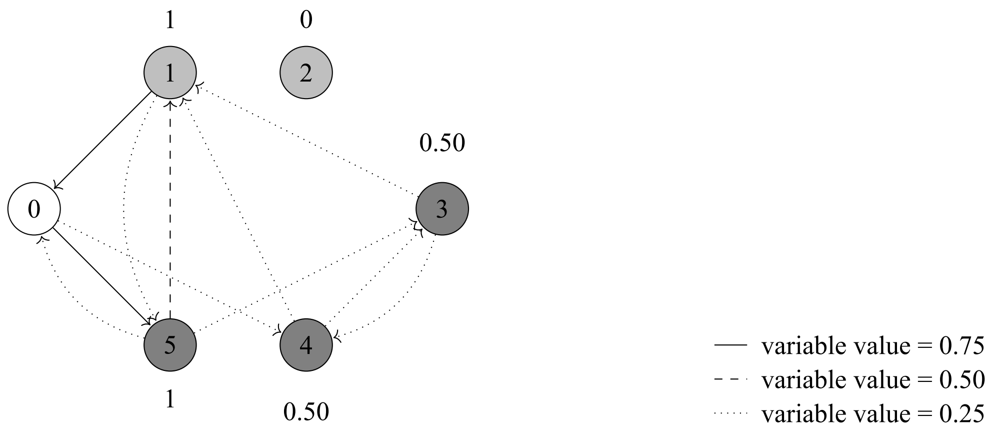


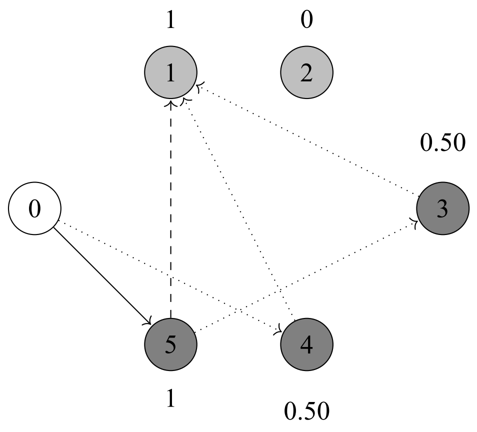
Figure 4.6: Feasible solution for the LP relaxation of the RV model.

As mentioned in Section 4.2.1, the compact models guarantee that there exists paths that have as initial node the depot that go through every visited node. More precisely, the SCF model ensures that a path goes through every visited nodes, the FCF model makes sure that there are L paths and each one traverses every visited node from the respective family and, finally, the NCF and the NCF$^+$ models guarantee that there is a path from the depot to every visited node. Note that the solutions that are feasible for the LP relaxation of the compact models are also connected, the main difference to the integer solutions of the compact models is that the paths may have fractional values and may be in bigger number. Obviously, in Figure 4.6, there is no path from the depot to nodes 3 and 4, which are visited nodes since $y_3 = y_4 = 0.25 > 0$. Therefore, this particular solution is feasible for the LP relaxation of the RV model but does not satisfy the LP relaxation of any of the compact models.

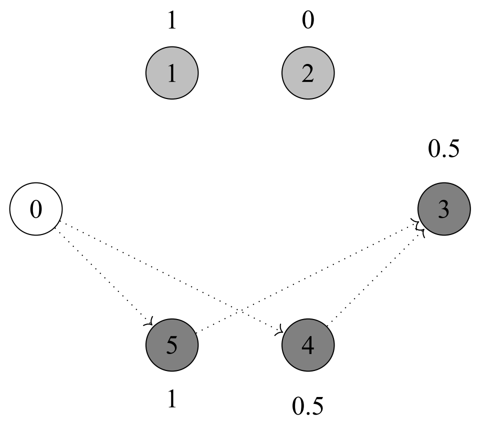
Focus now on Figure 4.7. As mentioned previously, the solution presented is feasible for the LP relaxation of the NCF$^+$ model considering the FTSP instance presented in Figure 3.1. Subfigure 4.7a shows the fractional solution in terms of the x variables whilst Subfigures 4.7b, 4.7c, 4.7d and 4.7e show the solutions in terms of variables z^1, z^3, z^4 and z^5, respectively. The filled arcs correspond to the ones in which variables have value 0.75, the dashed ones correspond to a variable value of 0.50 and the dotted arcs correspond to a variable with value 0.25. The values on the top or at the bottom of each node correspond to the value of the y variable associated with that node.



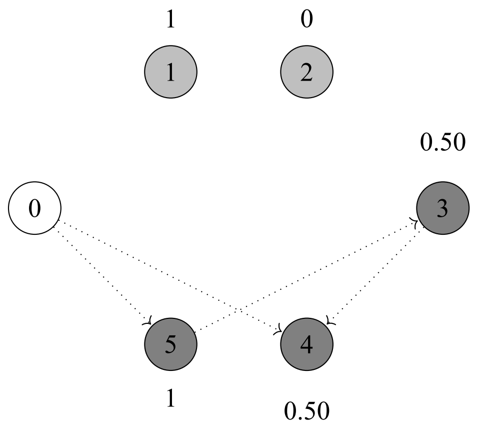
(a) Variables x.



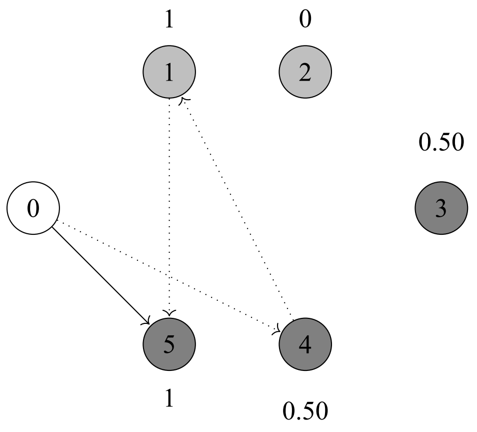
(b) Variables z^1.



(c) Variables z^3.



(d) Variables z^4.



(e) Variables z^5.

Figure 4.7: Feasible solution for the LP relaxation of the NCF$^+$ model.

The set $S = \{2, 3, 4\}$ is in the conditions of the RV inequalities since $|S| = 3 \geq |N| - V + 1 = 5 - 3 + 1 = 3$. However, considering the solution presented in Figure 4.7, $x(S', S) = 0.50$ $(= x_{04} + x_{53}) < 1$. Therefore, there is an RV inequality that is not satisfied by the feasible solution of the LP relaxation of the NCF$^+$ model. Moreover, due to the results presented in Propositions 14 and 16, we can construct, based on the former, a feasible solution of the LP relaxation of the SCF and the FCF models by using the solution presented in Figure 4.7 and the relationships between the f and the t variables (4.18) and between the t and the z variables (4.24), and, based on the latter, we know that a solution that is feasible for the LP relaxation of the NCF$^+$ model is also feasible for the LP relaxation of the NCF model. Consequently, we can construct feasible solutions for the LP relaxation of the several compact models that violate an RV inequality. Thus, considering also the conclusions drawn when we analyzed Figure 4.6, namely that there is a feasible solution for the LP relaxation of the RV model which is unfeasible for the LP relaxation of any compact model, we can deduce that the LP relaxation of the RV model is not comparable to the LP relaxation of the compact models. Proposition 17 summarizes these findings.

Proposition 17. *The LP relaxation of the RV model is not comparable to the LP relaxation of the SCF model, to the LP relaxation of the FCF model, to the LP relaxation of the NCF model and to the LP relaxation of the NCF$^+$ model.*

Observe that one might think that the non-dominance relationship between the LP relaxation of the RV model and the LP relaxation of the SCF model is not intuitive. In fact, in Section 4.2.2.2 we proved that the projection of the constraints (4.9)-(4.12) from the SCF model onto the space of the x and the y variables is defined as constraints $x(S', S) \geq \frac{y(S)}{V}$, $\forall S \subseteq N$ (4.29), and, that the RV inequalities dominate constraints (4.29). However, the RV inequalities are only valid inequalities for the FTSP when $|S| \geq |N| - V + 1$, while constraints (4.29) are valid for any subset of nodes S, thus, the LP relaxation of the models RV and SCF being not comparable. In order to construct a formulation with the RV inequalities that is a better formulation than the LP relaxation of the SCF model, we could add the inequalities (4.29) when $|S| \leq |N| - V$.

We can also show that the RFV model is not comparable to the compact models. In order to prove so, we will present a feasible solution for the LP relaxation of the RFV model which is unfeasible for the LP relaxation of the several compact models and use the solution presented in Figure 4.7 to show the reverse.

We start by considering the feasible solution for the NCF$^+$ model shown in Figure 4.7. We have already seen that: (i) using that solution we can construct a feasible solution for the LP relaxation of the SCF and the FCF models, based on Proposition 14; and (ii) the referred solution is feasible for the LP relaxation of the NCF model, due to Proposition 16. Recall the set $S = \{3, 4, 5\}$ defined previously. As we are interested in finding violated RFV inequalities, consider $S \cap F_2 = \{3, 4, 5\}$. Since $|S \cap F_2| = 3 \geq n_2 - v_2 + 1 = 3 - 2 + 1 = 2$, the set S is in the conditions of the RFV inequalities which implies that there is a violated RFV inequality since we verified previously that $x(S', S) = 0.50$. More precisely, we found a feasible solution for the LP relaxation of the compact models which is unfeasible for the LP relaxation of the RFV model.

All there is left now is to find a feasible solution for the LP relaxation of the RFV model which is unfeasible for the LP relaxation of the compact models. In order to do so, consider a new FTSP instance with two families. Family 1, which is represented by the light gray color, has two family members (nodes 1 and 2) while family 2, which is represented by the dark gray color, has five family members (nodes 3-7). Additionally, consider that we are required to visit one node from family 1 and two nodes from family 2. Figure 4.8 shows a feasible solution for the LP relaxation of the RFV model considering the FTSP instance described previously. The filled arcs correspond to the ones in which the x variables have value 0.50 and the dashed ones correspond to the x values equal to 0.25. The values on the top or at the bottom of each node correspond to the value of the y variable associated with that node.

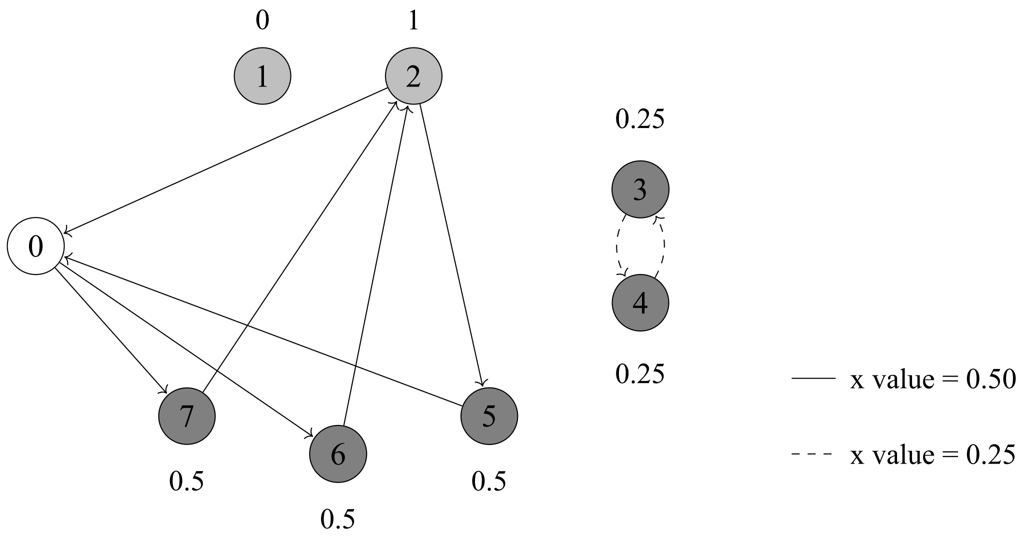


Figure 4.8: Feasible solution for the LP relaxation of the RFV model.

By using the same argument that we used when we analyzed Figure 4.6, namely that the compact models ensure that there exists paths, which may have fractional value, that have as initial node the depot and go through the visited nodes, we verify that the feasible solution for the LP relaxation of the RFV model presented in Figure 4.8 is not feasible for the LP relaxation of any of the compact models. Therefore, the LP relaxation of the RFV model is not comparable to the LP relaxation of the compact models, which is stated in Proposition 18.

Proposition 18. *The LP relaxation of the RFV model is not comparable to the LP relaxation of the SCF mode, to the LP relaxation of the FCF model, to the LP relaxation of the NCF model and to the LP relaxation of the NCF$^+$ model.*

The reason why the RFV model and the FCF model are not comparable is similar to the reason why the LP relaxation of the SCF model and the LP relaxation of the RV model are also not comparable. Also similarly, a formulation with the RFV inequalities that is a better formulation than the LP relaxation of the FCF model could be obtained by adding constraints $x(S', S) \geq \frac{y(S \cap F_l)}{v_l}$ (4.33) when $|S \cap F_l| \leq n_l - v_l,\ \forall l \in \mathcal{L}$ to the RFV model.

Finally, we must relate the LP relaxation of the CC model to the LP relaxation of the compact models. We start by relating the LP relaxation of the CC model to the LP relaxation of the NCF model as this relationship will help us to establish all the other ones.

Proposition 19. $\mathcal{V}^{LP}(NCF) = \mathcal{V}^{LP}(CC)$.

Proof. Consider the following polytopes:

- $P_{NCF} = \{(x, y, z) \in \mathbb{R}^{|A|+|N|+|N|\times|A|}$: satisfy the equation system $(4.2) - (4.5)$, $(4.19) - (4.21), (4.23), (4.25), (4.36)$ and $(4.37)\}$
- $P_{CC} = \{(x, y) \in \mathbb{R}^{|A|+|N|}$: satisfy the equation system $(4.2) - (4.5), (4.28), (4.36)$ and $(4.37)\}$

Proving the result stated in Proposition 19 is equivalent to proving that $proj_{x,y}P_{NCF} = P_{CC}$. More precisely, we need to prove that constraints (4.19)-(4.21), (4.23) and (4.25) from the NCF model projected onto the subspace of the x and the y variables are the CC inequalities (4.28).

Constraints (4.19)-(4.21) from the NCF model may be seen as $|N|$ different flows from the depot to each node $k \in N$. Consider a particular node $k \in N$. We wish to send y_k units of flow from the depot to node k in a capacitated network in which the lower and the upper capacities of the arc $(i, j) \in A$ are 0 and x_{ij}, respectively, which is stated in constraints (4.23) and (4.25). From the max-flow/min-cut theorem (Theorem 3), presented in Section 2.5.3, we know that y_k units of flow are sent from the depot to node k if and only if every cut-set separating the depot from k has a capacity of at least y_k, which is mathematically expressed by the CC inequalities (4.28) of the CC model. □

By using the result of the previous proposition and the one of Proposition 15, which states that the LP relaxation of the NCF model is not comparable to the LP relaxation of the SCF and the LP relaxation of the FCF models, we can deduce Proposition 20.

Proposition 20. *The LP relaxation of the CC model is not comparable to the LP relaxation of the SCF model and to the LP relaxation of the FCF model.*

Additionally, from Proposition 19 and from the fact that the LP relaxation of the NCF$^+$ model is a better formulation than the LP relaxation of the NCF model, which is stated in Proposition 16, we can conclude that the LP relaxation of the NCF$^+$ model is a better formulation than the LP relaxation of the CC model, which is stated in Proposition 21.

Proposition 21. $\mathcal{V}^{LP}(CC) \leq \mathcal{V}^{LP}(NCF^+)$.

4.3.3 Comparing the non-compact models

In this section we establish the relationships between the non-compact models, which include the CC model presented in Section 4.2.2.1, the RV model presented in Section 4.2.2.2 and the RFV model presented in Section 4.2.2.3.

Before comparing the RV and the RFV models, we define the sets:

$$\begin{aligned}\mathbb{S}^V &= \{S \subseteq N : |S| \geq |N| - V + 1\} \\ \mathbb{S}^{FV} &= \{S \subseteq N : \exists l \in \mathcal{L} : |S \cap F_l| \geq n_l - v_l + 1\},\end{aligned}$$

that is, intuitively, the set $\mathbb{S}^V$ is the set of sets $S \subseteq N$ in which the RV inequalities are defined while the set $\mathbb{S}^{FV}$ is the set of sets $S \subseteq N$ in which the RFV inequalities are defined.

Proposition 22. $\mathcal{V}^{LP}(RV) \leq \mathcal{V}^{LP}(RFV)$.

Proof. Proving that $\mathcal{V}^{LP}(RV) \leq \mathcal{V}^{LP}(RFV)$ is equivalent to proving that $\mathbb{S}^V \subseteq \mathbb{S}^{FV}$. Consider $S \in \mathbb{S}^V$. Suppose that $S \notin \mathbb{S}^{FV}$, that is, $\forall l \in \mathcal{L},\ |S \cap F_l| < n_l - v_l + 1 \iff |S \cap F_l| \leq n_l - v_l$. Since $\cup_{l \in \mathcal{L}} F_l$ is a partition of the set N, we have:

$$|S| = \sum_{l \in \mathcal{L}} |S \cap F_l| \leq \sum_{l \in \mathcal{L}} (n_l - v_l) = \sum_{l \in \mathcal{L}} n_l - \sum_{l \in \mathcal{L}} v_l = |N| - V,$$

which is a contradiction since $S \in \mathbb{S}^V$. □

An example of a set $S \in \mathbb{S}^{FV}$ that does not belong to $\mathbb{S}^V$ is the one presented in Figure 4.3, where we presented the motivation for the RFV model. Additionally, a consequence of Proposition 22 is that the RV inequalities are a particular case of the RFV inequalities. Nevertheless, the RV inequalities may be applied to routing problems in which the feasible solutions are single elementary circuits with a fixed number of nodes that are not Hamiltonian, while the RFV inequalities are specific for the FTSP.

We already verified that the LP relaxation of the NCF model is not comparable to the LP relaxation of the RV model and to the LP relaxation of the RFV model in Propositions 17 and 18, and since, according to Proposition 19, the CC model is the projection of the NCF model onto the space of the x and the y variables, we can deduce the following proposition

Proposition 23. *The LP relaxation of the CC model is not comparable to the LP relaxation of the RV model and to the LP relaxation of the RFV model.*

Note that it is obvious that when a set S belongs to $\mathbb{S}^{FV}$, the RFV inequalities dominate the CC inequalities, however, there are many subsets of nodes that do not belong to $\mathbb{S}^{FV}$.

Figure 4.9 shows a summary of the results stated throughout Section 4.3.

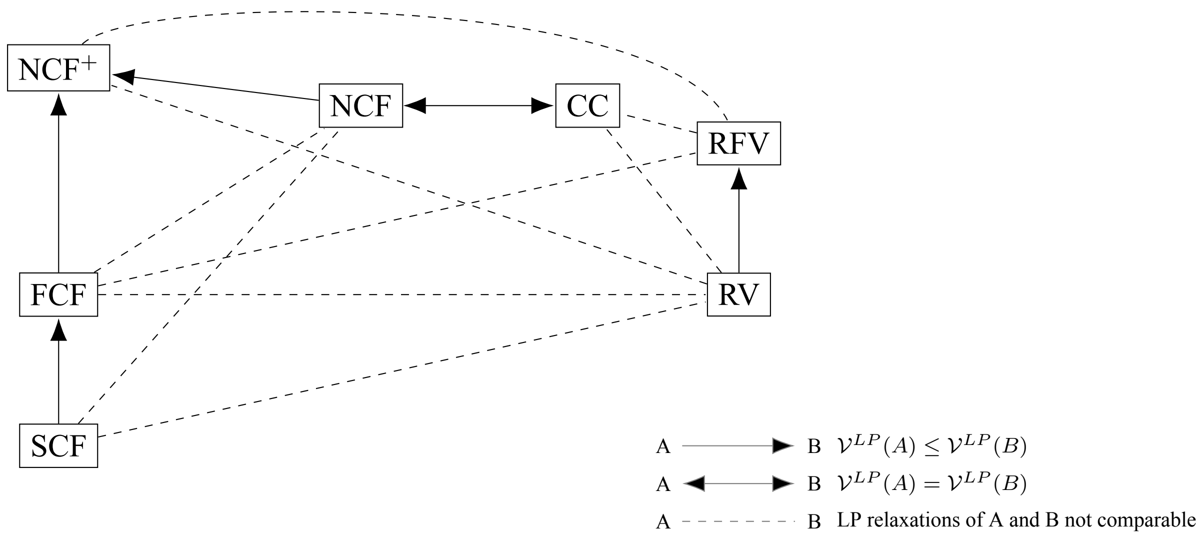

NCF$^+$
NCF
CC
RFV
FCF
RV
SCF
A B $\mathcal{V}^{LP}(A) \leq \mathcal{V}^{LP}(B)$
A B $\mathcal{V}^{LP}(A) = \mathcal{V}^{LP}(B)$
A B LP relaxations of A and B not comparable

Figure 4.9: Known relationships between the proposed formulations.

4.4 Empirical comparison of the several formulations

The purpose of this section is to establish which are the best models in practice. The evaluation of the different formulations takes into account two factors: (i) the LP relaxation value, and (ii) the computational efficiency, that is, the time taken to obtain the LP relaxation value. As we only wish to choose the best models, we only present computational results for a subset of small dimensioned benchmark instances, with a number of nodes between 14 and 48. A thorough computational study will be carried out further on in this dissertation, more specifically, in Chapter 5, where we present a branch-and-cut algorithm for the FTSP. Additionally, we also present the setting with which these results were obtained in Chapter 5, Section 5.4.

Tables 4.2 and 4.3 show the LP relaxation values obtained using the compact models and the non-compact ones, respectively. The referred tables are divided into several parts, each one dedicated to a different model. Each of those parts has three columns, one with the LP relaxation value ($\mathcal{V}^{LP}$), another with the percentage of gap between the LP relaxation value and the optimal value ($gap = 100\times$ *(optimal value - LP relaxation value)/optimal value*) and the final one with the time, in seconds, to obtain the LP relaxation value (t_s). The tables also contain, in the last row, the average of the results obtained.

To simplify the notation, the instance that is presented in Table A.1 as $tspinstancename|N| + 1\_L\_1001\_100i\_2$, with $i \in \{1, 2, 3\}$, will be designated by $tspinstancename\_i$.

Table 4.2: Linear programming relaxation results obtained with the compact models.

	SCF			FCF			NCF			NCF$^+$		
Instance	$\mathcal{V}^{LP}$	gap	t_s	$\mathcal{V}^{LP}$	gap	t_s	$\mathcal{V}^{LP}$	gap	t_s	$\mathcal{V}^{LP}$	gap	t_s
burma_1	10.00	28.19%	0	11.36	18.46%	0	12.07	13.34%	0	12.72	8.70%	0
burma_2	23.25	9.38%	0	24.33	5.18%	0	25.66	0.00%	0	25.66	0.00%	0
burma_3	7.92	33.40%	0	10.22	13.98%	0	9.93	16.47%	0	10.51	11.57%	0
bayg_1	4767.12	10.83%	0	4982.57	6.80%	0	5273.32	1.36%	2	5273.32	1.36%	6
bayg_2	4837.19	16.47%	0	5100.93	11.92%	0	5754.64	0.63%	3	5754.64	0.63%	16
bayg_3	4945.35	10.80%	0	5127.44	7.52%	0	5544.33	0.00%	1	5544.33	0.00%	9
att_1	18224.40	23.06%	1	18957.50	19.96%	1	23686.00	0.00%	53	23686.00	0.00%	447
att_2	14288.90	30.67%	0	14862.20	27.89%	2	20609.10	0.00%	73	20609.10	0.00%	2519
att_3	7262.57	19.52%	0	7925.97	12.17%	2	8742.08	3.13%	40	9017.84	0.07%	1307
average		20.26%	0		13.76%	1		3.88%	19		2.48%	478

Table 4.3: Linear programming relaxation results obtained with the non-compact models.

	CC			RV			RFV		
Instance	$\mathcal{V}^{LP}$	gap	t_s	$\mathcal{V}^{LP}$	gap	t_s	$\mathcal{V}^{LP}$	gap	t_s
burma_1	12.07	13.34%	0	12.57	9.79%	0	13.93	0.00%	0
burma_2	25.66	0.00%	0	25.02	2.50%	0	25.66	0.00%	0
burma_3	9.93	16.47%	0	8.39	29.44%	1	11.89	0.00%	0
bayg_1	5273.32	1.36%	0	4937.39	7.64%	10466	5316.85	0.54%	0
bayg_2	5754.64	0.63%	0	5349.98	7.62%	8575	5791.01	0.00%	0
bayg_3	5544.33	0.00%	0	5269.57	4.69%	6577	5544.33	0.00%	0
att_1	23686.00	0.00%	0	-	-	-	23580.50	0.45%	1
att_2	20609.10	0.00%	1	-	-	-	20609.10	0.00%	6
att_3	8742.08	3.13%	1	-	-	-	8760.03	2.93%	2
$average$		3.88%	0		10.32%	5087		0.44%	1

All the computational results obtained are consistent with the theoretical results stated in Section 4.3. By observing Table 4.2 we verify that, in general, the LP relaxation value obtained using the SCF model is the lowest followed by the one obtained with the FCF model and the highest LP relaxation value was obtained with the NCF$^+$ model. The computational time increases with the quality of the LP relaxation value. For instances att_3, the SCF model takes 0 seconds to provide the LP relaxation value with a percentage of gap of 19.52%, whilst the NCF$^+$ model finds the optimal value while solving the LP relaxation, but it takes 1307 seconds. According to Proposition 15, the NCF model is not comparable to either the SCF or the FCF models, nonetheless it provides a much lower average gap. More precisely, the LP relaxation values obtained with the NCF model are always better than the ones obtained with the SCF model and with the FCF model for the instances tested, except for instance $burma\_2$. Regarding the NCF and the NCF$^+$ models, the NCF$^+$ provides a better LP relaxation value than the NCF model in three instances but its computational time is significantly higher. By observing instance att_2 we verify that both models obtain the optimal value when solving the LP relaxation but the NCF$^+$ model takes 2519 seconds while the NCF model takes 73 seconds.

Consider now Table 4.3. As it was expected, due to Proposition 19, the LP values obtained with the NCF and the CC models are the same. However, if we focus on the computational time we verify that the CC model is much more efficient with an average time of 0 seconds compared to an average time of 19 seconds. For this reason, it is preferable to use the CC model in practice instead of the NCF model. Since the CC model is able to obtain significantly better LP relaxation values than the SCF model and the FCF model in a similar computational time, then it is also preferable to use the CC model instead of the SCF and the FCF models.

Due to the computational time needed to obtain the LP relaxation values for instances $bayg$ using the RV model, we did not compute the LP relaxation values of instances att, which have a bigger dimension. Even though the RV model is not comparable to the SCF model, it was able to obtain a higher LP relaxation value in every instance tested. Regarding the computational time, the RV model is much more time consuming than the SCF model as its average computational time is of 5087 seconds, whereas the SCF model, as mentioned before, obtains the LP relaxation value in an average of 0 seconds. The RV model and the FCF model are not comparable, however, the FCF model only outperforms the RV model, in terms of quality of the LP relaxation value, in two instances, namely instances $burma\_3$ and $bayg\_1$. Once again, in terms of computational time the FCF model is more efficient. When we consider only the non-compact models, the RV model is the worst non-compact model both in terms of average LP relaxation value and computational time. Even though it is not comparable to the CC model, the RV model only provides a better LP relaxation value than the CC model in one instance, namely instance $burma\_1$, and it is always significantly worse than the RFV model. From the previous arguments, we can conclude that there is no advantage in using the RV model instead of the other non-compact models.

In spite of the fact that the only theoretical dominance that we were able to establish with the RFV model was over the RV model, the RFV model was the model that provided the lowest average gap. More precisely, the average gap obtained with the RFV model was 0.44% while the second lowest gap obtained was 2.48%, which was obtained with the NCF$^+$ model. The RFV model outperforms the SCF, the FCF and the RV models, in terms of quality of the LP relaxation value, in every instance. Regarding the computational time, the RFV model is competitive with the fastest models that take an average of 0 seconds. If we compare the RFV model to the CC model, we verify that the CC model only provides a better LP relaxation value in one instance, while the RFV model is strictly better in five. Regarding the computational time, it is similar for both models. The conclusions when we compare the RFV model to the NCF$^+$ model are similar, except for the fact that the NCF$^+$ is much more time consuming as it took, on average, 478 seconds to obtain the LP relaxation values, while the RFV model took an average of 1 second. Since the RFV model provides a lower average gap than the NCF$^+$ model in a more efficient fashion we decided not to pursue the study of the NCF$^+$ model any further.

Recall that, as we saw in Section 4.2.2.3, the RFV inequalities $x(S', S) \geq 1$ (4.35) are only defined if $S \in \mathbb{S}^{FV}$, and, in that case, they dominate the CC inequalities $x(S', S) \geq y_k$ (4.28), however, the CC inequalities are valid for every set $S \subseteq N$ and, thus, they may always be applied.

Therefore, we decided to combine both sets of inequalities, which is shown in Section 4.4.1.

4.4.1 Combining the CC inequalities and the RFV inequalities

Since both the CC inequalities and the RFV inequalities may be used as subtour elimination constraints, as we showed in Sections 4.2.2.1 and 4.2.2.3, respectively, we can create a formulation for the FTSP in which the CC inequalities replace the generic subtour elimination constraints (4.6) as the subtour elimination constraints and the RFV inequalities are valid inequalities added to improve the quality of the LP relaxation value, or vice-versa. We denote this new formulation by the CC+RFV model. From Proposition 23, we know that the CC model and the RFV model are not comparable. Therefore, the LP relaxation of the CC+RFV model will be greater than or equal to the LP relaxation of both the CC model and the RFV model.

Table 4.4 shows the LP relaxation values obtained with the CC+RFV model. The referred table contains the LP relaxation value obtained with the CC+RFV model ($\mathcal{V}^{LP}$), the percentage of gap between the LP relaxation value and the optimal value (gap), the computational time, in seconds, to obtain the LP relaxation value (t_s) and the average of the results obtained, in the last row.

Table 4.4: Linear programming relaxation results obtained with the CC+RFV model.

Instance	$\mathcal{V}^{LP}$	gap	t_s
burma_1	13.93	0.00%	0
burma_2	25.66	0.00%	1
burma_3	11.89	0.00%	0
bayg_1	5330.17	0.29%	0
bayg_2	5791.01	0.00%	1
bayg_3	5544.33	0.00%	0
att_1	23686.00	0.00%	1
att_2	20609.10	0.00%	1
att_3	9024.58	0.00%	1
$average$		0.03%	1

With the CC+RFV model we were able to obtain the optimal value of all the instances tested when solving the LP relaxation, except for instance $bayg\_1$. The CC+RFV model was able to obtain significantly better results than the CC and the RFV models independently, which makes the CC+RFV model a promising formulation to pursue. Regarding the computational time, it is of the same magnitude as the computational time of the CC and the RFV models. Nevertheless, when addressing instances of higher dimension, the fact that we are adding two sets of constraints that are in exponential number may slow down the resolution process. Therefore, we continue to consider both the CC model and the RFV model.

Summary

Throughout this chapter we presented seven different formulations for the FTSP that can be divided into compact formulations and non-compact formulations. There are four compact formulations: the SCF model, presented in Section 4.2.1.1; the FCF model, presented in Section 4.2.1.2; the NCF model and the NCF$^+$ model, both presented in Section 4.2.1.3. We were able to establish that, in what concerns the LP relaxation, the NCF$^+$ model is a better formulation than all the other compact models and that the NCF model is not comparable to the SCF and the FCF models. Despite the LP relaxation of the NCF model not being comparable to the LP relaxation of the SCF model and to the LP relaxation of the FCF model, in the empirical experiment presented in Section 4.4, it provided a much lower average gap. Regarding the non-compact models, we developed three distinct models, namely the CC model, presented in Section 4.2.2.1, the RV model, presented in Section 4.2.2.2 and the RFV model, presented in Section 4.2.2.3. From the theoretical comparison of Section 4.3, we verified that the LP relaxation of the CC model is not comparable to the LP relaxation of the RV model and to the LP relaxation of the RFV model and that the RV inequalities are a particular case of the RFV inequalities. We also verified that the LP relaxation of the compact models is not comparable to the LP relaxation of the non-compact models, except for the LP relaxations of the NCF and the CC models, in which the latter is the projection of the former onto the subspace of the x and the y variables. Even though the LP relaxation of the RFV model is not comparable to the LP relaxation of the majority of the proposed models, it provided the lowest average gap. Therefore, we believe that the most promising formulations to use in practice are the CC model and the RFV model.

As the LP relaxations of the CC model and the RFV model are not comparable, we created a new formulation in which one set of constraints is added as subtour elimination constraints and the other is added as valid inequalities, which originated the CC+RFV model. The practical results obtained with the CC+RFV model were very promising since this model was able to obtain near-optimal solutions when solving the LP relaxation. More precisely, the CC+RFV model obtained an average gap of 0.03% on the tested instances.

Even though the compact models proved to be less effective than the more promising models, which are non-compact, in terms of LP relaxation value, they have characteristics that justify its usage. As mentioned previously, in order to use the non-compact models we have to resort to a branch-and-cut algorithm, while the compact models may be used with a standard ILP solver.

Regarding the RV inequalities, the results show that they were not efficient in practice for the FTSP instances tested. More precisely, the RV model is significantly more time consuming than all the other proposed models. Nonetheless, these inequalities may be used as valid inequalities to improve the LP relaxation value of any routing problem that has as feasible solutions single elementary circuits with a fixed number of nodes and that are not Hamiltonian.

In Chapter 5 we present the branch-and-cut algorithm which was designed to be applied to the CC model, the RFV model and the CC+RFV model, and, we also present a thorough computational study in which we use the most efficient model, in terms of computational time, to obtain the optimal value of the test instances presented in Section 3.4.

Chapter 5

The Branch-and-Cut Algorithm

The branch-and-cut (B&C) algorithm was introduced by Padberg and Rinaldi (1987, 1991) for the TSP and since then it has been applied to a large number of integer linear programming problems and, for many of them, the B&C algorithm is the state-of-the-art. In particular, B&C algorithms are the state-of-the-art when solving large dimensioned instances of routing problems, mostly because of the non-compact formulations for the subtour elimination constraints.

The purpose of this chapter is to present the B&C algorithm designed to solve the CC model, presented in Section 4.2.2.1, the RFV model, presented in Section 4.2.2.3, and the CC+RFV model, introduced in Section 4.4. In Section 5.1 we present the general outline of the B&C algorithm as well as some concepts related to the solver that we will use. In Section 5.2 we present the separation algorithms for the CC and the RFV inequalities. A heuristic procedure to provide upper bounds during the B&C algorithm is presented in Section 5.3 and we conclude, in Section 5.4, with the computational experiment.

5.1 The branch-and-cut algorithm outline

Since the basic framework of the B&C algorithm was already presented in Section 2.3, we will only present the polytopes that we consider when solving the non-compact models. The B&C algorithm was designed to solve the CC model and the RFV model. As the algorithm is similar for both models we consider the particular case of the CC model for this explanation. Let $P_{CC} = \{(x, y) \in \mathbb{R}^{|A|+|N|}$: satisfy the equation system (4.2)-(4.5), (4.28), (4.36) and (4.37)$\}$ be the polytope associated with the CC model. By using the B&C algorithm we wish to solve the following ILP problem:

$$\min\{cx : (x, y) \in P_{CC} \cap \{0, 1\}^{|A|+|N|}\}.$$

As the CC inequalities (4.28) are in exponential number, we start the B&C procedure considering the polytope without them, that is, $P^0_{CC} = \{(x, y) \in \mathbb{R}^{|A|+|N|}$: satisfy the equation system (4.2) $-$ (4.5), (4.36) and (4.37)$\}$, and then the CC inequalities are added by using the cutting plane algorithm described in Algorithm 2.1 until we find a solution that belongs to the polytope P_{CC}.

Nowadays, commercial solvers provide general purpose B&C algorithms, which use general purpose cuts and heuristics, to solve ILP problems and that can be tailored by the user. In order to implement the B&C algorithm described previously, we will use the built-in B&C algorithm provided by the commercial solver CPLEX (see, e.g., IBM, 2014). More precisely, we use CPLEX functions, called callback functions, which allow us to provide specific algorithms for the FTSP to be incorporated in the built-in B&C algorithm. The callbacks used are:

- the user cut callback, which is used to implement separation algorithms for fractional solutions;
- the lazy constraint callback, which is used to implement separation algorithms for integer solutions; and
- the heuristic callback, which is used to implement heuristic algorithms to provide feasible FTSP solutions during the B&C algorithm in an attempt to improve the upper bounds obtained and, hopefully, help to decrease the number of B&C subproblems to solve.

5.2 The separation algorithms

In Chapter 4 we proposed several sets of inequalities that prevent subtours, namely the CC inequalities in Section 4.2.2.1 and the RFV inequalities in Section 4.2.2.3. Throughout this section we present the separation algorithms for the CC and the RFV inequalities. Moreover, for each set of inequalities we propose heuristic separation algorithms that accelerate the separation process along with a summary of computational results that support this statement. More precisely, in Section 5.2.1 we present the separation algorithms for the CC inequalities, in Section 5.2.2 we present the separation algorithms for the RFV inequalities and, finally, in Section 5.2.2 we present a separation algorithm that separates, simultaneously, the CC and the RFV inequalities.

5.2.1 Separating the CC inequalities

Given a fractional solution (x^*, y^*) that satisfies the equation system (4.2)-(4.5) presented in Section 4.1 we must verify if it also satisfies the CC inequalities. Satisfying the CC inequalities is equivalent to verifying if the optimal solution of the LP problem

$$\omega = min_{S \subseteq N}\{x(S', S) - y_k : k \in S\}$$

is greater than or equal to zero. If $\omega \geq 0$, then the solution (x^*, y^*) satisfies all the CC inequalities and, consequently, it is feasible for the LP relaxation of the CC model. However, if $\omega < 0$ there is a subset S and a node $k \in S$ such that $x(S', S) < y_k$.

Consider a capacitated graph $G^* = (0 \cup N, A)$ where the capacities of the arc $(i, j) \in A$ are $l_{ij} = 0$ and $u_{ij} = x^*_{ij}$. For a node $k \in N$ we must verify if $\omega \geq 0$. From the Max-flow/Min-cut Theorem 3 of Section 2.5.3, we know that the value of the max-flow between the depot and k is equal to the value of the min-cut that separates the depot from k. Therefore, if the value of the maximum flow between the depot and k is greater than or equal to y^*_k, the minimum cut also has a value greater than or equal to y^*_k. Consequently, all cuts that separate the depot from k have a value greater than or equal to y^*_k and there is no violated CC inequality. Thus, if we compute the maximum flow from the depot to every node $k \in N$ such that $y^*_k > 0$ it is possible to determine whether all the CC inequalities are satisfied or not and, if not, it is possible to find a violated CC inequality. Algorithm 5.1 shows the pseudocode for the separation algorithm for the CC inequalities when (x^*, y^*) is a fractional solution.

Algorithm 5.1 Separation algorithm for the CC inequalities.

```
Require: A fractional solution $(x^*, y^*)$ that satisfies the equation system (4.2)-(4.5).
 1: Create a new complete graph $G^* = (0 \cup N, A)$.
 2: for all $(i, j) \in A$ do
 3:     Set its capacities to $l_{ij} = 0$ and $u_{ij} = x^*_{ij}$.
 4: end for
 5: for all $k \in N$ such that $y^*_k > 0$ do
 6:     Determine the max-flow between $0$ and $k$ in $G^*$. Let $\nu$ be the value of the max-flow.
 7:     if $\nu < y^*_k$ then
 8:         Determine the sets $S$ and $S' = (N \cup 0) \setminus S$ such that $S'$ is the set of nodes reachable from
            node $0$ in the residual network associated with the max-flow.
 9:         Add the violated inequality $x(S', S) \geq y_k$.
10:     end if
11: end for
```

The Separation Algorithm 5.1 may be used to separate the CC inequalities and, consequently, ensures that the solution obtained is feasible for the FTSP. However, there are several improvements that can be done in order to achieve a more efficient separation algorithm. When we use the separation algorithm presented previously, we usually find more than one violated inequality to be added. On the one hand, if we add too many inequalities to the LP problem, the LP problem becomes very time consuming to solve. On the other hand, if we add too few inequalities the separation algorithm has to be performed more times. Thus, we set a limit to the number of violated inequalities added before we re-solve the B&C subproblem. More precisely, the separation algorithm stops either when it computes the maximum flow from the depot to every node k such that $y_k^* > 0$ or when the maximum number of added violated inequalities is reached. Due to the limit in the number of added violated inequalities, instead of computing the maximum flow from the depot to each node using the lexicographic ordering of the nodes in Step 5, we use a permutation of the nodes generated randomly every time we execute the separation algorithm. This is done to avoid computing the maximum flow from the depot to the same target nodes. Let max_number_cuts be the maximum number of violated cuts added before re-solving the linear programming relaxation and $\pi(N)$ be a permutation of the set N. The pseudocode presented in Algorithm 5.2 corresponds to the improved separation algorithm for the CC inequalities used in the B&C algorithm.

Algorithm 5.2 Improved separation algorithm for the CC inequalities.

```
Require: A fractional solution $(x^*, y^*)$ that satisfies the equation system (4.2)-(4.5).
 1: Create a new complete graph $G^* = (0 \cup N, A)$.
 2: for all $(i, j) \in A$ do
 3:     Set its capacities to $l_{ij} = 0$ and $u_{ij} = x_{ij}^*$.
 4: end for
 5: Set $number\_added\_cuts = 0$ and compute $\pi(N)$.
 6: for all $k \in \pi(N)$ such that $y_k^* > 0$ do
 7:     Determine the max-flow between $0$ and $k$ in $G^*$. Let $\nu$ be the value of the max-flow.
 8:     if $\nu < y_k^*$ then
 9:         Determine the sets $S$ and $S' = (N \cup 0) \setminus S$ such that $S'$ is the set of nodes reachable from
            node $0$ in the residual network associated with the max-flow.
10:         Add the violated inequality $x(S', S) \geq y_k$.
11:         $number\_added\_cuts = number\_added\_cuts + 1$.
12:         if $number\_added\_cuts \geq max\_number\_cuts$ then
13:             BREAK.
14:         end if
15:     end if
16: end for
```

Heuristic separation of the CC inequalities

In order to accelerate the separation of the CC inequalities, in the case of fractional solutions, we developed a heuristic separation based on a standard heuristic separation for the TSP (see, e.g., Grötschel and Holland, 1991; Fischetti et al., 1997). By using this heuristic separation we were able to obtain a significant decrease in the computational time needed to obtain the LP relaxation value of the CC model.

The idea behind this heuristic separation is to add CC inequalities that are violated by the connected components induced by the fractional solution (x^*, y^*). We consider that nodes i and j belong to the same connected component if $x^*_{ij} > 0$. Note that, according to this definition, the value of the variables x associated with arcs which have the initial and the final nodes in different components is zero. Let $\{C_0, C_1, \ldots, C_p\}$ be the set of connected components associated with solution (x^*, y^*) and $\overline{S} = \{i \in N : y^*_i = 0\}$ be the set of non-visited nodes in the same solution. We assume that the connected component C_0 is the one that contains the depot (node 0). With these $p + 1$ components, we will add a maximum of $p + 1$ violated inequalities determined heuristically. Recall that the set S' always includes the depot, therefore, in the first violated inequality, we define $S' = C_0$ and $S = C_1 \cup \ldots \cup C_p \cup \overline{S}$ and for the remaining p violated inequalities sets S' and S are defined as follows: for the kth-inequality $S = C_k$ and $S' = (0 \cup N) \setminus S$, $k = 1, \ldots, p$. Notice that for every S' and S defined previously: (i) $0 \in S'$, (ii) set S contains visited nodes (i.e., nodes k with $y^*_k > 0$), and (iii) $x^*(S', S) = 0$, therefore all these sets originate violated CC inequalities. All there is left now is to choose the right-hand side of these constraints. As $x^*(S', S) = 0$, choosing for the right-hand side any node $i \in S$ such that $y^*_i > 0$ originates a violated CC inequality. However, we decided to choose the node $k \in S$ that maximizes the difference $x^*(S', S) - y^*_k$, which is equivalent to choosing the node with the highest y^* value. The pseudocode for the heuristic separation of the CC inequalities is presented in Algorithm 5.3. Once again, we decided to put a limit to the number of added violated inequalities. Therefore, the heuristic separation algorithm stops either when it has separated all the CC inequalities induced by the connected components or when the maximum number of added cuts was reached.

In the user cut callback for the CC inequalities, we start by applying the heuristic separation and in the end, when the heuristic procedure is not able to provide new violated inequalities, we use the exact separation algorithm, presented in Algorithm 5.2, either to find more violated CC inequalities or to conclude that there are no more violated CC inequalities. With this procedure we ensure that the solution obtained is the optimal solution for LP relaxation of the CC model.

Table 5.1 shows the impact of using the heuristic separation in terms of computational time. In Table 5.1 it is possible to see the average gap obtained ($\overline{gap}$), the average computational time,

Algorithm 5.3 Heuristic separation of the CC inequalities.

```
Require: A fractional solution $(x^*, y^*)$ that satisfies the equation system (4.2)-(4.5).
1: Determine the connected components $\{C_0, \ldots, C_p\}$ and the set of non-visited nodes $\overline{S}$ induced
   by the fractional solution.
2: $iter = 0$.
3: while $iter < p + 1$ do
4:     if $iter = 0$ then
5:         Set $S' = C_0$ and $S = C_1 \cup C_2 \cup \ldots \cup C_p \cup \overline{S}$.
6:     else
7:         Set $S = C_{iter}$ and $S' = (N \cup 0) \setminus S$.
8:     end if
9:     Determine $k \in S$ such that $y^*_k \geq y^*_i, \forall i \in S$.
10:    Add the violated inequality $x(S', S) \geq y_k$.
11:    $iter = iter + 1$.
12: end while
```

in seconds, to obtain the LP relaxation value ($\overline{t_s}$) and the average number of added violated CC inequalities ($\overline{\#CC}$) when we use the heuristic separation of the CC inequalities or when we only use the exact one. The detailed results obtained performing only the exact separation are available in appendix, Table B.1, and the ones obtained with the heuristic separation are in Section 5.4.

Table 5.1: Heuristic separation CC inequalities.

	$\overline{gap}$	$\overline{t_s}$	$\overline{\#CC}$
Without heuristic separation	3.17%	66	1407
With heuristic separation	3.17%	6	591

Average obtained with benchmark instances' $burma$, $bayg$, att and $bier$.

Table 5.1 shows that, by using the heuristic separation, the computational time decreases by a factor of ten. This reduction in the computational time is implied by the reduction in the number of added violated inequalities.

Separation algorithm for integer solutions

When the solution (x^*, y^*) is integer, that is, when it satisfies constraints (4.2)-(4.5),(4.7)-(4.8) presented in Section 4.1, we use the separation algorithm described in Algorithm 5.2 with $max\_number\_cuts =$

1, that is, we only add one violated CC inequality before re-solving the LP problem.

5.2.2 Separating the RFV inequalities

Searching for violated RFV inequalities is similar to searching for violated CC inequalities, except for the fact that there is an additional constraint in the cardinality of S. In order to obtain a violated RFV inequality, the cardinality of $S \cap F_l$, for some family $l \in \mathcal{L}$, has to be at least $n_l - v_l + 1$. Thus, if we fix $n_l - v_l + 1$ nodes from family l to S and determine the minimum cut $[S', S]$ we know that the number of nodes in S will be at least $n_l - v_l + 1$ and, consequently, $S \in \mathbb{S}^{FV}$. Hence, in order to determine all violated inequalities we need to compute, for each family $l \in \mathcal{L}$, all sets of nodes from that family with cardinality $n_l - v_l + 1$.

Let (x^*, y^*) be a fractional solution that satisfies the equation system (4.2)-(4.5) from Section 4.1 and we define

$$\mathbb{S}_l^{=} = \{S \subseteq F_l : |S| = n_l - v_l + 1\} \qquad \forall l \in \mathcal{L}.$$

Consider a capacitated graph $G^* = ((0 \cup N) \cup t, A \cup A_t)$ in which node t is a fictitious node and A_t is the set of arcs from N to t, that is, $A_t = \{(i,t) \ : \ i \in N\}$. The capacities of the arc $(i,j) \in A$ are $l_{ij} = 0$ and $u_{ij} = x^*_{ij}$ and currently we consider that the capacities of the arc $(i,t) \in A_t$ are both 0 ($l_{it} = u_{it} = 0$). The arc set A_t allows us to fix the subsets S_l of F_l with a cardinality equal to $n_l - v_l + 1$ to S by setting the upper capacities to infinity of the arcs in $(i,t) \in A_t$ with $i \in S_l$. After creating this modified graph, we just need to compute the maximum flow from the depot to t in order to separate the RFV inequalities. More precisely, if the value of the max-flow from the depot to t is greater than or equal to 1, then all the RFV inequalities are satisfied. Otherwise, there are violated RFV inequalities. The pseudocode for the separation algorithm for the RFV inequalities is available in Algorithm 5.4.

Analogously to what was done for the CC inequalities we also set a limit to the maximum number of added violated RFV inequalities per iteration. As the adaptation of the algorithm is similar to what was done for the CC inequalities, see Algorithm 5.2, we decided to omit it.

Heuristic separation of the RFV inequalities

A heuristic algorithm to accelerate the separation of the RFV inequalities when the solution (x^*, y^*) is a fractional solution can be devised by using a procedure similar to the heuristic separation algorithm for the CC inequalities. The only difference is that, after determining sets S and S', it still has to be verified whether $S \in \mathbb{S}^{FV}$ or not and we only add a violated inequality if $S \in \mathbb{S}^{FV}$. Once

Algorithm 5.4 Separation algorithm for the RFV inequalities.

```
Require: A fractional solution $(x^*, y^*)$ that satisfies the equation system (4.2)-(4.5).
 1: Create a new complete graph $G^* = (0 \cup N \cup t, A \cup A_t)$.
 2: for all $(i, j) \in A$ do
 3:     Set its capacities to $l_{ij} = 0$ and $u_{ij} = x^*_{ij}$.
 4: end for
 5: for all $l \in \mathcal{L}$ do
 6:     Compute all the sets in $\mathbb{S}^=_l$ (composed of subsets of $F_l$).
 7:     while There are sets $S_l \in \mathbb{S}^=_l$ to consider do
 8:         for all $(i, t) \in A_t$ do
 9:             Set its capacities to $l_{ij} = u_{ij} = 0$.
10:         end for
11:         Consider $S_l \in \mathbb{S}^=_l$ and set the capacities of the arcs $(i, t) \in A_t$, with $i \in S_l$, to $u_{it} = +\infty$.
12:         Determine the max-flow between $0$ and $t$ in $G^*$. Let $\nu$ be the value of the max-flow.
13:         if $\nu < 1$ then
14:             Determine the sets $S$ and $S' = (N \cup 0) \setminus S$ such that $S'$ is the set of nodes reachable
                from node $0$ in the residual network associated with the max-flow.
15:             Add the violated inequality $x(S', S) \geq 1$.
16:         end if
17:     end while
18: end for
```

again, we set a limit to the number of added violated inequalities before re-solving the LP problem. The pseudocode for the heuristic separation of the RFV inequalities is presented in Algorithm 5.5.

Algorithm 5.5 Heuristic separation of the RFV inequalities.

```
Require: A fractional solution $(x^*, y^*)$ that satisfies the equation system (4.2)-(4.5).
1: Determine the connected components $\{C_0, \dots, C_p\}$ and the set of non-visited nodes $\overline{S}$ induced
   by the fractional solution.
2: $iter = 0$.
3: while $iter < p + 1$ do
4:     if $iter = 0$ then
5:         Set $S' = C_0$ and $S = C_1 \cup C_2 \cup \ldots \cup C_p \cup \overline{S}$.
6:     else
7:         Set $S = C_{iter}$ and $S' = (N \cup 0) \setminus S$.
8:     end if
9:     if $S \in \mathbb{S}^{FV}$ then
10:        Add the violated inequality $x(S', S) \geq 1$.
11:    end if
12:    $iter = iter + 1$.
13: end while
```

To ensure that we obtain a feasible solution for the LP relaxation of the RFV model when it is not possible to find more violated inequalities by using the heuristic separation procedure, we use the exact separation algorithm presented in Algorithm 5.4 and, thus, we guarantee that the solution obtained is the optimal solution for the LP relaxation of the B&C subproblem that we are addressing.

Table 5.2 shows the impact of using the heuristic separation for the RFV inequalities in terms of computational time. In Table 5.2 it is possible to see the average gap obtained ($\overline{gap}$), the average computational time, in seconds, to obtain the LP relaxation value ($\overline{t_s}$) and the average number of added violated RFV inequalities ($\overline{\#RFV}$) when we use the heuristic separation or when we only use the exact one. The detailed results obtained without performing the heuristic separation are available in appendix, Table B.1, and the ones obtained with the heuristic separation are in Section 5.4.

Table 5.2: Heuristic separation RFV inequalities.

	$\overline{gap}$	$\overline{t_s}$	$\overline{\#RFV}$
Without heuristic separation	0.71%	10998	15645
With heuristic separation	0.71%	1442	7052

Average obtained with benchmark instances' $burma$, $bayg$, att and $bier$.

Similarly to what happened in the case of the CC inequalities, using the heuristic separation significantly accelerates the separation process. In fact, the heuristic separation reduces the computational time by more than seven times.

Separation algorithm for integer solutions

When (x^*, y^*) is integer, that is, satisfies the equation system (4.2)-(4.5),(4.7)-(4.8) of the generic model presented in Section 4.1, the separation algorithm is different. Instead of searching the set $\mathbb{S}_l^=$ we take into account the fact that the solution contains subtours. We start by determining all the subtours that are induced by the solution (x^*, y^*). Let $\mathbb{T}^*$ be the set of all subtours in (x^*, y^*) and $\overline{S} = \{i \in N : y_i^* = 0\}$ be the set of non-visited nodes in the referred solution. By using the same argument as in the proof of Proposition 13, which is there are no arcs used between different subtours, it is possible to construct several sets S that violate RFV inequalities by ensuring that S contains the nodes in $\overline{S}$ and the nodes from, at least, one subtour in $\mathbb{T}^*$. The pseudocode for the lazy constraint callback used for the RFV inequalities is available in Algorithm 5.6.

Algorithm 5.6 Separation algorithm for the RFV inequalities for integer solutions.

```
Require: An integer solution $(x^*, y^*)$ that satisfies the equation system (4.2)-(4.5).
1: Determine $\mathbb{T}^*$ and $\overline{S}$. Let $T_i$ be the $i$-th subtour in $\mathbb{T}^*$.
2: Set $k = 1$.
3: while $k \leq |\mathbb{T}^*|$ do
4:     Set $S = \cup_{i=k}^{|\mathbb{T}^*|} T_i \cup \overline{S}$ and $S' = (N \cup 0) \setminus S$.
5:     Add the violated inequality $x(S', S) \geq 1$.
6:     $k = k + 1$.
7: end while
```

5.2.3 Separating both the CC and the RFV inequalities

In order to devise a separation procedure for the CC+RFV model, we will use the exact separation algorithms for the CC and the RFV inequalities presented previously. Since the CC inequalities are faster to separate, observe the computational times in Tables 5.1 and 5.2, we start by performing the separation algorithm for the CC inequalities and then we perform the separation algorithm for the RFV inequalities. For the same reasons stated previously, we decided to limit the maximum number of added violated inequalities per iteration. When the solution obtained is integer, we apply the same separation algorithm used for the CC inequalities in the integer case, that is, Algorithm 5.2 with $max\_number\_cuts = 1$.

The heuristic separation for the CC+RFV model could simply be the heuristic separation algorithm for the CC inequalities and for the RFV inequalities applied sequentially. However, it is possible to separate both inequalities simultaneously. Consider the heuristic separation presented for the CC inequalities in Algorithm 5.3. Every time a set S is computed we verify whether $S \in \mathbb{S}^{FV}$ or not. If it is, we add an RFV inequality, otherwise we add a CC inequality. As mentioned in Section 4.3.3, the RFV inequalities dominate the CC inequalities. Therefore, when possible, it is preferable to add an RFV inequality instead of a CC inequality. The pseudocode for the heuristic separation algorithm for both the CC and the RFV inequalities is available in Algorithm 5.7.

Once again, to ensure that the solution obtained is feasible for the LP relaxation of the CC+RFV model, when it is not possible to find more violated inequalities by using the heuristic procedure, we apply the exact separation algorithm for the CC+RFV model guaranteeing that the LP relaxation is solved optimally.

Table 5.3 shows the impact of using the heuristic separation in the CC+RFV model in terms of computational time. In Table 5.3 it is possible to see the average gap obtained ($\overline{gap}$), the average computational time, in seconds, to obtain the LP relaxation value ($\overline{t_s}$) and the average number of added violated CC ($\overline{\#CC}$) and of RFV ($\overline{\#RFV}$) inequalities when we use the heuristic separation, described in Algorithm 5.7, or when we only use the exact ones, described in Algorithms 5.2 and 5.4. The detailed results obtained without performing the heuristic separation are available in appendix, Table B.2, and the ones obtained with the heuristic separation are available in Section 5.4.

Algorithm 5.7 Heuristic separation algorithm of both CC and RFV inequalities.

```
Require: A fractional solution $(x^*, y^*)$ that satisfies the equation system (4.2)-(4.5).
 1: Determine the connected components $\{C_0, \ldots, C_p\}$ and the set of non-visited nodes $\overline{S}$ induced
    by the fractional solution.
 2: Set $iter = 0$.
 3: while $iter < p + 1$ do
 4:   if $iter = 0$ then
 5:     Set $S' = C_0$ and $S = C_1 \cup C_2 \cup \ldots \cup C_p \cup \overline{S}$.
 6:   else
 7:     Set $S = C_{iter}$ and $S' = (N \cup 0) \setminus S$.
 8:   end if
 9:   if $S \in \mathbb{S}^{FV}$ then
10:     Add the violated inequality $x(S', S) \geq 1$.
11:     $iter = iter + 1$.
12:   else
13:     Determine $k \in S$ such that $y_k^* \geq y_i^*, \forall i \in S$.
14:     Add the violated inequality $x(S', S) \geq y_k$.
15:     $iter = iter + 1$.
16:   end if
17: end while
```

Table 5.3: Heuristic separation for both CC and RFV inequalities.

	$\overline{gap}$	$\overline{t_s}$	$\overline{\#CC}$	$\overline{\#RFV}$
Without heuristic separation	0.15%	161	1012	183
With heuristic separation	0.15%	42	634	137

Average obtained with benchmark instances' $burma$, $bayg$, att and $bier$.

Similarly to what happened with the previous heuristic separations, using the heuristic separation in the CC+RFV model leads to a significant decrease in the computational time. Note that the number of added CC inequalities is considerably higher than the number of added RFV inequalities due to the order by which the inequalities are separated.

5.2.3.1 Separating only some RFV inequalities

The calculation of the set $\mathbb{S}_l^{=}$ is very time consuming, as Table 5.2 shows. Therefore, the importance of the exact separation of the RFV is purely theoretical, since it could hardly be used to obtain the optimal values of the test instances within a reasonable computational time. Even in the case of the CC+RFV model, the RFV inequalities make the separation much more time consuming. In fact, for the smaller benchmark instances, the CC model takes an average of 6 seconds to obtain the LP relaxation value whilst the CC+RFV model takes an average of 42 seconds. Therefore, we decided to experiment using separation algorithms for the CC+RFV model that only separate some RFV inequalities. To clarify, we propose a new approach in which the RFV inequalities are added in a heuristic manner, in the sense that, it is not ensured that all RFV inequalities are satisfied. In the previous conditions, the RFV inequalities are added as valid inequalities for the FTSP and the CC inequalities are the subtour eliminations constraints. We developed two separation algorithms for the case in which the RFV inequalities are added in a heuristic manner: the y-separation algorithm and 1-separation algorithm. The former is a straightforward adaptation of the separation algorithm for the CC inequalities while the latter is an adaptation, which on average, finds more violated RFV inequalities. Summarizing, we have three distinct separation algorithms for the CC+RFV model, an exact one, which ensures that there are no more violated CC and RFV inequalities and two separation algorithms, the y-separation algorithm and the 1-separation algorithm, in which the RFV inequalities are separated in a heuristic manner. For simplification purposes, when the CC+RFV model is solved with the exact separation algorithm we call it the CC+RFV model and, when the CC+RFV model is solved with the y-separation algorithm and with the 1-separation algorithm, we will call it the y-separation and the 1-separation, respectively.

The y-separation algorithm is similar to the separation algorithm for the CC inequalities presented in Algorithm 5.2, with the difference that every time the set S is determined we check whether $S \in \mathbb{S}^{FV}$ or not. If $S \in \mathbb{S}^{FV}$ the violated inequality found is in the conditions of the RFV inequalities and so we add an RFV inequality, otherwise we add the violated CC inequality. Algorithm 5.8 shows the pseudocode for the y-separation algorithm.

Algorithm 5.8 The y-separation algorithm.

```
Require: A fractional solution $(x^*, y^*)$ that satisfies the equation system (4.2)-(4.5).
 1: Create a new complete graph $G^* = (0 \cup N, A)$.
 2: for all $(i, j) \in A$ do
 3:     Set its capacities to $l_{ij} = 0$ and $u_{ij} = x^*_{ij}$.
 4: end for
 5: for all $k \in N$ such that $y^*_k > 0$ do
 6:     Determine the max-flow between $0$ and $k$ in $G^*$. Let $\nu$ be the value of the max-flow.
 7:     if $\nu < y^*_k$ then
 8:         Determine the sets $S$ and $S' = (N \cup 0) \setminus S$ such that $S'$ is the set of nodes reachable from
            node $0$ in the residual network associated with the max-flow.
 9:         if $S \in \mathbb{S}^{FV}$ then
10:             Add the violated inequality $x(S', S) \geq 1$.
11:         else
12:             Add the violated inequality $x(S', S) \geq y_k$.
13:         end if
14:     end if
15: end for
```

The 1-separation algorithm is also based on max-flow/min-cut computations. We start by computing the maximum flow from the depot to every node $k \in N$. Let ν be the value of the maximum flow from the depot to node k. If $\nu \geq 1$ there is neither a violated RFV inequality nor a violated CC inequality. Otherwise, we start by verifying if $S \in \mathbb{S}^{FV}$ and if it is we add an RFV inequality. If $S \notin \mathbb{S}^{FV}$ we check if there is a violated CC inequality, that is, if $\nu < y^*_k$. Once again, if that is the case we add the violated CC inequality. The pseudocode for the 1-separation algorithm is available in Algorithm 5.9.

Note that by using the y-separation algorithm described in Algorithm 5.8 and the 1-separation algorithm presented in Algorithm 5.9 we ensure that all the CC inequalities are separated. Therefore, these separation algorithms can be used in the B&C algorithm to solve the FTSP. To better understand the difference between the y-separation and the 1-separation, consider the solution pre-

Algorithm 5.9 The 1-separation algorithm.

```
Require: A fractional solution $(x^*, y^*)$ that satisfies the equation system (4.2)-(4.5).
 1: Create a new complete digraph $G^* = (0 \cup N, A)$.
 2: for all $(i, j) \in A$ do
 3:    Set its capacities to $l_{ij} = 0$ and $u_{ij} = x^*_{ij}$.
 4: end for
 5: for all $k \in N$ such that $y^*_k > 0$ do
 6:    Determine the max-flow between $0$ and $k$ in $G^*$. Let $\nu$ be the value of the max-flow.
 7:    if $\nu < 1$ then
 8:       Determine the sets $S$ and $S' = (N \cup 0) \setminus S$ such that $S'$ is the set of nodes reachable from
          node $0$ in the residual network associated with the max-flow.
 9:       if $S \in \mathbb{S}^{FV}$ then
10:          Add the violated inequality $x(S', S) \geq 1$.
11:       end if
12:    else if $\nu < y^*_k$ then
13:       Determine the sets $S$ and $S' = (N \cup 0) \setminus S$ such that $S'$ is the set of nodes reachable from
          node $0$ in the residual network associated with the max-flow.
14:       Add the violated inequality $x(S', S) \geq y_k$.
15:    end if
16: end for
```

sented in Figure 4.7, which corresponds to a feasible solution for the LP relaxation of the NCF^+ model considering the FTSP instance presented in Figure 3.1. More precisely, consider the solution in terms of the x variables in Figure 4.7a. Recall that a feasible solution for the LP relaxation of the NCF$^+$ model is also feasible for the LP relaxation of the NCF model and, consequently, for the LP relaxation of the CC model. If we compute the maximum-flow between the depot and node 3 in a capacitated network in which the lower capacity of the arc (i, j) is zero and the upper capacity is given by the value of the variable x_{ij}, we verify that the maximum-flow has a value of 0.50 and as expected, there is no violated CC inequality in this case. Nevertheless, the set S that originates the minimum cut, which has a capacity of 0.50, is $S = \{2, 3, 4\}$. Observe that $S \in \mathbb{S}^{FV}$, thus, it originates a violated RFV inequality $x(S', S) \geq 1$. Consequently, with the 1-separation algorithm we would be able to find this violated RFV inequality, as we compare the value of the minimum cut to 1, whereas by using the y-separation algorithm we could not as the minimum cut is equal to y.

Observe that we cannot establish any relationship between the LP relaxation value obtained with the y-separation algorithm and the one obtained with the 1-separation algorithm due to the RFV inequalities being separated in a heuristic manner. Nevertheless, the y- and the 1-separation algorithms provide a lower LP relaxation value than the CC+RFV model and a higher LP relaxation value than the CC model. Additionally, we can also state that the LP relaxation of the y- and the 1-separation algorithms are not comparable to the LP relaxation of the RFV model.

Similarly to what we did with the separation algorithm for the CC inequalities, we limited to the total number of added violated inequalities per iteration and used a permutation of the nodes. Additionally, when the solution (x^*, y^*) is integer we apply the same separation algorithm that we applied when separating CC inequalities. Regarding the heuristic separation, we use the one presented in Algorithm 5.7.

Table 5.4 compares the LP relaxation value obtained with the CC+RFV model to the LP relaxation values obtained with the y-separation algorithm and the 1-separation algorithm. In Table 5.4 it is possible to see the average gap obtained ($\overline{gap}$), the average computational time, in seconds, to obtain the LP relaxation value ($\overline{t_s}$) and the average number of added violated CC inequalities ($\overline{\#CC}$) and RFV inequalities ($\overline{\#RFV}$). The detailed results are available in Section 5.4.

Table 5.4: Separation algorithms for the CC+RFV model.

	$\overline{gap}$	$\overline{t_s}$	$\overline{\#CC}$	$\overline{\#RFV}$
CC+RFV	0.15%	42	634	137
y-separation algorithm	0.17%	7	152	472
1-separation algorithm	0.17%	9	144	450

Average obtained with benchmark instances' $burma$, $bayg$, att and $bier$.

The difference in the average gap between separating all the RFV inequalities with the exact separation algorithm and separating them with the y-separation algorithm and the 1-separation algorithm is not meaningful, however the computational time decreases significantly. In fact, using the y-separation algorithm leads to a decrease in computational time of approximately 83%. Even though an average of 42 seconds seems a reasonable computational time, when we tried to obtain the LP relaxation value of instances a with the CC+RFV model, which have 280 nodes, we were unable to do so after 48 hours. When comparing the y-separation algorithm and the 1-separation algorithm to the CC model we verify that adding the RFV inequalities as valid inequalities significantly improves the LP relaxation value. By observing Table 5.1, we verify that the average gap obtained with the CC model was 3.17%. Concerning the computational time, the usage of the RFV inequalities as valid inequalities originates similar computational times as the CC model. More precisely, the CC model takes 6 seconds while the y-separation and 1-separations take 7 and 9 seconds, respectively. Finally, comparing the y-separation to the 1-separation, we verify that the average gap and the average time are similar.

Summary

Throughout this section we presented separation algorithms for sets of inequalities associated with the models: (i) CC model; (ii) RFV model; and (iii) CC+RFV model. Additionally, we presented heuristic separation algorithms for each of these sets of inequalities, which made the separation process much more efficient. Note that, even though we presented the separation algorithms considering a point that satisfies constraints (4.2)-(4.5) from the generic formulation presented in Section 4.1, these separation algorithms may be used to separate any point.

From the summary of results presented, it seems that all models that require the separation of all RFV inequalities are much worse in terms of computational time, therefore we proposed two different separation algorithms, the y-separation algorithm and the 1-separation algorithm, that only separate some RFV inequalities. Nevertheless, by applying these separation algorithms we ensure that a feasible solution for the FTSP is obtained as they guarantee that all the CC inequalities are separated.

As we saw in Section 4.4, the RV model is not competitive, both in terms of LP relaxation value and of computational time, with the CC, the RFV and the CC+RFV models. In order to obtain the results that allowed us to do that comparison, we had to devise an exact and a heuristic separation algorithm for the RV inequalities, which we decided to omit as they are very similar to the corresponding separation algorithms for the RFV inequalities. In the exact algorithm for the RV inequalities, instead of fixing $n_l - v_l + 1$, of some family $l \in \mathcal{L}$, to S, we fix $|N| - V + 1$ nodes of N to S and, in the heuristic separation, every time we compute a set S we need to verify whether $S \in \mathbb{S}^V$ or not.

5.3 Heuristic callback

During the B&C algorithm, as we explained in Section 2.3, whenever we find a feasible solution for the ILP problem, since we are addressing a minimization problem, its value provides an upper bound for the optimal value of the ILP problem. This upper bound allow us to prune B&C subproblems which have an LP relaxation value higher than the upper bound. Therefore, we devised a heuristic algorithm in an attempt to obtain better feasible solutions than the ones provided by the general purpose heuristics of CPLEX, that is, feasible solutions with a lower cost. As the heuristic algorithm is executed several times during the B&C algorithm, efficiency is of the utmost importance. Thus, we decided to use the basic heuristics presented in Section 3.3.

In the heuristic callback we used two constructive heuristics which were presented in Section 3.3.2, namely the nearest neighbor and the random heuristic. However, the nearest neighbor heuristic is applied considering a modified cost matrix in order to take into account the solution of the B&C subproblem that we are currently addressing. Let (x^*, y^*) be the solution of the current B&C subproblem. The arc costs are modified in such a way that the arcs that have a higher value of x^* are more likely to be on the solution constructed. The cost of the arc $(i, j) \in A$ is set to $c^*_{ij} = c_{ij} \times (1 - x^*_{ij})$. Note that, if $x^*_{ij} = 1$, then $c^*_{ij} = 0$ and, since the nearest neighbor is a greedy heuristic, the arc (i, j) becomes attractive to be selected for the heuristic solution.

After obtaining a feasible solution for the FTSP we apply a local search procedure which consists in searching the neighborhoods $\mathscr{N}_I$, $\mathscr{N}_O$ and 2-opt presented in Section 3.3. The pseudocode for the local search procedure is given in Algorithm 5.10.

All there is left now is to combine the several subroutines and create the heuristic callback used during the B&C algorithm. It is possible to execute the heuristic callback in every B&C subproblem,

Algorithm 5.10 The local search procedure used in the heuristic callback.

```
Require: A feasible solution $s$ for the FTSP
1: Search $\mathscr{N}_I(s)$ and obtain $s^*$. Set $s = s^*$.
2: Search $\mathscr{N}_O(s)$ and obtain $s^*$. Set $s = s^*$.
3: if The cost matrix is symmetric then
4:     Search 2-opt$(s)$ and obtain $s^*$.
5: end if
```

however we decided to apply it less frequently. During the first 250 B&C subproblems, we apply the heuristic callback with a frequency of 5 and, after that, the frequency drops to 10. We apply both constructive heuristics according to the following criterion: if the number of the B&C subproblem is even we use the nearest neighbor with the cost matrix c^* and if the number of the B&C subproblem is odd we use the random heuristic.

Table 5.5 shows the impact of using the heuristic algorithm to improve the upper bound in the B&C algorithm when solving the CC model or when using the CC+RFV model with the y- and the 1-separation algorithms. For the referred methods it is possible to observe the average computational time, in seconds, to obtain the optimal value ($\overline{t_s}$) and the average number of B&C subproblems solved during the B&C algorithm until reaching the optimal solution ($\overline{\#sub}$) when using or not the heuristic callback. The detailed results obtained without the heuristic callback are available in appendix, Tables B.3 and B.4, and the ones obtained with the heuristic callback are available in Section 5.4.

Table 5.5: Heuristic callback in the B&C algorithm.

	CC		y-separation		1-separation	
	$\overline{t_s}$	$\overline{\#sub}$	$\overline{t_s}$	$\overline{\#sub}$	$\overline{t_s}$	$\overline{\#sub}$
Without heuristic algorithm	44	89	25	95	38	83
With heuristic algorithm	33	85	10	58	11	59

Average obtained with benchmark instances' $burma$, $bayg$, att and $bier$.

In Table 5.5 we can observe that using the heuristic algorithm during the B&C algorithm allowed us to decrease the average computational time, which is a consequence of the decrease in the average number of B&C subproblems solved during the B&C algorithm. These results show that including the heuristic callback makes the B&C algorithm more efficient.

5.4 Computational experiment

Throughout this section we present the computational experiment carried out considering the several sets of test instances presented in Chapter 3. This section is divided into four sections each one devoted to one set of instances. We start by presenting the computational experiment for the instance set 1 in Section 5.4.1, then for the instance set 2 in Section 5.4.2, followed by the one for the instance set 3 in Section 5.4.3 and, finally, for the instance set 4 in Section 5.4.4.

The computational results were obtained adding a maximum of 20 violated inequalities per iteration. Additionally, due to the conclusions drawn in the previous sections, all results were obtained using heuristic separations and the heuristic callback when solving the instances optimally.

The LP relaxation of the several formulations were obtained without using the general purpose cuts generated by CPLEX to allow us to establish a fair comparison between the several models. However, the optimal solutions for the FTSP models were obtained using these CPLEX generated cuts and heuristics. Furthermore, we set as a time limit of 10800 seconds for the B&C algorithm to provide the optimal solutions for the FTSP instances, and, we consider that the B&C algorithm is an effective method to solve an instance if it is able to find its optimal solution within the time limit.

The implementation of the proposed methods is original, except for the max-flow algorithm, for which we used the algorithm to solve the max-flow problem proposed by Goldberg and Tarjan (1988). The models were implemented in C++ and were solved by using the Concert Technology from CPLEX 12.6.1 (see e.g., IBM, 2014). All computational experiments were carried out in an Intel Core i7, 3.60 gigahertz, 8 gigabytes RAM.

5.4.1 Benchmark instances

Tables 5.6 and 5.7 show the LP relaxation values obtained by using the CC model, the RFV model and the CC+RFV model, and, the y-separation and the 1-separation, respectively. The tables are divided into several parts, each one corresponding to a different method. Each of those parts has four or five columns depending on the number of different valid inequalities separated. The first column shows the LP relaxation value ($\mathcal{V}^{LP}$), the second the percentage of gap between the LP relaxation value and the optimal value ($gap = 100 \times (\textit{optimal value} - \textit{LP relaxation value})/\textit{optimal value}$), followed by the computational time, in seconds, to obtain the LP relaxation value (t_s) and, finally, the number of added violated inequalities (#$name\_of\_inequality$). Additionally, the tables contain, in the last row, the average of the results obtained. These tables correspond to the detailed results of the tables shown in the Section 4.4.

Table 5.6: Linear programming relaxation results for the instance set 1 with CC, RFV and CC+RFV models.

	CC				RFV				CC + RFV				
Instance	$\mathcal{V}^{LP}$	gap	t_s	#CC	$\mathcal{V}^{LP}$	gap	t_s	#RFV	$\mathcal{V}^{LP}$	gap	t_s	#CC	#RFV
burma_1	12.07	13.34%	0	34	13.93	0.00%	0	98	13.93	0.00%	0	35	53
burma_2	25.66	0.00%	0	90	25.66	0.00%	0	33	25.66	0.00%	1	22	26
burma_3	9.93	16.47%	0	55	11.89	0.00%	0	51	11.89	0.00%	0	22	22
bayg_1	5273.32	1.36%	0	184	5316.85	0.54%	0	707	5330.17	0.29%	0	259	67
bayg_2	5754.64	0.63%	0	307	5791.01	0.00%	0	261	5791.01	0.00%	1	270	111
bayg_3	5544.33	0.00%	0	146	5544.33	0.00%	0	491	5544.33	0.00%	0	151	39
att_1	23686.00	0.00%	0	501	23580.50	0.45%	1	1219	23686.00	0.00%	1	451	29
att_2	20609.10	0.00%	1	720	20609.10	0.00%	6	2851	20609.10	0.00%	1	841	39
att_3	8742.08	3.13%	1	446	8760.03	2.93%	2	2021	9024.58	0.00%	1	478	102
bier_1	33227.80	1.43%	13	1321	33314.70	1.17%	5484	20937	33446.00	0.78%	29	1365	43
bier_2	88308.90	0.48%	19	1389	87336.20	1.58%	7913	39032	88479.50	0.29%	247	1721	802
bier_3	47162.50	1.18%	35	1903	46830.70	1.88%	3898	16926	47504.40	0.46%	217	1988	305
$average$		3.17%	6	591		0.71%	1442	7052		0.15%	42	634	137

Table 5.7: Linear programming relaxation results for the instance set 1 with y- separation and 1-separation.

	y-separation					1-separation				
Instance	$\mathcal{V}^{LP}$	gap	t_s	#CC	#RFV	$\mathcal{V}^{LP}$	gap	t_s	#CC	#RFV
burma_1	13.93	0.00%	0	23	43	13.93	0.00%	0	19	40
burma_2	25.66	0.00%	0	0	27	25.66	0.00%	0	0	35
burma_3	11.89	0.00%	0	11	19	11.89	0.00%	0	11	36
bayg_1	5330.17	0.29%	0	54	235	5330.17	0.29%	0	39	201
bayg_2	5791.01	0.00%	0	9	254	5791.01	0.00%	0	13	224
bayg_3	5544.33	0.00%	0	21	160	5544.33	0.00%	0	21	167
att_1	23686.00	0.00%	0	18	502	23686.00	0.00%	0	18	502
att_2	20609.10	0.00%	1	97	710	20609.10	0.00%	1	110	759
att_3	9024.58	0.00%	0	133	300	9024.58	0.00%	0	167	281
bier_1	33446.00	0.78%	11	349	952	33446.00	0.78%	5	394	747
bier_2	88447.60	0.33%	18	396	1116	88448.10	0.32%	22	261	1008
bier_3	47397.00	0.69%	51	711	1370	47417.20	0.65%	80	672	1394
$average$		0.17%	7	152	474		0.17%	9	144	450

As we already established, separating all the RFV inequalities is very time consuming. In fact, the average gap obtained with the RFV model is much smaller than the one obtained with the CC model, however, the computational times are the reverse, that is, the RFV model is much more time consuming. When we combine both valid inequalities in the CC+RFV model we obtain very small gap values. In particular, we were able to obtain the optimal value when solving the LP relaxation in eight of the 12 instances presented. Regarding the computational time, the CC+RFV model is much more efficient than the RFV model but, as we already mentioned, for instances of higher dimension, it is still very time consuming.

The y-separation and 1-separation proved to be a good alternative for the CC+RFV model as they take advantage of the RFV inequalities without using its time consuming exact separation. In fact, these separation algorithms were able to provide LP relaxation values similar to the ones provided by the CC+RFV model but in a more efficient manner. For instance $bier\_2$, the y-separation algorithm took 18 seconds to find the LP relaxation value whilst the CC+RFV model took 247 seconds. Additionally, they also provide better results than both the CC and the RFV models, thus, from now, we will only focus on the y-separation and the 1-separation. Comparing both separation algorithms, we verify that the y-separation algorithm is usually faster while the 1-separation algorithm provides, in general, lower gap values. These results can be explained due to the fact that the 1-separation algorithm was designed to find more RFV inequalities, thus the increase in time and the decrease in gap.

With the y- and 1-separation algorithms we tried to obtain the LP relaxation value of instances a, which are available in Table 5.8. Table 5.8 is organized in a similar manner as Table 5.7.

Table 5.8: Linear programming relaxation results for instances a with y-separation and 1-separation.

	y-separation					1-separation				
Instance	$\mathcal{V}^{LP}$	*gap*	t_s	#CC	#RFV	$\mathcal{V}^{LP}$	*gap*	t_s	#CC	#RFV
a_1	1676.43	11.35%*	1087	1318	2965	1677.52	11.30%*	2626	1359	3206
a_2	1508.99	11.10%*	1375	1660	2877	1509.88	11.05%*	2674	1651	3263
a_3	1386.06	1.57%	1174	1928	2608	1389.31	1.34%	2066	2047	2914
average		8.01%	1212	1635	2817		7.90%	2455	1686	3128

*Gap calculated with the best upper bound obtained by Morán-Mirabal et al. (2014).

The gap values marked with an asterisk (*) were computed by using the best upper bound available in the literature instead of the optimal value, as the optimal value is unknown. Therefore, these values are an upper bound for the real gap value. The best upper bounds were obtained by Morán-Mirabal et al. (2014).

By observing Table 5.8 we see that the difference between the y-separation and the 1-separation is more noticeable for instances a. The latter provides a lower average gap value while the former is more efficient to obtain the LP relaxation value. However, the difference between the average gap obtained by both methods is not meaningful, as it is a difference of 0.11%, but, regarding the computational time, the y-separation takes less than half of the time to obtain the LP relaxation value. These conclusions are due to the fact that the number of RFV inequalities found during the 1-separation is higher. In fact, the average number of RFV inequalities added with the referred separation is 3128 whereas the average number of RFV inequalities added with the y-separation is 2817.

We tried to compute the LP relaxation values for the remaining benchmark instances, instances gr and pr, which have 666 and 1002 nodes respectively, however it was impossible to do so due to lack of computational memory. These instances will be addressed by using heuristic methods in Chapter 6.

Table 5.9 compares the y-separation to the 1-separation in terms of efficiency, that is, computational time, to obtain the optimal value. Table 5.9 is divided into two parts, one for each method. Each part shows the optimal value obtained ($\mathcal{V}$), the computational time, in seconds, to obtain the optimal value (t_s), the number of B&C subproblems solved during the B&C algorithm (#sub) and the number of added violated CC inequalities (#CC) and RFV inequalities (#RFV). Additionally, it contains, in the last row, the average of the results obtained.

Table 5.9: Optimal values for the instance set 1 with y-separation and 1-separation.

	y-separation					1-separation				
Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV	$\mathcal{V}$	t_s	#sub	#CC	#RFV
burma_1	13.93	0	0	15	24	13.93	0	0	14	48
burma_2	25.66	1	0	5	27	25.66	1	0	5	35
burma_3	11.89	0	0	10	16	11.89	0	0	5	32
bayg_1	5345.89	0	1	24	169	5345.89	1	0	25	227
bayg_2	5791.01	0	0	13	200	5791.01	0	0	13	223
bayg_3	5544.33	0	0	17	105	5544.33	0	0	17	112
att_1	23686.00	1	0	26	461	23686.00	1	0	19	385
att_2	20609.10	1	0	70	571	20609.10	1	0	129	691
att_3	9024.58	0	0	142	213	9024.58	0	0	154	256
bier_1	33709.70	23	252	226	2071	33709.70	15	175	184	1133
bier_2	88736.40	48	243	716	2999	88736.40	51	270	624	2505
bier_3	47726.30	49	205	1006	2578	47726.30	63	263	964	3175
average		10	58	189	786		11	59	179	735

As Table 5.9 shows, the computational time to solve instances $burma$, $bayg$ and att is negligible with any of the separation algorithms. If we only focus on instances $bier$, we verify that the y-separation algorithm is faster on average, as it takes an average of 40 seconds while the 1-separation takes an average of 46 seconds even though the 1-separation provided slightly better LP relaxation values with an average of 0.58% compared to an average of 0.60% provided by the y-

separation. Nonetheless, these values are very similar which makes it difficult to establish which is the best method. Summarizing, both methods are suited to solve the FTSP benchmark instances up to 127 nodes as they were able to provide the optimal value of instances that had never been solved, optimally, in less than 63 seconds.

Tables 5.10 and 5.11 show the computational results obtained when we tried to solve instances a optimally with the y-separation and the 1-separation, respectively. Each table contains the optimal value ($\mathcal{V}$), the computational time, in seconds, to obtain the optimal value (t_s), the number of B&C subproblems solved during the B&C algorithm (#sub) and the number of added violated CC inequalities (#CC) and RFV inequalities (#RFV). These tables also contain, in the last row, the average of the results obtained.

Table 5.10: Optimal value for instances a with y-separation.

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
a_1	[1688.59, 1692.92]	10803	2012	11290	44992
a_2	[1514.30, 1612.39]	10803	816	8730	30605
a_3	1408.14	3730	176	5056	8876
$average$		8445	1001	8358	28157

Table 5.11: Optimal value for instances a with 1-separation.

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
a_1	[1686.95, 1694.79]	10804	1080	12834	42351
a_2	[1512.40, 1625.84]	10808	421	6557	24709
a_3	1408.14	5005	287	7405	17614
$average$		8872	596	8932	28225

The results of Tables 5.10 and 5.11 show that, for instances a_1 and a_2, neither of the separations was able to solve them. Therefore, instead of showing the optimal value we show an interval $[LB, UB]$, in which LB represents the best lower bound and UB represents the best upper bound for the optimal value found by the B&C algorithm after 10800 seconds. Regarding instance a_3, both the y-separation and 1-separation were able to obtain its optimal value within the time limit, however, we verify that the y-separation algorithm was faster as it took 3730 seconds whereas the 1-separation algorithm took 5005 seconds, despite the fact that the percentage of gap obtained with the y-separation was higher than the one obtained with the 1-separation (1.57% compared to 1.34%).

Additionally, for the instances that we could not solve within the time limit, the y-separation algorithm provided better lower and upper bounds than the 1-separation algorithm within the same time limit.

5.4.2 Generated instances based on symmetric TSP instances

In this section we present the computational results regarding the instance set 2, which corresponds to symmetric instances generated based on TSP instances. As we saw in Section 5.4.1, both the y-separation and the 1-separation are efficient tools to solve FTSP instances. In fact, they are able to solve instances with 127 nodes in less than 63 seconds. However, the proposed separation algorithms were only able to solve one instance with 280 nodes within the time limit. Thus, instance set 2 was designed to help us analyze how the proposed exact approach handles instances with a number of nodes between 127 and 280.

Table 5.12 shows the average LP relaxation results for the instance set 2 obtained with the y-separation and the 1-separation. This table contains the average gap between the LP relaxation value and the optimal value ($\overline{gap}$), the average computational time, in seconds, to obtain the LP relaxation values ($\overline{t_s}$) and the average number of added violated CC ($\overline{\#CC}$) and RFV ($\overline{\#RFV}$) inequalities. The results obtained for each instance from the instance set 2 are available in appendix, Table B.5. Additionally, to simplify the notation, the instance that is presented as $tspinstancename.sftsp\_L\_i$, with $i \in \{1, \ldots, 4\}$ in Table A.2 is designated as $tspinstancename\_i$.

Table 5.12: Average LP relaxation results for the instance set 2.

	$\overline{gap}$	$\overline{t_s}$	$\overline{\#CC}$	$\overline{\#RFV}$
y-separation	1.15%	1142	1109	2132
1-separation	1.13%	1694	1134	2290

Once again, the 1-separation algorithm provides a slightly lower average gap value, nonetheless there are instances where the y-separation algorithm provides a lower gap value as, for example instance $gr136\_2$ (see Table B.5), which highlights the fact that these separation algorithms are heuristic regarding the RFV inequalities. Regarding the computational time, the y-separation algorithm is more efficient, on average, than the 1-separation algorithm.

By observing the detailed results of Table B.5 we verify that there are some gap values marked with an asterisk (*), which means that those gap values were obtained by using the best upper bound obtained with the B&C algorithm after 10800 seconds of computational time instead of the optimal value as we could not obtain it within the time limit, as we shall see in Table 5.14. If we consider the average gap obtained without the instances for which the optimal value is not known, both the y-separation and the 1-separation algorithms provide an average gap of 0.75%, which shows that the proposed methods provide very good LP relaxation values for this instance set.

In order to verify if there is a pattern relating the instance type to the difficulty to solve it we clustered the instances by instance type and evaluated the LP relaxation value and the computational time. Therefore, Table 5.13 shows the average gap ($\overline{gap}$) and the average computational time, in seconds, to obtain the LP relaxation value ($\overline{t_s}$) for each instance type. Recall that instances of type $reference$ are the instances of reference, instances of type low were designed to have a smaller number of visits per family than instances of type $reference$ while instances of type $high$ have a higher number of visits per family than instances of type $reference$, and, finally, instances of type $mixed$ are a random combination of instances of type low and $high$.

Table 5.13: Average gap and time by instance type for the instance set 2.

	y-separation		1-separation	
	$\overline{gap}$	$\overline{t_s}$	$\overline{gap}$	$\overline{t_s}$
Type $reference$	1.03%	1783	0.93%	2497
Type low	1.13%	2069	1.20%	2775
Type $high$	1.15%	214	1.14%	121
Type $mixed$	1.29%	503	1.26%	1384

Table 5.13 shows that the y-separation algorithm is more efficient than the 1-separation algorithm for all types of instance, except for type $high$. These results are not surprising considering the nature of the instances of type $high$, which were designed to have a higher number of visits, and consequently, the RFV inequalities are more effective. For illustration purposes, consider a family l with five nodes ($n_l = 5$). Firstly, we assume that we are required to make few visits, so $v_l = 1$. As the RFV inequalities are valid if there are in S at least $n_l - v_l + 1$ nodes from family l, in this particular case the RFV inequalities are valid if the number of nodes from family l in S is at least $n_l - v_l + 1 = 5 - 1 + 1 = 5$. Therefore, there is only one subset of F_l that originates an RFV inequality. Conversely, consider that the number of visits is high, for instance, $v_l = 4$. By using the same reasoning as previously, in order to obtain an RFV inequality the number of nodes from family l in S has to be, at least, $n_l - v_l + 1 = 5 - 4 + 1 = 2$. Consequently, there are 10 subsets of F_l that originate RFV inequalities. Thus, the RFV inequalities are more effective for instances of type $high$ and, consequently, the separation that favors them performs better.

Regarding the average gap value, the 1-separation algorithm provides lower average gap values for all types of instances, except for the type low. Once again, these results are a consequence from the fact that instances of type low have a small number of visits per family and the RFV inequalities are less effective in this case.

According to the results shown in Table 5.13, the logical decision would be to use the y-separation algorithm to obtain the optimal value of instances of types $reference$, low and $mixed$ and use the 1-separation algorithm to obtain the optimal value of instances of type $high$. However, we experimented obtaining the optimal solution of the instances of type $high$ using both algorithms and realized that the 1-separation algorithm is worse than the y-separation algorithm in terms of average computational time to obtain the optimal solution, with an average of 5423 seconds compared to an average of 4161 seconds of the y-separation algorithm. Therefore, we decided to use the y-separation algorithm to obtain the optimal values of all instances from the instance set 2 as well.

Table 5.14 shows the results obtained by using the y-separation algorithm to solve the instances from the instance set 2 to optimality. This table shows the optimal values of the instance set 2 ($\mathcal{V}$), the computational time, in seconds, to obtain the optimal value (t_s), the number of B&C subproblems solved during the B&C algorithm (#sub) and the number of added violated CC (#CC) and RFV (#RFV) inequalities. Table 5.14 also contains, in the last row, the average of the results obtained. The results obtained with the 1-separation algorithm for the instances of type $high$ from the instance set 2 are available in appendix, Table B.6.

Table 5.14: Optimal values for the instance set 2 with y-separation.

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
pr136_1	61448	43	56	709	3494
pr136_2	43522	74	47	1724	2222
pr136_3	81481	141	1863	335	9690
pr136_4	63246	19	7	318	1327
gr137_1	44263	12	43	187	987
gr137_2	36435	36	3	677	1270
gr137_3	55919	18	304	50	1511
gr137_4	46620	12	3	125	1228
pr144_1	46376	20	7	300	1603
pr144_2	36518	144	0	1673	1892
pr144_3	54635	992	13878	374	12633

Continues on the next page

Table 5.14

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
pr144_4	[49182.80, 49403]	10801	6050	12152	56573
kroA150_1	14307	33	0	518	1007
kroA150_2	9481	250	11	2344	1034
kroA150_3	[20764.00, 20880]	10801	54566	1912	49762
kroA150_4	13404	190	88	1036	2114
kroB150_1	14522	41	11	439	978
kroB150_2	9555	187	18	3285	2114
kroB150_3	19925	4	9	40	859
kroB150_4	12532	31	19	901	1370
pr152_1	51806	112	60	977	4324
pr152_2	45810	600	38	3341	3676
pr152_3	[64274.50, 64425]	10804	28982	7499	64854
pr152_4	57337	2076	4641	4457	31185
u159_1	29821	181	42	734	3635
u159_2	23404	304	28	1499	3481
u159_3	36399	29	207	201	2220
u159_4	30845	1083	1044	6474	23914
rat195_1	[1258.33, 1285]	10803	1444	12126	52697
rat195_2	912	376	80	3007	3346
rat195_3	[1779.89, 1814]	10805	6231	12140	116441
rat195_4	[1300.13, 1320]	10804	2849	7893	87117
d198_1	10945	1070	244	2194	15348
d198_2	10212	1537	158	3546	7445
d198_3	13843	153	137	484	4855
d198_4	12418	529	488	1505	10735
kroA200_1	[16331.50, 16441]	10803	6890	8436	52307
kroA200_2	12416	785	17	2747	1676
kroA200_3	[24017.30, 24471]	10805	9903	5070	119139
kroA200_4	16518	217	11	880	2010
kroB200_1	17527	1669	870	4406	19511
kroB200_2	12881	1654	180	5489	3452

Continues on the next page

Table 5.14

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
kroB200_3	[23863.50, 24088]	10805	16399	2584	104010
kroB200_4	17583	623	165	1430	4582
gr202_1	23394	2282	1175	2739	32776
gr202_2	14957	813	8	2237	2415
gr202_3	34547	2669	13322	1220	29388
gr202_4	[27948.10, 28039]	10804	2506	6794	66623
pr226_1	52109	828	33	896	3615
pr226_2	47585	10039	54	3667	19235
pr226_3	66812	839	45	209	4353
pr226_4	51905	1162	12	1170	7251
gr229_1	[70423.00, 70741]	10804	4402	5387	66560
gr229_2	31653	3833	214	4387	6392
gr229_3	102841	1613	4928	935	22736
gr229_4	46231	176	29	811	2900
gil262_1	[1506.26, 1529]	10804	1516	4758	48683
gil262_2	1069	3652	33	3860	3015
gil262_3	[1961.23, 2012]	10801	4140	1439	69503
gil262_4	[1642.00, 1773]	10804	649	2251	40714
pr264_1	33904	5010	617	3197	16690
pr264_2	[28151.52, 28748]	10801	1498	12277	13617
pr264_3	40705	1942	1378	704	30622
pr264_4	35153	7099	1637	5920	27513
average		3426	3067	3017	22035

By observing Table 5.14 we verify that there are two distinct cases regarding the resolution of each instance, which are: (i) we were able to obtain the optimal value within the time limit; or (ii) we could not obtain the optimal value within the time limit. In instances in which case (ii) happens, instead of the optimal value we can see the pair of values $[LB, UB]$ corresponding to the lower bound and the upper bound for the optimal value, respectively, obtained by the B&C algorithm after 10800 seconds of computational time.

From the instance set 2, we were able to obtain the optimal value, within the time limit, of

49 out of the 64 instances, which have a maximum of 264 nodes. These results are satisfactory considering the dimension of these instances. Considering only the instances that were solved within the time limit, the average computational time obtained was 1167 seconds. We were able to solve all instances $pr136$, $gr137$, $kroB150$, $u159$, $d198$ and $pr226$. The hardest instances were $rat195$ and $gil262$, since we could only solve one of them within the time limit. These results show that the number of visits per family has more influence in the efficiency of the method than the instance dimension, as we can solve instances with 198 nodes in an average of 822 seconds but we cannot solve instance $pr144\_4$ within the time limit.

Although we could not obtain the optimal value of the instances in case (ii), the B&C algorithm was able to provide an upper bound for their optimal values. Additionally, we can even evaluate the quality of the upper bound obtained by comparing it to the lower bound. These instances will be addressed in Chapter 6 with heuristic methods.

For a simplified analysis of the results obtained when solving the instances from the instance set 2 up to optimality, we clustered them by instance type. In Table 5.15 we can see the average time, in seconds, to obtain the optimal value ($\overline{t_s}$) and the number of instances solved optimally within the time limit (#solved) for each instance type. Recall that there are 16 instances of each type.

Table 5.15: Statistics for the optimal value by instance type for the instance set 2.

	$\overline{t_s}$	#solved
Type $reference$	3407	12
Type low	2193	15
Type $high$	4576	10
Type $mixed$	3527	12

The y-separation algorithm was more effective when solving instances of type low, not only is the instance type which was solved in a lower average computational time, but also is the type of instances where the number of instances solved optimally was higher. Despite the RFV inequalities being more effective when there is a higher number of visits per family as we saw previously, there are several single-visit families in instances of type low and, when that happens, we can use the valid inequalities for the single-visit families, which state that we can never use an arc between two nodes from a single-visit family. Thus, the instances of type low were easier to solve than the other instance types. However, if we compute the average computational time ignoring the instances in which the case (ii) occurred, we obtain an average computational time of 942, 1639, 840 and 1101 seconds for the instances of type $reference$, low, $high$ and $mixed$, respectively. By removing the instances that we could not solve within the time limit, we obtain the expected results, namely the

y-separation is more efficient when addressing instances of type $high$ and less efficient for instances of type low.

5.4.3 Generated instances based on asymmetric TSP instances

In this section we present the results obtained for the instance set 3, which corresponds to generated asymmetric instances based on TSP instances. Table 5.16 shows the average LP relaxation results for the instance set 3 obtained with the y-separation and the 1-separation and it is organized in a similar manner as Table 5.16. The LP relaxation values for each instance from the instance set 3 may be seen in appendix, Table B.7. To simplify the notation the instance that is presented as $atspinstancename.aftsp\_L\_i$, with $i \in \{1, \dots, 4\}$ in Table A.3 is designated by $atspinstancename\_i$ henceforth.

Table 5.16: Average LP relaxation results for the instance set 3.

	$\overline{gap}$	$\overline{t_s}$	$\overline{\#CC}$	$\overline{\#RFV}$
y-separation	1.46%	17	112	388
1-separation	1.24%	45	132	482

The conclusions drawn from Table 5.16 are similar to the ones drawn for the instance set 2, namely the y-separation is faster but the 1-separation provides lower gap values. For example and considering the results of Table B.7, for the instance $ftv170\_1$ the y-separation took 307 seconds to find the LP relaxation value whilst the 1-separation took 1358 seconds and, considering instance $ftv170\_4$, the gap value obtained with the y-separation was of 2.28% while the one obtained with the 1-separation was of 1.71%.

By observing the detailed results of instances rbg in Table B.5 we verify that both methods provide the same LP relaxation value in the majority of the instances. This unexpected behavior led us to experiment obtaining the LP relaxation values of the rbg instances using the SCF model, which was the model that provided the lowest LP relaxation values on average, and we saw that the SCF model provided the same LP relaxation value as the y-separation for all instances, except for instance $rbg323\_2$, in which the LP relaxation value obtained with the SCF model was of 67.26. The detailed results obtained with the SCF model are available in appendix, Table B.8. By analyzing the cost matrix of the rbg instances we realized that these instances have several arc costs with value zero and, since we are only required to visit some nodes, the majority of the arcs in the solution has cost zero thus the similarity of the LP relaxation values obtained by the different formulations.

We could not obtain the LP relaxation value for instance $rbg443\_2$ due to lack of computational memory, therefore this instance will be solved with heuristic methods in Chapter 6.

Table 5.17 shows the LP relaxation results for the instance set 3 grouped by instance type. The referred table contains the average gap ($\overline{gap}$) and the average computational time, in seconds, to obtain the LP relaxation value ($\overline{t_s}$) for each instance type.

Table 5.17: Average gap and time by instance type for the instance set 3.

	y-separation		1-separation	
	$\overline{gap}$	$\overline{t_s}$	$\overline{gap}$	$\overline{t_s}$
Type $reference$	1.16%	21	1.02%	109
Type low	3.09%	30	2.46%	49
Type $high$	0.86%	8	0.82%	5
Type $mixed$	0.79%	3	0.72%	17

Table 5.17 shows that the 1-separation provided a lower average gap than the y-separation for every type of instances. The biggest difference was obtained in the instances of type low and it corresponds to a difference of 0.63%. Regarding the computational time, the y-separation was faster in the instances of types $reference$, low and $mixed$, while the 1-separation was more efficient in instance of type $high$, however, the difference in computational time for the instances of type $high$ is not meaningful, as the y-separation, which is the less efficient algorithm for this instance type, took an average of 8 seconds. Consequently, we will use the y-separation to obtain the optimal values of the instances from the instance set 3. Considering the results obtained with the y-separation, we verify that the algorithm is more efficient, in terms of computational time, for the instances of type $high$ and less efficient for the instances of type low, the opposite of what happened in the instance set 2. Regarding the gap values, as expected, the highest average gap was obtained for the instances of type low and the lowest average gap was obtained in instances of type $mixed$. By observing the results obtained with the 1-separation, we verify that the referred method provided the highest average gap for instances of type low and the lowest average gap for instances of type $mixed$. Regarding the computational time, the 1-separation took a significantly higher amount of time to solve the instances of type $reference$ than any other type of instances. When we observe in detail Table B.7, we conclude that this increase in the computational time in the instances of type $reference$ is mostly due to instances $ftv170\_1$ and $rbg323\_1$.

Table 5.18 shows the optimal values of the instances from the instance set 3 and it has the same layout as Table 5.14.

Table 5.18: Optimal values for the instance set 3 with y-separation.

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
br17_1	31	0	0	14	46
br17_2	28	0	0	46	73
br17_3	39	0	0	15	48
br17_4	36	0	0	10	129
ftv33_1	868	0	5	28	205
ftv33_2	401	1	0	52	94
ftv33_3	1286	0	0	7	73
ftv33_4	829	0	0	10	111
ftv35_1	1008	0	0	10	263
ftv35_2	530	1	0	35	161
ftv35_3	1232	0	18	12	243
ftv35_4	1008	0	0	8	347
ftv38_1	830	0	0	9	93
ftv38_2	391	1	0	54	111
ftv38_3	1449	0	0	10	289
ftv38_4	774	0	0	16	124
p43_1	5483	0	0	31	123
p43_2	5473	1	0	210	341
p43_3	5530	0	0	4	86
p43_4	5492	0	0	11	119
ftv44_1	996	1	0	27	220
ftv44_2	625	0	3	132	277
ftv44_3	1343	0	0	13	207
ftv44_4	998	1	60	86	678
ftv47_1	1179	0	22	62	529
ftv47_2	729	1	37	268	623
ftv47_3	1472	1	32	28	385
ftv47_4	1099	0	0	9	135
ry48p_1	10318	1	1	75	483
ry48p_2	6787	1	5	344	314
ry48p_3	12752	1	81	38	372

Continues on the next page

Table 5.18

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
ry48p_4	10139	1	13	61	512
ft53_1	3572	0	3	52	348
ft53_2	2819	2	41	330	1009
ft53_3	5972	1	35	27	619
ft53_4	4734	0	4	60	525
ftv55_1	570	1	0	104	346
ftv55_2	365	1	0	492	190
ftv55_3	1021	1	0	38	569
ftv55_4	579	1	0	55	263
ftv64_1	977	2	3	124	879
ftv64_2	660	4	14	728	776
ftv64_3	1617	1	0	1	255
ftv64_4	1515	5	1	18	2287
ft70_1	21226	1	0	26	335
ft70_2	15360	1	0	10	180
ft70_3	29573	1	5	16	416
ft70_4	26817	1	15	22	426
ftv70_1	849	13	180	761	3881
ftv70_2	676	10	102	821	2210
ftv70_3	1342	1	6	6	552
ftv70_4	1261	1	0	18	464
kro124p_1	25251	9	79	103	2486
kro124p_2	12421	38	77	2397	1729
kro124p_3	32025	2	35	29	665
kro124p_4	18552	40	148	2317	3745
ftv170_1	1652	5781	2560	10365	77580
ftv170_2	1108	9071	952	16941	42632
ftv170_3	2215	19	55	95	1765
ftv170_4	1610	141	132	1270	4186
rbg323_1	337	36	12	46	279
rbg323_2	70	6684	41	510	13065

Continues on the next page

Table 5.18

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
rbg323_3	822	11	0	5	100
rbg323_4	335	31	13	21	155
rbg358_1	209	916	121	434	8711
rbg358_2	50	328	68	79	2825
rbg358_3	658	8	0	3	19
rbg358_4	409	73	20	66	528
rbg403_1	345	17	0	7	74
rbg403_2	32	1316	9	610	5318
rbg403_3	1427	134	90	202	910
rbg403_4	486	12	0	4	60
rbg443_1	539	28	0	3	39
rbg443_2			596*		
rbg443_3	1353	42	0	12	210
rbg443_4	729	2138	194	19	4682
average		359	71	534	2615

*Stopped before the time limit due to lack of computational memory.

We were able to obtain the optimal value for all instances from set 3 within the time limit, except for instance $rbg443\_2$, which is not surprising since we could not even obtain its LP relaxation value. The optimal values were obtained in an average time of 359 seconds. The computational time to solve the instances from the instance set 3 up to 70 nodes is negligible, as the maximum computational time is of 13 seconds. Instances $kro124p$ are also solved very efficiently, in an average of 22 seconds. The hardest instances to solve, in terms of computational time, were instances $ftv170$, which took an average of 3753 seconds to obtain their optimal value. Although instances rbg have a much higher number of nodes than instances $ftv170$, as they have at least 323 nodes while instances $ftv170$ have 171, they were faster to solve as they were solved in an average of 785 seconds. This is a consequence of the characteristics of instances rbg that we mentioned previously, namely having several arcs with cost zero.

Table 5.19 shows the average computational time ($\overline{t_s}$) and the maximum computational time (t_{max}), in seconds, to obtain the optimal value for each instance type.

Table 5.19: Statistics for the optimal value by instance type for the instance set 3.

	$\overline{t_s}$	t_{max}
Type $reference$	358	5781
Type low	970	9071
Type $high$	12	134
Type $mixed$	129	2138

As expected, the y-separation algorithm is more efficient, in terms of computational time, when solving instances of type $high$ and less efficient when addressing instances of type low due to the number of visited nodes per family in each type of instances. As we could solve all the instances from the instance set 3 within the time limit, except for instance $rbg443\_2$, we decided to present the maximum computational time to obtain the optimal value for each instance type instead of the number of instances solved. Regarding the maximum computational time, in the instances of type $high$, it was of 134 seconds while for all the other types of instances it was bigger than 2000 seconds, which highlights the fact that the y-separation provides better results, in terms of computational time, when addressing instances with a high number of visits per family.

5.4.4 Generated instances based on asymmetric UTPP instances

In this section we present results for the instance set 4, which corresponds to asymmetric instances generated based on UTPP instances. Table 5.20 shows the average LP relaxation results for the instance set 4 obtained with the y-separation and the 1-separation and it is organized in a similar manner as Table 5.16. The LP relaxation values of each instance from the instance set 4 are available in appendix, Table B.9. To simplify the notation the instance that is presented as $tppinstancename.aftsp\_|N|+1\_L\_i$, with $i \in \{1,\dots,4\}$ in Table A.4 will be designated by $tppinstancename\_|N|+1\_i$ henceforth.

Table 5.20: Average LP relaxation results for the instance set 4.

	$\overline{gap}$	$\overline{t_s}$	$\overline{\#CC}$	$\overline{\#RFV}$
y-separation	0.02%	3	2	100
1-separation	0.01%	4	6	137

The instance set 4 was created since instances rbg have characteristics that make them not interesting to be used as a base for generating asymmetric FTSP instances. In addition, the majority of the other instances from the instance set 3 has a small number of nodes, which makes them easily solvable by the proposed methods. However, the instance set 4 proved to be very easy to solve. In fact, by observing Table 5.20 we verify that both the y-separation and the 1-separation obtained a near optimal average gap. More precisely, when we observe the detailed results of Table B.9 we verify that both separation algorithms obtained the optimal value when solving the LP relaxation in 21 of the 24 instances from the instance set 4. Additionally, in the instances in which the optimal value is not obtained when solving the LP relaxation, the maximum value of gap obtained was 0.29%. Regarding the computational time, both separation algorithms provide the LP relaxation values in a negligible amount of time. These results show that the proposed methods are very efficient to solve the instances from set 4.

Even though both separation algorithms provide identical gap values and computational times for the instances from set 4, we decided to use the y-separation to obtain the optimal values of this instance set, similarly to what was done for the other instance sets. Table 5.21 shows the optimal values of the instance set 4 ($\mathcal{V}$), the computational time, in seconds, to obtain the optimal value (t_s), the number of B&C subproblems solved during the B&C algorithm (#sub) and the number of added violated CC (#CC) and RFV inequalities (#RFV). Table 5.21 also shows, in the last row, the average of the results obtained.

Table 5.21: Optimal values for the instance set 4 with y-separation.

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
AsimSingh50_1	438	0	0	0	38
AsimSingh50_2	211	0	0	2	31
AsimSingh50_3	694	0	0	0	38
AsimSingh50_4	367	0	0	0	39
AsimSingh100_1	1005	1	0	0	178
AsimSingh100_2	465	1	0	9	78
AsimSingh100_3	1350	1	0	0	80
AsimSingh100_4	705	1	0	0	40
AsimSingh150_1	1185	2	0	5	127
AsimSingh150_2	735	3	0	4	151
AsimSingh150_3	1770	2	0	0	187
AsimSingh150_4	885	3	0	6	161
AsimSingh200_1	1545	5	0	10	140

Continues on the next page

Table 5.21

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
AsimSingh200_2	1050	4	0	5	82
AsimSingh200_3	2400	2	0	0	135
AsimSingh200_4	1455	10	0	5	369
AsimSingh250_1	2145	7	0	1	93
AsimSingh250_2	1200	11	0	12	129
AsimSingh250_3	3135	2	0	0	20
AsimSingh250_4	2445	11	0	1	149
AsimSingh300_1	2595	12	0	1	79
AsimSingh300_2	1605	19	0	7	165
AsimSingh300_3	3825	19	0	2	313
AsimSingh300_4	2655	10	0	0	80
$average$		5	0	3	21

As expected from the results of the LP relaxation, we were able to obtain the optimal values of the instances from set 4 in a short amount of computational time. In fact, by observing column #sub we verify that all instances were solved without requiring branching, in a maximum time of 19 seconds.

Summary

Throughout this section we concluded that the best methods to solve the FTSP are the ones that use the RFV inequalities but that do not separate them all, which are the y-separation and the 1-separation. After some testing, we realized that, usually, the y-separation obtains the LP relaxation values in the shortest amount of computational time while the 1-separation obtains the highest LP relaxation values. Nevertheless, the results obtained with the y-separation and the 1-separation were very similar. Regarding the optimal values, we verified that the y-separation is the most efficient method to obtain the optimal value of FTSP instances. With this method we were able to obtain the optimal value of 13 of the 21 instances of the instance set 1. Note that six of the instances from the instance set 1 that we could not solve optimally have a considerable number of nodes, namely 666 and 1002 nodes. The instance set 1 is the set of the benchmark instances generated by Morán-Mirabal et al. (2014) and we were able to obtain the optimal value of instances for which the optimal value was unknown. In particular, we were able to obtain the optimal value of instances $bier$ in a maximum of 49 seconds and the optimal value of instance a_3, with 280 nodes, in 3730 seconds.

Recall that the instance sets 2, 3 and 4 were generated by us and we created four different types of instances in each of those sets. Instances of type $reference$ are considered as the instances of reference, instances of type low have a smaller number of visits per family than instances of type $reference$ while instances of type $high$ have a higher number of visits per family than instances of type $reference$, finally, instances of type $mixed$ are a random combination of instances of type low and $high$.

Regarding the instance set 2, by using the y-separation, we were able to obtain the optimal value of 49 of 64 instances. Considering that the instances of the instance set 2 have a number of nodes varying between 136 and 264, these are very satisfactory results. Due to the nature of the different instance types, we expected instances of type $high$ to be the easiest to solve and the instances of type low the hardest, as the RFV inequalities are more effective when the number of visits is high. However, in this set, the instances of type low were the easiest to solve and the instances of type $reference$ were the hardest. We believe that these results are due to the single-visit families present in the instances of type low and the valid inequalities added that state that we cannot use an arc between two nodes from a single-visit family.

From the instance set 3, we were able to obtain the optimal value of all instances except for instance $rbg443\_2$. Although instances rbg were the instances with the highest dimension from the instance set 3, the most time consuming instances were instances ftv. In the instance set 3, the easiest type of instance to solve in terms of computational time was the type $reference$.

Finally, we were able to obtain the optimal value of all instances from the instance set 4 within the time limit. More precisely, we were able to obtain the optimal value of the referred instances in less than 20 seconds. The y-separation proved to be very effective when addressing this instance set.

We used a directed graph to model the FTSP instead of a nondirected one since it is more versatile. More precisely, with a directed graph and, consequently, formulation we may solve both symmetric and asymmetric instances while the reverse is not true. If we group the results obtained in terms of symmetric and asymmetric instances we verify that from the 85 symmetric instances we were able to obtain the optimal solution of 62 while from the 96 asymmetric instances we were able to solve 95. Therefore, the percentage of symmetric instances solved is of 73% whereas the percentage of asymmetric instances solved is 99%, which shows that the symmetric instances are more challenging to solve with the proposed B&C algorithm.

For the instances that we could not solve optimally within the time limit of 10800 seconds we present, in Table 5.22, a summary of the best upper bounds obtained with the B&C algorithm. These instances will be addressed in Chapter 6 with heuristic methods.

Table 5.22: Best upper bounds obtained with the B&C algorithm.

Instance	Best upper bounds B&C algorithm
Instance set 1	
a_1	1692.92
a_2	1612.39
Instance set 2	
pr144_4	49403
kroA150_3	20880
pr152_3	64425
rat195_1	1285
rat195_2	1814
rat195_4	1320
kroA200_1	16441
kroA200_3	24471
kroB200_3	24088
gr202_4	28039
gr229_1	70741
gil262_1	1529
gil262_3	2012
gil262_4	1773
pr264_2	28748
Instance set 3	
rbg443_2	596

Chapter 6

Heuristic Algorithms

The objective of this chapter is two-fold. Firstly, we want to find an efficient method able to provide good quality feasible solutions for the FTSP instances that the exact methods were unable to solve up to optimality within the time limit, and, secondly, we wish to improve the best upper bounds available in the literature for the benchmark instances, which were obtained by Morán-Mirabal et al. (2014) and are available in Table 6.1. In Table 6.1 the value t_s represents the computational time, in seconds, reported by the authors to obtain the referred best upper bounds.

Table 6.1: Best upper bounds for the benchmark instances obtained by Morán-Mirabal et al. (2014).

Instance	Best upper bounds by Morán-Mirabal et al. (2014)	t_s
a_1	1891.16	218
a_2	1697.48	2701
gr_1	1817.06	6601
gr_2	1443.05	4005
gr_3	1384.18	7200
pr_1	163461.79	21
pr_2	182144.13	9
pr_3	149456.63	228

In order to evaluate the quality of the solutions obtained by the proposed heuristic algorithms we will use the percentage of gap. However, since the instances that we want to solve with the heuristic algorithms have an unknown optimal value we will use the best upper bound obtained by Morán-Mirabal et al. (2014) presented in Table 6.1 to compute the percentage of gap. Therefore, the formula used to compute the percentage of gap in the subsequent sections is the following: $gap = 100 \times (\textit{heuristic solution} - \textit{best upper bound})/\textit{best upper bound}$.

We developed three distinct heuristic algorithms for the FTSP. A genetic algorithm which uses permutations as chromosomes and is presented in Section 6.1. An iterated local search algorithm that iterates between a local search algorithm and a perturbation algorithm, which is presented in Section 6.2, and a hybrid algorithm, presented in Section 6.3, which combines the CC+RFV model, using the y-separation, with heuristic procedures. We conclude with Section 6.4 where we present the results obtained with the best heuristic algorithms proposed.

6.1 The genetic algorithm

Genetic algorithms were created with the aim of imitating the theory of evolution. Usually, genetic algorithms are composed by: (i) chromosomes, which encode the solution of the problem; (ii) a fitness function, which measures the quality of each chromosome; (iii) a population, which is composed by several chromosomes, also called individuals; (iv) selection, which is the measure used to select the fittest individuals; and (v) crossover and mutation, which allow us to generate new individuals - the children. For a more detailed explanation of genetic algorithms see, for instance, Goldberg and Holland (1988); Holland (1992).

Bernardino and Paias (2018b) proposed two genetic algorithms for the UTPP, one using permutations as chromosomes and other using random keys, and concluded that the former proved to be a more efficient method when the routing part of the problem takes precedence over the acquisition part. Even though Morán-Mirabal et al. (2014) developed a biased random key genetic algorithm for the FTSP which proved to be less effective when compared to the other heuristic method proposed, which was a GRASP heuristic with a path-relinking procedure, we decided to develop a genetic algorithm for the FTSP which uses permutations as chromosomes, since in the FTSP we wish to establish a circuit and Bernardino and Paias (2018b) obtained good results in a similar problem.

Throughout this section we present several versions of the algorithms created until we obtained the one we consider is the best. We also present some computational results to justify some parameter choices, which led to the creation of other algorithms. We start by presenting the genetic algorithm.

As mentioned previously, each chromosome is a permutation of the nodes that are family members, that is, a permutation of the set $\{1, \dots, |N|\}$. The depot is where the circuit must start and end and, therefore, there is no need to include it in the chromosome. Each value of the chromosome is called an allele. The decoding of a permutation into a feasible solution for the FTSP that we use is the one used in the random constructive heuristic described in Section 3.3 and that results in a solution that visits the nodes by the same order as they appear in the chromosome. For a more detailed explanation of the decoder we refer the reader to Section 3.3, more specifically, to Example 28.

The fitness value of a chromosome is the cost of the solution obtained when the chromosome is decoded. Thus, the fittest individuals are the ones with the lowest fitness value.

The initial population has η individuals randomly generated. The parents are selected by using the tournament method, which consists in choosing, at random, τ individuals from the population and selecting the individual with the best fitness value to enter the mating pool. The tournament method is repeated until we have selected all the parents necessary to originate the descendants. The crossover operator chosen was the known order crossover (OX) operator for permutations (see, e.g., Goldberg and Tarjan, 1988). In the crossover OX we mate two parents, p_1 and p_2, and we obtain two children, c_1 and c_2. We start by generating at random two crossover points. Child c_1 inherits the portion of chromosome between the two crossover points from parent p_2 while child c_2 inherits it from parent p_1. The remaining alleles of child c_1 are filled, with the nodes that are not in the chromosome yet, by the same order as they appear in parent p_1, starting at the crossover point with the highest value. The same applies for child c_2, but it inherits the order from parent p_2. We generate ζ children to be part of the next population. The children replace the ζ individuals with the highest fitness function value, that is, the less fit individuals. Since we use a crossover operator that generates two children, ζ ought to be an even number.

Example 30 (The OX operator)**.** Consider parents $p_1 = (1, 2, 3, 4, 5)$ and $p_2 = (3, 5, 4, 2, 1)$ associated with the FTSP instance presented in Figure 3.1. Additionally, consider that the crossover points are 1 and 4. Therefore, child $c_1 = (*, 5, 4, 2, *)$ and child $c_2 = (*, 2, 3, 4, *)$, as child c_1 inherits the portion of chromosome between the crossover points from parent p_2 and child c_2 inherits it from parent p_1. Now, to fill the remaining alleles of the chromosome c_1, we start with the allele 5 from the parent p_1. Since 5 is already in c_1 we go to the next allele, in this case the allele 1. As 1 is not in c_1 we obtain $c_1 = (*, 5, 4, 2, 1)$. Then, we keep going through the alleles until we reach the one that contains 3 and we obtain $c_1 = (3, 5, 4, 2, 1)$. Analogously, $c_2 = (5, 2, 3, 4, 1)$.

To add randomness to the population, the children suffer a mutation with probability ϵ. The mutation consists in determining two random mutation points and flipping the alleles of the chromosome between the mutation points.

Example 31 (The mutation operator)**.** Consider the chromosome $(1, 2, 3, 4, 5)$ associated with the FTSP instance presented in Figure 3.1 and the mutation points 1 and 4. The chromosome that we obtain after applying the mutation operator is $(1, 4, 3, 2, 5)$.

This process of selecting parents, mating them and inserting the children in the population is repeated for a fixed number of iterations. The pseudocode for the genetic algorithm, which will be designated by algorithm GA, is available in Algorithm 6.1. In order to evaluate the quality of the solutions provided by the GA algorithm we decided to experiment two sets of parameters, which differ significantly on the size of the population. The set of parameters 1 is characterized by the following setting: $\eta^1 = 20$, $\tau^1 = 4$, $\zeta^1 = 16$, $\epsilon^1 = 0.25$, while the set of parameters 2 contains the following parameters: $\eta^2 = 120$, $\tau^2 = 10$, $\zeta^2 = 100$, $\epsilon^2 = 0.50$. The number of iterations performed with both parameter sets is 10000. Table 6.2 compares the two sets of parameters both in terms of quality of the solutions obtained and of computational time. This table shows the average percentage of gap between the best solution provided by the GA algorithm and the best upper bound obtained by Morán-Mirabal et al. (2014) ($\overline{gap}$) and the average computational time, in seconds, to obtain the best solution using the GA algorithm ($\overline{t_s}$). The detailed results are available in appendix, Table C.1.

Algorithm 6.1 The basic framework of the genetic algorithm.

```
Require: $\eta$, $\zeta$, $\tau$, $\epsilon$
1: Initialize the population with $\eta$ random individuals (permutations).
2: while The maximum number of iterations is not reached do
3:     Select $\zeta$ parents from the population using the tournament method.
4:     Mate the parents to produce children, that is, to each pair of parents apply the crossover operator OX.
5:     Apply to children, with probability $\epsilon$, the mutation operator.
6:     Substitute the $\zeta$ individuals of the population with the highest fitness value by the children.
7: end while
```

Table 6.2: Experimenting parameter sets for the GA algorithm.

	$\overline{gap}$	$\overline{t_s}$
Parameter set 1	584.07%	1
Parameter set 2	330.35%	15

Average obtained with instances a_1, a_2, gr and pr.

The average gap obtained with the GA algorithm with both parameter settings was of at least 300%, which means that the best solution provided by the GA algorithm is too expensive when compared to the best upper bounds obtained by Morán-Mirabal et al. (2014). In fact, the cost of the solutions obtained with the GA algorithm is more than the triple of the best upper bound provided by Morán-Mirabal et al. (2014). Note that the decoder used is a surjective function but not an injective one as every feasible solution has several chromosomes that could be decoded into it. Due to this property, it is possible that different chromosomes have the same fitness value. Additionally, we verified that the values of the elements of the cost matrices associated with the benchmark instances vary significantly. More precisely, in instances pr, the cost of the arc $(42, 43)$ is 100 while the cost of the arc $(58, 65)$ is 18200.30, which implies that structurally similar solutions may have costs that are significantly different. The parameter set 2 provided a lower average gap then the parameter set 1. Regarding the computational time, since the parameter set 2 has a bigger population, it was expected that the parameter set 1 would be faster. Nonetheless, 15 seconds is a reasonable computational time considering the dimension of the instances we are addressing. Therefore, we decided to choose the parameter set 2.

By observing the detailed results in Table C.1, we verify that the percentage of gap increases with the instance's dimension, which led us to believe that the GA algorithm did not converge. In order to overcome this situation we could either increase the number of iterations, which would make our algorithm less efficient in terms of computational time, or populate the initial population with solutions of good quality. We decided to adopt the latter methodology, so we populated the initial population with feasible solutions for the FTSP obtained by using the random nearest neighbor heuristic presented in Section 3.3, which originated the GA+NN algorithm.

Table 6.3 shows a summary of the results obtained by using the GA+NN algorithm. We experimented generating 60 and 120 individuals with the random nearest neighbor, which corresponds to changing the Step 1 of Algorithm 6.1 to generating 60 individuals of η by using the random nearest neighbor or to generating 120 individuals of η by using the random nearest neighbor, respectively. The detailed results are available in appendix, Table C.2. Table 6.3 shows the average percentage of gap between the best solution provided by the GA+NN algorithm and the best upper bound obtained by Morán-Mirabal et al. (2014) ($\overline{gap}$) and the average computational time, in seconds, to obtain the best solution by the GA+NN algorithm ($\overline{t_s}$).

Table 6.3: Experimenting generating individuals for the GA+NN algorithm.

	$\overline{gap}$	$\overline{t_s}$
Generate 60 individuals	11.28%	31
Generate 120 individuals	11.34%	46

Average obtained with instances a_1, a_2, gr and pr.

The results obtained with the GA+NN algorithm were satisfactory since the average percentage of gap plummeted. In fact, the average gap decreased more than 300%. Even though the difference between generating 60 or 120 individuals with the random nearest neighbor is not very significant, generating only 60 individuals provided a sightly lower average percentage gap. This is not surprising since it is known that genetic algorithms behave better when the population is varied. When we generate all the population, that is, 120 individuals, with the random nearest neighbor the likelihood of obtaining similar solutions is high since the random nearest neighbor is a greedy constructive heuristic. However, when we generate 60 individuals with the random nearest neighbor the rest of the population is more varied. Additionally, generating 60 individuals also leads to a slightly lower computational time. Consequently, all the results presented subsequently were obtained using the GA+NN algorithm with 60 individuals in the initial population generated with the random nearest neighbor and the remaining 60 individuals are randomly generated chromosomes.

To further improve the quality of the solutions provided by the GA+NN algorithm we developed a local search procedure, which is presented next.

The local search procedure

The local search procedure uses as input a feasible solution for the FTSP. This procedure searches the three neighborhoods $\mathcal{N}_I$, $\mathcal{N}_O$ and 2-opt, which were introduced in Section 3.3, and has two random procedures called $switchOutRandom$ and $insertAllRemoveExtra$, which have the purpose of adding some randomness to the local search and will be presented further on. Recall that the neighborhood $\mathcal{N}_I$ consists in switching two nodes that belong to the solution, the neighborhood $\mathcal{N}_O$ in switching a visited node with a non-visited node from the same family and, when we have a symmetric cost matrix, the neighborhood 2-opt in inverting the order of a path in the solution.

The first local search procedure that we developed is very simple as it only consists in searching neighborhoods $\mathcal{N}_I$, 2-opt and $\mathcal{N}_O$ iteratively during a predefined number of iterations and then, once the iterations are complete, we search neighborhood $\mathcal{N}_I$ and 2-opt once more. The algorithm used to search the neighborhoods is the one presented in Algorithm 3.1 in Section 3.3. This local search procedure, which will be designated by LS algorithm, is applied to the best solution found by the GA+NN algorithm, that is, to the solution with the lowest cost. Algorithm 6.2 shows the pseudocode of the LS algorithm.

Table 6.4 shows the average percentage of gap between the best solution provided by the LS algorithm and the best upper bound obtained by Morán-Mirabal et al. (2014) ($\overline{gap}$) and the average computational time, in seconds, to obtain the best solution ($\overline{t_s}$). The results presented in Table 6.4 were obtained after 1000 and 5000 iterations of the LS algorithm. The detailed results are available

Algorithm 6.2 The LS algorithm.

```
Require: A feasible solution $s$ for the FTSP.
 1: Set $number\_iterations = 0$.
 2: while $number\_iterations < maximum\_number\_iterations$ do
 3:     Search $\mathscr{N}_I(s)$ and obtain $s^*$. Set $s = s^*$.
 4:     if The cost matrix is symmetric then
 5:         Search 2-opt(s) and obtain $s^*$. Set $s = s^*$.
 6:     end if
 7:     Search $\mathscr{N}_O(s)$ and obtain $s^*$. Set $s = s^*$.
 8: end while
 9: Search $\mathscr{N}_I(s)$ and obtain $s^*$. Set $s = s^*$.
10: if The cost matrix is symmetric then
11:     Search 2-opt(s) and obtain $s^*$. Set $s = s^*$.
12: end if
Ensure: Solution $s^*$ such that $Cost(s^*) \leq Cost(s)$.
```

in appendix, Table C.3.

Table 6.4: Experimenting the LS algorithm.

	$\overline{gap}$	$\overline{t_s}$
1000 iterations	-2.49%	33
5000 iterations	-2.49%	37

Average obtained with instances a_1, a_2, gr and pr.

The average gap obtained by applying the LS algorithm is significantly lower than the one obtained with the GA+NN algorithm. In fact, the average gap decreased 13.77%. Moreover, the average gap is negative which means that the LS algorithm provided solutions with a lower cost than the best upper bounds from the literature. More precisely, when we observe Table C.3 we verify that we were able to improve the best upper bound available in the literature of five benchmark instances with unknown optimal values, performing both 1000 and 5000 iterations. Regarding the computational time, if we consider the 1000 iterations we verify that the heuristic algorithm proposed is very efficient as it takes a maximum of 61 seconds to obtain solutions with a lower cost than the best upper bounds obtained by Morán-Mirabal et al. (2014) considering the highest dimensioned benchmark instances. The increase in the number of iterations of the LS algorithm from

1000 iterations to 5000 did not provide better quality solutions, which makes us believe that the local search procedure converged to a local optimum and it was unable to escape it.

In an attempt to improve the quality of the solutions obtained with the LS algorithm we decided to randomize it so that it could escape from local optima. Thus, we developed the procedure $switchOutRandom$, which is based on the search of neighborhood $\mathcal{N}_O$, as we replace one node that belongs to the solution by one node that does not, from the same family. The main difference is that, the node that is inserted in the solution is chosen randomly. Then, we remove from the solution the node from the same family that originates the switch that leads to the lowest increase in the solution value. Note that this procedure does not guarantee that the solution obtained has a lower cost than the initial solution.

The LS_random algorithm is obtained by replacing the search of the neighborhood $\mathcal{N}_O$ with the procedure $switchOutRandom$ in the LS algorithm. This algorithm is applied to the best solution found by the GA+NN algorithm. Table 6.5 shows the average percentage of gap between the best solution provided by the LS_random algorithm and the best upper bound obtained by Morán-Mirabal et al. (2014) ($\overline{gap}$) and the average computational time, in seconds, to obtain the best solution with the LS_random algorithm ($\overline{t_s}$). The results shown in Table 6.5 were obtained after 1000 iterations of the LS_random algorithm and the detailed results are available in appendix, Table C.4.

Table 6.5: Experimenting the LS_random algorithm.

	$\overline{gap}$	$\overline{t_s}$
1000 iterations	-0.48%	42

Average obtained with instances a_1, a_2, gr and pr.

The solutions obtained with the LS_random algorithm are worse than the ones obtained with the LS algorithm considering the same number of iterations. More precisely, the LS_random provided an average gap of -0.48% whereas the LS algorithm obtained an average gap of -2.49%. Nonetheless, by using the LS_random we continue to obtain a negative average gap. In fact, we were able to improve the best upper bound available in the literature of four benchmark instances. Due to these results we realized that, probably, the $switchOutRoute$ algorithm did not provided enough randomness to the algorithm, which motivated the creation of the procedure $insertAllRemoveExtra$. This procedure was designed not only to add more randomness to the local search procedure but also to accelerate its convergence. The main idea of the procedure is to make the solution unfeasible and then restore its feasibility, in the hope of guiding the search of the solution space in a different direction.

In the $insertAllRemoveExtra$ procedure we start by destroying a feasible solution by inserting extra nodes in the solution. We randomly choose a node that does not belong to the solution and determine to which family it belongs. Let $l \in \mathcal{L}$ be that family. Then, we insert all the nodes from family l, which do not belong to the solution, in the solution, in the best possible position, that is, in the position where the insertion leads to the lowest increase in the solution cost. Now, we have an unfeasible solution since we are visiting n_l nodes from family l and we are only required to visit v_l. Note that $n_l \neq v_l$ as we chose the family based on the non-visited nodes, which implies that there exists at least one node from the chosen family that is not in the solution. In order to restore the solution's feasibility we must remove the extra nodes. As it was said before, we need to remove $n_l - v_l$ nodes from family l. In order to do that we compute the value of removing every node from family l and we remove the node which the removal originates the biggest decrease in the solution cost. This process is repeated until we have v_l nodes from family l in the solution.

A new local search procedure, called LS_insertRemove, is obtained by replacing the search of neighborhood $\mathcal{N}_O$ with the procedure $insertAllRemoveExtra$ in the LS Algorithm 6.2. The algorithm LS_insertRemove is applied to the best solution found by the GA+NN algorithm. Table 6.6 shows the average percentage of gap between the best solution provided by the LS_insertRemove algorithm and the best upper bound obtained by Morán-Mirabal et al. (2014) ($\overline{gap}$) and the average computational time, in seconds, to obtain the best solution with the LS_insertRemove algorithm ($\overline{t_s}$). These results were obtained performing 1000 and 5000 iterations of the LS_insertRemove algorithm. The detailed results are available in appendix, Table C.5.

Table 6.6: Experimenting the LS_insertRemove algorithm.

	$\overline{gap}$	$\overline{t_s}$
1000 iterations	-8.19%	52
5000 iterations	-8.19%	141

Average obtained with instances a_1, a_2, gr and pr.

With the LS_insertRemove algorithm we obtained a significant improvement in the average gap. More precisely, considering 1000 iterations, we were able to decrease the average percentage of gap from -2.49%, which were obtained with the LS algorithm, to -8.19%. By observing the detailed results in Table C.5 we verify that the solution obtained with the LS_insertRemove algorithm is better than the best upper bound available in the literature for all benchmark instances, except for the instance a_2. By performing 5000 iterations, the average gap is the same but the computational time is approximately three times bigger. Nonetheless, 141 seconds continues to be a reasonable computational time considering that we are addressing instances with 1002 nodes. Instead of increasing the number of iterations of the $LS\_insertRemove$ algorithm we experimented applying it to more solutions present in the final population of the GA+NN algorithm. More precisely, we decided to apply the $LS\_insertRemove$ algorithm to 12 solutions from the final population of the GA+NN algorithm. These solutions are varied as they span from the best solution to the worst solution present in the final population of the GA+NN algorithm. With this procedure and considering the benchmark instances a_1, a_2, gr and pr, the average percentage of gap obtained was of -9.83% and the average computational time was of 305 seconds. The detailed results obtained when we applied the $LS\_insertRemove$ algorithm to several solutions from the final population of the NN+GA algorithm are available in appendix, Table C.6. When we compare these results to the ones obtained with 5000 iterations of the $LS\_insertRemove$ algorithm, which were the best results obtained so far, we verify that the average gap decreased 1.64% and that the average computational time increased by a factor of two. By applying the LS_insertRemove algorithm to several solutions we were able to obtain solutions with a lower value than the best upper bound obtained by Morán-Mirabal et al. (2014) for all benchmark instances with an unknown optimal value, which are very good results. We could experiment applying the $LS\_insertRemove$ algorithm to several solutions and performing 5000 iterations, however, from our experience, the increase in the number of iterations of the local search procedure does not improve significantly the quality of the solutions obtained but increases the computational time considerably.

Even though the best heuristic method presented until now consists in applying the local search procedure LS_insertRemove to 12 solutions present in the final population obtained with the genetic algorithm GA+NN, we will refer to it as genetic algorithm to be easily identified.

6.2 The iterated local search algorithm

We decided to develop an iterated local search (ILS) algorithm since the local search procedure that provided the best results in the genetic algorithm presented in Section 6.1 was the LS_insertRemove algorithm, which consists in a neighborhood search method mixed with a random procedure. The purpose of the LS_insertRemove algorithm was to use the random procedure to escape from local optima, which corresponds to the basic idea of an ILS algorithm.

As mentioned in the previous paragraph, the main idea behind the ILS algorithm is to find a local optimum, by using a local search procedure, and then to apply a perturbation method in order to escape from that local optimum and continue the search of the solution space. This process is done iteratively, for example during a fixed number of iterations. For more details on ILS algorithms see, for instance, Lourenço et al. (2003) and Boussaïd et al. (2013). Algorithm 6.3 shows the basic framework of an ILS algorithm with generic procedures. The procedures used in practice are presented throughout this section as well as some computational results to justify the parameter choices made. Note that every time we are testing a parameter the rest of the ILS algorithm remains the same, the only change is the parameter under testing.

Algorithm 6.3 The basic framework of the ILS algorithm.

```
1: Determine a feasible solution $s$ for the FTSP.
2: Apply a local search algorithm to $s$ and obtain a new solution $s^*$.
3: $number\_iterations = 0$
4: while $number\_iterations < maximum\_number\_iterations$ do
5:     Apply a perturbation procedure to $s^*$ and obtain solution $p$.
6:     Apply a local search procedure to $p$ and obtain a new solution $p^*$.
7:     if $Cost(p^*) < Cost(s^*)$ then
8:         Update the best solution to $p^*$.
9:     end if
10:    $s^* = p^*$.
11: end while
```

In order to obtain an initial feasible solution for the FTSP we use the nearest neighbor heuristic presented in Section 3.3. The local search procedure used in the ILS algorithm consists in searching the neighborhoods $\mathscr{N}_I$, $\mathscr{N}_O$ and 2-opt, which were presented in Section 3.3. Note that we only search the neighborhood 2-opt if we have a symmetric cost matrix since the referred neighborhood is only easily searched when the cost of the arcs (i, j) and (j, i) are the same. Additionally, recall that the neighborhood $\mathscr{N}_I$ consists in switching two nodes in the circuit and the neighborhood $\mathscr{N}_O$ consists in switching a visited node with a non-visited node from the same family. The pseudocode of the neighborhood search is available in Algorithm 3.1 presented in Section 3.3 while the pseudocode for the local search algorithm used in the ILS algorithm is available in Algorithm 6.4. Note that this local search is different from the local search presented in Algorithm 6.2 used in the genetic algorithm since this algorithm searches the neighborhoods until reaching a local optimum in all of them while Algorithm 6.2 searches the neighborhoods during a fixed number of iterations.

All there is left now is to present the perturbation method. The idea behind the perturbation method is similar to the idea of the procedure $insertAllRemoveExtra$ presented in Section 6.1, which consists in making a feasible solution for the FTSP unfeasible by inserting nodes and then restore its feasibility by removing nodes. The perturbation method has as input a feasible solution

Algorithm 6.4 The local search procedure used in the ILS algorithm.

```
Require: A feasible solution $s$ for the FTSP.
1: $cost\_old = Cost(s)$.
2: $cost\_new = -\infty$.
3: while $cost\_old > cost\_new$ do
4:     $cost\_old = Cost(s)$.
5:     Search $\mathscr{N}_I(s)$ and obtain $s^*$. Set $s = s^*$.
6:     Search $\mathscr{N}_O(s)$ and obtain $s^*$. Set $s = s^*$.
7:     if The cost matrix is symmetric then
8:         Search 2-opt(s) and obtain $s^*$.
9:     end if
10:    Set $cost\_new = Cost(s^*)$ and $s = s^*$.
11: end while
Ensure: Solution $s^*$ such that $Cost(s^*) \leq Cost(s)$.
```

for the FTSP, which will most likely become unfeasible since we will insert v_l nodes, from each family $l \in \mathcal{L}$, chosen according to a criterion. These chosen nodes are inserted in the solution in the best possible position, that is, in the position that leads to the lowest increase in the solution cost. After inserting the chosen nodes, we must remove the extra nodes to restore the solutions' feasibility, which is also done according to a criterion. The subroutines used in the perturbation method are presented next.

We start by presenting the choosing phase of the perturbation method, which, essentially, consists in choosing v_l nodes from each family $l \in \mathcal{L}$. As in the FTSP there are no costs associated with the nodes, we decided to create metrics which allow us to establish relationships between them. Therefore, we created three different metrics: (i) $mean$ in which the cost of node $i \in N$ is the average cost of the arcs that have i as initial node, that is, $\frac{\sum_{j:(i,j)\in A} c_{ij}}{|N|}$; (ii) min which assigns to the node $i \in N$ the cost of the arc with the lowest cost between the arcs that have i as initial node, that is, $\min_{j:(i,j)\in A}\{c_{ij}\}$; and (iii) max which assigns to node $i \in N$ the cost of the arc with the highest cost between the arcs that have i as initial node, that is, $\max_{j:(i,j)\in A}\{c_{ij}\}$. We developed five different criteria to choose the nodes to be inserted in the solution, three of them based on the metrics presented previously, which are:

- $Mean$: Choose the nodes with the lowest metric $mean$.
- Min: Choose the nodes with the lowest metric min.

- Max: Choose the nodes with lowest metric max.
- $Random\_choice$: Choose the nodes randomly.
- $Least\_choosen$: Choose the nodes that were chosen fewer times during the ILS algorithm.

The $Least\_chosen$ criterion is the only criterion that is updated during the ILS algorithm.

Table 6.7 compares the several choosing criteria in terms of quality of the solution obtained and of computational time. Table 6.7 shows the average gap between the best solution obtained with the ILS algorithm and the best upper bound provided by Morán-Mirabal et al. (2014) ($\overline{gap}$) and the average computational time, in seconds, to obtain the best solution with the ILS algorithm ($\overline{t_s}$). The results were obtained performing 1000 iterations of the ILS algorithm and maintaining all the other parameters unchanged. The detailed results are available in appendix, Tables C.7 and C.8.

Table 6.7: Comparison of the different choosing criteria in the perturbation method.

	$\overline{gap}$	$\overline{t_s}$
Criterion $Mean$	-3.75%	98
Criterion Min	-4.83%	542
Criterion Max	-3.09%	105
Criterion $Random\_choice$	-9.73%	177
Criterion $Least\_chosen$	-9.74%	172

Average obtained with instances a_1, a_2, gr and pr.

From Table 6.7 we verify that all the criteria were able to obtain a negative average gap, which means that we were able to improve, on average, the best upper bounds available in the literature with all the proposed choosing criteria. The criteria $Random\_choice$ and $Least\_chosen$ provided solutions of better quality. In fact, the average gap obtained with the criterion $Random\_choice$ and $Least\_chosen$ was of -9.74% whereas the average gap obtained with the criteria $Mean$, Min and Max was of -3.89%. Consequently, we will not consider criteria $Mean$, Min and Max any longer. Criteria $Random\_choice$ and $Least\_chosen$ are identical in terms of average gap. Regarding the computational time, the criterion $Least\_chosen$ is only slightly faster. Essentially, both criteria $Random\_choice$ and $Least\_chosen$ produce similar results and, thus, we will continue to consider both of them for now.

As explained before, after choosing the nodes according to the criteria presented in the insertion phase, we insert them in the solution in the best possible position. Note that the chosen nodes that already belong to the solution do not need to be inserted. After the insertion of the chosen nodes we will, most likely, have an unfeasible solution. Therefore, in order to restore the solution's feasibility we must remove the extra nodes, which corresponds to the removal phase of the ILS algorithm. We also developed two different removal criteria, which are:

- $Greedy$: Remove the nodes that lead to the highest decrease in the solution cost.
- $Random\_removal$: Remove nodes chosen randomly.

Table 6.8 shows the average gap between the best solution obtained with the ILS algorithm and the best upper bound obtained by Morán-Mirabal et al. (2014) ($\overline{gap}$) and the average computational time, in seconds, to obtain the best solution with the ILS algorithm ($\overline{t_s}$) using as removal criterion the criterion $Random\_removal$ and using as choosing criterion both criteria $Random\_choice$ and $Least\_chosen$. The results were obtained performing 1000 iterations of the ILS algorithm. The detailed results are available in appendix, Table C.9.

Table 6.8: Evaluation of the removal criterion $Random\_removal$.

	$\overline{gap}$	$\overline{t_s}$
Criterion $Random\_choice$	-2.02%	126
Criterion $Least\_chosen$	-1.79%	148

Average obtained with instances a_1, a_2, gr and pr.

Regarding the removal criterion $Greedy$, the results are the ones presented in Table 6.7, which provided an average gap of -9.73% and -9.74% for the choosing criteria $Random\_removal$ and $Least\_chosen$, respectively. The results for the removal criterion $Random\_removal$ are significantly worse than the ones obtained with the removal criterion $Greedy$. More precisely and considering the choosing criteria $Random\_choice$ and $Least\_chosen$, the removal criterion $Random\_removal$ provided an average gap of -1.70% while the removal criterion $Greedy$ provided an average gap of -9.74%. Thus, we will not use the removal criterion $Random\_removal$.

We experimented combining both removal criteria to increase the randomness of the removal phase as we already eliminated the removal criterion $Random\_removal$ due to the worse results obtained. Therefore, we decided to use the removal criterion $Greedy$ but once in every ρ iterations we apply the criterion $Random\_removal$. Table 6.9 shows the results obtained using several ρ values and using the choosing criteria $Random\_choice$ and $Least\_chosen$. This table shows the average gap between the best solution obtained with the ILS algorithm and the best upper bound obtained by Morán-Mirabal et al. (2014) ($\overline{gap}$) and the average computational time, in seconds, to obtain the best solution with the ILS algorithm ($\overline{t_s}$). The results presented were obtained performing 1000 iterations of the ILS algorithm. The detailed results obtained by using as choosing criterion the criterion $Random\_choice$ are available in appendix, Tables C.10 and C.11, while the ones obtained by using the choosing criterion $Least\_chosen$ are available in Tables C.12 and C.13.

Table 6.9: Combining both removal criteria using several ρ values.

	$Random\_choice$		$Least\_chosen$	
	$\overline{gap}$	$\overline{t_s}$	$\overline{gap}$	$\overline{t_s}$
$\rho = 200$	-10.21%	148	-10.09%	164
$\rho = 100$	-9.73%	165	-9.56%	185
$\rho = 50$	-10.58%	165	-10.15%	169
$\rho = 25$	-10.91%	166	-10.29%	157
$\rho = 10$	-10.91%	163	-11.38%	176
$\rho = 5$	-10.59%	242	-10.67%	176

Average obtained with instances a_1, a_2, gr and pr.

The combination of both removal criteria provides better average gap values than using only the removal criterion $Greedy$ for all values of ρ tested, except for $\rho = 100$. Therefore, we will use the combination of both removal criteria instead of using the removal criterion $Greedy$. Both choosing criteria $Random\_choice$ and $Least\_chosen$ continue to provide similar results in terms of average gap and computational time, except for $\rho = 5$ in which the difference in the average computational time is of 66 seconds. Since the lowest average gap was obtained considering $\rho = 10$ and using the choosing criterion $Least\_chosen$ we decided to use this setting.

To summarize, the best parameter setting for the perturbation phase of the ILS algorithm according to the test results is to use the choosing criterion $Least\_chosen$ and a combination of both removing criteria, that is, we use the criterion $Greedy$ but once in every $\rho = 10$ iterations we use criterion $Random$. Algorithm 6.5 shows the pseudocode of the perturbation method, specified for the chosen setting.

Table 6.10 compares the results obtained with the ILS algorithm, using the setting described previously, considering different numbers of iterations, namely 1000 and 5000 iterations. This table shows the average gap between the best solution obtained with the ILS algorithm and the best upper bound obtained by Morán-Mirabal et al. (2014) ($\overline{gap}$) and the average computational time, in seconds, to obtain the best solution with the ILS algorithm ($\overline{t_s}$). The complete results obtained performing 1000 iterations are available in appendix, Table C.11, whereas the ones obtained per-

Algorithm 6.5 The perturbation method used in the ILS algorithm.

```
Require: A feasible solution $s$ for the FTSP.
1: for all $l \in \mathcal{L}$ do
2:     Choose $v_l$ nodes from family $l$ according to the choosing criterion $Least\_chosen$.
3:     Insert the chosen nodes in $s$ in the best possible position.
4: end for
5: Remove the extra nodes by using the combination of both removal criteria: apply criterion
   $Greedy$ but once in every $10$ iterations apply criterion $Random$.
```

forming 5000 iterations are available in Table C.14.

Table 6.10: Comparing different number of iterations in the ILS algorithm.

	$\overline{gap}$	$\overline{t_s}$
1000 iterations	-11.43%	207
5000 iterations	-12.20%	735

Average obtained with instance a_1, a_2, gr and pr.

As expected, when we increase the number of iterations the average percentage of gap decreases whilst the computational time increases. In particular, the average computational time increased by a factor of approximately four, while the average gap decreased 0.92%. By observing in detail Tables C.11 and C.14, we verify that we were able to obtain a solution with a lower cost than the best upper bound from the literature for all benchmark instances with unknown optimal value performing both 1000 and 5000 iterations. There is a clear trade-off between the quality of the solutions obtained and the computational time. Therefore, the number of iterations that we should use depends only on the purpose of the method. If we wish to have a more efficient method that still provides solutions of good quality it is preferable to use less iterations. However, as we wish to improve the best upper bound obtained by Morán-Mirabal et al. (2014) as much as possible, within a reasonable computational time, we decided to perform 5000 iterations of the ILS algorithm.

6.3 The hybrid algorithm

In this section we present the hybrid algorithm, which consists in combining the methods that provided the best results gathered so far. For a more detailed explanation of hybrid heuristics see, for example, Raidl and Puchinger (2008) and Raidl (2015). On the one hand, the CC+RFV model with the y-separation proposed in Chapter 4 is very efficient when addressing benchmark instances up to 127 nodes as it is able to provide its optimal solution in less than 50 seconds, however, it cannot solve the highest dimensioned benchmark instances due to the number of variables associated with them. On the other hand, the ILS algorithm is able to improve the best upper bound available in the literature of all benchmark instances, obtaining the best results so far with the lowest average gap of -12.20%.

The hybrid algorithm combines the referred exact model and the ILS algorithm and has two phases: the constructive phase, where we use the exact method, and the improvement phase, where we apply the ILS algorithm. The purpose of the constructive phase is to obtain a feasible solution for the FTSP. In order to do so we use the exact method to provide an intelligent initialization for the algorithm. Nonetheless, as we are addressing high dimensioned instances, the exact method only considers and solves a partial problem. In order to complete the partial solution obtained with the exact method, we use a savings heuristic. The purpose of the second phase is to improve the solution obtained in the first phase. Consequently, due to the successful results achieved previously, we apply the ILS algorithm to the solution obtained in the constructive phase. We also created an improvement method that transfers information from the first phase to the second phase.

We start by presenting in Section 6.3.1 the constructive phase of the hybrid algorithm and then we present the improvement phase in Section 6.3.2.

6.3.1 Constructive phase

As mentioned previously, the exact method cannot address the highest dimensioned benchmark instances. Nevertheless, we can use the CC+RFV model with the y-separation to solve optimally a partial problem of the FTSP, which we designated by core problem. The core problem is induced by considering a subset of $\mathcal{L}$. Therefore, the core problem only contains a subset of nodes of $0 \cup N$. We start by deciding which families will be on the core problem.

From the computational study carried out in Section 5.4 it seems that the exact method is more efficient when the number of visits per family is high. Thus, we created several variations of instance $bier$ to better evaluate the efficiency of the CC+RFV model with the y-separation depending on the number of visits per family. The several variations have the same families and cost matrix, they only differ on the number of visits per family. There are ten different variations of instance $bier$, which are designated by $bier^i$ in which the number of visits for family $l \in \mathcal{L}$ is computed in the following manner: $v_l^i = \lceil (0.1 \times i) \times n_l \rceil$, with $i \in \{1, \dots, 10\}$. For example, in instance $bier^3$ the number of visits per family is, approximately, 30% of the number of family members. Table 6.11 shows the computational time, in seconds, to obtain the optimal value of instances $bier$ with the

CC+RFV model with the y-separation (t_s) depending on the percentage of nodes visited per family ($i\%$). The detailed results, with the optimal value and the number of added violated inequalities, are available in appendix, Table C.15.

Table 6.11: Optimal solution times for the variations of instance $bier$.

$i\%$	10	20	30	40	50	60	70	80	90	100
t_s	33	147	252	95	306	45	78	38	7	5

The results of Table 6.11 show that the exact method is more efficient when solving instances $bier^9$ and $bier^{10}$, which is not surprising since we already concluded that the CC+RFV model with the y-separation is more efficient when the number of family visits is high due to the RFV inequalities being more effective. Note that solving instance $bier^{10}$ is equivalent to solving a TSP, which shows that the FTSP is more challenging, computationally, than the TSP. Instance $bier^1$ is also easily solved, which we believe is a consequence from the fact that instance $bier^1$ has several single-visit families, and, thus, we add the valid inequalities that state that an arc between two nodes from a single-visit family can never be used in a feasible solution.

Due to these results, we defined a metric for each family $l \in \mathcal{L}$ which we called ratio of family l, with the notation r_l, which is defined as the ratio between the number of family visits and the number of family members, that is, $r_l = \frac{v_l}{n_l}, \forall l \in \mathcal{L}$. The families chosen to be in the core problem are the ones with a ratio bigger than r^*, which is a parameter of the hybrid algorithm. Let $Core = 0 \cup \{i \in N : i \in F_l \text{ and } r_l \geq r^*\}$. Observe that families with a high ratio have a similar number of family nodes and of family visits.

From Table 6.11 we know that the y-separation is able to solve instances with 127 nodes and families with a high ratio very efficiently. However, the number of nodes in the core problem may be significantly higher. In fact, if we were to consider $r^* = 0.8$ the number of nodes in the core problem of instance pr_2 would be 301 and, from the computational experiment carried out in Section 5.4.1, we know that the CC+RFV model with the y-separation cannot solve optimally instances with 280 nodes efficiently. Therefore, we decided to create an iterative process to obtain a solution of the core problem, which may not be the optimal solution as we may have to apply variable fixing. In the iterative process we start by solving, optimally, a partial core problem, which has a maximum of Δ nodes, then we apply variable fixing to reduce the size of the partial core problem and, finally, we insert more families in the partial core problem, until reaching a maximum number of Λ nodes. Notice that the last two steps are repeated until we have inserted all the families $l \in \mathcal{L}$ with $r_l \geq r^*$ in the core problem. Next we present the iterative process in more detail.

If $|Core| \leq \Delta$, we insert all the families in the initial core problem and solve it up to optimality.

Note that this is the only case in which the solution obtained after solving the core problem corresponds to its optimal solution. If $|Core| > \Delta$, then we must use the iterative process described previously, which consists in obtaining the optimal solution of a partial core problem, applying variable fixing and inserting more families. In the initial partial core problem, while the number of nodes in the initial core problem is less than Δ, we insert families $l \in \mathcal{L}$ such that $r_l \geq r^*$ according to a descending order of r_l. Then, we solve the initial partial core problem optimally and obtain its optimal solution (x^*, y^*). Before inserting more families in the partial core problem, we apply variable fixing, which in this particular case consists in removing variables from the core problem as we will fix to zero the variables that in the optimal solution of the partial core problem have value zero. Equivalently, we are removing from the partial core problem arcs and nodes that were not used in its optimal solution. This implies that, for now, we are not allowing to change the relative ordering of the nodes in the optimal solution of the partial core problem as well as the visited nodes from the families which have a ratio bigger than r^*. After reducing the partial core problem's size with variable fixing, we will insert families with the ratio bigger than r^* that were not inserted yet, according to a descending order of r_l. Then, we solve the resulting core problem up to optimality. Once again, we decided to set a limit to the number of nodes included in the several iterations of Λ. The last two steps, that is, the variable fixing and the insertion of more families in the core problems are repeated until all families which have a ratio greater than or equal to r^* are in the core problem.

After this process we obtain a feasible solution for an FTSP instance which only contains the families $l \in \mathcal{L}$ such that $r_l \geq r^*$. Therefore, in order to obtain a feasible solution for the original FTSP instance we must complete solution with the families which have a ratio less than r^*, this is done by using a savings heuristic. Consider a family $l \in \mathcal{L}$ such that $r_l < r^*$. For every node in F_l, that is not in the partial solution, we determine the cost of inserting it in every possible position of the partial solution. Then, we insert the node which the insertion cost is the lowest in the position that originated the lowest insertion cost. After inserting a node we recalculate the insertion costs. This process is repeated until we obtain a feasible solution for the FTSP. Note that the solution obtained preserves the relative ordering of the nodes from the core problem.

Algorithm 6.6 shows the pseudocode of the constructive phase of the hybrid algorithm.

Parameter testing

In order to choose the best possible parameter setting for the constructive phase of the hybrid algorithm we carried out a computational study considering the benchmark instances with unknown optimal value, which will be presented next. All the computational results presented during this section were obtained by performing only the constructive phase of the hybrid algorithm. Addi-

Algorithm 6.6 Constructive phase of the hybrid algorithm.

```
Require: $r^*$, $\Delta$, $\Lambda$
 1: Let $\mathcal{R} = \{l \in \mathcal{L} : r_l \geq r^*\}$.
 2: Set $i = 0$ and $Core_i = \{0\}$.
 3: while Number of nodes in $Core_0 < \Delta$ do
 4:     Insert family $l \in \mathcal{R}$ in $Core_0$ according to a descending order of $r_l$.
 5: end while
 6: Solve to optimality $Core_0$ as an FTSP and obtain the solution $(x^*, y^*)_{Core_0}$.
 7: while There are families in $\mathcal{R}$ not in $Core_i$ do
 8:     Remove arcs $(i, j)$ and nodes $i$ which have variables associated that satisfy: $x^*_{Core_i} = 0$ and
        $y^*_{Core_i} = 0$.
 9:     $i = i + 1$.
10:     while Number of inserted nodes in $Core_i \leq \Lambda$ do
11:         Insert a family $l \in \mathcal{R} \setminus Core_{i-1}$ in $Core_i$ according to a descending order of $r_l$.
12:     end while
13:     Solve to optimality $Core_i$ as an FTSP and obtain the solution $(x^*, y^*)_{Core_i}$.
14: end while
15: Complete the families $l \in \mathcal{L} \setminus \mathcal{R}$ with a savings heuristic.
Ensure: A feasible solution for the FTSP.
```

tionally, every time we are testing a parameter the rest of the algorithm remains the same, the only change is the parameter under testing.

Table 6.12 evaluates the usage of different r^* values, in particular, the r^* values of 0.70, 0.80 and 0.90. Table 6.12 shows the average percentage of gap between the solution obtained with the constructive phase of the hybrid algorithm using the different r^* values and the best upper bounds presented in Table 6.1 ($\overline{gap}$) and the average computational time, in seconds, to obtain the solution using the constructive phase of the hybrid algorithm ($\overline{t_s}$). The complete results are available in appendix, Table C.16.

Table 6.12: Comparing several values of r^*.

	$\overline{gap}$	$\overline{t_s}$
$r^* = 0.70$	-7.10%	1988
$r^* = 0.80$	-2.17%	67
$r^* = 0.90$	2.98%	1

Average obtained with instances a_1, a_2, gr and pr.

The computational time is inversely proportional to the r^* value, since lower values of r^* imply that more families must be included in the core problem. As the computational time increased significantly from $r^* = 0.80$ to $r^* = 0.70$ we did not experiment lower values of r^*. Regarding the average gap, with r^* equal to 0.80 and 0.70 we were able to obtain a negative average gap, which shows that only the constructive phase of the hybrid algorithm is able to provide better quality solutions than the other heuristic methods available in the literature. More precisely, by using $r^* = 0.70$ we were able to obtain a solution with a lower value than the best upper bound obtained by Morán-Mirabal et al. (2014) for every instance tested. Observe that $r^* = 0.70$ makes the method very time consuming and $r^* = 0.90$ worsens the quality of the solutions obtained significantly due to the core problem being too small. To reach a compromise between the quality of the solutions obtained and the computational time we decided to choose $r^* = 0.80$.

Next, we experimented several Δ values, that is, different values for the maximum number of nodes in the initial partial core problem. More precisely, we considered Δ values of 140, 160, 180 and 200. Table 6.13 shows the results obtained with the constructive phase of the hybrid algorithm considering the values of Δ mentioned previously in terms of average gap between the solution obtained with the constructive phase of the hybrid algorithm and the best upper bound presented in Table 6.1 ($\overline{gap}$) and of average computational time, in seconds, to obtain the solution using the constructive phase of the hybrid algorithm ($\overline{t_s}$). Recall that all the results were obtained using the same parameter setting, except for the Δ value. The complete results considering $\Delta = 180$ are available in appendix, Table C.16, while the detailed results considering the other Δ values are available in Table C.17.

Table 6.13: Comparing several values of Δ.

	$\overline{gap}$	$\overline{t_s}$
$\Delta = 140$	-1.69%	35
$\Delta = 160$	-1.96%	2158
$\Delta = 180$	-2.17%	67
$\Delta = 200$	-1.74%	387

Average obtained with instances a_1, a_2, gr and pr.

The results of Table 6.13 are not the expected ones both in terms of percentage of gap and of computational time. Regarding the percentage of gap, we expected it to decrease when the Δ value increased as we are considering more nodes in the first partial core problem and, consequently, we should be able to obtain a better quality solution. However, that is not the case since the highest value of Δ provides the second highest average gap. If we only consider the results for $\Delta = 140, 160$ and 180 we obtain the expected relationship, which is, lower Δ values provide higher gap values. Concerning the computational time, we were expecting it to be proportional to the Δ value. By observing the detailed results in Table C.17 we verify that the computational time for instance pr_2 is very high as it took 16782 seconds, which shows that even by limiting the number of nodes in the initial partial core problem there is an unpredictability associated with this method. If we compute the average time ignoring instance pr_2 we obtain the expected results as the average computational time is 11 seconds with $\Delta = 140$, 69 seconds with $\Delta = 160$, 65 seconds with $\Delta = 180$ and, finally, 84 seconds with $\Delta = 200$. We decided to choose $\Delta = 180$ as it is the value of Δ that allowed us to obtain the best average gap in an average time of 67 seconds, which is reasonable considering the dimension of the test instances that we are using to tune the parameters.

Table 6.14 shows the average percentage of gap between the solution obtained by using the constructive phase of the hybrid algorithm and the best upper bound obtained by Morán-Mirabal et al. (2014) considering several Λ values ($\overline{gap}$) and the average computational time, in seconds, to obtain the solution by using the constructive phase of the hybrid algorithm ($\overline{t_s}$). The complete results obtained with the Λ values of 30, 50 and 90 are available in appendix, Table C.18 while the ones obtained with $\Lambda = 70$ are available in Table C.16.

Table 6.14: Comparing several values of Λ.

	$\overline{gap}$	$\overline{t_s}$
$\Lambda = 30$	-2.31%	67
$\Lambda = 50$	-2.31%	69
$\Lambda = 70$	-2.17%	67
$\Lambda = 90$	-2.17%	69

Average obtained with instances a_1, a_2, gr and pr.

The results presented in Tables 6.12 and 6.13 were obtained by considering $\Lambda = 70$. The results shown in Table 6.14 are similar for the several Λ values, both in terms of average gap and of average computational time, which shows that the value of the parameter Λ has a small impact on the overall quality of the solution obtained by the constructive phase of the hybrid algorithm. In fact, there was only one instance in which the solution obtained with the several Λ values was different, which was instance pr_2. Since the average gap is the same for $\Lambda = 30$ and $\Lambda = 50$ and with $\Lambda = 30$ we obtained a sightly lower average computational time we decided to use $\Lambda = 30$.

According to the parameter tuning performed, the best parameter setting, amongst the ones tested, for the constructive phase of the hybrid algorithm is the following: $r^* = 0.80$, $\Delta = 180$ and $\Lambda = 30$. By using this parameter setting we were able to obtain an average gap of -2.31%, which means that with the constructive phase of the hybrid algorithm we were able to improve, on average, the best upper bounds obtained by Morán-Mirabal et al. (2014) of the benchmark instances with unknown optimal value. More precisely, we were able to improve the best upper bound presented in Table 6.1 of five instances, namely instances a_1, gr_1, pr_1, pr_2 and pr_3. Regarding the computational time, taking into account that we are using the y-separation to solve the core problem, an average of 67 seconds seems reasonable. Now, we must apply to the solution obtained in the constructive phase of the hybrid algorithm an improvement algorithm, which is presented next.

6.3.2 Improvement phase

Due to the satisfactory results obtained with the ILS algorithm we experimented applying it directly to the feasible FTSP solution obtained in the constructive phase of the hybrid algorithm as an improvement algorithm. The ILS algorithm used is the one presented in Algorithm 6.3 with the first step being the constructive phase presented in the previous section, in Algorithm 6.6. Table 6.15 shows the average gap between the best upper bound obtained by Morán-Mirabal et al. (2014) and the best solution provided by the hybrid algorithm performing 1000 and 5000 iterations of the ILS algorithm ($\overline{gap}$) and the computational time, in seconds, to obtain the best solution with the hybrid algorithm ($\overline{t_s}$). The complete results are available in appendix, Table C.19.

Table 6.15: Results obtained with the hybrid algorithm.

	$\overline{gap}$	$\overline{t_s}$
1000 iterations	-11.97%	244
5000 iterations	-12.77%	904

Average obtained with instances a_1, a_2, gr and pr.

By comparing the results presented in Table 6.15 to the ones presented in Table 6.10, which were obtained with the ILS algorithm presented in Section 6.2 in which the initial FTSP solution is obtained by using the nearest neighbor heuristic, we verify that the hybrid algorithm obtained the lowest average gap performing both 1000 and 5000 iterations. Even though the difference is not very significant, considering 5000 iterations, the hybrid algorithm was able to obtain a better solution than the ILS algorithm in six of the eight benchmark instances which have an unknown optimal value. Regarding the computational time, as expected, the hybrid algorithm is more time consuming than the ILS algorithm. Nevertheless, as our main objective is to improve the best upper bounds available in the literature for the benchmark instances which have an unknown optimal value, the hybrid algorithm seems to be a suitable method.

By applying the ILS directly to the solution obtained in the constructive phase of the hybrid algorithm we are not exchanging information between both phases. Thus, in Section 6.3.2.1 we present a procedure that exchanges information between both phases of the hybrid algorithm.

6.3.2.1 Transferring information from the constructive phase to the improvement phase

In the hybrid algorithm we complete the partial solution provided by the core problem by using a savings heuristic, however, when completing the solution, we only take into account the costs associated to the arcs. Therefore, we decided to use dual information to choose nodes which are not contemplated in a greedy procedure, such as the savings heuristic, but which may be good candidates to be visited in higher quality solutions.

Recall that in linear programming the optimality conditions may be expressed in terms of dual feasibility, meaning that a variable does not satisfy the optimality conditions if the corresponding dual constraint is violated. Therefore, we decided to use dual variables to identify violated dual constraints associated with variables x_{ij} linking nodes in the core problem to nodes not in the core problem and consider the corresponding arcs $(i, j) \in A$ as potentially interesting arcs to be included in a feasible solution for the FTSP.

To obtain the value of the dual variables we will use the SCF model as this is the fastest compact model to obtain the LP relaxation value. Consequently, we reintroduce the SCF model, more precisely, the objective and the constraints that contain x variables:

$$\textit{Minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{4.1}$$

Subject to:

$$x(0, N) = 1 \qquad \alpha_0 \tag{4.2}$$

$$x(i, 0 \cup N) - y_i = 0 \qquad \forall i \in N \qquad \alpha_i \tag{4.3}$$

$$x(0 \cup N, i) - x(i, 0 \cup N) = 0 \qquad \forall i \in 0 \cup N \qquad \beta_i \tag{4.4}$$

(4.5) – (4.9) and (4.10)

$$f_{ij} - Vx_{ij} \leq 0 \qquad \forall (i, j) \in A \setminus A_0 \qquad \mu_{ij} \tag{4.11}$$

$$0 \leq x_{ij} \leq 1 \qquad \forall (i, j) \in A \qquad \gamma_{ij} \tag{4.36}$$

(4.37) and (4.12).

The LP relaxation of the SCF model is used to solve the FTSP instance defined by the set of nodes in $Core$. As we are only interested in the arcs, we only presented the constraints, and the corresponding dual variables, that contain x variables. Let α_0, α_i, β_i, μ_{ij} and γ_{ij} be the dual variables associated with constraints (4.2), (4.3), (4.4), (4.11) and (4.36), respectively. The dual constraint associated with variable x_{ij}, with $(i, j) \in A$ is: $\alpha_i - \beta_i + \beta_j - V\mu_{ij} + \gamma_{ij} \leq c_{ij}$. Consider now that $j \in N \setminus Core$. In this case, we can assume that the dual variables associated with j have value zero and the dual constraint associated with x_{ij} becomes $\alpha_i - \beta_i \leq c_{ij}$. If there is a variable x_{ij} such that $\alpha_i - \beta_i - c_{ij} > 0$, then the dual constraint associated with x_{ij} is violated and the variable does not satisfy the corresponding optimality condition. Therefore, we will choose the arcs $(i, j) \in A$, with $i \in Core$ and $j \in N \setminus Core$, such that $\alpha_i - \beta_i - c_{ij} > 0$ to be in a pool of arcs.

Before applying the savings heuristic, we insert some of the arcs from the pool of arcs in the partial solution to visit more nodes and be closer to obtaining a feasible solution for the FTSP. We start by inserting the arcs (i, j) which have the highest value of $\alpha_i - \beta_i - c_{ij}$, meaning that, we prioritize the insertion of the arcs which are associated with the most violated dual constraints. Assume that we wish to insert the arc (i, j) in the partial solution, with $i \in Core$ and $j \in N \setminus Core$. To do so we remove node i from the partial solution and verify which is the best position, in terms of solution cost, to insert the arc (i, j). When choosing the arc to insert we also need to make sure that node j's family is not complete. Additionally, for each node $i \in Core$, we insert at the most one arc from the pool of arcs which has node i as its initial node. The arcs that were chosen according to dual information are considered taboo, that is, they cannot be removed from the solution. If after inserting all the possible arcs from the pool of arcs the partial solution remains unfeasible, then we apply the savings heuristic.

After obtaining a feasible solution for the FTSP, we apply the ILS algorithm presented in Algorithm 6.3. We experimented considering the arcs from the pool of arcs taboo not only during the savings heuristic and the first local search procedure (step 2 Algorithm 6.3) but also during the first 100 iterations of the perturbation and the local search procedures. The results obtained are shown in Table 6.16. This table shows the average percentage of gap between the best upper bound obtained by Morán-Mirabal et al. (2014) and the best solution obtained by the hybrid algorithm with dual information ($\overline{gap}$) and the average computational time, in seconds, to obtain the solution with the hybrid algorithm with dual information ($\overline{t_s}$). The results shown were obtained by performing 1000 iterations of the ILS algorithm. The detailed results obtained with the hybrid algorithm with dual information are available in appendix, Table C.20.

Table 6.16: Using the hybrid algorithm with dual information.

	$\overline{gap}$	$\overline{t_s}$
Taboo first local search	-11.88%	283
Taboo first 100 iterations	-11.36%	283

Average obtained with instances a_1, a_2, gr and pr.

By comparing the results presented in the previous table we conclude that the best approach, in terms of average percentage of gap, is to consider the arcs from the pool of arcs as taboo only during the initial local search. When we compare these results to the ones presented in Table 6.15 we verify that these results are slightly worse. More precisely, without using the dual information we obtain an average percentage of gap of -11.97%. When we analyze the detailed results obtained with the hybrid algorithm with dual information we verify that it only provided a better solution than the hybrid algorithm in instance a_1. The results obtained with and without using dual information are similar, however we expected that the dual information would lead us to better quality solutions. We believe that these results are a consequence from the fact that we are solving an ILP problem using information of its LP relaxation. Additionally, when we insert an arc from the pool of arcs in the partial solution we may be forcing the node that belongs to the solution to change place. Nevertheless, the hybrid algorithm with dual information was able to obtain a better solution than the hybrid algorithm in one instance which means that this idea could be promising provided that we improve the quality of the approximation used. Consequently, we will use the hybrid algorithm without the dual information, that is, we use the algorithm presented in Algorithm 6.6 to obtain a feasible solution for the FTSP and then we apply, to that solution, the ILS algorithm presented in Section 6.2.

6.4 Computational experiment

In this section we start by comparing the results obtained with the proposed heuristics to verify which are the best heuristic methods. Then, we present the final computational experiment with the heuristic algorithms.

All the algorithms presented in this chapter were developed within the scope of this dissertation and were implemented in C++ and the computational experiment was carried out in an Intel Core i7, 3.60 gigahertz, 8 gigabytes RAM, as before.

Table 6.17 shows a summary of the best results obtained with the genetic algorithm, the ILS algorithm and the hybrid algorithm. We refer the reader to Sections 6.1, 6.2 and 6.3 to verify which is the best parameter setting for the genetic algorithm, the ILS algorithm and the hybrid algorithm, respectively. Table 6.17 contains the average percentage of gap ($\overline{gap}$) and the average computational time, in seconds, to obtain the best solution with the several heuristic algorithms ($\overline{t_s}$).

Table 6.17: Summary of the best results obtained with the proposed heuristic algorithms.

	$\overline{gap}$	$\overline{t_s}$
Genetic algorithm	-9.83%	305
ILS algorithm	-12.20%	735
Hybrid algorithm	-12.77%	904

Average obtained with instances a_1, a_2, gr and pr.

In order to establish a fairer comparison, the results obtained with the ILS algorithm and the hybrid algorithm where obtained performing 5000 iterations while the ones obtained with the genetic algorithm were obtained by applying 1000 iterations of the local search algorithm to 12 solutions present in the final population of the genetic algorithm. By observing Table 6.17 we verify that the average percentage of gap is inversely proportional to the average computational time as less time consuming heuristics lead to higher percentage of gap values. Nevertheless, all the proposed heuristics were able to improve the best upper bound obtained by Morán-Mirabal et al. (2014) for all the benchmark instances with unknown optimal value. As our main objective is to obtain the best solution possible for the instances that the exact methods cannot solve within the time limit we analyzed the detailed results and verified which methods provided the best solution. Tables C.6, C.14 and C.19 show the detailed results of the best version of the genetic algorithm, the ILS algorithm and the hybrid algorithm, respectively. From these tables we verify that the hybrid algorithm obtained the solution with the lowest cost in six instances and the ILS algorithm obtained the lowest cost solution in two, namely instances gr_2 and pr_2. Even though the genetic algorithm is the most efficient method in terms of computational time, it is dominated by the ILS algorithm and the hybrid algorithm in terms of quality of the solutions obtained as it never provides the solution with the lowest cost. Therefore, as we are more interested in the quality of the solutions obtained than in the computational efficiency of the algorithm and as neither the ILS algorithm nor the hybrid algorithm provided the lowest cost solution in every benchmark instance with unknown optimal value we decided to carry out a computational study with both methods.

Since there are instances from the instance sets 2 and 3 that the exact methods could not find their optimal value within the time limit, we decided to use the heuristic methods to solve them. Note that since these instances were never addressed in the literature there are no upper bounds available for us to use to evaluate the quality of the solutions obtained by the proposed heuristics. Therefore, we decided to use as reference values the upper bounds provided by the B&C algorithm, which are available in Table 5.22. In the case of the instances from the instance sets 2 and 3 the percentage of gap is computed by using the following formula: $gap = 100 \times (\textit{heuristic solution} - \textit{B\&C solution})/\textit{B\&C solution}$.

As the ILS algorithm and the hybrid algorithm use random procedures we did five runs, considering different seeds, for each instance that we intend to solve. This allowed us to evaluate the robustness of the methods. Tables 6.18 and 6.19 show a summary of the results obtained in the five runs with the ILS algorithm and the hybrid algorithm, respectively, with the most important statistics. The detailed results for all the different runs are available in appendix, Table C.21. Tables 6.18 and 6.19 contain, besides the instance name, the value of the best solution obtained by the proposed method (Best value), the minimum percentage of gap (min), the average percentage of gap ($average$) and the maximum percentage of gap (max) obtained in the five runs, the range between the maximum and the minimum percentage of gap ($range = max - min$) and the average computational time, in seconds, also considering the five distinct runs ($\overline{t_s}$).

Table 6.18: Summary of the final results obtained with the ILS algorithm.

Instance	Best value	min	$average$	max	$range$	$\overline{t_s}$
Instance set 1						
a_1	1771.25	-6.34%	-5.76%	-4.86%	1.48%	33
a_2	1573.27	-7.32%	-5.62%	-4.38%	2.94%	33
gr_1	1554.80	-14.43%	-13.93%	-13.56%	0.87%	452
gr_2	1246.21	-13.64%	-13.27%	-12.69%	0.95%	461
gr_3	1227.50	-11.32%	-10.58%	-9.78%	1.53%	478
pr_1	134283.00	-17.85%	-16.99%	-15.32%	2.53%	1436
pr_2	144621.00	-20.60%	-19.90%	-19.24%	1.36%	1417
pr_3	128562.00	-13.98%	-12.81%	-11.69%	2.29%	1608
Instance set 2						
pr144_4	49527	0.25%	0.60%	1.04%	0.79%	4
kroA150_3	21645	3.66%	3.68%	3.75%	0.08%	4
pr152_3	65339	1.42%	1.71%	1.86%	0.44%	5
rat195_1	1300	1.17%	2.26%	3.27%	2.10%	12
rat195_3	1841	1.49%	1.86%	2.21%	0.72%	9
rat195_4	1359	2.95%	3.83%	4.09%	1.14%	10
kroA200_1	17318	5.33%	6.62%	8.02%	2.69%	12
kroA200_3	25487	4.15%	4.51%	4.70%	0.55%	9
kroB200_3	25410	5.49%	5.77%	6.28%	0.79%	8
gr202_4	28766	2.59%	4.28%	5.02%	2.43%	11
gr229_1	72238	2.12%	2.39%	2.63%	0.52%	19
gil262_1	1567	2.49%	3.96%	5.30%	2.81%	23
gil262_3	2069	2.83%	3.06%	3.53%	0.70%	16
gil262_4	1737	-2.03%	-1.55%	-1.07%	0.96%	20
pr264_2	28892	0.50%	0.80%	1.00%	0.50%	25
Instance set 3						
rbg443_2	400	-32.89%	-29.80%	-27.52%	5.37%	131

Table 6.19: Summary of the final results obtained with the hybrid algorithm.

Instance	Best value	min	$average$	max	$range$	$\overline{t_s}$
Instance set 1						
a_1	1741.14	-7.93%	-6.82%	-6.00%	1.93%	36
a_2	1568.56	-7.59%	-6.64%	-5.37%	2.23%	33
gr_1	1551.27	-14.63%	-13.43%	-12.06%	2.57%	462
gr_2	1236.96	-14.28%	-13.69%	-13.22%	1.07%	481
gr_3	1240.23	-10.40%	-10.01%	-9.72%	0.68%	497
pr_1	135848.93	-16.89%	-16.28%	-15.04%	1.85%	2046
pr_2	146596.93	-19.52%	-19.06%	-18.74%	0.78%	1621
pr_3	123164.62	-17.59%	-14.56%	-12.81%	4.78%	1887
Instance set 2						
pr144_4	49650	0.50%	0.58%	0.88%	0.38%	4
kroA150_3	21038	0.76%	0.76%	0.76%	0.00%	5
pr152_3	64718	0.45%	0.62%	0.83%	0.38%	5
rat195_1	1314	2.26%	2.54%	2.96%	0.70%	12
rat195_3	1840	1.43%	1.71%	2.54%	1.10%	12
rat195_4	1331	0.83%	1.05%	1.21%	0.38%	11
kroA200_1	17099	4.00%	4.78%	5.41%	1.41%	12
kroA200_3	24580	0.45%	0.46%	0.54%	0.09%	5152
kroB200_3	24542	1.88%	1.88%	1.88%	0.00%	196
gr202_4	28117	0.28%	0.98%	1.66%	1.38%	12
gr229_1	71667	1.31%	1.78%	2.42%	1.11%	19
gil262_1	1577	3.14%	3.14%	3.14%	0.00%	24
gil262_3	1987	-1.24%	-0.67%	-0.30%	0.94%	8196
gil262_4	1712	-3.44%	-2.64%	-1.30%	2.14%	64
pr264_2	28863	0.40%	0.80%	1.21%	0.81%	28
Instance set 3						
rbg443_2	399	-33.05%	-31.24%	-28.86%	4.19%	132

We start by analyzing Table 6.18. The average gap obtained considering all instances and all seeds with the ILS algorithm was of -3.54% while the average computational time was of 260 seconds. Considering the instance set 1, the ILS algorithm was able to find solutions with a lower value than the best upper bound obtained by Morán-Mirabal et al. (2014) for every instance for which the optimal value is not known. In fact, even the worst solution obtained in the five runs is better than the best upper bound from the literature. The method seems to provide good quality solutions consistently as the maximum range obtained was 2.94%, thus this is a robust method. Regarding the computational time, taking into account the size of instances gr and pr, which have 666 and 1002 nodes, respectively, we can state that the ILS algorithm is able to obtain good quality solutions in a very reasonable amount of computational time. We cannot establish a fair comparison to the metaheuristics proposed by Morán-Mirabal et al. (2014) for the FTSP in terms of computational time due to the different stopping criterion used. The referred metaheuristics stops either when they reach the time limit, which is 7200 seconds, or when they find a solution with a cost which is less than or equal to the target solution. The target solution is the best solution found by the referred metaheuristics, after 36000 seconds of computational time. We reported the computational times obtained by Morán-Mirabal et al. (2014) in Table 6.1.

From the 15 instances of the instance set 2 which have an unknown optimal value, the ILS algorithm could only improve the reference solution, which was obtained with the B&C algorithm, in one instance, namely instance $gil262\_4$. We were expecting that the ILS algorithm would obtain better solutions than the B&C algorithm, however that was not the case. These results make us believe that the heuristic callback presented in Section 5.3 is able to provide good quality solutions during the B&C algorithm. Regarding the computational time, the B&C algorithm took 10800 seconds to obtain the reference solutions while the ILS algorithm provided solutions for these instances in an average of 12 seconds, which shows that we cannot establish a fair comparison between the solutions obtained by the B&C algorithm and the ones obtained with the ILS algorithm due to the discrepancy in the average computational time.

Finally, there was only one instance from the instance set 3 that the exact method could not solve within the time limit, which was instance $rbg443\_2$. Recall that this is one of the instances which have arcs with cost zero. The ILS algorithm was able to provide a good quality solution for this instance. In fact, the solution provided by the B&C algorithm has a value of 596 whilst the best solution obtained by the ILS algorithm has a value of 400. Moreover, the ILS algorithm took an average of 131 seconds to obtain that solution.

Consider now the results obtained with the hybrid algorithm presented in Table 6.19. The average gap obtained considering all instances and all runs was -4.75% while the average computational time was 873 seconds. Observing the referred table, we verify that the high average computational time is due to instances $kroA200\_3$ and $gil262\_3$. If we compute the average computational time without these two instances we obtain an average of 345 seconds, which is of the same magnitude as the time obtained with the ILS algorithm.

From the benchmark instances with unknown optimal values, the hybrid algorithm was able to obtain a solution of better quality in every run as the maximum value of gap obtained is negative. From these results, it is clear that the hybrid algorithm outperforms the heuristics methods available in the literature in terms of the quality of the solutions obtained. Regarding the robustness of the method, it seems to be less robust than the ILS algorithm as the maximum range obtained was 4.78%. When we compare the hybrid algorithm to the ILS algorithm, the hybrid algorithm provides a lower average gap while the ILS algorithm is faster. Moreover, both methods were able to improve the best upper bounds from the literature. In fact, the hybrid algorithm obtained the solution with the lowest value in five instances whereas the ILS algorithm provided the best quality solution in three instances, more precisely, instances gr_3, pr_1 and pr_2.

Regarding the instance set 2, the hybrid algorithm was able to improve the upper bounds obtained by the B&C algorithm in two instances, namely instances $gil262\_3$ and $gil262\_4$. Once again, the B&C algorithm is able to provide better quality solutions after 10800 seconds of computational time which makes the comparison of the hybrid algorithm to the B&C algorithm unfair. The hybrid algorithm was very time consuming when addressing instances $kroA200\_3$ and $gil262\_3$, which shows that even with all the measures taken to ensure the efficiency of the resolution of the core problem, there is an underlying uncertainty. This characteristic is a drawback of the hybrid algorithm. However, even considering the time consuming instances, the average time to solve the instance set 2 is 917 seconds, which is significantly smaller than the time limit used in the B&C algorithm. Without the time consuming instances, the average computational time for the instance set 2 is 38 seconds. When we compare the best solutions provided by the hybrid algorithm to the one obtained with the ILS algorithm, we verify that the hybrid algorithm provided the best solution in 14 instances whereas the ILS algorithm only obtained the solution of better quality in three instances, namely instances $pr144\_4$, $rat195\_1$ and $gil262\_1$.

The hybrid algorithm was able to improve the reference value obtained for instance $rbg443\_2$ from the instance set 3. In fact, the hybrid algorithm was the method that provided the best solution for the referred instance. In terms of computational time, it was similar to the one obtained with the ILS algorithm.

Summary

We proposed three different heuristics for the FTSP to solve the instances that the exact methods could not solve optimally within the time limit, which are a genetic algorithm, an ILS algorithm and a hybrid algorithm.

All the proposed heuristics were able to improve the best upper bound obtained by Morán-

Mirabal et al. (2014) for the benchmark instances with unknown optimal value. The genetic algorithm was the most efficient heuristic algorithm as considering its best version took an average of 305 seconds to provide feasible solutions for the FTSP while the ILS algorithm and the hybrid algorithm took 735 and 904 seconds, respectively. Regarding the quality of the solutions obtained, the genetic algorithm is the heuristic algorithm that provides the worst quality solutions on average. In fact, the other heuristic methods provide solutions of better quality than the genetic algorithm in every instance tested. As our main objective is to obtain the solutions with the lowest possible cost, we decided to carry out a computational study with the ILS algorithm and the hybrid algorithm, since we could not establish any relationship of dominance between them.

From the 24 instances that the exact methods failed to obtain the optimal solution within the time limit, the hybrid algorithm was able to obtain the best solution in 18 instances whilst the ILS algorithm provided the best solution in six. This shows that even though the methods are similar, there is a dominance of the hybrid algorithm in terms of quality of the solutions obtained. Regarding the computational time, the ILS algorithm is more efficient than the hybrid algorithm. Additionally, as we are using an exact method in the hybrid algorithm there is an unpredictability in the computational time of the referred method since it is highly dependent on the difficulty of the core problem. Thus, when the first concern is to obtain the best solutions possible in a reasonable amount of computational time the hybrid algorithm is the best heuristic method. However, if the goal is too achieve good quality solutions quickly the genetic algorithm is preferable.

Consider now the feasible solutions obtained by the B&C algorithm. These solutions are of very good quality. In fact, the B&C algorithm was able to obtain the best solution in 14 out of the 24 instances with unknown optimal value. Nonetheless, the B&C algorithm requires an effective formulation and it has limitations regarding the instance's dimension, for example, we were unable to apply the y-separation to the benchmark instances gr and pr. Table 6.20 shows a summary of the best known upper bounds for the instances with unknown optimal value, which were all obtained by applying methods developed within the scope of this dissertation.

Table 6.20: Current best known upper bounds.

Instance	Best known upper bounds
Instance set 1	
a_1	1692.92
a_2	1568.56*
gr_1	1551.27
gr_2	1236.96
gr_3	1227.50
pr_1	134283.00
pr_2	144621.00
pr_3	123164.62
Instance set 2	
pr144_4	49403*
kroA150_3	20880*
pr152_3	64425*
rat195_1	1285*
rat195_2	1814*
rat195_4	1320*
kroA200_1	16441*
kroA200_3	24471*
kroB200_3	24088*
gr202_4	28039*
gr229_1	70741*
gil262_1	1529*
gil262_3	1987
gil262_4	1712
pr264_2	28748*
Instance set 3	
rbg443_2	399

*Obtained with the B&C algorithm.

Chapter 7

The Family Traveling Salesman Problem: a Variant

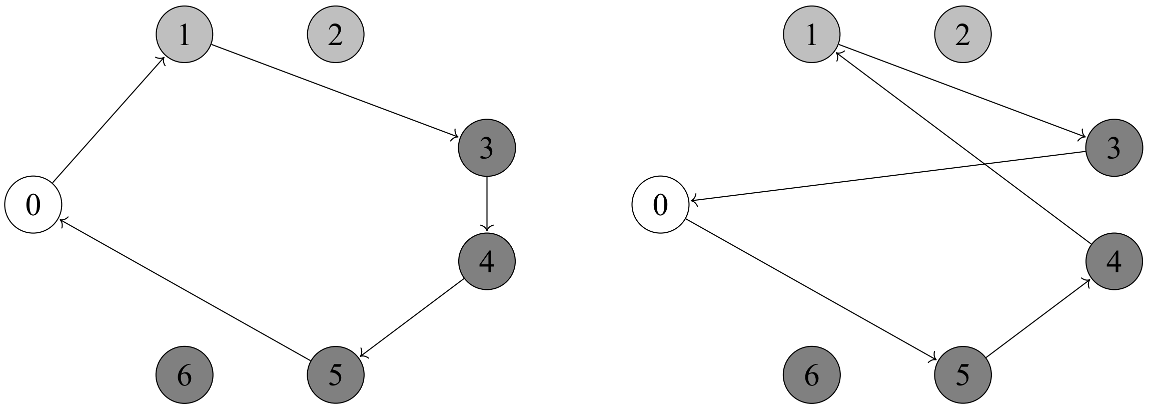
In this chapter we address a variant of the FTSP. Assume that, besides the usual requirements for a circuit to be a feasible solution for the FTSP, it also needs to visit the nodes from the same family consecutively. This assumption makes sense as we may want to cluster the products of the same type for delivery purposes. This additional constraint originates the restricted family traveling salesman problem (RFTSP). To clarify, the objective of the RFTSP is to find the minimum cost elementary circuit that: (i) starts and ends at the depot; (ii) visits a predefined number of nodes per family; and (iii) visits the nodes from the same family consecutively. We will refer to condition (iii) as the consecutiveness condition henceforth.

This chapter is organized as follows. In Section 7.1 we formally introduce the RFTSP. In Section 7.2 we present some basic constructive heuristics and neighborhoods for the RFTSP, which will be used as subroutines in the exact and the heuristic algorithms presented in the subsequent sections. In Section 7.3 we present mathematical formulations for the RFTSP. Some are adaptations of formulations for the FTSP while others are specific for the RFTSP. In Section 7.4 we present the B&C algorithm to solve the non-compact formulations proposed and, finally, in Section 7.5 we present the heuristic algorithms for the RFTSP.

7.1 The restricted family traveling salesman problem

In order to illustrate the difference between the FTSP and the RFTSP, consider a new FTSP instance which has two families. Family 1, which is represented by the light gray color, has two family nodes (nodes 1 and 2) while family 2, represented by the dark gray color, has four family nodes (nodes

3-6). The number of nodes that we are required to visit in family 1 is one and in family 2 is three. Figure 7.1 shows an example of a feasible (Figure 7.1a) and of an unfeasible solution (Figure 7.1b) for the RFTSP considering the instance described previously.



(a) Feasible solution. (b) Unfeasible solution.

Figure 7.1: An example of a feasible and an unfeasible solution for the RFTSP.

Both solutions presented in Figure 7.1 are feasible for the FTSP but the solution presented in Figure 7.1b is not feasible for the RFTSP since we are visiting node 1 between nodes 4 and 3 which are from family 2. By analyzing Figure 7.1a we verify that the number of arcs used in the cut-set $[(N \cup 0) \setminus F_2, F_2]$ is one. Note that if the number of arcs used in the referred cut-set was lower, we would have a disconnected solution, while if the number of arcs used was higher, like in the case of Figure 7.1b, the solution would not satisfy the consecutiveness condition. Additionally and considering family 2, we see that there is an elementary three node path using only arcs linking nodes from family 2. Generalizing, given a family $l \in \mathcal{L}$, a feasible solution for the RFTSP that satisfies the consecutiveness condition has two characteristics:

(a) there is one, and only one, arc used in the cut-set $[F_l, (N \cap 0) \setminus F_l]$; and

(b) there is an elementary path with v_l nodes and with arcs $(i, j) \in A$, such that $i, j \in F_l$.

Given a feasible solution for the FTSP, if we ensure either one of the conditions (a) or (b) stated previously, we obtain a feasible solution for the RFTSP. Note that conditions (a) or (b) are redundant for the single-visit families as, firstly, condition (a) is ensured by the constraints (4.2)-(4.5) presented in the generic model of Section 4.1 and, secondly, it is not possible to define a path that only visits one node, according to the definition of a path given in Section 2.1. Therefore, we only need to ensure that conditions (a) or (b) are satisfied for the multi-visit families, which we recall is the set of families $\mathcal{M} = \mathcal{L} \setminus \mathcal{U}$.

7.2 Basic heuristics and neighborhoods for the RFTSP

Throughout this section we present some basic constructive heuristics and neighborhoods for the RFTSP. The proposed methods are an adaptation of the basic FTSP heuristics and neighborhoods presented in Section 3.3.

Regarding the basic FTSP heuristics presented in Section 3.3, we will adapt the nearest neighbor algorithm and the random constructive heuristic. The adaptation of both heuristic procedures is similar and it relies on the idea that once we start visiting nodes from a family we can only visit nodes from a different family when the former family is complete.

In the case of the nearest neighbor, if the family of the last node inserted in the circuit is not complete, then we must choose the nearest node which belongs to the same family as the last node inserted in the circuit. Otherwise, that is, when we complete a family, we may choose the nearest node to be inserted in the circuit like we did in the FTSP.

The adaptation of the random constructive heuristic is similar to the one made for the nearest neighbor. More precisely, the permutation defines the order of the visits but we need to visit all the nodes from the same family consecutively. This implies that the families are visited according to the order as their first nodes appear in the permutation and, for each family, we will visit the nodes of that family by the order as they appear in the permutation ignoring the remaining families until the current family is complete. Example 32 shows how the random constructive heuristic transforms a permutation into a feasible solution for the RFTSP.

Example 32 (Random Heuristic RFTSP)**.** Consider the RFTSP instance presented in Figure 7.1 and the permutation $\pi = (3, 1, 4, 2, 5, 6)$. The circuit starts at node 0 and then visits node 3. As we cannot leave family 2 before visiting all the required nodes, we will visit nodes 4 and 5. Family 2 is now complete, so we will return to the second position of the permutation and we will visit node 1 from family 1. The circuit obtained is $\{(0, 3), (3, 4), (4, 5), (5, 1), (1, 0)\}$.

The neighborhoods for the FTSP defined in Section 3.3 consist in switching the position of nodes that belong to the circuit, in switching a visited node with a non-visited node and, in the case of a symmetric cost matrix, in inverting the order of one path in the circuit. As the moves that define the neighborhoods must originate feasible solutions, we have to adapt them for the case of the RFTSP. We propose five different neighborhoods for the RFTSP, four of which are similar to the neighborhoods $\mathcal{N}_I, \mathcal{N}_O$ and 2-opt and the fifth one is specific for the RFTSP. The neighborhoods that are similar to FTSP neighborhoods will be called the same for simplification purposes. Therefore, the five RFTSP neighborhoods are $\mathcal{N}_I$, $\mathcal{N}_O$, 2-opt-intra, 2-opt-inter and $\mathcal{N}_F$. Considering s as a feasible solution for the RFTSP, these neighborhoods are defined as follows:

- $\mathscr{N}_I(s)$ $=\{s'$ feasible : s' can be obtained from s by switching a maximum of two nodes, from the same family, in the circuit$\}$
- $\mathscr{N}_O(s)$ $=\{s'$ feasible : s' can be obtained from s by switching a maximum of two nodes, from the same family, such that, one belongs to the circuit and the other does not$\}$
- 2-opt-$intra(s)$ $=\{s'$ feasible : s' can be obtained from s by inverting the order of one path in the circuit such that the initial and the final node of the path belong to the same family$\}$
- 2-opt-$inter(s)$ $=\{s'$ feasible : s' can be obtained from s by inverting the order of one path in the circuit that has i and j ($i \neq j$) as its initial and final nodes, respectively, such that: i is either the depot or the first visited node from family $l_1 \in \mathcal{L}$ and j is either the depot or the last visited node from family $l_2 \in \mathcal{L} : l_1 \neq l_2\}$
- $\mathscr{N}_F(s)$ $=\{s'$ feasible : s' can be obtained from s by switching the position of a maximum of two families in the circuit$\}$

Neighborhoods $\mathcal{N}_I$ and $\mathcal{N}_O$ are similar to the ones presented in Section 3.3 for the FTSP. More precisely, the former has an additional condition that states that we can only switch the position of nodes in the circuit which are from the same family while the latter is exactly the same. Neighborhood 2-opt presented in Section 3.3 originated neighborhoods 2-opt-intra and 2-opt-inter, which intuitively consist in inverting a path inside a family or a path with several families, respectively. Additionally, the path that is inverted in neighborhood 2-opt-inter may contain the depot and it must contain complete families. Finally, neighborhood $\mathcal{N}_F$ is similar to neighborhood $\mathcal{N}_I$ for the FTSP if we look at each family as a node.

Example 33 (Neighborhoods RFTSP)**.** Consider the feasible solution for the RFTSP instance presented in Figure 7.1a, that is, $s = \{(0,1),(1,3),(3,4),(4,5),(5,0)\}$. Then, the neighborhoods associated with s are:

$$\begin{aligned}\mathscr{N}_I(s) = \quad & \{\{(0,1),(1,3),(3,4),(4,5),(5,0)\},\ \{(0,1),(1,4),(4,3),(3,5),(5,0)\},\\ & \{(0,1),(1,5),(5,4),(4,3),(3,0)\},\ \{(0,1),(1,3),(3,5),(5,4),(4,0)\}\}\end{aligned}$$

$$\begin{aligned}\mathscr{N}_O(s) = \quad & \{\{(0,1),(1,3),(3,4),(4,5),(5,0)\},\ \{(0,2),(2,3),(3,4),(4,5),(5,0)\},\\ & \{(0,1),(1,6),(6,4),(4,5),(5,0)\},\ \{(0,1),(1,3),(3,6),(6,5),(5,0)\},\\ & \{(0,1),(1,3),(3,4),(4,6),(6,0)\}\}\end{aligned}$$

$$\begin{aligned}2\text{-}opt\text{-}intra(s) = \quad & \{\{(0,1),(1,3),(3,4),(4,5),(5,0)\},\ \{(0,1),(1,4),(4,3),(3,5),(5,0)\},\\ & \{(0,1),(1,5),(5,4),(4,3),(3,0)\},\ \{(0,1),(1,3),(3,5),(5,4),(4,0)\}\}\end{aligned}$$

$$\begin{aligned}2\text{-}opt\text{-}inter(s) = \quad & \{\{(0,1),(1,3),(3,4),(4,5),(5,0)\},\ \{(0,3),(3,4),(4,5),(5,1),(1,0)\},\\ & \{(0,5),(5,4),(4,3),(3,1),(1,0)\}\}\end{aligned}$$

$$\mathscr{N}_F(s) = \quad \{\{(0,1),(1,3),(3,4),(4,5),(5,0)\},\ \{(0,3),(3,4),(4,5),(5,1),(1,0)\}\}$$

The sizes of the neighborhoods $\mathscr{N}_I(s)$ and 2-opt-intra(s) are at most $1 + \sum_{l\in\mathcal{M}} \frac{v_l\times(v_l-1)}{2}$, the size of the neighborhood 2-opt-inter(s) is, at most, $1 + 2 \times (L-1) + \frac{L\times(L-1)}{2}$ and the size of neighborhood $\mathscr{N}_F(s)$ has a maximum value of $1 + \frac{(L+1)\times L}{2}$.

To conclude, all the neighborhoods for the RFTSP are searched by using Algorithm 3.1 presented in Section 3.3.

7.3 Mathematical formulations for the RFTSP

We introduced the RFTSP in the beginning of Chapter 7 as being the FTSP with the additional consecutiveness condition, thus, any formulation for the FTSP can be used as a formulation for the RFTSP as long as we include constraints to model the consecutiveness condition. This section in divided into two main parts. The first part consists in adapting the mathematical models proposed for the FTSP in Chapter 4 by including constraints to guarantee the satisfaction of the consecutiveness condition, while in the second part we propose a new modeling approach for the RFTSP in which we address the interfamily and the intrafamily relationships as different subproblems. The latter formulation is called the inter- and intrafamily formulation. Considering a multi-visit family $l \in \mathcal{M}$, the set of constraints developed to formulate the consecutiveness condition is based on characteristic

(a) presented in Section 7.1, which states that there exists one and only one arc used in the cut-set $[(0 \cup N) \setminus F_l, F_l]$, whereas the formulation for the intrafamily subproblem of the inter- and intrafamily formulation is based on characteristic (b), which states that there must be a path with v_l nodes that only uses arcs linking nodes from the same family.

In Section 7.3.1 we present a formulation for the consecutiveness condition, and, consequently, how to adapt the FTSP formulations into RFTSP formulations. In Section 7.3.2 we present the inter- and intrafamily formulation. We conclude in Section 7.3.3 with an empirical comparison between the two modeling approaches proposed.

7.3.1 Formulating the consecutiveness condition

As we mentioned previously, in a feasible solution for the RFTSP, for a given multi-family $l \in \mathcal{M}$, there must be one and only one arc used in the cut-set $[F_l, (0 \cap N) \setminus F_l]$. Therefore, a formulation for the RFTSP may be obtained by adding the following constraints

$$x(F_l, (N \cup 0) \setminus F_l) = 1 \qquad \forall l \in \mathcal{M}, \tag{7.1}$$

to an FTSP formulation. Note that the consecutiveness constraints (7.1) are in polynomial number, since $|\mathcal{M}| \leq L$.

Any FTSP formulation may be used to solve the RFTSP as long as we include in the formulation the consecutiveness constraints (7.1). For simplification purposes, the RFTSP formulation which consists in adding the consecutiveness constraints to an FTSP formulation will be designated by the name of the FTSP formulation.

7.3.2 The inter- and intrafamily formulations

This formulation is based on the fact that the interfamily and the intrafamily relationships in the RFTSP may be seen as two different subproblems. As we must visit the nodes from the same family consecutively, the families may be seen as "supernodes". Consequently, as we must visit every family, a feasible solution for the RFTSP is an Hamiltonian circuit that goes through the "supernodes". Hence, we can formulate the interfamily subproblem as a TSP in which the nodes are the different families. Regarding the intrafamily subproblem, as we mentioned before, for each family l, with $l \in \mathcal{M}$, we wish to have an elementary path that only uses arcs (i, j), such that i and j are from family l. Consequently, we can formulate the intrafamily subproblem as an elementary path problem, with the additional constraints that: (i) the initial and the final nodes of the path are not predetermined; and (ii) the path must have v_l nodes. Recall that we are only required to formulate the intrafamily subproblem for the multi-visit families.

Consider the binary x variables defined in the generic formulation for the FTSP presented in Section 4.1, which have value 1 if the arc $(i, j) \in A$ is used in the circuit and value 0 otherwise. We start by presenting a generic formulation in which the inter- and intrafamily subproblems are modeled in a generic manner and then we provide formulations for the generic constraints. Thus, a generic formulation for the RFTSP based on the intrafamily and the interfamily subproblems is the following:

$$\textit{Minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{4.1}$$

Subject to:

$$\{(i,j) \in A : x_{ij} = 1\} \textit{ is a solution for the intrafamily subproblem} \tag{7.2}$$

$$\{(i,j) \in A : x_{ij} = 1\} \textit{ is a solution for the interfamily subproblem} \tag{7.3}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A. \tag{4.7}$$

The objective is to minimize the cost of the circuit, which is represented in (4.1). Constraints (7.3) and (7.2) are a generic representation of the inter- and intrafamily subproblems, respectively. Constraints (4.7) define the domain of the x variables.

In Section 7.3.2.1 we present different formulations for the intrafamily generic constraints (7.2), which will originate several inter- and intrafamily formulations, in Section 7.3.2.2 we present a formulation for the interfamily generic constraints (7.3) and in Section 7.3.2.3 we present a theoretical comparison of the several inter- and intrafamily formulations. Before doing so, we denote by A_l the set of arcs which have its initial and final nodes belonging to family $l \in \mathcal{M}$, that is, $A_l = \{(i, j) \in A : i \in F_l, j \in F_l\}$ and by R a subset of family nodes $R \subseteq F_l$ and by R' its complementary subset, that is, $R' = F_l \setminus R$.

7.3.2.1 Formulating the intrafamily subproblem

Considering a multi-visit family $l \in \mathcal{M}$, we wish to determine an elementary path that visits v_l nodes. Additionally, the path does not have a predefined initial nor final node. Thus, in order to formulate the intrafamily subproblem of the inter- and intrafamily formulation, we define binary variables $w^k_{ij} = 1$ if the arc $(i, j) \in A_l : j \neq k$ is used in the path that has $k \in F_l$ as its initial node, and $w^k_{ij} = 0$ otherwise. Binary variables p^k have value 1 if node $k \in F_l$ is the initial node of the path, and value 0 otherwise. Variables q^k_i are binary and have value 1 if node $i \in F_l : k \neq i$ is visited in the path which has $k \in F_l$ as its initial node, and value 0 otherwise and, finally, binary variables $u^k_i = 1$ if node $i \in F_l : i \neq k$ is the final node of the path that has $k \in F_l$ as its initial node, and $u^k_i = 0$ otherwise. Note that the w variables and the p, the q and the u variables may be seen as a disaggregation of the x variables and the y variables introduced in the generic formulation for the FTSP in Section 4.1, respectively, per possible initial node of the path. More precisely, if an arc $(i,j) \in A_l$ is used in the path then it is used in the circuit, thus, $x_{ij} = \sum_{k \in F_l : k \neq j} w_{ij}^k$ and if $k \in F_l$ is the initial node of the path or is visited in a path that has other initial node, then k is visited in the circuit, that is, $y_k = p^k + \sum_{i \in F_l : i \neq k} q_k^i$. The relationship between the x and the w variables is explicitly incorporated in the interfamily subproblem in Section 7.3.2.2, while the relationship between the y variables and the p and the q variables is implicit in the formulation of the intrafamily subproblem.

In order to obtain a formulation for the intrafamily subproblem, the generic intrafamily constraints (7.2) can be replaced with:

$$\sum_{j \in F_l} w_{kj}^k = p^k \qquad \forall l \in \mathcal{M},\ \forall k \in F_l \tag{7.4}$$

$$\sum_{j \in F_l} w_{ji}^k = \sum_{j \in F_l : k \neq j} w_{ij}^k + u_i^k \qquad \forall l \in \mathcal{M},\ \forall k \in F_l,\ \forall i \in F_l : i \neq k \tag{7.5}$$

$$\sum_{j \in F_l} w_{ji}^k = q_i^k \qquad \forall l \in \mathcal{M},\ \forall k \in F_l,\ \forall i \in F_l : i \neq k \tag{7.6}$$

$$\{k \in F_l, (i,j) \in A_l : w_{ij}^k = 1\} \textit{ is a connected path} \tag{7.7}$$

$$q_i^k \leq p^k \qquad \forall l \in \mathcal{M},\ \forall k \in F_l,\ \forall i \in F_l : i \neq k \tag{7.8}$$

$$u_i^k \leq p^k \qquad \forall l \in \mathcal{M},\ \forall k \in F_l,\ \forall i \in F_l : i \neq k \tag{7.9}$$

$$\sum_{k \in F_l} p^k = 1 \qquad \forall l \in \mathcal{M} \tag{7.10}$$

$$\sum_{k \in F_l} (p^k + \sum_{i \in F_l : i \neq k} q_k^i) = v_l \qquad \forall l \in \mathcal{M} \tag{7.11}$$

$$w_{ij}^k \in \{0,1\} \qquad \forall l \in \mathcal{M},\ \forall k \in F_l,\ \forall (i,j) \in A_l : j \neq k \tag{7.12}$$

$$p^k \in \{0,1\} \qquad \forall l \in \mathcal{M},\ \forall k \in F_l \tag{7.13}$$

$$q_i^k \in \{0,1\} \qquad \forall l \in \mathcal{M},\ \forall k \in F_l,\ \forall i \in F_l : i \neq k \tag{7.14}$$

$$u_i^k \in \{0,1\} \qquad \forall l \in \mathcal{M},\ \forall k \in F_l,\ \forall i \in F_l : i \neq k. \tag{7.15}$$

Constraints (7.4)-(7.7) define the path in each multi-visit family and are similar to constraints (4.2)-(4.4) of the generic model for the FTSP presented in Section 4.1. Constraints (7.4) state that if a path has k as its initial node, then there is an arc leaving k in the path. Constraints (7.5) ensure that the indegree and the outdegree of every node in the path, except for the initial and the final nodes of the path, are the same. The indegree of the final node of the path is 1, which is expressed by the value of the variable u. Constraints (7.6) guarantee that if node i is visited in the path which has k

as its initial node, then there is an arc in the referred path that has i as its final node. Constraints (7.8) and (7.9) state, respectively, that a node can only be used or be the final node of the path which has k as initial node, if k is, in fact, the initial node of the path. Constraints (7.10) ensure that there exists exactly one initial node in each multi-visit family, and consequently, in each path. The set of constraints (7.11) guarantees the family visits as, we already mentioned, if a node is visited, it is either the initial node of the path (variables p) or a visited node in a path that has a different node as initial node (variables q). Finally, constraints (7.12), (7.13), (7.14) and (7.15) define the domain of the w, the p, the q and the u variables, respectively. Similarly to what happened in Section 4.1, for a given family $l \in \mathcal{M}$ a solution that satisfies the equation system (7.4)-(7.6), (7.8)-(7.15) has an arc leaving $k \in F_l$, visits v_l nodes and each node that is visited, except for the initial and the final nodes of the path, has the same indegree and outdegree. However, it is not ensured that the solution is a connected path. If the solution is not connected, then there exists a subtour, which in this case means that there exists a circuit that does not contain the node chosen as the initial node of the path. Therefore, in order to guarantee that the path obtained is connected we may use an adaptation of the subtour elimination constraints proposed in Section 4.2 for the FTSP. We propose four different formulations for the path connectivity constraints (7.7) which are based on the SCF model presented in Section 4.2.1.1, the NCF model presented in Section 4.2.1.3, the CC model presented in Section 4.2.2.1 and the RV model presented in Section 4.2.2.2 for the FTSP. Note that we do not have formulations for the path connectivity based on the FCF model presented in Section 4.2.1.2 nor on the RFV model presented in Section 4.2.2.3 since the paths are already defined for a single family.

The several formulations for the path connectivity are presented considering a multi-visit family $l \in \mathcal{M}$. The formulation that is based on the SCF model is called the path single-commodity flow (P-SCF) and ensures the path connectivity by sending one flow with $v_l - 1$ units from the initial node of the path to the remaining nodes of the path. In the path multi-commodity flow (P-MCF) formulation, which is based on the NCF model, the path connectivity is guaranteed by sending $v_l - 1$ different flows, each one with one unit, from the initial node of the path to the remaining nodes of the path. The path connectivity cuts (P-CC) formulation, which is similar to the CC model, guarantees the path connectivity by ensuring that if there is a path from $k \in F_l$ to $m \in F_l, m \neq k$, then there exists an arc that is used in the path in every k-m cut. Finally, the path rounded visits (P-RV) formulation is similar to the RV model since it ensures that there exists an arc that is used in the cut-set $[R', R]$ if there are not enough nodes in R' to fulfill the family visits, however, as the initial node of the path is not predefined, the right-hand side of these constraints must be adapted.

In the ensuing sections we present the sets of constraints that ensure the path connectivity and that will allow us to develop several inter- and intrafamily formulations, which will only differ on the path connectivity constraints and which will be formalized in Section 7.3.2.3. More precisely, in Section 7.3.2.1.1 we present the P-SCF formulation, in Section 7.3.2.1.2 we present the P-MCF formulation, the P-CC formulation is presented in Section 7.3.2.1.3 and the P-RV formulation is presented in Section 7.3.2.1.4.

7.3.2.1.1 The path single-commodity flow

Variables g_{ij}^k represent the amount of flow that traverses arc $(i,j) \in A_l$ sent from node $k \in F_l, k \neq j$, which corresponds to the number of nodes that will still be visited in family $l \in \mathcal{M}$. Variables g are non-negative as their integrality is ensured by the other constraints of the model. The P-SCF formulation is the following:

$$\sum_{j \in F_l} g_{kj}^k = (v_l - 1)p^k \qquad \forall l \in \mathcal{M},\ \forall k \in F_l \tag{7.16}$$

$$\sum_{j \in F_l} g_{ji}^k = \sum_{j \in F_l : j \neq k} g_{ij}^k + q_i^k \qquad \forall l \in \mathcal{M},\ \forall k \in F_l,\ \forall i \in F_l : i \neq k \tag{7.17}$$

$$g_{ij}^k \leq (v_l - 1)w_{ij}^k \qquad \forall l \in \mathcal{M},\ \forall k \in F_l,\ \forall (i,j) \in A_l : j \neq k \tag{7.18}$$

$$g_{ij}^k \geq 0 \qquad \forall l \in \mathcal{M},\ \forall k \in F_l,\ \forall (i,j) \in A_l : j \neq k. \tag{7.19}$$

Constraints (7.16)-(7.19) are similar to constraints (4.9)-(4.12) from the SCF model presented in Section 4.2.1.1. More precisely, constraints (7.16) ensure that $v_l - 1$ units of flow must leave k if k is the initial node of the path. Otherwise, if $p^k = 0$ then there is no flow leaving k. Note that we only send $v_l - 1$ units of flow from k because we already visited k, which is a family node. Constraints (7.17) are the flow conservation constraints for the nodes in the path that are not its initial node and state that if node i is visited in the path that starts in k then one unit of flow from k must be left in i. Constraints (7.18) guarantee that the flow sent from k can only traverse an arc if that arc is used in the path that has k as its initial node and that the maximum amount of flow in each arc is $v_l - 1$. Finally, constraints (7.19) define the domain of the g variables.

7.3.2.1.2 The path multi-commodity flow

Variables h_{ij}^{km} are binary and have value 1 if the arc $(i,j) \in A_l$ is used to send one unit of flow from node $k \in F_l, k \neq j$ to node $m \in F_l, m \neq i$, with $l \in \mathcal{M}$. The P-MCF formulation is the following:

$$\sum_{j \in F_l} h_{kj}^{km} = q_m^k \qquad \forall l \in \mathcal{M},\ \forall k \in F_l,\ \forall m \in F_l : k \neq m \tag{7.20}$$

$$\sum_{j \in F_l} h_{jm}^{km} = q_m^k \qquad \forall l \in \mathcal{M},\ \forall k \in F_l,\ \forall m \in F_l : k \neq m \tag{7.21}$$

$$\sum_{j \in F_l : j \neq m} h_{ji}^{km} = \sum_{j \in F_l : j \neq k} h_{ij}^{km} \quad \forall l \in \mathcal{M},\ \forall k \in F_l,\ \forall m \in F_l : m \neq k,\ \forall i \in F_l : i \neq k, i \neq m \tag{7.22}$$

$$h_{ij}^{km} \leq w_{ij}^k \qquad \forall l \in \mathcal{M},\ \forall k \in F_l,\ \forall m \in F_l : m \neq k,\ \forall (i,j) \in A_l : i \neq m, j \neq k \tag{7.23}$$

$$h_{ij}^{km} \geq 0 \qquad \forall l \in \mathcal{M},\ \forall k \in F_l,\ \forall m \in F_l : m \neq k,\ \forall (i,j) \in A_l : i \neq m, j \neq k. \tag{7.24}$$

Constraints (7.20)-(7.24) are similar to constraints (4.19)-(4.21), (4.25) and (4.23) from the NCF model presented in Section 4.2.1.3. Constraints (7.20) and (7.21) ensure that if k is the initial node of the path and m is a visited node in the same path, then there is an arc leaving k and an arc entering m, respectively, which are used to send one unit of flow from k to m. Constrains (7.17) are the flow conservation constraints for the nodes that are neither the origin nor the destination of the flow and guarantee that the flow remains constant. Constraints (7.23), which represent the relationship between the h and the w variables, state that flow originated from node k can only traverse an arc that was chosen to be in the path that has k as its initial node. Finally, constraints (7.19) define the domain of the h variables.

7.3.2.1.3 The path connectivity cuts

Consider that for a specific multi-visit family $l \in \mathcal{M}$ we have a cut-set $[R', R]$ with $R \subseteq F_l$. Additionally, consider that $k \in R'$ and $m \in R$. Assuming that k is the initial node of the path and m is a visited node of the path, that is, $q_m^k = 1$, then there is an arc in the cut-set $[R', R]$ that has to be used in the path that starts in k in order to obtain a feasible solution for the RFTSP. The P-CC formulation is as follows:

$$w^k(R', R) \geq q_m^k \qquad \forall l \in \mathcal{M},\ \forall R \subseteq F_l,\ \forall k \in R',\ \forall m \in R. \tag{7.25}$$

For simplification purposes, constraints (7.25) will be designated as P-CC constraints henceforth.

7.3.2.1.4 The path rounded visits

Considering $R \subseteq F_l$, the P-RV formulation is the following:

$$w^k(R', R) \geq p^k \qquad \forall l \in \mathcal{M},\ \forall R \subseteq F_l : |R| \geq n_l - v_l + 1,\ \forall k \in R'. \tag{7.26}$$

Constraints (7.26), which will be called P-RV inequalities henceforth, are similar to the RV inequalities presented in Section 4.2.2.2 for the FTSP. However, as the initial node of the path is not predefined, the P-RV inequalities have a variable in their right-hand side instead of the value 1. Note that when $p^k = 0$ the P-RV inequalities are redundant, whereas, when $p^k = 1$ the path has $k \in R'$ as its initial node and there are not enough nodes in R' to fulfill the family visits since $|R| \geq n_l - v_l + 1 \implies |R'| \leq v_l$, therefore there exists an arc in the cut-set $[R', R]$ that has to be used in the path in order to obtain a feasible solution for the RFTSP.

7.3.2.2 Formulating the interfamily subproblem

As we mentioned at the beginning of Section 7.3.2, the interfamily subproblem of the inter- and intrafamily formulation may be seen as a TSP in which each node is a family. Therefore, the generic interfamily subproblem (7.3) may be formulated as follows:

$$\sum_{j \in N} x_{0j} = 1 \tag{4.2}$$

$$\sum_{j \in N} x_{j0} = \sum_{j \in N} x_{0j} \tag{7.27}$$

$$\sum_{j \in 0 \cup N} x_{ji} = \sum_{j \in 0 \cup N} x_{ij} \qquad \forall l \in \mathcal{U},\ \forall i \in F_l \tag{7.28}$$

$$\sum_{i \in (0 \cup N) \setminus F_l} \sum_{j \in F_l} x_{ij} = 1 \qquad \forall l \in \mathcal{U} \tag{7.29}$$

$$\sum_{i \in S'} \sum_{j \in S} x_{ij} \geq 1 \qquad \forall S \subseteq N : \exists l \in \mathcal{L} : F_l \subseteq S \tag{7.30}$$

$$\sum_{j \in (0 \cup N) \setminus F_l} x_{jk} = p^k \qquad \forall l \in \mathcal{M},\ \forall k \in F_l \tag{7.31}$$

$$\sum_{j \in F_l} x_{ji} = \sum_{k \in F_k : k \neq i} q_i^k \qquad \forall l \in \mathcal{M},\ \forall i \in F_l \tag{7.32}$$

$$\sum_{j \in (0 \cup N) \setminus F_l} x_{ij} = \sum_{k \in F_k : k \neq i} u_i^k \qquad \forall l \in \mathcal{M},\ \forall i \in F_l \tag{7.33}$$

$$\sum_{k \in F_l : k \neq j} w_{ij}^k = x_{ij} \qquad \forall l \in \mathcal{M},\ \forall (i,j) \in A_l. \tag{7.34}$$

Constraint (4.2) was already presented in the generic formulation for the FTSP in Section 4.1. Additionally, constraints (7.27) and (7.28) are a particular case of constraints (4.4) when $i = 0$ and when node i belongs to a single-visit family, respectively. Constraints (7.29) guarantee that one node is visited in a single-visit family and constraints (7.30) are the subtour elimination constraints for the interfamily subproblem. Constraints (7.30) may be seen as the connectivity cuts for the TSP however, since we are considering that each family is a node, we must ensure that S contains at least one family. Constraints (7.30) will be designated as the subtour elimination constraints for the interfamily subproblem henceforth. Constraints (7.31)-(7.34) relate the x variables to the other decision variables of the inter- and intrafamily formulation, namely the p, the q, the u and the w variables. Constraints (7.31) state that if k is the initial node of a family path there exists an arc entering k which has as initial node the depot or a node from a different family. The set of constraints (7.32) guarantees that if node i is visited in a family path that does not start in i, then there is an arc entering i which has as initial node a node from the same family as i. Constraints (7.33) ensure that if i is the last node to be visited in a multi-visit family, then there is an arc leaving i which has as final node the depot or a node from a different family. Finally, constraints (7.34) state that if an arc is in a family path then it is also in the circuit.

7.3.2.3 Theoretical comparison of the inter- and intrafamily formulations

As mentioned previously, we will define four distinct inter- and intrafamily models that only differ on how the generic path connectivity constraints (7.7) are formulated and which are designated by the same name as the path connectivity formulation. Consequently, all the inter- and intrafamily formulations are comprised of the domain of the x variables (4.7), the intrafamily subproblem (7.4)-(7.15) and the interfamily subproblem (4.2), (7.27)-(7.34). The P-SCF model is obtained by replacing the generic path connectivity constraints (7.7) with the P-SCF formulation (7.16)-(7.19). Similarly, the P-MCF model is obtained by replacing the generic path connectivity constraints (7.7) with the P-MCF formulation (7.20)-(7.24). The P-CC model is obtained by changing the generic path connectivity constraints (7.7) to the P-CC inequalities (7.25). Finally, the P-RV model is obtained by replacing the generic path connectivity constraints (7.7) with the P-RV inequalities (7.26).

Since the four inter- and intrafamily models, namely the P-SCF, the P-MCF, the P-CC and the P-RV models, only differ on the path connectivity constraints and such constraints are similar to the subtour elimination constraints for the FTSP presented in Section 4.2, their relationships in terms of LP relaxation are also similar. Therefore, we only present the results and we refer to reader to Section 4.2 for the detailed proofs. Recall that we defined, in Section 2.3, $\mathcal{V}^{LP}(M)$ as the LP relaxation value of the model M.

Proposition 24. *The LP relaxation of the P-SCF model is not comparable to the LP relaxation of the P-MCF model, to the LP relaxation of the P-CC model and to the LP relaxation of the P-RV model.*

Proof. The proofs are similar to the ones of Propositions 15, 17 and 20. □

Proposition 25. $\mathcal{V}^{LP}(P\text{-}MCF) = \mathcal{V}^{LP}(P\text{-}CC)$

Proof. Similar to the proof of Proposition 19. □

Proposition 26. *The LP relaxation of the P-MCF model is not comparable to the LP relaxation of the P-RV model.*

Proof. Similar to the proof of Proposition 17. □

Proposition 27. *The LP relaxation of the P-CC model is not comparable to the LP relaxation of the P-RV model.*

Proof. Similar to the proof of Proposition 23. □

7.3.3 Empirical comparison between the adapted formulations and the inter- and intrafamily formulations

This section is devoted to establishing an empirical comparison between the RFTSP formulations obtained by adapting the FTSP formulations and the inter- and intrafamily formulations. In order to do so we selected a subset of test instances, namely the symmetric instances $burma$, $bayg$ and att from the instance set 1 and the asymmetric instances $br17$, $ftv33$ and $ftv35$ from the instance set 3. We wish to compare the adapted FCF, CC, RFV and CC+RFV models for the RFTSP to the inter- and intrafamily formulations, which are the P-SCF, the P-MCF, the P-CC and the P-RV models. Recall that we denote by adapted model the FTSP model which has the same name with the additional consecutiveness constraints (7.1). Additionally, even though the P-SCF and the P-RV models are based on the SCF and the RV models for the FTSP, respectively, we are interested in comparing them to the FCF and the RFV models. Note that the P-SCF model may be seen as a flow model disaggregated per family, which intuitively corresponds to the FCF model, and in the P-RV model we ensure that, in each multi-visit family, an arc is used in the cut-set $[R', R]$ if the number of nodes in R' is not enough to fulfill the family visits, which is the underlying idea of the RFV model.

We expect that the relationships between the FCF, the CC, the RFV and the CC+RFV models remain the same for the RFTSP both in terms of LP relaxation value and of computational efficiency as the ones established in Sections 4.3 and 4.4 for the FTSP, since the only difference to the case of the FTSP is that we add the consecutiveness constraints (7.1). Even though the most efficient model to obtain the optimal values of the FTSP was the CC+RFV model with the y-separation algorithm, we will use the CC+RFV model with the exact separation for the RFTSP in the comparison since, in a first phase, we are only interested in comparing the LP relaxation values of adapted models when solving the RFTSP. Table 7.1 shows the average percentage of gap between the LP relaxation value and the optimal value ($\overline{gap}$) and the average computational time, in seconds, to obtain the LP relaxation value ($\overline{t_s}$) with the several adapted models. The percentage of gap for each instance was computed with the following formula: $gap = 100 \times (\textit{optimal value} - \textit{LP relaxation value})/\textit{optimal value}$. The detailed results for the instance set 1 are available in appendix, Table D.1 while the ones for the instance set 3 are available in appendix, Table D.2.

Table 7.1: Average of linear programming relaxation results with the adapted models for the RFTSP.

	Instance set 1		Instance set 3	
	$\overline{gap}$	$\overline{t_s}$	$\overline{gap}$	$\overline{t_s}$
FCF model	12.05%	1	7.79%	0
CC model	4.64%	0	1.53%	0
RFV model	0.71%	11	0.45%	5
CC+RFV model	0.34%	0	0.24%	0

As we expected the results shown in Table 7.1 are consistent with the theoretical and practical comparison established in Sections 4.3 and 4.4 for the FTSP. Even though the FCF model is not comparable to the CC and to the RFV models it provides a significantly higher average percentage of gap. In fact, it was the model that provided the highest gap in every instance tested. The CC model and the RFV model are not comparable, by observing the detailed Table D.1 we verify that the CC model provides a lower percentage of gap in instance $bayg\_1$ while the RFV model obtains a lower percentage of gap in instance $bayg\_2$. Similarly to what happened in the FTSP, the RFV model is the model that provides the lowest average gap amongst the FCF model, the CC model and the RFV model. When we combine the CC inequalities with the RFV inequalities in the CC+RFV model, which was solved with the exact separation, we were able to further improve the results obtained. More precisely, the CC+RFV model was the method that provided the lowest average gap both in the instance set 1 and in the instance set 3.

Table 7.2 shows the LP relaxation value obtained with the inter- and intrafamily formulations, namely the P-SCF, the P-MCF, the P-CC and the P-RV models, for the instances from the instance set 1. This table is divided into four parts, each one devoted to a different formulation and shows the LP relaxation value ($\mathcal{V}^{LP}$), the percentage of gap between the LP relaxation value and the optimal value (gap) and the computational time, in seconds, to obtain the LP relaxation value (t_s). The tables also contain, in the last row, the average of the results obtained.

Table 7.2: Linear programming relaxation results for the instance set 1 with the inter- and intrafamily formulations.

	P-SCF			P-MCF			P-CC			P-RV		
Instance	$\mathcal{V}^{LP}$	gap	t_s	$\mathcal{V}^{LP}$	gap	t_s	$\mathcal{V}^{LP}$	gap	t_s	$\mathcal{V}^{LP}$	gap	t_s
burma_1	14.70	0.00%	0	14.70	0.00%	0	14.70	0.00%	0	14.70	0.00%	0
burma_2	26.78	0.00%	1	26.78	0.00%	0	26.78	0.00%	0	26.78	0.00%	0
burma_3	11.89	0.00%	0	11.89	0.00%	0	11.89	0.00%	1	11.89	0.00%	1
bayg_1	7131.08	2.44%	1	7300.47	0.12%	1	7300.47	0.12%	0	7309.21	0.00%	0
bayg_2	6084.63	20.34%	0	7483.84	2.02%	1	7483.84	2.02%	0	7638.47	0.00%	0
bayg_3	7223.81	11.58%	1	8101.96	0.83%	1	8101.96	0.83%	0	8157.82	0.15%	0
att_1	42194.40	12.53%	2	45623.80	5.42%	11	45623.80	5.42%	2	45997.60	4.64%	4
att_2	26241.20	8.89%	4	28554.70	0.85%	8	28554.70	0.85%	1	28800.10	0.00%	2
att_3	11302.60	5.76%	2	11992.80	0.00%	5	11992.80	0.00%	1	11992.80	0.00%	0
$average$		6.84%	1		1.03%	3		1.03%	1		0.53%	1

The relationships between the several inter- and intrafamily formulations are consistent with the ones stated in Section 7.3.2.3. Even though the P-SCF is not comparable to all the other models, it provides the highest percentage of gap in every instance tested from the instance set 1. The P-MCF and the P-CC models provide the same LP relaxation value. The P-CC and the P-RV models are not comparable, see for example instances $bayg\_1$ and $bayg\_2$. The P-RV model was the model that provided the lowest average gap. More precisely, with the P-RV model we obtained an average gap of 0.53% while the second lowest average gap was of 1.03% obtained with the P-CC model and with the P-MCF model.

By comparing the results obtained with the inter- and intrafamily formulation to the ones obtained with the adapted formulations presented in Table 7.1, and ignoring the CC+RFV model, we verify that the best results in terms of LP relaxation value for the instance set 1 were obtained with the P-RV model. The P-SCF model provided lower gap values than the FCF model for every instance tested. The same conclusions can be drawn for the CC model and the P-CC model. More precisely, the P-CC model always provides a higher LP relaxation value than the CC model. Additionally, the P-RV model also provides a lower average gap than the RFV model.

Table 7.3 shows the LP relaxation value obtained with the inter- and intrafamily formulations for the test instances from the instance set 3 and it has the same layout as Table 7.2.

Table 7.3: Linear programming relaxation results for the instance set 3 with the inter- and intrafamily formulations.

	P-SCF			P-MCF			P-CC			P-RV		
Instance	$\mathcal{V}^{LP}$	gap	t_s	$\mathcal{V}^{LP}$	gap	t_s	$\mathcal{V}^{LP}$	gap	t_s	$\mathcal{V}^{LP}$	gap	t_s
br17_1	36	0.00%	0	36	0.00%	4	36	0.00%	0	36	0.00%	1
br17_2	28	0.00%	1	28	0.00%	16	28	0.00%	1	28	0.00%	2
br17_3	71.36	8.52%	0	76.33	2.14%	2	76.33	2.14%	1	78	0.00%	0
br17_4	70	0.00%	1	70	0.00%	4	70	0.00%	0	70	0.00%	0
ftv33_1	934.50	2.96%	1	956.50	0.67%	1	956.50	0.67%	0	956.50	0.67%	0
ftv33_2	438.00	1.35%	0	438.00	1.35%	0	438.00	1.35%	1	442.00	0.45%	0
ftv33_3	1720.20	4.65%	1	1801.00	0.17%	1	1801.00	0.17%	0	1801.00	0.17%	0
ftv33_4	922.86	3.47%	0	956	0.00%	0	956	0.00%	0	956	0.00%	0
ftv35_1	1080.96	2.70%	1	1111	0.00%	1	1111	0.00%	0	1111	0.00%	0
ftv35_2	521.43	6.89%	0	560	0.00%	1	560	0.00%	1	560	0.00%	0
ftv35_3	1365.46	4.11%	1	1424	0.00%	1	1424	0.00%	0	1424	0.00%	1
ftv35_4	1039.67	1.73%	1	1058	0.00%	1	1058	0.00%	0	1058	0.00%	0
$average$		3.03%	1		0.36%	3		0.36%	0		0.11%	0

Once again, the results presented in Table 7.3 are compatible with the theoretical results shown in Section 7.3.2.3 and we can draw conclusions similar to ones drawn when we analyzed Table 7.2. More precisely, the P-SCF model was the formulation that provided the highest average gap, followed by the P-MCF and the P-CC models, which provided the same average gap, and the model that provided the lowest average gap was the P-RV model. Additionally, when we compare these results to the ones shown in Table 7.2 we verify the lowest average gap was obtained with the P-RV model in the instance set 3, which is not surprising since all the proposed formulations are asymmetric and they usually provide higher LP relaxation values in instances with an asymmetric cost matrix.

Consider the detailed results shown in Table D.2. We verify that the inter- and intrafamily formulation provides a lower percentage of gap than the corresponding adapted formulation for every instance tested from the instance set 3. More precisely, the P-SCF model provides a lower gap than the FCF model, the P-CC model than the CC model and the P-RV model than the RFV model. Additionally, considering instance $ftv33\_2$, we verify that the LP relaxation value obtained with the P-SCF model is higher than the LP relaxation value obtained with the CC model. With these results we may conclude that the P-SCF model is not comparable to the CC model.

From the results shown in Tables 7.2 and 7.3 we verify that the P-CC and the P-RV models are not comparable. Therefore, by using a similar reasoning as the one used when we created the

CC+RFV model in Section 4.4.1 we decided to create the P-CC+RV model, which consists in using the P-CC inequalities (7.25) to ensure the path connectivity and the P-RV inequalities (7.26) as valid inequalities to improve the LP relaxation value, or vice-versa. Table 7.4 shows the average percentage of gap between the LP relaxation value and the optimal value ($\overline{gap}$) and the average computational time, in seconds, to obtain the LP relaxation value ($\overline{t_s}$) with the P-CC+RV model. The detailed results for the instances from the instance set 1 are available in appendix, Table D.3 and the ones for the instances from the instance set 3 are available in Table D.4.

Table 7.4: Average of linear programming relaxation results with P-CC+RV model.

	$\overline{gap}$	$\overline{t_s}$
Instance set 1	0.32%	1
Instance set 3	0.11%	0

Considering the instance set 1, the average gap obtained with the P-CC+RV model is lower than the one obtained with the best inter- and intrafamily formulation so far, which was the P-RV model. Nonetheless, the P-CC+RV model was only able to obtain a lower percentage of gap in one instance, instance att_1. Regarding the instance set 3, when we compare the P-CC+RV model to the P-RV model we verify that the P-CC+RV model did not provide a higher LP relaxation value in any instance.

Throughout this section we presented and analyzed the LP relaxation values provided by the proposed formulations for the RFTSP and concluded that the inter- and intrafamily formulations provide, in general, higher LP relaxation values than the adapted formulations. During our first analysis we did not take into account the computational time since the test instances are of small dimension and, thus, we cannot properly evaluate the efficiency of the several formulations. Consequently, we decided to solve the LP relaxation of the benchmark instances with higher dimensions. However, using the inter- and intrafamily formulations we could not solve the LP relaxation of instances $bier$, which have 127 nodes, due to lack of computational memory. Note that the number of variables and constraints in the inter- and intrafamily formulation are significantly higher than in the adapted formulations. Therefore, even though the inter- and intrafamily formulations are promising formulations considering their LP relaxation values, they require a more thorough study to be used in practice, namely in what regards the resolution method. Consequently, we will use the adapted formulations to solve the RFTSP as they are the most efficient ones to be used within a B&C approach.

7.4 The branch-and-cut algorithm for the RFTSP

In Chapter 5 we presented the B&C algorithm developed for the FTSP. More precisely, we presented the callback functions created, namely the user cut callback, the lazy constraint callback and the heuristic callback, which is what we are going to do throughout this section for the RFTSP. In Section 7.3 we proposed two distinct approaches to formulate the RFTSP. From the results shown in Section 7.3.3, we verified that the adapted formulations are more efficient in practice than the inter- and intrafamily formulations. Additionally, since the results of Section 7.3.3 are consistent with the ones for the FTSP presented in Section 4.4 we decided to use the most efficient method for the FTSP to solve the RFTSP, which is the CC+RFV model with the y-separation. Nevertheless, we decided to compute the LP relaxation values of the test instances used in Section 7.3.3 with the CC+RFV model with the y-separation to fully understand which is the impact, in terms of LP relaxation value, of separating the RFV inequalities heuristically instead of exactly. Table 7.5 shows the average percentage of gap between the LP relaxation value and the optimal value ($\overline{gap}$) and the average computational time, in seconds, to obtain the LP relaxation value ($\overline{t_s}$) obtained with the y-separation. The detailed results for the instance set 1 are available in appendix, Table D.5 and the ones for the instance set 3 are available in Table D.6.

Table 7.5: Average of linear programming relaxation results with y-separation.

	$\overline{gap}$	$\overline{t_s}$
Instance set 1	0.36%	1
Instance set 3	0.92%	0

Throughout this section we present the callback functions used in the B&C algorithm for the RFTSP considering the CC+RFV model with the y-separation. The B&C algorithm for the inter- and intrafamily formulations is presented in Appendix E.

Since the consecutiveness constraints (7.1) are in polynomial number, the constraints that are in exponential number in CC+RFV model for the RFTSP are the same as in the CC+RFV model for the FTSP, which are the CC inequalities (4.28) and the RFV inequalities (4.35). Consequently, the outline of the B&C algorithm for the CC+RFV model is similar to the one presented in Section 5.1 for the corresponding model for the FTSP. Therefore, the user cut callback and the lazy constraint callback for the RFTSP are the ones presented in Section 5.2. More precisely, the user cut callback is comprised of the heuristic separation algorithm that finds violated CC and RFV inequalities presented in Algorithm 5.7 and of the y-separation algorithm presented in Algorithm 5.8, which we recall separates all the CC inequalities but the RFV inequalities are separated in a heuristic manner.

Regarding the lazy constraint callback, we use Algorithm 5.2 with $max\_number\_cuts = 1$.

All there is left now is to present the heuristic callback used in the B&C algorithm for the RFTSP, which is done in Section 7.4.1.

7.4.1 Heuristic callback

The heuristic callback presented in Section 5.3 provides feasible solutions for the FTSP, consequently, it has to be adapted to provide feasible solutions for the RFTSP. Recall that the heuristic callback used in the B&C algorithm for the FTSP consisted in two constructive heuristics: the nearest neighbor heuristic considering a different cost matrix in which the cost of the arcs $(i, j) \in A$ is $c^*_{ij} = c_{ij} \times (1 - x^*_{ij})$, being (x^*, y^*) the solution of the LP relaxation of the B&C subproblem that we are addressing; and the random heuristic. These heuristics were presented in Section 3.3. After obtaining a feasible solution for the FTSP with the previous heuristics, we applied the local search procedure presented in Algorithm 5.10.

The heuristic callback used in a B&C algorithm to solve the RFTSP is similar to the heuristic callback described in the previous paragraph, with the heuristics and neighborhoods defined for the RFTSP in Section 7.2. Regarding the local search procedure, we search the RFTSP neighborhoods $\mathcal{N}_I, \mathcal{N}_O, \mathcal{N}_F$, 2-opt-inter and 2-opt-intra. Algorithm 7.1 shows the pseudocode for the local search algorithm used in the B&C algorithm for the RFTSP.

Algorithm 7.1 The local search procedure for the B&C algorithm for the RFTSP.

```
Require: A feasible solution $s$ for the RFTSP
1: Search $\mathcal{N}_I(s)$ and obtain $s^*$. Set $s = s^*$.
2: Search $\mathcal{N}_O(s)$ and obtain $s^*$. Set $s = s^*$.
3: Search $\mathcal{N}_F(s)$ and obtain $s^*$. Set $s = s^*$.
4: if The cost matrix is symmetric then
5:     Search 2-opt-inter($s$) and obtain $s^*$. Set $s = s^*$.
6:     Search 2-opt-intra($s$) and obtain $s^*$. Set $s = s^*$.
7: end if
Ensure: A feasible solution $s^*$ such that $Cost(s^*) \leq Cost(s)$.
```

The heuristic callback used in the B&C algorithm for the RFTSP is similar to the one used in the B&C algorithm for the FTSP, that is, during the first 250 B&C subproblems, we apply the heuristic callback with a frequency of 5 and, after that, the frequency drops to 10. We apply both constructive heuristics according to the following criterion: if the number of the B&C subproblem is even we use the nearest neighbor with the cost matrix c^* and if the number of the B&C subproblem is odd we use the random heuristic. Then, we apply the local search algorithm presented in Algorithm 7.1.

7.4.2 Computational experiment

In this section we present the computational experiment for the RFTSP considering the several sets of instances presented in Section 3.4. As we already established which formulation is the best in practice, we only present the results in terms of optimal value obtained with the CC+RFV model with the y-separation. In Section 7.4.2.1 we present the results obtained for the benchmark instances (instance set 1), the results for the generated instances based on symmetric TSP instances are presented in Section 7.4.2.2 (instance set 2), in Section 7.4.2.3 we present the results obtained for the generated instances based on asymmetric TSP instances (instance set 3) and, finally, in Section 7.4.2.4 we show the results obtained for the generated instances based on asymmetric UTPP instances (instance set 4).

The results were obtained by using the setting described in Section 5.4. More precisely, the results were obtained using the heuristic separation, considering 20 as the maximum of violated inequalities added per iteration and using the heuristic callback presented in Section 7.4.1. As we are focused on obtaining the optimal value of the test instances, we use the general purpose cuts and heuristics from CPLEX. To conclude, similarly to what was done when we addressed the FTSP, we set a time limit of 10800 seconds. Therefore, we consider that the B&C algorithm proposed for the RFTSP is an efficient method to solve an instance if it is able to obtain its optimal value within the time limit. Otherwise, the instance must be solved with heuristic methods.

The implementation of the proposed methods is original, except for the max-flow algorithm, for which we used the algorithm to solve the max-flow problem proposed by Goldberg and Tarjan (1988). The models were implemented in C++ and were solved by using the Concert Technology from CPLEX 12.6.1 (see e.g., IBM, 2014). All computational experiments were carried out in an Intel Core i7, 3.60 gigahertz, 8 gigabytes RAM.

7.4.2.1 Benchmark instances

Table 7.6 shows the optimal values for the RFTSP considering the instance set 1. This table contains the optimal value ($\mathcal{V}$), the computational time, in seconds, to obtain the optimal value (t_s), the number of B&C subproblems solved during the B&C algorithm (#sub) and the number of added violated CC inequalities (#CC) and RFV inequalities (#RFV). Table 7.6 also contains, in the last row, the average of the results obtained.

Table 7.6: Optimal values of the benchmark instances considering the RFTSP.

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
burma_1	14.70	0	0	8	25
burma_2	26.78	1	0	0	0
burma_3	11.89	0	0	6	17
bayg_1	7309.21	0	0	4	46
bayg_2	7638.47	1	2	12	171
bayg_3	8169.95	0	0	16	110
att_1	48237.80	2	387	172	1499
att_2	28800.10	0	0	42	158
att_3	11992.80	0	0	53	130
bier_1	46113.10	33	102	506	1203
bier_2	133285.00	225	743	2074	10988
bier_3	58809.90	157	473	1911	8012
a_1	1817.34	5634	567	9236	33598
a_2	[1616.94, 1654.95]	10803	696	6885	31844
a_3	[1479.00, 1497.69]	10801	942	8686	28966
average		1844	261	1974	7784

Table 7.6 does not contain all the benchmark instances, in fact, it does not contain the instances gr and pr, which have 666 and 1002 nodes, respectively, as we could not compute its optimal value due to lack of computational memory.

By observing Table 7.6 we verify that all instances have the optimal value in the column $\mathcal{V}$, except for instances a_2 and a_3. For these instances instead of the optimal value we have a pair $[LB, UB]$, where the value LB represents the best lower bound and the value UB the best upper bound for the optimal value obtained by the B&C algorithm after 10800 seconds of computational time. The average computational time to obtain the optimal value of the benchmark instances, ignoring instances a_2 and a_3, is of 643 seconds. The computational time of instances $burma$, $bayg$ and att is negligible, similarly to what happen when we obtained the optimal values of the FTSP for this instance set. However, the average time to obtain the optimal value of the instances $bier$ is of 152 seconds. When we compare this value to the average computational time obtained when we addressed the FTSP, which was 20 seconds, we verify that obtaining the optimal values of the RFTSP is more time consuming for the instances $bier$. Regarding instances a, similarly to what happened when we solved these instances for the FTSP we could only obtain the optimal value of one instance a within the time limit. Nevertheless, the instance that we could solve considering the FTSP was instance a_3 while in the case of the RFTSP we were able to obtain the optimal value of instance a_1. The average computational time to solve instances a in the RFTSP is higher, in fact, it took an average of 9079 seconds whereas for the FTSP the average computational time considering only instances a presented in Table 5.10 was of 8872 seconds.

To conclude, the number of benchmark instances that we could solve optimally is the same when we address the FTSP or the RFTSP. Nonetheless, the computational time to obtain the optimal value of the RFTSP is higher. More precisely, the average time to obtain the optimal value of instances that we could solve within the time limit considering the FTSP was of 296 seconds while the average computational time considering the RFTSP was of 466 seconds.

7.4.2.2 Generated instances based on symmetric TSP instances

Table 7.7 shows the optimal values for the RFTSP considering the instance set 2. This table shows the optimal value ($\mathcal{V}$), the computational time, in seconds, to obtain the optimal value (t_s), the number of B&C subproblems solved during the B&C algorithm (#sub) and the number of added violated CC inequalities (#CC) and RFV inequalities (#RFV). Table 7.7 also presents, in the last row, the average of the results obtained.

Table 7.7: Optimal values of the instances from the instance set 2 considering the RFTSP.

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
pr136_1	[91782.80, 92294]	10803	16318	2853	41391
pr136_2	47137	37	5	1244	1436
pr136_3	143663	2276	42907	264	11630
pr136_4	95354	35	14	275	1430
gr137_1	46595	8	23	131	612
gr137_2	37524	38	220	678	1665
gr137_3	61858	5	22	40	664
gr137_4	51862	24	168	216	1823
pr144_1	[52857.00, 53739]	10801	11135	4921	41400
pr144_2	36780	214	198	1610	4952
pr144_3	64614	506	8006	171	7506
pr144_4	54534	88	161	819	3313

Continues on the next page

Table 7.7

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
kroA150_1	42531	161	274	1886	3294
kroA150_2	21093	464	146	3706	3450
kroA150_3	73944	771	4329	2925	17903
kroA150_4	31952	494	25	3327	3224
kroB150_1	45505	449	514	3004	8231
kroB150_2	20531	4105	1846	10396	16296
kroB150_3	74338	34	6	372	1700
kroB150_4	32037	22	0	402	558
pr152_1	67407	274	202	1522	2971
pr152_2	47198	587	22	2619	3071
pr152_3	95005	163	896	962	5838
pr152_4	73109	578	2858	1560	9023
u159_1	30605	26	3	430	1591
u159_2	24356	276	23	1505	3048
u159_3	37397	8	0	82	973
u159_4	33435	115	223	805	3060
rat195_1	[1483,54, 1548]	10801	690	8303	23949
rat195_2	964	663	97	2150	2072
rat195_3	2148	649	1141	1548	9841
rat195_4	1447	335	42	1321	3852
d198_1	[11588.00, 11607]	10802	3016	5596	75015
d198_2	10524	2682	98	3783	5779
d198_3	16012	185	86	401	4194
d198_4	13974	2100	1331	1957	23417
kroA200_1	56410	1098	631	2316	8258
kroA200_2	[30284.60, 32282]	10801	671	6883	22231
kroA200_3	112865	879	2678	1446	14072
kroA200_4	60220	7950	1841	6573	42356
kroB200_1	54520	1749	167	3351	7434
kroB200_2	[29890.30, 30684]	10805	2178	14650	45342
kroB200_3	107239	2227	6256	1827	24564

Continues on the next page

Table 7.7

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
kroB200_4	52541	1721	74	3975	8412
gr202_1	27231	2049	1495	2887	21950
gr202_2	15811	900	231	2366	6110
gr202_3	40966	65	123	197	1852
gr202_4	[31014.00, 31203]	10801	2486	5503	57767
pr226_1	[61828.80, 65194]	10802	159	2863	58880
pr226_2	59677	9994	0	4036	3398
pr226_3	[88633.20, 89682]	10801	3155	4742	93671
pr226_4	63670	741	0	1570	4378
gr229_1	74432	1500	1716	2116	13043
gr229_2	31761	2205	65	3286	3240
gr229_3	109222	267	275	1116	8017
gr229_4	48338	1527	283	2100	13117
gil262_1	5305	9944	2557	3891	33046
gil262_2	[2794.47, 2965]	10807	382	9682	13282
gil262_3	9331	508	852	640	5412
gil262_4	[6669.50, 6696]	10802	2316	3422	45760
pr264_1	37561	5197	661	4345	26145
pr264_2	[28564.30, 33626]	10802	0	2907	3133
pr264_3	[53991.40, 56144]	10802	1897	2472	56032
pr264_4	[42778.00, 45737]	10803	1000	3427	50848
average		3439	2050	2787	16280

From Table 7.7 we verify that, similarly to what happened when we addressed the FTSP, there are several instances that we could not solve within the time limit. For these instances in column $\mathcal{V}$ we present a pair $[LB, UB]$ where LB is the best lower bound and UB is the best upper bound for the optimal value found by the B&C algorithm after 10800 seconds of computational time.

We were able to obtain the optimal value of 50 instances from the instance set 2 within the time limit. More precisely, we were able to obtain the optimal value of instances $gr137$, $kroA150$, $kroB150$, $pr152$, $u159$ and $gr229$. As for the instances $pr136$, $pr144$, $rat195$, $d198$, $kroA200$, $kroB200$ and $gr202$, we were able to obtain the optimal value of three of the four existing types of each instance. Instances $pr226$, $gil262$ and $pr264$ were the most difficult instances to solve since we could only obtain the optimal value of two types of instances $pr226$ and $gil262$ and of instance $pr264\_1$. Similarly to what we saw in Section 5.4.2 when we addressed the FTSP, the difficulty of each instance is more correlated with the family visits than with the dimension of the instance, as we could obtain the optimal value of instance $pr264\_1$ within the time limit but we could not solve instance $pr136\_1$, which corresponds to one of the smallest instances from this instance set. In order to evaluate the behavior of the exact method for the RFTSP for each instance type we clustered the results obtained by instance type. Table 7.8 shows the average computational time, in seconds, to obtain the optimal value ($\overline{t_s}$) and the number of instances solved optimally (#solved) for each instance type. Recall that, in the instance set 2, there are 16 instances of each type.

Table 7.8: Statistics for the optimal value by instance type for the instance set 2 considering the RFTSP.

	$\overline{t_s}$	#solved
Type $reference$	4779	11
Type low	4086	12
Type $high$	1884	14
Type $mixed$	3009	13

Table 7.8 shows that CC+RFV model with the y-separation algorithm for the RFTSP obtained the best results for the instances of type $high$. More precisely, the average computational time for instances of type $high$ was 1884 while the second lowest average computational time was of 3009, obtained in instances of type $mixed$, which is significantly higher. In addition, the largest number of instances solved were of the type $high$, for which we could obtain the optimal value of all instances, except for instances $pr226\_3$ and $pr264\_3$. These results are not surprising since we concluded in Section 5.4.2 that the RFV inequalities are more effective when the number of visits per family is high, which characterizes instances of type $high$. The worst results both in terms of average computational time and of number of instances solved within the time limit were obtained for the $reference$ type instances.

We were able to solve up to optimality 50 instances from the instance set 2 while, when addressing the FTSP in Section 5.4.2, we were only able to solve to optimality 49 instances. Even though the number of instances solved in the RFTSP is slightly higher, the average computational time is also slightly higher. More precisely, the average computational time to obtain the optimal values of the FTSP was of 3426 seconds while for the RFTSP was of 3439. This shows that the exact method for the RFTSP is as efficient as the corresponding method for the FTSP.

7.4.2.3 Generated instances based on asymmetric TSP instances

Table 7.9 shows the optimal values for the RFTSP considering the instances from the set 3. This table displays the optimal value ($\mathcal{V}$), the computational time, in seconds, to obtain the optimal value (t_s), the number of B&C subproblems solved during the B&C algorithm (#sub) and the number of added violated CC inequalities (#CC) and RFV inequalities (#RFV). Table 7.9 also contains, in the last row, the average of the results obtained.

Table 7.9: Optimal values of the instances from the instance set 3 considering the RFTSP.

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
br17_1	36	1	0	15	54
br17_2	28	0	0	17	46
br17_3	78	0	0	5	24
br17_4	70	0	0	6	30
ftv33_1	963	1	0	8	184
ftv33_2	444	0	0	85	137
ftv33_3	1804	0	0	6	115
ftv33_4	956	0	0	8	134
ftv35_1	1111	1	0	12	284
ftv35_2	560	0	0	35	147
ftv35_3	1424	0	0	8	163
ftv35_4	1058	0	0	12	133
ftv38_1	961	1	22	45	432
ftv38_2	392	0	0	76	81
ftv38_3	1782	0	6	23	123
ftv38_4	830	1	28	72	417
p43_1	5510	1	7	70	729
p43_2	5481	0	0	174	418
p43_3	5633	1	3	7	208
p43_4	5508	0	0	5	199
ftv44_1	1237	1	0	33	367
ftv44_2	732	0	1	104	278
ftv44_3	1734	1	0	6	266
ftv44_4	1207	0	29	61	557

Continues on the next page

Table 7.9

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
ftv47_1	1740	1	27	52	446
ftv47_2	986	1	20	318	446
ftv47_3	2205	0	0	5	229
ftv47_4	1769	1	0	33	279
ry48p_1	15969	1	4	70	598
ry48p_2	9035	3	21	1002	1235
ry48p_3	21602	0	0	4	122
ry48p_4	15578	1	31	61	433
ft53_1	4146	1	13	163	517 3
ft53_2	2925	0	0	111	263 1
ft53_3	7571	1	25	21	359
ft53_4	5514	1	51	108	711 8
ftv55_1	590	0	0	49	252
ftv55_2	365	1	0	364	164
ftv55_3	1402	2	119	182	1447
ftv55_4	737	2	6	280	758
ftv64_1	1885	5	100	466	1998
ftv64_2	1101	3	25	971	753
ftv64_3	3194	1	0	5	309
ftv64_4	2825	1	17	20	675
ft70_1	22735	1	0	18	364
ft70_2	16182	4	45	150	1338
ft70_3	331454	0	0	10	392
ft70_4	28560	1	4	16	496
ftv70_1	1089	3	10	317	896
ftv70_2	741	6	47	594	769
ftv70_3	2078	8	411	85	3322
ftv70_4	1919	3	115	127	1674
kro124p_1	57908	37	522	960	7158
kro124p_2	19059	61	119	2548	2215
kro124p_3	82931	8	231	81	1540

Continues on the next page

Table 7.9

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
kro124p_4	35002	17	45	725	1895
ftv170_1	2516	8203	5124	7978	80957
ftv170_2	[1489.93, 1612]	10803	640	16177	45936
ftv170_3	3657	175	790	280	8706
ftv170_4	[2449.66, 2522]	10802	2594	5729	70697
rbg323_1	[1485.84, 1668]	10813	0	388	7554
rbg323_2	[624.91, 836]	10807	0	2135	4405
rbg323_3	[2604.84, 2887]	10806	0	274	11166
rbg323_4	[1391.19, 1572]	10815	0	372	7208
rbg358_1	[1623.63, 1974]	10807	0	383	6958
rbg358_2	[650.09, 908]	10807	0	825	5974
rbg358_3	2976	717	13	135	3374
rbg358_4	2135	848	35	75	5830
rbg403_1	[1453.81, 1966]	10809	0	314	4887
rbg403_2	[598.05, 1039]	10819	0	932	4248
rbg403_3	[3077.33, 3619]	10801	0	56	9644
rbg403_4	[1655.86, 2113]	10809	0	294	5627
rbg443_1	[2005.56, 2511]	10811	0	404	4454
rbg443_2	[1034.65, 1392]	10808	0	461	3733
rbg443_3	3393	4057	0	69	3619
rbg443_4	[2184.87, 2649]	10808	0	70	5210
average		2320	149	634	4471

From the instance set 3 we were able to obtain the optimal value of 57 out of the 76 instances within the time limit. For the instances that we could not solve within the time limit we present, in column $\mathcal{V}$, a pair $[LB, UB]$ in which LB is the best lower bound and UB is the best upper bound for the optimal value found by the B&C algorithm after 10800 seconds of computational time. By observing the computational times to obtain the optimal value we verify that they are negligible for the instances up to 70 nodes, namely instances $br17$, $ftv33$, $ftv35$, $ftv38$, $p43$, $ftv44$, $ftv47$, $ry48p$, $ft53$, $ftv55$, $ftv64$, $ft70$ and $ftv70$, as the maximum computational time obtained in these instances was of 8 seconds. Instances $kro124p$ were also efficiently solved optimally, in fact, the average computational time to obtain the optimal value of these instances was of 31 seconds. From the 20 instances which have a number of nodes greater than or equal to 170, we were only able to obtain the optimal value of five instances within the time limit, which were instances $ftv170\_1$, $ftv170\_3$, $rbg358\_3$, $rbg358\_4$ and $rbg443\_3$. Notice that, three of the five instances that we could solve within the time limit are instances of type $high$. Therefore, similarly to what we did in Section 7.4.2.2, we clustered the results obtained by instance type to verify whether or not, for this instance set, there are types of instance that are easier to solve than the others. Table 7.10 shows the average computational time, in seconds, to obtain the optimal value ($\overline{t_s}$) and the number of instances solved optimally (#solved) for each instance type. Recall that, in the instance set 3, there are 19 instances of each type.

Table 7.10: Statistics for the optimal value by instance type for the instance set 3 considering the RFTSP.

	$\overline{t_s}$	#solved
Type $reference$	2710	15
Type low	2849	14
Type $high$	1399	17
Type $mixed$	2322	15

Table 7.10 shows that the instance type $high$ was not only one with the lowest average computational time but also the type of instances with the highest number of instances solved to optimality within the time limit, whereas for the instance type low was the opposite. More precisely, the instance type low was the one with the highest computational time and with the lowest number of instances solved up to optimality within the time limit. These results are not surprising due to the effectiveness of the RFV inequalities when the number of visits per family is high.

In order to compare the RFTSP to the FTSP regarding the instance set 3, consider the results presented in Section 5.4.3. We observe that for the instance set 3 and concerning the RFTSP, the number of instances that we could not solve within the time limit increases significantly. More precisely, when considering the FTSP we could not solve one instance within the time limit, whereas when addressing the RFTSP we could not solve 15 instances. Considering only the instances that we could solve within the time limit in both problems, the average computational time obtained was of 101 and 233 seconds for the FTSP and the RFTSP, respectively. These results show that, for the instance set 3, the RFTSP is more computationally challenging than the FTSP.

7.4.2.4 Generated instances based on asymmetric UTPP instances

Table 7.11 shows the optimal values for the RFTSP considering the instances from the set 4. This table shows the optimal value ($\mathcal{V}$), the computational time, in seconds, to obtain the optimal value (t_s), the number of B&C subproblems solved during the B&C algorithm (#sub) and the number of added violated CC inequalities (#CC) and RFV inequalities (#RFV). In addition, Table 7.11 presents, in the last row, the average of the results obtained.

Table 7.11: Optimal values of the instances from the instance set 4 considering the RFTSP.

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
AsimSingh50_1	466	0	9	30	380
AsimSingh50_2	212	0	0	1	3
AsimSingh50_3	750	1	0	11	137
AsimSingh50_4	383	0	0	1	19
AsimSingh100_1	1101	15	54	244	3560
AsimSingh100_2	476	2	0	179	169
AsimSingh100_3	1492	1	0	8	223
AsimSingh100_4	748	80	90	1433	8084
AsimSingh150_1	[1274.33, 1285]	10803	0	2436	21124
AsimSingh150_2	770	22	0	267	6031
AsimSingh150_3	1933	24	8	124	1895
AsimSingh150_4	[946.87, 951]	10805	0	2672	34295
AsimSingh200_1	1631	17	0	29	603
AsimSingh200_2	1090	18	0	39	445
AsimSingh200_3	2612	138	58	757	3753
AsimSingh200_4	1563	52	0	44	948
AsimSingh250_1	2279	65	0	15	751
AsimSingh250_2	1241	17	0	21	281
AsimSingh250_3	3420	1309	101	224	12994
AsimSingh250_4	2630	25	0	12	334
AsimSingh300_1	[2717.01, 2732]	10804	0	228	10297
AsimSingh300_2	1647	66	0	52	926
AsimSingh300_3	[4099.90, 4145]	10801	0	201	15429
AsimSingh300_4	[2820.75, 2845]	10801	0	52	10401
average		2328	13	378	5545

When addressing the RFTSP, we could obtain the optimal value of 19 of the 24 instances from the instance set 4. The instances that we could not obtain the optimal value within the time limit were instances $AsimSingh150\_1$, $AsimSingh150\_4$, $AsimSingh300\_1$, $AsimSingh300\_2$ and $AsimSingh300\_4$. The computational time to solve instances $AsimSingh50$ is negligible. The B&C algorithm proposed is also efficient when addressing instances $AsimSingh100$ and $AsimSingh200$ since it takes an average computational time of 25 and 56 seconds, respectively, to find their optimal value. Regarding instances $AsimSingh250$, we were also able to find their optimal value in a reasonable amount of computational time. More precisely, the average computational time to find the optimal value of instances $AsimSingh250$ is 354 seconds. Apart from the instances $AsimSingh150$ and $AsimSingh300$ that we could not solve within the time limit, the others were solved very efficiently. In fact, we were able to obtain the optimal value of instances $AsimSingh150\_2$, $AsimSingh150\_3$ and $AsimSingh300\_2$ in 22, 24 and 66 seconds, respectively. Once again, we decided to group the instances by instances type. Table 7.12 shows the average computational time, in seconds, to obtain the optimal value ($\overline{t_s}$) and the number of instances solved optimally (#solved) for each instance type. Note that, in the instance set 4, there are 6 instances of each type.

Table 7.12: Statistics for the optimal value by instance type for the instance set 4 considering the RFTSP.

	$\overline{t_s}$	#solved
Type $reference$	3617	4
Type low	21	6
Type $high$	2046	5
Type $mixed$	3627	4

Unlike what happens in the instance sets 2 and 3, the easiest type of instance to solve was the type low. In addition, the instance type low was the only one for which we could obtain the optimal value of every instance within the time limit. Note that the significant difference in the average computational time between the instance type low and the other types of instances is due to the fact that for the latter types there are instances that were not solved within the time limit. However, if we compute the average computational time only considering the instances that we could solve within the time limit we obtain an average of 24, 295 and 39 seconds for the type of instances $reference$, $high$ and $mixed$.

When we compare these results to the ones obtained for the FTSP for this instance set, which were presented in Section 5.4.4, we verify that the RFTSP is significantly more challenging to solve than the FTSP. More precisely, when we were addressing the FTSP we were able to obtain the optimal value of every instance from the instance set 4 within the time limit, in fact, we were able to obtain the optimal values of the instances from the instance set 4 in a maximum computational time of 20 seconds.

Summary

In this section we presented the B&C algorithm with the CC+RFV model with the y-separation algorithm for the RFTSP. We used the test instances presented in Section 3.4 as RFTSP instances and tried to solve them within the time limit of 10800 seconds. Regarding the benchmark instances, and similarly to what happened for the FTSP, we were able to obtain the optimal value of 13 instances within the time limit, which correspond to all instances up to 127 nodes and one instance with 280 nodes.

Considering the instance set 2, we were able to find the optimal value of 50 out of 64 instances when solving the RFTSP, which are similar results to the ones obtained when we solved the FTSP. The B&C algorithm proposed was more efficient, in terms of computational time, for the instances of type $high$ and less efficient for the instances of type $reference$.

From the 72 instances from the instance set 3, we were able to obtain the optimal value of 57 instances. The type of instance from this instance set for which the B&C algorithm was more efficient was type $high$ whereas the more time consuming was type low. When we solved this instance set in the FTSP case we were able to obtain the optimal value of all instances, except for one. Therefore, we can conclude that these instances are significantly more complicated to address in the RFTSP case.

Finally, we were able to obtain the optimal value of 19 instances from the 24 instances from the instance set 4. Unlike the other instance sets, the B&C algorithm was more efficient for the instances of type low and less efficient for the instances of type $mixed$. When we solved the FTSP, we were able to obtain the optimal value of every instance from the instance set 4 within the time limit. Therefore, the RFTSP is significantly more difficult to solve than the FTSP considering this instance set.

The methods proposed for the FTSP behave similarly when addressing the RFTSP, which was expected as the RFTSP only has to satisfy the additional consecutiveness constraints (7.1). Nonetheless, the number of instances that we could solve optimally when addressing the RFTSP is lower when compared to the number of instances solved when considering the FTSP, specially for the asymmetric instances of the instance sets 3 and 4. Although the adapted B&C algorithm presented for the RFTSP provide satisfactory results in practice, we believe that we would have obtained a better performance if we had developed a specific method for the RFTSP that takes into account the specificities of this variant.

Similarly to what we did when we addressed the FTSP, the instances that we could not solve within the time limit will be solved with heuristic procedures in Section 7.5. Table 7.13 shows a summary of the best upper bounds found by the B&C algorithm after 10800 seconds of computational time for the instances which still have an unknown optimal value.

Table 7.13: Best upper bounds found by the B&C algorithm for the RFTSP.

Instance	Best upper bounds B&C algorithm
Instance set 1	
a_2	1654.95
a_3	1497.69
Instance set 2	
pr136_1	92294
pr144_1	53739
rat195_1	1548
d198_1	11607
kroA200_2	32282
kroB200_2	30684
gr202_4	31203
pr226_1	65194
pr226_3	89682
gil262_2	2965
gil262_4	6696
pr264_2	33626
pr264_3	59144
pr264_4	45737
Instance set 3	
ftv170_2	1612
ftv170_4	2522
rbg323_1	1668
rbg323_2	836
rbg323_3	2887

Continues on the next page

Table 7.13

Instance	Best upper bounds B&C algorithm
rbg323_4	1572
rbg358_1	1974
rbg358_2	908
rbg403_1	1966
rbg403_2	1039
rbg403_3	3619
rbg403_4	2113
rbg443_1	2511
rbg443_2	1392
rbg443_4	2449
Instance set 4	
AsimSingh150_1	1285
AsimSingh150_4	951
AsimSingh300_1	2732
AsimSingh300_3	4145
AsimSingh300_4	2845

7.5 Heuristic algorithms for the RFTSP

In Chapter 6 we proposed three distinct heuristics for the FTSP, namely a genetic algorithm in Section 6.1, an ILS algorithm in Section 6.2 and an hybrid algorithm in Section 6.3. From the summary of the results presented in Section 6.4, we concluded that the methods that provided the best quality solutions were the ILS algorithm and the hybrid algorithm. Therefore, these are the methods that we are going to use to obtain feasible solutions for the RFTSP instances that the exact methods could not solve within the time limit.

We present the adaptation of the ILS algorithm for the RFTSP in Section 7.5.1 while the adaptation of the hybrid algorithm is shown in Section 7.5.2. We conclude this section with the computational experiment in Section 7.5.3.

7.5.1 The iterated local search algorithm

We presented the ILS algorithm in Section 6.2 as an iterative process that iterates between a local search algorithm and a perturbation method. To apply the ILS algorithm to the RFTSP we must adapt the referred method.

The first step of the ILS algorithm (see Algorithm 6.3) is to determine a feasible solution for the RFTSP. In order to do so we use the nearest neighbor heuristic presented in Section 7.2. The local search procedure used when we are addressing the RFTSP consists in searching the neighborhoods $\mathscr{N}_I$, $\mathscr{N}_O$, 2-opt-intra, 2-opt-inter and $\mathscr{N}_F$, which were presented in Section 7.2. Recall that we only search neighborhoods 2-opt-intra and 2-opt-inter when we have a symmetric cost matrix. The neighborhoods are searched by using the Algorithm 3.1 presented in Section 3.3 and the pseudocode for the local search algorithm used in the ILS algorithm developed for the RFTSP is available in Algorithm 7.2.

Algorithm 7.2 The local search procedure used in the ILS algorithm for the RFTSP.

```
Require: A feasible solution $s$ for the RFTSP.
1: $cost\_old = Cost(s)$.
2: $cost\_new = -\infty$.
3: while $cost\_old > cost\_new$ do
4:     $cost\_old = Cost(s)$.
5:     Search $\mathscr{N}_I(s)$ and obtain $s^*$. Set $s = s^*$.
6:     Search $\mathscr{N}_O(s)$ and obtain $s^*$. Set $s = s^*$.
7:     Search $\mathscr{N}_F(s)$ and obtain $s^*$. Set $s = s^*$.
8:     if The cost matrix is symmetric then
9:         Search 2-opt-inter(s) and obtain $s^*$. Set $s = s^*$.
10:        Search 2-opt-intra(s) and obtain $s^*$. Set $s = s^*$.
11:    end if
12:    Set $cost\_new = Cost(s^*)$ and $s = s^*$.
13: end while
Ensure: Solution $s^*$ such that $Cost(s^*) \leq Cost(s)$.
```

Regarding the perturbation method and the FTSP, we recall that given a feasible solution it consists in choosing v_l nodes from each family $l \in \mathcal{L}$ according to the choosing criterion $Least\_chosen$, inserting the chosen nodes in the solution in the best possible position and, finally, removing the extra nodes according to a combination of the removal criteria $Greedy$ and $Random\_removal$. In order to apply the perturbation method to the RFTSP we must ensure that the solution obtained after removing the extra nodes is feasible for the RFTSP. Therefore, the only adaptation that has to be made is concerning the insertion of the chosen nodes since if we ensure that the solution with the extra nodes satisfies the consecutiveness condition, then when we remove the extra nodes the solution obtained also satisfies the consecutiveness condition. In order for the solution with the extra nodes to satisfy the consecutiveness condition we can only insert the chosen nodes from the family $l \in \mathcal{L}$ in the middle of arcs $(i, j) \in A$ such that $i \in F_l$ or $j \in F_l$. To conclude, the perturbation method used in the ILS algorithm for the RFTSP is similar to the perturbation method presented in Algorithm 6.5 except for the insertion of the chosen nodes, which must be inserted in the best possible position such that at least one of the adjacent nodes is from the same family. Algorithm 7.3 shows the pseudocode for the perturbation algorithm used in the ILS algorithm for the RFTSP.

Algorithm 7.3 The perturbation method used in the ILS algorithm for the RFTSP.

```
Require: A feasible solution $s$ for the RFTSP.
1: for all $l \in \mathcal{L}$ do
2:     Choose $v_l$ nodes from family $l$ according to the choosing criterion $Least\_chosen$.
3:     Insert the chosen nodes in $s$ in the best possible position such that at least one adjacent node
       is from family $l$.
4: end for
5: Remove the extra nodes by using the combination of both removal criteria: apply criterion
   $Greedy$ but once in every 10 iterations apply criterion $Random$.
```

7.5.2 The hybrid algorithm

The hybrid algorithm presented in Section 6.3 has two phases: the constructive phase and the improvement phase. The improvement phase consists in applying the ILS algorithm described in Section 6.2 directly to the solution obtained in the constructive phase of the hybrid algorithm. As we already presented the ILS algorithm for the RFTSP in Section 7.5.1, we only need to show how the constructive phase of the hybrid algorithm should be adapted to provide a feasible solution for the RFTSP.

In the constructive phase of the hybrid algorithm applied to the FTSP, we use the CC+RFV model with the y-separation to obtain a solution for a partial FTSP which is defined by the nodes in the set $Core$. Then, the partial solution is completed by using a savings heuristics. In order for the procedure described previously to be used to provide feasible solutions for the RFTSP we must use the exact method for the RFTSP described in Section 7.3, which is basically using the y-separation to solve the CC+RFV model with the additional consecutiveness constraints (7.1). The savings procedure consists in completing the partial solution by inserting the necessary nodes to make it feasible in terms of visited nodes. The nodes are inserted in the best possible position. Since we must ensure that the nodes from the same family are visited consecutively, if the number of family nodes in the partial solution is zero then the node has to be inserted between two nodes such that: one is the depot or both are from different families. Otherwise, that is, if there are family nodes in the partial solution, then the node can only be inserted in a position in which at least one of the adjacent nodes is from the same family.

7.5.3 Computational experiment

A preliminary computational experiment showed that, similarly to what happened in Chapter 6 when we addressed the FTSP, we cannot establish a relationship of dominance between the ILS algorithm and the hybrid algorithm since there are instances in which the ILS algorithm provides the solution with the lowest cost while there are others where the hybrid algorithm provides the best solution. Therefore, we decided to carry out a computational study with both heuristic algorithms.

All the algorithms presented in this chapter were developed within the scope of this dissertation and were implemented in C++. The computational experiment was carried out in an Intel Core i7, 3.60 gigahertz, 8 gigabytes RAM, as before.

Since the RFTSP was never addressed in the literature there are no upper bounds available. Therefore, in order to evaluate the quality of the solutions provided by the proposed heuristics we decided to use as reference values the upper bounds provided by the B&C algorithm available in Table 7.13. Consequently, the percentage of gap is computed by using the following formula: $gap = 100 \times (\textit{heuristic solution} - \textit{B\&C solution}) / \textit{B\&C solution}$. Recall that we could not compute any lower or upper bounds for instances gr and pr from the instance set 1 due to its dimension. Thus, we decided to use as reference value the best solution found by the proposed heuristics. Table 7.14 shows the reference values for the instances gr and pr. Note that for these instances the percentage of gap is computed by using the formula presented previously with the values presented in Table 7.14 instead of the *B&C solution*.

Table 7.14: Reference values for the instance gr and pr considering the RFTSP.

Instance	Reference value
gr_1	1842.25
gr_2	1342.17
gr_3	1327.26
pr_1	158362.92
pr_2	176086.63
pr_3	129436.70

As the ILS algorithm and the hybrid algorithm use random procedures we did five runs, with different seeds, for each instance. This allowed us to evaluate the robustness of the methods. Tables 7.15 and 7.16 show a summary of the results obtained in the five runs with the ILS algorithm and the hybrid algorithm, respectively, with some statistics. The detailed results for all the different runs are available in appendix, Table D.7. Tables 6.18 and 6.19 contain, besides the instance name, the value of the best solution obtained by the proposed method (Best value), the minimum percentage of gap (min), the average percentage of gap ($average$) and the maximum percentage of gap (max) obtained in the five runs, the range between the maximum and the minimum percentage of gap ($range = max - min$) and the average computational time, in seconds, also considering the five distinct runs ($\overline{t_s}$).

Table 7.15: Summary of the final results obtained with the ILS algorithm for the RFTSP.

Instance	Best value	min	$average$	max	$range$	$\overline{t_s}$
Instance set 1						
a_2	1662.03	0.43%	1.62%	2.63%	2.20%	33
a_3	1644.12	9.78%	10.11%	10.36%	0.58%	29
gr_1	1892.55	2.73%	3.12%	3.87%	1.14%	435
gr_2	1344.53	0.18%	0.61%	1.22%	1.05%	440
gr_3	1338.52	0.85%	1.92%	4.12%	3.27%	459
pr_1	158433.00	0.04%	0.79%	1.62%	1.58%	1439
pr_2	176788.00	0.40%	2.19%	4.06%	3.66%	1423
pr_3	131433.00	1.54%	3.45%	4.94%	3.40%	1630
Instance set 2						

Continues on the next page

Table 7.15

Instance	Best value	min	$average$	max	$range$	$\overline{t_s}$
pr136_1	92581	0.31%	2.01%	3.46%	3.15%	4
pr144_1	58125	8.16%	8.21%	8.28%	0.12%	4
rat195_1	1544	-0.26%	-0.04%	0.26%	0.52%	12
d198_1	11632	0.22%	0.77%	1.46%	1.24%	12
kroA200_2	31824	-1.42%	-0.10%	0.80%	2.22%	12
kroB200_2	31425	2.41%	2.84%	3.35%	0.93%	12
gr202_4	32142	3.01%	4.28%	4.93%	1.92%	11
pr226_1	63178	-3.09%	-2.92%	-2.85%	0.24%	19
pr226_3	92715	3.38%	3.42%	3.44%	0.06%	10
gil262_2	3014	1.65%	2.36%	3.44%	1.79%	27
gil262_4	6918	3.32%	4.02%	4.58%	1.27%	20
pr264_2	30670	-8.79%	-5.16%	-3.06%	5.73%	26
pr264_3	55914	-0.41%	0.01%	0.36%	0.77%	17
pr264_4	45342	-0.86%	-0.64%	-0.46%	0.40%	23

Instance set 3

ftv170_2	1727	7.13%	13.61%	21.34%	14.21%	7
ftv170_4	2650	5.08%	8.20%	10.94%	5.87%	8
rbg323_1	1644	-1.44%	-0.48%	0.54%	1.98%	46
rbg323_2	763	-8.73%	-7.68%	-5.86%	2.87%	50
rbg323_3	2804	-2.87%	-1.61%	-0.59%	2.29%	29
rbg323_4	1571	-0.06%	0.78%	1.34%	1.40%	41
rbg358_1	1814	-8.11%	-7.39%	-6.79%	1.32%	68
rbg358_2	764	-15.86%	-11.06%	-5.62%	10.24%	68
rbg403_1	1682	-14.45%	-13.58%	-12.82%	1.63%	99
rbg403_2	782	-24.74%	-23.60%	-21.17%	3.56%	99
rbg403_3	3373	-6.80%	-6.02%	-4.53%	2.27%	56
rbg403_4	1933	-8.52%	-6.94%	-4.59%	3.93%	77
rbg443_1	2222	-11.51%	-9.74%	-8.48%	3.03%	103
rbg443_2	1262	-9.34%	-7.46%	-5.60%	3.74%	120
rbg443_4	2437	-0.49%	0.28%	0.86%	1.35%	84

Instance set 4

Continues on the next page

Table 7.15

Instance	Best value	min	$average$	max	$range$	$\overline{t_s}$
AsimSingh150_1	1296	0.86%	1.04%	1.32%	0.47%	6
AsimSingh150_4	973	2.31%	2.78%	3.36%	1.05%	5
AsimSingh300_1	2798	2.42%	2.61%	2.75%	0.33%	40
AsimSingh300_3	4236	2.20%	2.33%	2.41%	0.22%	24
AsimSingh300_4	2882	1.30%	1.88%	2.11%	0.81%	31

Table 7.16: Summary of the final results obtained with the hybrid algorithm for the RFTSP.

Instance	Best value	min	$average$	max	$range$	$\overline{t_s}$
Instance set 1						
a_2	1662.48	0.45%	1.00%	1.53%	1.07%	32
a_3	1518.09	1.36%	1.48%	1.57%	0.20%	29
gr_1	1842.25	0.00%	0.56%	1.23%	1.23%	455
gr_2	1342.17	0.00%	0.47%	1.13%	1.13%	456
gr_3	1327.26	0.00%	1.02%	1.76%	1.76%	506
pr_1	158362.92	0.00%	0.28%	0.54%	0.54%	1406
pr_2	176086.63	0.00%	2.25%	4.42%	4.42%	1368
pr_3	129436.70	0.00%	1.82%	4.00%	4.00%	1521
Instance set 2						
pr136_1	93478	1.28%	1.51%	2.16%	0.88%	4
pr144_1	53739	0.00%	0.16%	0.61%	0.61%	4
rat195_1	1522	-1.68%	-0.37%	0.97%	2.65%	12
d198_1	11640	0.28%	0.56%	0.94%	0.65%	13
kroA200_2	32640	1.11%	1.68%	2.22%	1.11%	12
kroB200_2	32031	4.39%	5.03%	6.01%	1.62%	12
gr202_4	31303	0.32%	0.35%	0.49%	0.17%	11
pr226_1	63178	-3.09%	-2.95%	-2.78%	0.31%	18
pr226_3	89524	-0.18%	0.34%	1.10%	1.28%	19
gil262_2	3053	2.97%	3.87%	4.42%	1.45%	26
gil262_4	6921	3.36%	3.98%	4.72%	1.36%	24

Continues on the next page

Table 7.16

Instance	Best value	min	$average$	max	$range$	$\overline{t_s}$
pr264_2	30614	-8.96%	-8.16%	-7.49%	1.47%	24
pr264_3	55493	-1.16%	-1.10%	-1.05%	0.11%	23138
pr264_4	44758	-2.14%	-2.04%	-1.99%	0.15%	69
Instance set 3						
ftv170_2	1715	6.39%	12.08%	18.24%	11.85%	7
ftv170_4	2726	8.09%	10.44%	12.53%	4.44%	8
rbg323_1	1654	-0.84%	-0.32%	0.30%	1.14%	46
rbg323_2	778	-6.94%	-5.93%	-5.26%	1.67%	49
rbg323_3	2726	-5.58%	-5.15%	-4.81%	0.76%	37
rbg323_4	1547	-1.59%	-0.45%	0.70%	2.29%	40
rbg358_1	1819	-7.85%	-7.28%	-6.53%	1.32%	68
rbg358_2	809	-10.90%	-8.52%	-6.72%	4.19%	68
rbg403_1	1682	-14.45%	-14.05%	-13.73%	0.71%	98
rbg403_2	778	-25.12%	-23.46%	-20.98%	4.14%	99
rbg403_3	3214	-11.19%	-9.96%	-9.17%	2.02%	107
rbg403_4	1908	-9.70%	-8.99%	-8.38%	1.33%	84
rbg443_1	2243	-10.67%	-9.85%	-9.12%	1.55%	110
rbg443_2	1260	-9.48%	-9.11%	-8.62%	0.86%	124
rbg443_4	2379	-2.86%	-2.43%	-1.55%	1.31%	92
Instance set 4						
AsimSingh150_1	1292	0.54%	0.68%	0.86%	0.31%	6
AsimSingh150_4	965	1.47%	1.75%	2.00%	0.53%	5
AsimSingh300_1	2776	1.61%	1.94%	2.27%	0.66%	52
AsimSingh300_3	4157	0.29%	0.54%	0.65%	0.36%	110
AsimSingh300_4	2883	1.34%	1.37%	1.51%	0.18%	34

Considering Table 7.15, the average gap obtained was of -0.46% and the average computational time was of 170 seconds. Focus now on the instance set 1, the average gap obtained was of 2.98% and the average computational time was of 736 seconds, which is higher than the average computational time obtained when we consider all sets of instances. Nonetheless, the instance set 1 contains the highest dimensioned test instances, which are instances pr with 1002 nodes, thus, 1497

seconds seems to be a reasonable computational time to obtain feasible solutions for these instances. The ILS algorithm is a reasonably robust method for this instance set as the average range obtained was of 2.11%. When we compare the ILS algorithm with the B&C algorithm, we conclude that the ILS algorithm was unable to obtain better solutions than the B&C algorithm in instances a_2 and a_3. Nevertheless, the ILS algorithms provided solutions for instances a in an average of 31 seconds while the B&C algorithm took 10800 seconds. Additionally, with the ILS algorithm we could obtain feasible solutions for the instances gr and pr, which we could not address with the B&C algorithm.

Regarding the instance set 2, the ILS algorithm provided an average gap of 1.36% while the average computational time was of 15 seconds. By observing the column min we verify that there are several instances with a negative percentage of gap, which implies that the ILS algorithm was able to obtain solutions with a lower cost than the solutions obtained by the B&C algorithm. More precisely, the ILS algorithm was able to improve the best bound obtained by the B&C algorithm in six instances, namely instances $rat195\_1$, $kroA200\_2$, $pr226\_1$, $pr264\_2$, $pr264\_3$ and $pr264\_4$. Regarding the robustness of the method for this instance set, the average range obtained was of 1.45% and the maximum range obtained was of 5.73%, which seems to be an outlier.

The average gap obtained considering the instance set 3 was of -4.85% and the average computational time was of 64 seconds. Since the average gap obtained is negative, the ILS algorithm was able to improve, on average, the upper bounds provided by the B&C algorithm. More precisely, with the ILS algorithm we were able to improve the best upper bound obtained by the B&C algorithm in 13 instances, which correspond to all the instances $rbg323$, $rbg358$, $rbg403$ and $rbg443$ which have an unknown optimal value. These results can be explained by the cost matrix of these instances having arcs with zero cost, which makes the B&C algorithm less effective. Additionally, this was the instance set where the ILS algorithm provided the best results in terms of average gap obtained. The ILS algorithm was less robust in this instance set. In fact, the average range was of 5.98% and the maximum range was of 14.21%. Nonetheless, considering the maximum value of gap obtained in the five runs we were still able to improve the solution provided by the B&C algorithm in 10 instances.

Finally, considering the instance set 4, the average gap obtained was of 2.13% and the average computational time was of 21 seconds. By observing the column min, we verify that all the gap values are positive thus we could not improve the best solutions obtained by the B&C algorithm in any instance from the instance set 4. The average range obtained was of 0.58% and the maximum range was of 1.05%, which makes this the instance set for which the ILS algorithm was the most robust.

Consider now Table 7.16, which contains the results obtained with the hybrid algorithm. The average gap obtained was of -1.55% whereas the average computational time was of 724 seconds. The former is significantly lower than the average gap obtained with the ILS algorithm while the latter is significantly higher than the average computational time obtained with the ILS algorithm. If we observe the column $\overline{t_s}$ we verify that the computational time of instance $pr264\_3$ was abnormally high. More precisely, the average computational time to obtain a feasible solution for instance $pr264\_3$ was of 23138 seconds. These results are a consequence of the drawback of the hybrid algorithm, which is the fact that the efficiency of the algorithm is highly dependent on the difficulty of the core problem. When we remove instance $pr264\_3$ the average gap obtained was of -1.56% and the average computational time of 177 seconds, which is similar to the average computational time obtained with the ILS algorithm.

The hybrid algorithm provided an average gap of 1.11% and an average computational time of 722 seconds for the instance set 1. By comparing these to the ones obtained with the B&C algorithm, we verify that the B&C algorithm obtained a better quality solution than the hybrid algorithm for instances a. Nonetheless, the B&C algorithm took 10800 seconds to obtain those solutions while the hybrid algorithm took 31 seconds. The average range obtained was of 1.79% while the maximum range obtained was of 4.42%. By comparing the results obtained with both heuristic algorithms for the instance set 1, we verify that the hybrid algorithm provided a lower average gap and a lower computational time. In fact, the hybrid algorithm obtained the solution with the lowest cost for every instance with unknown optimal value from the instance set 1, except for instance a_2.

The average gap and the average computational time obtained with the hybrid algorithm for the instance set 2 was of 0.20% and of 1670 seconds. Recall that this was the instance set which had an instance with an abnormally high computational time; if we remove that instance we obtain an average gap of -0.18% and an average computational time of 19 seconds. By observing the column min we verify that we were able to obtain a better solution than the one provided by the B&C algorithm in six instances, namely instances $rat195\_1$, $pr226\_1$, $pr226\_3$, $pr264\_2$, $pr264\_3$ and $pr264\_4$. Note that some of these instances are not the same ones which we could improve with the ILS algorithm. The average range obtained was of 0.99% while the maximum range obtained was of 2.65%. By comparing the hybrid algorithm to the ILS algorithm we verify that the hybrid algorithm provided the solution with the lowest cost in seven instances, namely instances $pr144\_1$, $rat195\_1$, $gr202\_4$, $pr226\_3$, $pr264\_2$, $pr264\_3$ and $pr264\_4$, which corresponds to half of the instances from the instance set 2 which have an unknown optimal value.

Consider now the instance set 3, the average gap obtained was of -5.53% and the average computational time was of 69 seconds. We were able to obtain a solution with a lower cost than the solution obtained by the B&C algorithm in 13 instances. More precisely, the hybrid algorithm was able to provide a lower cost solution than the B&C algorithm in every instance from the instance set 3 with an unknown optimal value, except for instances $ftv170$. The variability of the solutions provided by the hybrid algorithm for this instance set was high, in fact, the average range obtained was of 2.64% and the maximum range obtained was of 11.35%. Similarly to what happened with the ILS algorithm, this is the set of instances with the biggest average range. When we compare the lowest cost solution found by the proposed heuristic algorithms for this set of instances we verify that the hybrid algorithm provided the best solution in eight instances, which are instances $ftv170\_2$, $rbg323\_3$, $rbg323\_4$, $rbg403\_2$, $rbg403\_3$, $rbg403\_4$, $rbg443\_2$ and $rbg443\_4$.

Finally, the hybrid algorithm provided an average gap of 1.26% and an average computational time of 41 seconds when addressing the instance set 4. By observing the column min we conclude that the hybrid algorithm could not improve the solutions obtained with the B&C algorithm. The average range obtained was of 0.41% and the maximum range obtained was of 0.66%, making this the instance set where the hybrid algorithm provided the most robust results. The average gap obtained with the ILS algorithm for this instance set was significantly higher. More precisely, the ILS algorithm provided an average gap of 2.13%. When we analyze the best solution obtained for each instance with the ILS algorithm and with the hybrid algorithm we verify that the hybrid algorithm provided the lowest cost solution in every instance from the instance set 4 with unknown optimal value, except for instance $AsimSingh300\_4$.

Summary

We adapted the ILS algorithm and the hybrid algorithm developed for the FTSP to the RFTSP, since these were the heuristic algorithms that provided the best quality solutions for the FTSP. These methods were applied to the instances that the exact methods could not solve within the time limit.

There were 42 instances that the exact method could not solve to optimality within the time limit. To further support the claim that the RFTSP is more challenging than the FTSP, recall that the number of instances with unknown optimal value in the FTSP was 24. From the 42 instances with unknown optimal value, the hybrid algorithm provided the lowest cost solution in 26 instances whereas the ILS algorithm obtained the best solution in 16 instances. Similarly to what we concluded when we addressed the FTSP, the hybrid algorithm provides better quality solutions than the ILS algorithm and the ILS algorithm is more efficient than the hybrid algorithm. The hybrid algorithm is more robust than the ILS algorithm, on average, since the average range obtained with the hybrid algorithm was of 1.46% while the average range obtained with the ILS algorithm was of 2.61%. Therefore, when addressing the RFTSP, it is preferable to use the hybrid algorithm when the first concern is to obtain the best solutions possible in a reasonable amount time, however, if the objective is to achieve good quality solutions quickly then the ILS algorithm is the most suited method.

When we compare the solutions obtained with the heuristic methods to the ones provided by the B&C algorithm, we verify that the B&C algorithm was able to obtain the solution with the lowest cost in 26 of the 42 instances with unknown optimal value. However, the B&C algorithm provided these solutions in 10800 seconds. Even though we concluded in Section 7.4.2 that the performance of the B&C algorithm is worse for the RFTSP than for the FTSP, specially for asymmetric instances, the proportion of instances in which the B&C algorithm provides the lowest cost solution is similar in the FTSP and in the RFTSP. In fact, the proportion of instances in the FTSP where the B&C algorithm provided the lowest cost solution was of 58% while in the RFTSP is of 62%. Table 7.17 shows a summary of the best known upper bounds for the RFTSP for the instances with unknown optimal value, which were obtained by applying the methods proposed during this dissertation.

Table 7.17: Current best known upper bounds for the RFTSP.

Instance	Best known upper bounds
Instance set 1	
a_2	1654.95*
a_3	1497.69*
gr_1	1842.25
gr_2	1342.17
gr_3	1327.26
pr_1	158362.92
pr_2	176086.63
pr_3	129436.70
Instance set 2	
pr136_1	92294*
pr144_1	53739*
rat195_1	1522
d198_1	11607*
kroA200_2	31824
kroB200_2	30684*

Continues on the next page

Table 7.17

Instance	Best known upper bounds
gr202_4	31203*
pr226_1	63178
pr226_3	89524
gil262_2	2965*
gil262_4	6696*
pr264_2	30614
pr264_3	55493
pr264_4	44758
Instance set 3	
ftv170_2	1612*
ftv170_4	2522*
rbg323_1	1644
rbg323_2	763
rbg323_3	2726
rbg323_4	1547
rbg358_1	1814
rbg358_2	764
rbg403_1	1682
rbg403_2	782
rbg403_3	3214
rbg403_4	1908
rbg443_1	2222
rbg443_2	1260
rbg443_4	2379
Instance set 4	
AsimSingh150_1	1285*
AsimSingh150_4	951*
AsimSingh300_1	2732*
AsimSingh300_3	4145*
AsimSingh300_4	2845*

*Obtained with B&C algorithm.

Chapter 8

Conclusion

We start by summarizing and presenting the main conclusions of the work developed within the scope of this dissertation in Section 8.1 and, in Section 8.2, we present some ideas for future work.

8.1 Main conclusions

This dissertation focused on the family traveling salesman problem, a recent problem which may be seen as a generalization of well-known problems from the literature, such as the traveling salesman problem. Given a depot and a partition of the set of cities into families, the FTSP consists in establishing the minimum cost elementary circuit that starts and ends at the depot and visits a given number of cities in each family.

The literature regarding the FTSP is limited, in fact, there are only two articles that address it, namely by Morán-Mirabal et al. (2014) and Bernardino and Paias (2018a), and the latter is a result from the work developed during this dissertation. Morán-Mirabal et al. (2014) proposed the FTSP and focus their study on heuristic methods. Additionally, they proposed an ILP model for the FTSP which consists in a TSP model with the additional constraints that ensure the family visits. Therefore, we decided to carry out a comprehensive study concerning the development of formulations for the FTSP.

We developed seven different formulations for the FTSP, three of which are adaptations of known formulations for the TSP, namely the SCF model, the NCF model and the CC model. The other four formulations, which are the FCF model, the NCF$^+$ model, the RV model and the RFV model, take into account the specificities of the FTSP, particularly, the fact that, usually, we are not required to visit every node from each family and that the total number of visits may be disaggregated per family. The formulations may also be classified as compact formulations and non-compact formulations. The compact formulations, namely the SCF model, the FCF model, the NCF model and the NCF$^+$ model, use flow variables to ensure that the solution obtained does not contain subtours while the non-compact formulations, which are the CC model, the RV model and the RFV model, use cut inequalities. We established a theoretical comparison between the proposed formulations and concluded that, except for the NCF model and the CC model, the compact formulations are not comparable to the non-compact formulations. Nevertheless, the non-compact formulations provide, in general, higher LP relaxation values. In fact, the RFV model was the one that provided the lowest average gap amongst the proposed formulations. After the preliminary computational tests we concluded that the best models in practice were the CC model and the RFV model. Since the CC model and the RFV model are not comparable we decided to combine them which originated the CC+RFV model in which one set of exponential many constraints is added as subtour elimination constraints and the other is added as valid inequalities to increase the LP relaxation value. With the CC+RFV model we were able to further improve the LP relaxation values obtained with the RFV model.

As the CC, RFV and CC+RFV models have sets of constraints that are in exponential number, we have to resort to a branch-and-cut algorithm to solve them. Therefore, we developed a B&C algorithm for these models. More precisely, we developed separation algorithms for the CC and the RFV inequalities. With more computational testing we concluded that the exact separation algorithm for the RFV inequalities is very time consuming, which led us to create two distinct separation algorithms for the CC+RFV model which separate the RFV inequalities in a heuristic manner but ensure the separation of all CC inequalities, which were the y-separation algorithm and the 1-separation algorithm. In practice we verified that, when addressing the CC+RFV model, it was preferable to use the y-separation and the 1-separation instead of the exact separation algorithms of both the CC and the RFV inequalities. Additionally, we developed a very simple heuristic procedure to obtain feasible solutions and, consequently, upper bounds during the B&C algorithm.

Since the number of existing benchmark FTSP instances was small, we developed an instance generator. This generator has as input a cost matrix and creates several FTSP instances with that cost matrix and randomly generated families and number of visits per family. We used cost matrices from symmetric and asymmetric TSP instances as well as from asymmetric UTPP instances to create instances for the FTSP. We generated a total of 164 FTSP instances. Each cost matrix originated four different FTSP instances, which were designed to have different characteristics, for example instances of type low were designed to have a small number of visits per family while instances of type $high$ were designed to have a big number of visits per family.

We used the y-separation and the 1-separation to obtain the optimal value of the benchmark instances and the generated instances. Before solving the instances to optimality, we conducted a comparison in terms of LP relaxation of the referred separation algorithms in order to determine which algorithm was the best both in terms of the quality of the LP relaxation value and of computational time. As these separation algorithms separate the RFV inequalities in a heuristic manner we cannot establish any theoretical comparison between them. Nevertheless, from the computational results, the y-separation solved the LP relaxation faster while the 1-separation provided higher LP relaxation values, since the latter separates more RFV inequalities on average. Moreover, the y-separation was the most efficient method, in terms of computational time, to obtain the optimal values of the test instances. For all the reasons stated previously, we advise the usage of the CC+RFV model with the y-separation to obtain the optimal values of FTSP instances. From the benchmark instances, we were able to obtain the optimal value of all instances with a maximum of 127 nodes and of one instance with 280 nodes. The optimal value of the benchmark instances $bier$ and a_3 was unknown. Regarding the generated instances, we were able to obtain the optimal value of 148 instances within the time limit. The majority of the instances that remain with an unknown optimal value has a symmetric cost matrix.

For the instances which have an unknown optimal value we developed three heuristic algorithms: a genetic algorithm, an iterated local search algorithm and a hybrid algorithm. The computational tests showed that the ILS algorithm and the hybrid algorithm provided lower cost solutions for the FTSP than the genetic algorithm. Since we could not establish any relationship of dominance between the ILS algorithm and the hybrid algorithm we carried out a computational experiment with both heuristic methods. The solutions obtained with the proposed heuristic methods were compared to the best upper bounds provided by Morán-Mirabal et al. (2014), in the case of the benchmark instances, and to the best upper bound obtained by the B&C algorithm after 10800 seconds, in the case of the instances generated. By comparing the results obtained with the ILS algorithm to the ones obtained with the hybrid algorithm we verify that, in general, the ILS algorithm is faster while the hybrid algorithm provides better quality solutions. By using both the ILS algorithm and the hybrid algorithm we were able to improve the best upper bounds provided by Morán-Mirabal et al. (2014) for all the benchmark instances with unknown optimal value. Regarding the generated instances, the upper bounds obtained during the B&C algorithm were better, on average, than the ones obtained with the heuristic methods. Nevertheless, the upper bounds obtained with the B&C algorithm were obtained after 10800 seconds of computational time. We advise the usage of the ILS algorithm as this is the heuristic method proposed that reaches a better compromise between the quality of the solutions obtained and the computational time, since the genetic algorithm is faster but provides worse quality solutions and the hybrid algorithm may be significantly slower but, in general, provides better quality solutions.

We also created a variant of the FTSP, which we called the restricted family traveling salesman problem. This variant is obtained by adding the additional condition to the FTSP that nodes from the same family must be visited consecutively. To the best of our knowledge this variant has never been proposed in the literature. We created two distinct exact methods for the RFTSP, one is a straightforward adaptation of the CC+RFV model with the y-separation and the other is the inter- and intrafamily formulation, which is a formulation that considers the interfamily subproblem and the intrafamily subproblem. Preliminary computational tests showed that the inter- and intrafamily formulations provide better LP relaxation values than the corresponding adapted FTSP formulations, however, they cannot be used efficiently within a B&C framework due to the large number of variables and constraints associated with them. For this reason we used the adapted CC+RFV model with the y-separation to obtain the optimal values of the RFTSP. With the same time limit, 10800 seconds, the number of symmetric instances that we could solve is similar to the ones solved when addressing the FTSP while, considering the asymmetric instances, the number of instances solved decreased significantly. We also developed heuristic methods for the RFTSP, which are adaptations of the heuristic methods developed for the FTSP, namely the ILS algorithm and the hybrid algorithm. The conclusions that we can draw from the computational results obtained with the heuristic algorithms are similar to the case of the FTSP: the ILS algorithm is faster but the hybrid algorithm provides better quality solutions.

Summarizing, the main contributions of this dissertation are:

- an efficient B&C algorithm to solve the FTSP;
- heuristic methods that are able to provide good quality solutions for the FTSP in an efficient manner;
- the introduction of the RFTSP as well as some resolution methods; and
- a set of test instances and their optimal values or upper bounds, when the optimal value is unknown, which were made available for the scientific community in a website devoted to the FTSP.

8.2 Future work

There are essentially two main points that could be further explored in this dissertation, namely the polyhedral analysis of the FTSP and the study of the RFTSP, more precisely, the inter- and intrafamily formulations.

We started a polyhedral study for the FTSP and we were able to establish which is the dimension of the polytope associated with the FTSP. More formally, let P_{FTSP} be the polytope which corresponds to the convex hull of the integer points which are feasible solutions for the FTSP. Then:

$$dim(P_{FTSP}) = |N|^2 - 1 - L - \sum_{l \in \mathcal{U}} n_l \times (n_l - 1) - \sum_{l \in \mathcal{W}} (n_l - 1). \tag{8.1}$$

We were able to prove the dimension (8.1) by using the result of Proposition 1 and by presenting a sufficient number of affinely independent points. In Appendix F we present the reasoning and the affinely independent points which led us to the dimension (8.1).

The advantage of knowing the dimension of the polytope P_{FTSP} is that we can use it to verify if the CC and the RFV inequalities define facets. We did a brief study and concluded that, for a given family $l \in \mathcal{L} \backslash \mathcal{U}$, when $S \subset F_l$ and $|S \cap F_l| = n_l - v_l + 1$ the RFV inequality $x(S', S) \geq 1$ (4.35) is a facet of P_{FTSP}. Observe that establishing whether or not the CC and the RFV inequalities define facets would guide our research into different directions. In the former case, we would focus our study on finding inequalities that dominate the CC and the RFV inequalities while, in the latter case, we would investigate new sets of valid inequalities for the FTSP.

As mentioned in the introduction, the study of the RFTSP was not as exhaustive as the study of the FTSP, namely in what concerns the theoretical comparison of the adapted formulations to the inter- and intrafamily formulations. Moreover, we showed that the B&C algorithm is not the best approach for the inter- and intrafamily formulations. Nevertheless, these formulations seem promising since they provide higher LP relaxation values than the adapted formulations, thus, it would be interesting to develop a different solution approach. The idea that we intend to pursue in the future is to use a decomposition approach in which the interfamily subproblem is the master problem.

Bibliography

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. *The traveling salesman problem: a computational study*. Princeton University Press, 2006.

C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

M. Bellmore and G. L. Nemhauser. The traveling salesman problem: a survey. *Operations Research*, 16(3):538–558, 1968.

R. Bernardino and A. Paias. Solving the family traveling salesman problem. *European Journal of Operational Research*, 267(2):453–466, 2018a.

R. Bernardino and A. Paias. Metaheuristics based on decision hierarchies for the traveling purchaser problem. *International Transactions in Operational Research*, 25(4):1269–1295, 2018b.

F. F. Boctor, G. Laporte, and J. Renaud. Heuristics for the traveling purchaser problem. *Computers & Operations Research*, 30(4):491–504, 2003.

I. Boussaïd, J. Lepagnot, and P. Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013.

N. Christofides. *Graph theory: An algorithmic approach (Computer science and applied mathematics)*. Academic Press, Inc., 1975.

M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer Programming*. Springer, 2014.

G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.

M. Fischetti, J.-J. Salazar-González, and P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45(3):378–394, 1997.

B. Gavish and S. C. Graves. The travelling salesman problem and related problems. *Working paper*, 1978.

A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.

D. E. Goldberg and J. H. Holland. Genetic algorithms and machine learning. *Machine learning*, 3 (2):95–99, 1988.

B. Golden, Z. Naji-Azimi, S. Raghavan, M. Salari, and P. Toth. The generalized covering salesman problem. *INFORMS Journal on Computing*, 24(4):534–553, 2012.

M. Gondran and M. Minoux. *Graphs and algorithms*. Wiley, 1984.

L. Gouveia, J. Riera-Ledesma, and J.-J. Salazar-González. Reverse multistar inequalities and vehicle routing problems with a lower bound on the number of customers per route. *Networks*, 61 (4):309–321, 2013.

M. Grötschel and O. Holland. Solution of large-scale symmetric travelling salesman problems. *Mathematical Programming*, 51(1):141–202, 1991.

G. Gutin and A. P. Punnen. *The traveling salesman problem and its variations*. Springer Science & Business Media, 2006.

J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

IBM. IBM ILOG CPLEX Optimization Studio 12.6.1. `https://www.ibm.com/analytics/cplex-optimizer`, 2014.

D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. *Local search in combinatorial optimization*, 1:215–310, 1997.

E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys. *The traveling salesman problem: a guided tour of combinatorial optimization*. John Wiley & Sons, 1985.

H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

L. Morán-Mirabal, J. González-Velarde, and M. G. Resende. Randomized heuristics for the family traveling salesperson problem. *International Transactions in Operational Research*, 21(1):41–57, 2014.

G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.

M. Padberg and G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6(1):1–7, 1987.

M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

G. R. Raidl. Decomposition based hybrid metaheuristics. *European Journal of Operational Research*, 244(1):66–76, 2015.

G. R. Raidl and J. Puchinger. Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In *Hybrid metaheuristics*, pages 31–62. Springer, 2008.

G. Reinelt. Tsplib — a traveling salesman problem library. *ORSA journal on computing*, 3(4): 376–384, 1991.

A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

K. N. Singh and D. L. van Oudheusden. A branch and bound algorithm for the traveling purchaser problem. *European Journal of Operational Research*, 97(3):571–579, 1997.

S. Srivastava, S. Kumar, R. Garg, and P. Sen. Generalized traveling salesman problem through n sets of nodes. *CORS journal*, 7:97–101, 1969.

L. A. Wolsey. *Integer Programming*. John Wiley & Sons, 1998.

R. T. Wong. Integer programming formulations of the traveling salesman problem. In *Proceedings of the IEEE international conference of circuits and computers*, pages 149–152. IEEE Press Piscataway, NJ, 1980.

Appendix A

Instance Description

This chapter contains a complete description of the test instances. Tables A.1, A.2, A.3, A.4 show the description of the instance set 1, 2, 3 and 4, respectively, presented in Section 3.4. Each table has six columns with the name of the instance (Instance name), the number of nodes ($|N| + 1$), the number of families (L), the total number of visits (V) and the enumeration, per family, of both the number of family members (n_l) and the number of family visits (v_l).

The tables are sorted according to an ascending order of the number of nodes of the instance.

Table A.1: Complete description of the instance set 1.

Instance Name	$\|N\| + 1$	L	V	n_l	v_l
bruma14_3_1001_1001_2	14	3	6	[4, 5, 4]	[2, 2, 2]
bruma14_3_1001_1002_2			10		[4, 2, 4]
bruma14_3_1001_1003_2			4		[2, 1, 1]
bayg29_4_1001_1001_2	29	4	16	[7, 9, 6, 6]	[6, 4, 5, 1]
bayg29_4_1001_1002_2			17		[2, 9, 1, 5]
bayg29_4_1001_1003_2			18		[6, 6, 1, 5]
att48_5_1001_1001_2	48	5	34	[12, 9, 9, 7, 10]	[10, 4, 9, 7, 4]
att48_5_1001_1002_2			25		[8, 2, 9, 1, 5]
att48_5_1001_1003_2			15		[6, 1, 3, 3, 2]
bier127_10_1001_1001_2	127	10	62	[12, 12, 14, 8, 13, 16, 13, 8, 17, 13]	[10, 4, 13, 1, 12, 4, 6, 1, 5, 6]
bier127_10_1001_1002_2			85		[8, 2, 12, 7, 9, 9, 5, 5, 17, 11]
bier127_10_1001_1003_2			60		[6, 1, 13, 3, 3, 13, 13, 2, 2, 4]
a280_20_1001_1001_2	280	20	179	[15, 14, 16, 11, 19, 15, 18, 10, 17, 16, 16, 8, 7, 15, 24, 8, 11, 13, 15, 11]	[14, 10, 14, 4, 13, 9, 15, 4, 5, 14, 6, 7, 6, 8, 13, 7, 9, 13, 6, 2]
a280_20_1001_1002_2			156		[8, 2, 12, 9, 9, 5, 17, 6, 3, 9, 7, 2, 6, 11, 4, 6, 11, 7, 11, 11]
a280_20_1001_1003_2			141		[14, 14, 6, 1, 13, 3, 18, 3, 2, 4, 10, 4, 4, 8, 5, 4, 9, 4, 14, 1]
gr666_30_1001_1001_2	666	30	357	[27, 24, 24, 17, 29, 19, 20, 17, 27, 24, 26, 15, 15, 30, 40, 11, 19, 28, 27, 20 28, 22, 24, 14, 23, 15, 17, 18, 20, 25]	[14, 10, 15, 4, 13, 9, 15, 4, 22, 5, 14, 6, 15, 30, 24, 7, 2, 1, 19, 5, 6, 13, 18, 9, 21, 10, 15, 2, 10, 19]
gr666_30_1001_1002_2			328		[8, 2, 15, 9, 21, 17, 14, 3, 9, 7,

Continues on the next page

Table A.1

Instance Name	$\lvert N\rvert + 1$	L	V	n_l	v_l
					10, 6, 11, 4, 39, 11, 11, 26, 7, 8 1, 8, 14, 7, 19, 5, 6, 9, 9, 12]
gr666_30_1001_1003_2			328		[6, 17, 13, 2, 4, 12, 4, 5, 12, 14, 15, 9, 4, 14, 33, 10, 17, 27, 17, 8, 6, 8, 2, 5, 8, 9, 17, 15, 6, 9]
pr1002_40_1001_1001_2	1002	40	486	[22, 28, 27, 30, 32, 24, 21, 22, 29, 30, 27, 16, 20, 30, 38, 16, 21, 23, 27, 28 23, 25, 26, 26, 21, 24, 20, 30, 18, 25 25, 27, 27, 21, 26, 24, 28, 28, 25, 21]	[14, 10, 15, 4, 13, 9, 15, 4, 22, 25, 5, 14, 6, 30, 24, 14, 13, 7, 25, 22, 2, 1, 19, 5, 6, 13, 18, 9, 15, 2, 22, 10, 19, 11, 1, 8, 3, 8, 6, 17]
pr1002_40_1001_1002_2			538		[8, 2, 15, 25, 9, 21, 17, 14, 22, 22, 3, 9, 7, 10, 6, 11, 4, 22, 27, 7 11, 7, 8, 1, 8, 14, 19, 21, 6, 9 9, 12, 26, 8, 23, 21, 8, 28, 18, 20]
pr1002_40_1001_1003_2			463		[6, 17, 13, 19, 19, 18, 19, 2, 4, 26, 12, 4, 5, 12, 15, 9, 4, 14, 1, 15, 17, 17, 8, 6, 8, 2, 5, 8, 17, 15, 6, 9, 3, 20, 15, 5, 14, 26, 18, 10]

Table A.2: Complete description of the instance set 2.

Instance Name	$\lvert N\rvert + 1$	L	V	n_l	v_l
$pr136.sftsp$_21_1	136	21	82	[5, 3, 4, 8, 6, 5, 4, 4, 5, 8, 5, 3, 3, 8, 8, 7, 8, 12, 11, 11, 7]	[4, 1, 3, 3, 4, 2, 4, 1, 3, 2, 3, 2, 2, 2, 8, 2, 6, 10, 9, 8, 3]
$pr136.sftsp$_21_2			48		[3, 1, 2, 1, 1, 2, 2, 1, 3, 2, 3, 1, 2, 2, 2, 2, 6, 2, 8, 1, 1]
$pr136.sftsp$_21_3			114		[5, 2, 4, 4, 5, 5, 4, 3, 4, 7, 4, 3, 3, 3, 8, 5, 7, 11, 10, 11, 6]
$pr136.sftsp$_21_4			83		[3, 1, 4, 4, 1, 5, 2, 3, 3, 7, 3, 1, 3, 2, 2, 5, 6, 11, 10, 1, 6]
$gr137.sftsp$_21_1	137	21	87	[5, 3, 4, 8, 6, 5, 7, 4, 5, 8, 5, 3, 3, 5, 8, 7, 8, 12, 11, 11, 8]	[4, 1, 3, 3, 4, 2, 7, 1, 3, 2, 3, 2, 2, 2, 8, 2, 6, 10, 9, 8, 5]
$gr137.sftsp$_21_2			56		[3, 1, 2, 1, 1, 2, 6, 1, 3, 2, 3, 1, 2, 2, 2, 2, 6, 2, 8, 1, 5]
$gr137.sftsp$_21_3			117		[5, 2, 4, 4, 5, 5, 7, 3, 4, 7, 4, 3, 3, 3, 8, 5, 7, 11, 10, 11, 6]
$gr137.sftsp$_21_4			87		[3, 1, 4, 4, 1, 5, 6, 3, 3, 7, 3, 1, 3, 2, 2, 5, 6, 11, 10, 1, 6]
$pr144.sftsp$_22_1	144	22	90	[5, 3, 4, 8, 6, 5, 7, 4, 5, 8, 5, 3, 3, 5, 8, 7, 8, 5, 9, 6,	[5, 2, 3, 4, 2, 5, 7, 2, 5, 1, 1, 2, 2, 4, 4, 6, 2, 4, 1, 3,

Continues on the next page

Table A.2

Instance Name	$\|N\|+1$	L	V	n_l	v_l
				5, 24]	5, 20]
$pr144.sftsp$_22_2			53		[5, 1, 1, 4, 2, 1, 1, 2, 3, 1, 1, 2, 2, 2, 2, 2, 1, 1, 1, 1, 2, 15]
$pr144.sftsp$_22_3			122		[5, 3, 4, 6, 3, 5, 7, 3, 5, 7, 2, 3, 3, 5, 6, 7, 3, 5, 7, 4, 5, 24]
$pr144.sftsp$_22_4			81		[5, 3, 4, 4, 3, 1, 7, 2, 5, 1, 1, 3, 2, 2, 6, 2, 3, 5, 1, 4, 2, 15]
$kroA150.sftsp$_16_1	150	16	78	[8, 9, 4, 5, 10, 5, 6, 6, 3, 4, 12, 16, 10, 8, 8, 35]	[8, 6, 2, 4, 5, 4, 1, 5, 3, 4, 8, 4, 7, 2, 2, 13]
$kroA150.sftsp$_16_2			48		[2, 5, 2, 4, 5, 2, 1, 4, 1, 1, 3, 4, 6, 1, 1, 6]
$kroA150.sftsp$_16_3			117		[8, 7, 3, 5, 6, 5, 6, 6, 3, 4, 12, 6, 10, 8, 4, 24]
$kroA150.sftsp$_16_4			58		[2, 5, 2, 5, 5, 2, 1, 4, 3, 4, 3, 4, 10, 1, 1, 6]
$kroB150.sftsp$_16_1	150	16	78	[8, 9, 4, 5, 10, 5, 6, 6, 3, 4, 12, 16, 10, 8, 8, 35]	[8, 6, 2, 4, 5, 4, 1, 5, 3, 4, 8, 4, 7, 2, 2, 13]
$kroB150.sftsp$_16_2			48		[2, 5, 2, 4, 5, 2, 1, 4, 1, 1, 3, 4, 6, 1, 1, 6]
$kroB150.sftsp$_16_3			117		[8, 7, 3, 5, 6, 5, 6, 6, 3, 4, 12, 6, 10, 8, 4, 24]
$kroB150.sftsp$_16_4			58		[2, 5, 2, 5, 5, 2, 1, 4, 3, 4, 3, 4, 10, 1, 1, 6]
$pr152.sftsp$_23_1	152	23	77	[5, 3, 4, 8, 6, 5, 7, 4, 5, 8, 5, 3, 3, 5, 10, 10, 4, 5, 11, 7, 8, 3, 22]	[3, 3, 4, 8, 6, 2, 3, 2, 3, 2, 4, 2, 2, 5, 4, 4, 1, 1, 5, 2, 4, 2, 5]
$pr152.sftsp$_23_2			47		[1, 3, 2, 1, 3, 2, 3, 1, 2, 2, 2, 2, 2, 2, 3, 1, 1, 1, 2, 1, 4, 1, 5]
$pr152.sftsp$_23_3			116		[4, 3, 4, 8, 6, 4, 4, 3, 5, 5, 5, 3, 3, 5, 5, 5, 4, 4, 7, 6, 5, 3, 15]
$pr152.sftsp$_23_4			86		[4, 3, 4, 1, 6, 2, 3, 3, 2, 2, 5, 2, 3, 5, 3, 5, 1, 1, 2, 6, 5, 3, 15]
$u159.sftsp$_17_1	159	17	100	[8, 9, 4, 5, 10, 5, 6, 6, 5, 12, 12, 10, 8, 7, 5, 15, 31]	[2, 3, 3, 5, 9, 5, 5, 3, 2, 8, 12, 7, 2, 5, 3, 11, 15]
$u159.sftsp$_17_2			64		[2, 2, 2, 4, 4, 4, 1, 3, 1, 4, 11, 7, 1, 3, 1, 2, 12]
$u159.sftsp$_17_3			139		[8, 6, 4, 5, 10, 5, 6, 5, 5, 10, 12, 9, 7, 6, 4, 13, 24]
$u159.sftsp$_17_4			98		[2, 2, 2, 4, 10, 5, 1, 3, 5, 4, 11, 7, 1, 3, 1, 13, 24]
$rat195.sftsp$_29_1	195	29	94	[5, 3, 4, 6, 4, 9, 7, 4, 5, 8, 5, 3, 5, 7, 8, 3, 4, 4, 11, 9, 9, 7, 10, 8, 6, 4, 14, 9, 13]	[3, 2, 1, 2, 1, 8, 2, 4, 4, 2, 4, 1, 4, 2, 4, 2, 1, 1, 4, 7, 8, 1, 6, 5, 1, 4, 6, 2, 2]

Continues on the next page

Table A.2

Instance Name	$\lvert N\rvert + 1$	L	V	n_l	v_l
$rat195.sftsp$_29_2			62		[3, 2, 1, 1, 1, 2, 1, 3, 4, 1, 2, 1, 1, 1, 1, 2, 1, 1, 2, 3, 7, 1, 6, 2, 1, 2, 5, 2, 2]
$rat195.sftsp$_29_3			145		[4, 3, 3, 3, 3, 9, 3, 4, 5, 3, 5, 3, 5, 6, 5, 3, 4, 3, 8, 8, 9, 2, 10, 6, 3, 4, 13, 3, 5]
$rat195.sftsp$_29_4			97		[4, 2, 3, 1, 1, 2, 1, 4, 5, 1, 2, 1, 5, 1, 1, 2, 4, 1, 8, 3, 7, 1, 10, 2, 3, 2, 13, 2, 5]
$d198.sftsp$_29_1	198	29	94	[5, 3, 8, 8, 6, 5, 7, 4, 5, 8, 5, 3, 5, 7, 8, 3, 4, 4, 11, 9, 9, 7, 10, 5, 6, 4, 14, 3, 21]	[3, 2, 1, 2, 5, 2, 2, 4, 4, 2, 4, 1, 4, 2, 4, 2, 1, 1, 4, 7, 8, 1, 6, 3, 1, 4, 6, 2, 6]
$d198.sftsp$_29_2			67		[3, 2, 1, 1, 5, 2, 1, 3, 4, 1, 2, 1, 1, 1, 1, 2, 1, 1, 2, 3, 7, 1, 6, 1, 1, 2, 5, 2, 4]
$d198.sftsp$_29_3			160		[4, 3, 8, 5, 6, 5, 3, 4, 5, 3, 5, 3, 5, 6, 5, 3, 4, 3, 8, 8, 9, 2, 10, 5, 3, 4, 13, 3, 15]
$d198.sftsp$_29_4			115		[4, 2, 8, 1, 5, 2, 1, 4, 5, 1, 2, 1, 5, 1, 1, 2, 4, 1, 8, 3, 7, 1, 10, 1, 3, 2, 13, 2, 15]
$kroA200.sftsp$_29_1	200	29	102	[5, 3, 8, 6, 4, 9, 7, 4, 5, 8, 5, 3, 5, 7, 10, 3, 4, 4, 11, 9, 9, 7, 10, 5, 6, 4, 14, 3, 21]	[3, 2, 1, 2, 1, 8, 2, 4, 4, 2, 4, 1, 4, 2, 10, 2, 1, 1, 4, 7, 8, 1, 6, 3, 1, 4, 6, 2, 6]
$kroA200.sftsp$_29_2			69		[3, 2, 1, 1, 1, 2, 1, 3, 4, 1, 2, 1, 1, 1, 7, 2, 1, 1, 2, 3, 7, 1, 6, 1, 1, 2, 5, 2, 4]
$kroA200.sftsp$_29_3			159		[4, 3, 8, 3, 3, 9, 3, 4, 5, 3, 5, 3, 5, 6, 10, 3, 3, 4, 8, 9, 9, 3, 7, 5, 6, 4, 8, 3, 13]
$kroA200.sftsp$_29_4			96		[3, 3, 1, 3, 1, 2, 1, 3, 5, 3, 2, 1, 1, 6, 7, 2, 1, 4, 2, 9, 7, 1, 6, 5, 1, 4, 5, 3, 4]
$kroB200.sftsp$_29_1	200	29	102	[5, 3, 8, 6, 4, 9, 7, 4, 5, 8, 5, 3, 5, 7, 10, 3, 4, 4, 11, 9, 9, 7, 10, 5, 6, 4, 14, 3, 21]	[3, 2, 1, 2, 1, 8, 2, 4, 4, 2, 4, 1, 4, 2, 10, 2, 1, 1, 4, 7, 8, 1, 6, 3, 1, 4, 6, 2, 6]
$kroB200.sftsp$_29_2			69		[3, 2, 1, 1, 1, 2, 1, 3, 4, 1, 2, 1, 1, 1, 7, 2, 1, 1, 2, 3, 7, 1, 6, 1, 1, 2, 5, 2, 4]
$kroB200.sftsp$_29_3			159		[4, 3, 8, 3, 3, 9, 3, 4, 5, 3, 5, 3, 5, 6, 10, 3, 3, 4, 8, 9, 9, 3, 7, 5, 6, 4, 8, 3, 13]
$kroB200.sftsp$_29_4			96		[3, 3, 1, 3, 1, 2, 1, 3, 5, 3, 2, 1, 1, 6, 7, 2, 1, 4, 2, 9, 7, 1, 6, 5, 1, 4, 5, 3, 4]
$gr202.sftsp$_30_1	202	30	128	[5, 3, 4, 6, 4, 9, 7, 4, 5, 8, 5, 3, 5, 7, 8, 3, 4, 4, 9, 9, 9, 7, 10, 8, 6, 4, 4, 10, 15, 16]	[5, 3, 2, 5, 2, 2, 2, 2, 4, 5, 1, 3, 5, 4, 3, 1, 1, 4, 7, 8, 6, 4, 3, 5, 2, 4, 2, 4, 15, 14]
$gr202.sftsp$_30_2			74		[2, 2, 1, 2, 1, 1, 2, 1, 2, 1,

Continues on the next page

Table A.2

Instance Name	$\lvert N\rvert + 1$	L	V	n_l	v_l
					1, 2, 2, 2, 3, 1, 1, 1, 4, 5, 6, 3, 2, 4, 1, 2, 2, 1, 9, 7]
$gr202.sftsp$_30_3			172		[5, 3, 3, 6, 4, 8, 4, 3, 5, 6, 2, 3, 5, 7, 4, 3, 4, 4, 9, 9, 8, 6, 5, 7, 4, 4, 4, 7, 15, 15]
$gr202.sftsp$_30_4			118		[2, 3, 1, 6, 1, 1, 2, 1, 5, 6, 1, 2, 2, 7, 3, 1, 1, 4, 4, 9, 6, 3, 2, 7, 1, 4, 2, 7, 9, 15]
$pr226.sftsp$_28_1	226	28	121	[7, 9, 6, 10, 4, 9, 10, 4, 3, 4, 4, 10, 10, 8, 8, 7, 8, 5, 11, 11, 3, 7, 6, 5, 12, 16, 16, 12]	[7, 7, 2, 3, 2, 5, 2, 4, 2, 2, 2, 9, 6, 5, 3, 4, 3, 3, 2, 4, 1, 5, 3, 1, 9, 13, 8, 4]
$pr226.sftsp$_28_2			73		[2, 6, 2, 2, 1, 1, 1, 2, 1, 1, 2, 4, 6, 1, 2, 1, 2, 2, 1, 3, 1, 5, 1, 1, 9, 3, 8, 2]
$pr226.sftsp$_28_3			188		[7, 8, 4, 6, 3, 8, 9, 4, 3, 4, 4, 10, 8, 6, 8, 5, 7, 5, 5, 10, 3, 7, 5, 2, 11, 14, 16, 6]
$pr226.sftsp$_28_4			112		[2, 8, 2, 2, 3, 1, 9, 2, 1, 1, 2, 10, 8, 1, 2, 1, 7, 2, 1, 3, 3, 5, 5, 1, 9, 3, 16, 2]
$gr229.sftsp$_24_1	229	24	133	[8, 9, 4, 5, 10, 5, 6, 10, 11, 12, 6, 6, 6, 10, 14, 7, 8, 3, 9, 17, 6, 15, 5, 36]	[3, 4, 4, 5, 7, 3, 2, 3, 10, 5, 2, 2, 2, 4, 14, 5, 6, 3, 3, 3, 5, 13, 1, 24]
$gr229.sftsp$_24_2			55		[1, 1, 1, 1, 4, 1, 2, 1, 4, 4, 2, 2, 1, 1, 3, 2, 3, 1, 2, 1, 5, 6, 1, 5]
$gr229.sftsp$_24_3			186		[5, 9, 4, 5, 10, 4, 4, 6, 11, 11, 4, 5, 6, 6, 14, 6, 8, 3, 7, 4, 6, 14, 2, 32]
$gr229.sftsp$_24_4			101		[1, 1, 4, 1, 4, 4, 2, 6, 11, 4, 4, 2, 1, 1, 14, 6, 8, 3, 2, 4, 5, 6, 2, 5]
$gil262.sftsp$_37_1	262	37	150	[3, 8, 8, 6, 4, 9, 7, 4, 5, 8, 5, 3, 5, 7, 8, 3, 4, 4, 9, 9, 9, 7, 9, 5, 6, 10, 6, 9, 9, 4, 5, 8, 5, 4, 16, 7, 23]	[2, 2, 5, 4, 1, 3, 4, 3, 3, 1, 3, 1, 1, 1, 6, 3, 1, 4, 7, 2, 6, 7, 2, 2, 5, 5, 4, 3, 1, 4, 3, 6, 1, 1, 13, 7, 23]
$gil262.sftsp$_37_2			102		[1, 1, 3, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 1, 3, 2, 1, 4, 7, 1, 2, 5, 2, 2, 4, 4, 1, 2, 1, 4, 2, 5, 1, 1, 12, 1, 16]
$gil262.sftsp$_37_3			212		[3, 3, 8, 5, 3, 4, 6, 4, 5, 7, 4, 3, 2, 7, 7, 3, 2, 4, 9, 3, 8, 7, 3, 4, 6, 6, 6, 7, 4, 4, 4, 7, 5, 3, 16, 7, 23]
$gil262.sftsp$_37_4			167		[3, 1, 8, 5, 3, 4, 6, 4, 5, 1, 4, 1, 1, 7, 3, 2, 1, 4, 7, 3, 8, 5, 3, 2, 6, 4, 1, 7, 1, 4, 4, 5, 1, 1, 12, 7, 23]
$pr264.sftsp$_37_1	264	37	144	[3, 8, 8, 6, 4, 9, 7, 4, 5, 8,	[2, 2, 5, 4, 1, 3, 4, 3, 3, 1,

Continues on the next page

Table A.2

Instance Name	$\lvert N\rvert + 1$	L	V	n_l	v_l
				5, 3, 5, 7, 8, 3, 4, 4, 9, 9, 9, 7, 9, 5, 6, 4, 11, 9, 9, 4, 5, 12, 11, 4, 15, 10, 14]	3, 1, 1, 1, 6, 3, 1, 4, 7, 2, 6, 7, 2, 2, 5, 3, 11, 3, 1, 4, 3, 6, 9, 1, 11, 7, 6]
$pr264.sftsp$_37_2			93		[1, 1, 3, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 1, 3, 2, 1, 4, 7, 1, 2, 5, 2, 2, 4, 3, 10, 2, 1, 4, 2, 5, 8, 1, 2, 1, 2]
$pr264.sftsp$_37_3			214		[3, 3, 8, 5, 3, 4, 6, 4, 5, 7, 4, 3, 2, 7, 7, 3, 2, 4, 9, 3, 8, 7, 3, 4, 6, 4, 11, 7, 7, 4, 4, 11, 10, 3, 13, 10, 10]
$pr264.sftsp$_37_4			158		[1, 3, 8, 5, 3, 4, 6, 4, 2, 7, 2, 1, 2, 1, 3, 2, 1, 4, 9, 3, 2, 7, 2, 4, 4, 3, 11, 2, 7, 4, 2, 5, 8, 1, 13, 10, 2]

Table A.3: Complete description of the instance set 3.

Instance Name	$\lvert N\rvert + 1$	L	V	n_l	v_l
$br17.aftsp$_3_1	17	3	9	[3, 2, 11]	[2, 2, 5]
$br17.aftsp$_3_2			6		[1, 2, 3]
$br17.aftsp$_3_3			14		[3, 2, 9]
$br17.aftsp$_3_4			8		[3, 2, 3]
$ftv33.aftsp$_8_1	34	5	23	[5, 8, 4, 10, 6]	[3, 7, 4, 8, 1]
$ftv33.aftsp$_8_2			10		[1, 3, 4, 1, 1]
$ftv33.aftsp$_8_3			33		[5, 8, 4, 10, 6]
$ftv33.aftsp$_8_4			19		[5, 8, 4, 1, 1]
$ftv35.aftsp$_5_1	36	5	21	[3, 9, 8, 6, 9]	[1, 9, 8, 2, 1]
$ftv35.aftsp$_5_2			12		[1, 1, 8, 1, 1]
$ftv35.aftsp$_5_3			29		[3, 9, 8, 6, 3]
$ftv35.aftsp$_5_4			20		[1, 9, 8, 1, 1]
$ftv38.aftsp$_5_1	39	5	23	[3, 9, 6, 4, 16]	[1, 9, 2, 2, 9]
$ftv38.aftsp$_5_2			11		[1, 1, 2, 1, 6]
$ftv38.aftsp$_5_3			37		[3, 9, 6, 4, 15]
$ftv38.aftsp$_5_4			21		[3, 9, 2, 1, 6]
$p43.aftsp$_6_1	43	6	27	[3, 8, 8, 6, 4, 13]	[3, 8, 2, 1, 2, 11]
$p43.aftsp$_6_2			17		[2, 1, 1, 1, 2, 10]
$p43.aftsp$_6_3			39		[3, 8, 8, 5, 3, 12]
$p43.aftsp$_6_4			27		[2, 1, 8, 1, 3, 12]
$ftv44.aftsp$_6_1	45	6	31	[3, 9, 6, 10, 4, 12]	[3, 5, 2, 9, 2, 10]
$ftv44.aftsp$_6_2			20		[2, 3, 1, 6, 2, 6]
$ftv44.aftsp$_6_3			39		[3, 7, 5, 10, 3, 11]
$ftv44.aftsp$_6_4			29		[2, 7, 1, 10, 3, 6]
$ftv47.aftsp$_6_1	48	6	34	[3, 9, 6, 5, 12, 12]	[3, 5, 2, 4, 10, 10]
$ftv47.aftsp$_6_2			20		[2, 3, 1, 4, 4, 6]
$ftv47.aftsp$_6_3			42		[3, 7, 5, 5, 11, 11]
$ftv47.aftsp$_6_4			32		[2, 7, 1, 5, 11, 6]

Continues on the next page

Table A.3

Instance Name	$\|N\|+1$	L	V	n_l	v_l
$ry48p.aftsp$_6_1	48	6	34	[3, 9, 6, 5, 12, 12]	[3, 5, 2, 4, 10, 10]
$ry48p.aftsp$_6_2			20		[2, 3, 1, 4, 4, 6]
$ry48p.aftsp$_6_3			42		[3, 7, 5, 5, 11, 11]
$ry48p.aftsp$_6_4			32		[2, 7, 1, 5, 11, 6]
$ft53.aftsp$_8_1	53	7	29	[3, 9, 6, 10, 4, 5, 15]	[2, 2, 1, 10, 2, 4, 8]
$ft53.aftsp$_8_2			22		[1, 2, 1, 6, 2, 3, 7]
$ft53.aftsp$_8_3			45		[3, 5, 4, 10, 3, 5, 15]
$ft53.aftsp$_8_4			49		[3, 2, 4, 6, 3, 3, 15]
$ftv55.aftsp$_7_1	56	7	17	[3, 9, 6, 5, 10, 6, 16]	[2, 2, 1, 5, 4, 2, 1]
$ftv55.aftsp$_7_2			9		[1, 2, 1, 1, 2, 1, 1]
$ftv55.aftsp$_7_3			33		[3, 5, 4, 5, 7, 6, 3]
$ftv55.aftsp$_7_4			21		[3, 2, 4, 1, 7, 1, 3]
$ftv64.aftsp$_8_1	65	8	34	[7, 9, 6, 10, 4, 11, 10, 7]	[2, 1, 4, 4, 4, 4, 9, 6]
$ftv64.aftsp$_8_2			23		[2, 1, 2, 3, 3, 3, 4, 5]
$ftv64.aftsp$_8_3			59		[6, 5, 6, 10, 4, 11, 10, 7]
$ftv64.aftsp$_8_4			48		[6, 1, 6, 3, 4, 11, 10, 7]
$ft70.aftsp$_8_1	70	8	39	[7, 9, 6, 5, 10, 5, 15, 12]	[2, 1, 4, 4, 4, 3, 9, 12]
$ft70.aftsp$_8_2			27		[2, 1, 2, 3, 3, 1, 4, 11]
$ft70.aftsp$_8_3			55		[6, 5, 6, 5, 5, 5, 11, 12]
$ft70.aftsp$_8_4			49		[6, 1, 6, 3, 5, 5, 11, 12]
$ftv70.aftsp$_9_1	71	9	35	[3, 9, 6, 10, 4, 11, 7, 10, 10]	[1, 1, 4, 4, 1, 8, 6, 4, 6]
$ftv70.aftsp$_9_2			25		[1, 1, 3, 3, 1, 7, 3, 4, 2]
$ftv70.aftsp$_9_3			54		[3, 2, 6, 6, 4, 10, 7, 6, 10]
$ftv70.aftsp$_9_4			50		[3, 2, 6, 6, 4, 10, 3, 6, 10]
$kro124p.aftsp$_14_1	100	14	66	[3, 8, 8, 6, 4, 9, 7, 4, 3, 3, 10, 13, 5, 16]	[3, 8, 8, 6, 2, 9, 6, 3, 1, 2, 9, 3, 2, 4]
$kro124p.aftsp$_14_2			30		[1, 2, 2, 3, 2, 5, 2, 2, 1, 2, 4, 1, 2, 1]
$kro124p.aftsp$_14_3			89		[3, 8, 8, 6, 3, 9, 7, 4, 2, 3, 10, 9, 4, 13]
$kro124p.aftsp$_14_4			46		[3, 2, 2, 3, 2, 5, 2, 2, 2, 3, 4, 1, 2, 13]
$ftv170.aftsp$_19_1	171	19	97	[7, 9, 4, 5, 9, 11, 7, 4, 11, 12, 6, 6, 6, 5, 10, 10, 12, 14, 22]	[7, 9, 3, 5, 8, 8, 3, 4, 2, 1, 4, 2, 3, 1, 9, 2, 8, 2, 16]
$ftv170.aftsp$_19_2			65		[7, 4, 1, 4, 3, 4, 2, 1, 1, 1, 2, 1, 3, 1, 9, 1, 8, 2, 10]
$ftv170.aftsp$_19_3			142		[7, 9, 4, 5, 9, 10, 6, 4, 4, 11, 6, 3, 4, 5, 10, 8, 9, 7, 21]
$ftv170.aftsp$_19_4			100		[7, 9, 1, 4, 3, 4, 2, 1, 4, 11, 2, 1, 4, 5, 9, 8, 8, 7, 10]
$rbg323.aftsp$_34_1	323	34	190	[8, 9, 4, 5, 9, 11, 7, 4, 11, 12, 6, 6, 5, 11, 12, 6, 8, 11, 9, 6, 8, 12, 6, 5, 5, 16, 13, 10, 12, 7, 6, 8, 25, 29]	[2, 2, 4, 4, 4, 9, 5, 1, 3, 8, 5, 1, 1, 4, 10, 5, 1, 3, 9, 1, 8, 4, 2, 4, 5, 14, 7, 1, 7, 4, 3, 7, 16, 26]
$rbg323.aftsp$_34_2			105		[2, 1, 1, 1, 1, 2, 5, 1, 2, 1, 4, 1, 1, 3, 8, 4, 1, 3, 4, 1, 3, 3, 1, 4, 2, 13, 5, 1, 4, 2, 2, 3, 5, 10]
$rbg323.aftsp$_34_3			271		[8, 6, 4, 5, 9, 11, 6, 3, 5, 10,

Continues on the next page

Table A.3

Instance Name	$\|N\|+1$	L	V	n_l	v_l
					6, 3, 3, 6, 11, 6, 6, 4, 9, 3, 8, 11, 5, 5, 5, 15, 13, 8, 10, 6, 6, 8, 19, 28]
$rbg323.aftsp$_34_4			174		[8, 1, 1, 1, 9, 2, 6, 1, 5, 1, 6, 1, 3, 6, 11, 6, 6, 4, 9, 1, 8, 3, 1, 5, 2, 13, 5, 1, 4, 6, 6, 3, 19, 10]
$rbg358.aftsp$_49_1	358	49	219	[3, 8, 8, 6, 4, 9, 7, 4, 5, 8, 5, 3, 5, 7, 8, 3, 4, 4, 9, 9, 9, 7, 9, 5, 6, 10, 6, 3, 4, 4, 5, 4, 7, 10, 4, 12, 6, 6, 11, 10, 7, 5, 10, 3, 15, 3, 21, 18, 18]	[2, 1, 6, 3, 1, 5, 6, 2, 4, 6, 2, 2, 1, 3, 2, 3, 3, 4, 4, 6, 5, 6, 8, 2, 2, 5, 1, 2, 1, 3, 5, 2, 4, 8, 2, 5, 6, 4, 5, 9, 7, 5, 4, 3, 6, 2, 19, 12, 10]
$rbg358.aftsp$_49_2			131		[1, 1, 3, 2, 1, 1, 2, 1, 4, 1, 2, 2, 1, 3, 1, 3, 3, 1, 3, 5, 3, 2, 7, 1, 2, 5, 1, 1, 1, 2, 3, 1, 2, 3, 1, 1, 4, 2, 4, 2, 7, 4, 2, 3, 2, 2, 3, 10, 9]
$rbg358.aftsp$_49_3			312		[3, 6, 7, 4, 4, 7, 7, 3, 5, 7, 3, 3, 2, 4, 3, 3, 4, 4, 6, 9, 9, 7, 9, 4, 6, 7, 5, 3, 3, 4, 5, 4, 6, 10, 3, 7, 6, 6, 9, 10, 7, 5, 10, 3, 15, 3, 21, 17, 14]
$rbg358.aftsp$_49_4			243		[3, 1, 7, 4, 4, 7, 7, 1, 4, 1, 3, 2, 1, 4, 1, 3, 3, 1, 6, 9, 9, 2, 9, 4, 6, 7, 1, 3, 1, 4 , 5, 1, 2, 3, 3, 1, 4, 6, 9, 2, 7, 5, 10, 3, 15, 2, 21, 17, 9]
$rbg403.aftsp$_50_1	403	50	227	[7, 9, 6, 10, 4, 9, 10, 4, 3, 4, 4, 10, 3, 5, 10, 10, 4, 4, 9, 9, 9, 7, 10, 5, 6, 4, 11, 9, 9, 4, 5, 4, 11, 4, 6, 12, 6, 5, 11, 10, 11, 5, 11, 3, 15, 3, 8, 15, 19, 30]	[1, 6, 3, 5, 4, 7, 4, 2, 3, 2, 3, 1, 1, 2, 7, 5, 4, 3, 6, 5, 11, 2, 6, 7, 5, 3, 3, 4, 7, 4, 1, 2, 3, 8, 2, 5, 6, 4, 3, 9, 7, 8, 4, 8, 3, 2, 1, 4, 1, 7, 24]
$rbg403.aftsp$_50_2			148		[1, 2, 3, 1, 2, 3, 4, 1, 2, 2, 2, 1, 1, 2, 2, 1, 3, 2, 3, 2, 1, 3, 2, 5, 3, 1, 4, 6, 4, 1, 2, 2, 3, 1, 5, 4, 3, 2, 5, 5, 2, 4, 4, 1, 2, 1, 3, 1, 5, 23]
$rbg403.aftsp$_50_3			341		[2, 9, 6, 9, 4, 8, 5, 3, 3, 4, 4, 7, 2, 3, 8, 7, 4, 4, 9, 8, 8, 7, 8, 5, 6, 4, 9, 9, 5, 3, 4, 4, 10, 3, 6, 10, 6, 5, 10, 8, 10, 5, 11, 3, 10, 3, 6, 9, 16, 29]
$rbg403.aftsp$_50_4			243		[2, 2, 3, 1, 4, 3, 4, 3, 2, 2, 2, 1, 2, 3, 8, 1, 4, 4, 9, 8, 1, 7, 2, 5, 6, 1, 4, 6, 5, 1, 2, 4, 10, 1, 5, 10, 6, 2, 10, 5, 10, 5, 4, 3, 10, 1, 6, 9, 5, 29]
$rbg443.aftsp$_64_1	443	64	265	[5,3,8, 8, 6, 5, 7, 4, 5, 8, 5, 3, 5, 7, 8, 3, 5, 8, 9, 8, 5, 5,	[2, 1, 8, 3, 6, 1, 6, 1, 2, 6, 5, 1, 3, 3, 7, 1, 2, 7, 8, 6,

Continues on the next page

Table A.3

Instance Name	$\|N\|+1$	L	V	n_l	v_l
				5, 5, 3, 3, 5, 9, 9, 9, 5, 7, 8, 5, 6, 7, 4, 6, 8, 4, 7, 8, 7, 5, 6, 5, 3, 6, 8, 9, 10, 8, 8, 11, 3, 8, 12, 6, 6, 6, 4, 9, 9, 13, 12, 25]	1, 5, 1, 3, 4, 8, 9, 2, 2, 6, 6, 5, 6, 1, 1, 6, 5, 2, 2, 8, 4, 5, 2, 4, 2, 4, 2, 3, 7, 2, 5, 1, 1, 1, 9, 2, 3, 3, 2, 3, 9, 1, 4, 24]
$rbg443.aftsp$_64_2			180		[2, 1, 6, 2, 5, 1, 4, 1, 1, 5, 2, 1, 2, 3, 7, 1, 2, 2, 3, 6, 1, 2, 1, 3, 2, 7, 7, 1, 1, 4, 6, 2, 5, 1, 1, 4, 4, 2, 1, 7, 2, 5, 1, 3, 2, 2, 2, 2, 3, 2, 3, 1, 1, 1, 7, 2, 3, 1, 2, 2, 9, 1, 2, 2]
$rbg443.aftsp$_64_3			363		[5, 2, 8, 4, 6, 4, 7, 2, 5, 7, 5, 3, 5, 5, 8, 3, 4, 8, 9, 7, 5, 5, 2, 3, 5, 9, 9, 5, 3, 7, 8, 5, 6, 3, 3, 6, 6, 3, 6, 8, 5, 5, 3, 5, 3, 5, 4, 8, 10, 4, 8, 2, 3, 2, 10, 6, 4, 5, 3, 8, 9, 2, 10, 25]
$rbg443.aftsp$_64_4			272		[5, 1, 6, 2, 6, 1, 7, 1, 1, 7, 2, 3, 5, 5, 8, 3, 4, 8, 3, 7, 1, 5, 1, 3, 2, 9, 9, 1, 1, 4, 6, 5, 6, 1, 1, 4, 4, 3, 1, 7, 2, 5, 1, 5, 2, 2, 4, 2, 10, 2, 8, 2, 1, 1, 10, 2, 3, 1, 2, 2, 9, 2, 10, 25]

Table A.4: Complete description of the instance set 4.

Instance Name	$\|N\|+1$	L	V	n_l	v_l
$AsimSingh.aftsp$_50_6_1	50	6	28	[7, 9, 6, 5, 7, 15]	[7, 5, 2, 4, 6, 4]
$AsimSingh.aftsp$_50_6_2			13		[1, 3, 1, 4, 2, 2]
$AsimSingh.aftsp$_50_6_3			45		[7, 7, 5, 5, 7, 14]
$AsimSingh.aftsp$_50_6_4			23		[1, 7, 1, 5, 7, 2]
$AsimSingh.aftsp$_100_14_1	100	14	66	[3, 8, 8, 6, 4, 9, 7, 4, 3, 3, 10, 13, 5, 16]	[3, 8, 8, 6, 2, 9, 6, 3, 1, 2, 9, 3, 2, 4]
$AsimSingh.aftsp$_100_14_2			30		[1, 2, 2, 3, 2, 5, 2, 2, 1, 2, 4, 1, 2, 1]
$AsimSingh.aftsp$_100_14_3			89		[3, 8, 8, 6, 3, 9, 7, 4, 2, 3, 10, 9, 4, 13]
$AsimSingh.aftsp$_100_14_4			46		[3, 2, 2, 3, 2, 5, 2, 2, 2, 3, 4, 1, 2, 13]
$AsimSingh.aftsp$_150_16_1	150	16	78	[8, 9, 4, 5, 10, 5, 6, 6, 3, 4, 12, 16, 10, 8, 8, 35]	[8, 6, 2, 4, 5, 4, 1, 5, 3, 4, 8, 4, 7, 2, 2, 13]
$AsimSingh.aftsp$_150_16_2			48		[2, 5, 2, 4, 5, 2, 1, 4, 1, 1, 3, 4, 6, 1, 1, 6]
$AsimSingh.aftsp$_150_16_3			117		[8, 7, 3, 5, 6, 5, 6, 6, 3, 4,

Continues on the next page

Table A.4

Instance Name	$\lvert N\rvert + 1$	L	V	n_l	v_l
					12, 6, 10, 8, 4, 24]
$AsimSingh.aftsp$_150_16_4			58		[2, 5, 2, 5, 5, 2, 1, 4, 3, 4, 3, 4, 10, 1, 1, 6]
$AsimSingh.aftsp$_200_29_1	200	29	102	[5, 3, 8, 6, 4, 9, 7, 4, 5, 8, 5, 3, 5, 7, 10, 3, 4, 4, 11, 9, 9, 7, 10, 5, 6, 4, 14, 3, 21]	[3, 2, 1, 2, 1, 8, 2, 4, 4, 2, 4, 1, 4, 2, 10, 2, 1, 1, 4, 7, 8, 1, 6, 3, 1, 4, 6, 2, 6]
$AsimSingh.aftsp$_200_29_2			69		[3, 2, 1, 1, 1, 2, 1, 3, 4, 1, 2, 1, 1, 1, 7, 2, 1, 1, 2, 3, 7, 1, 6, 1, 1, 2, 5, 2, 4]
$AsimSingh.aftsp$_200_29_3			159		[4, 3, 8, 3, 3, 9, 3, 4, 5, 3, 5, 3, 5, 6, 10, 3, 3, 4, 8, 9, 9, 3, 7, 5, 6, 4, 8, 3, 13]
$AsimSingh.aftsp$_200_29_4			96		[3, 3, 1, 3, 1, 2, 1, 3, 5, 3, 2, 1, 1, 6, 7, 2, 1, 4, 2, 9, 7, 1, 6, 5, 1, 4, 5, 3, 4]
$AsimSingh.aftsp$_250_35_1	250	35	142	[3, 8, 8, 6, 4, 9, 7, 4, 5, 8, 5, 3, 5, 7, 8, 3, 4, 4, 9, 9, 9, 7, 9, 5, 6, 4, 12, 9, 9, 12, 5, 8, 5, 4, 26]	[2, 4, 6, 4, 1, 1, 3, 3, 5, 3, 3, 1, 3, 6, 1, 3, 2, 1, 1, 5, 7, 2, 6, 5, 2, 3, 5, 7, 1, 9, 5, 8, 3, 2, 19]
$AsimSingh.aftsp$_250_35_2			79		[1, 1, 5, 2, 1, 1, 2, 1, 4, 1, 3, 1, 2, 2, 1, 3, 2, 1, 1, 2, 7, 2, 6, 1, 2, 1, 5, 1, 1, 6, 1, 2, 3, 2, 2]
$AsimSingh.aftsp$_250_35_3			208		[3, 8, 7, 6, 2, 3, 6, 4, 5, 7, 4, 2, 4, 7, 8, 3, 4, 4, 4, 7, 8, 4, 7, 5, 6, 4, 6, 8, 7, 11, 5, 8, 4, 3, 24]
$AsimSingh.aftsp$_250_35_4			162		[1, 8, 5, 6, 1, 3, 2, 4, 4, 7, 4, 2, 4, 7, 8, 3, 2, 4, 1, 2, 8, 2, 6, 1, 2, 1, 6, 8, 1, 11, 1, 8, 3, 2, 24]
$AsimSingh.aftsp$_300_34_1	300	34	172	[7, 9, 6, 10, 4, 11, 7, 4, 3, 3, 3, 6, 5, 11, 12, 6, 8, 12, 11, 7, 8, 12, 6, 10, 7, 10, 14, 3, 12, 14, 15, 12, 16, 15]	[5, 2, 2, 4, 2, 9, 5, 1, 3, 2, 2, 1, 1, 4, 10, 5, 1, 6, 6, 6, 8, 4, 2, 4, 7, 2, 7, 2, 7, 4, 12, 7, 16, 13]
$AsimSingh.aftsp$_300_34_2			106		[5, 1, 1, 1, 1, 2, 5, 1, 2, 1, 1, 1, 1, 3, 8, 4, 1, 6, 4, 3, 3, 3, 1, 4, 7, 1, 5, 1, 4, 2, 5, 3, 5, 10]
$AsimSingh.aftsp$_300_34_3			254		[7, 6, 4, 9, 4, 10, 7, 2, 3, 3, 3, 3, 3, 6, 11, 6, 6, 9, 8, 7, 8, 7, 4, 7, 7, 8, 14, 3, 11, 12, 14, 12, 16, 14]
$AsimSingh.aftsp$_300_34_4					[7, 1, 1, 1, 4, 2, 7, 1, 3, 1, 3, 1, 3, 6, 11, 6, 6, 9, 8, 3, 8, 3, 1, 7, 7, 1, 5, 1, 4, 12, 14, 3, 16, 10]

Appendix B

Branch-and-cut algorithm detailed results

B.1 Linear programming relaxation results without heuristic separation

Tables B.1 and B.2 show the LP relaxation results obtained with the CC and the RFV models and with the CC+RFV model, respectively, without using the heuristic separation algorithms. These tables contain the LP relaxation value ($\mathcal{V}^{LP}$), the computational time, in seconds, to obtain the LP relaxation value (t_s), the number of added violated CC inequalities (#CC) and the number of added violated RFV inequalities ($\#RFV$). Additionally, the tables also contain, in the last row, the average of the results obtained.

Table B.1: Linear programming relaxation results of the CC and RFV models without the heuristic separation algorithm.

	CC				RFV			
Instance	$\mathcal{V}^{LP}$	gap	t_s	#CC	$\mathcal{V}^{LP}$	gap	t_s	#RFV
burma_1	12.07	13.34%	1	76	13.93	0.00%	0	92
burma_2	25.66	0.00%	0	68	25.66	0.00%	0	33
burma_3	9.93	16.47%	0	48	11.89	0.00%	0	46
bayg_1	5273.32	1.36%	1	286	5316.85	0.54%	1	673
bayg_2	5754.64	0.63%	0	365	5791.01	0.00%	1	420
bayg_3	5544.33	0.00%	0	352	5544.33	0.00%	0	458
att_1	23686.00	0.00%	1	1092	23580.50	0.45%	2	1537
att_2	20609.10	0.00%	1	960	20609.10	0.00%	14	4165
att_3	8742.08	3.13%	0	732	8760.03	2.93%	5	2566
bier_1	33227.80	1.43%	70	3260	33314.70	1.17%	49743	57101
bier_2	88308.90	0.48%	116	3538	87336.20	1.58%	46501	61278
bier_3	47162.50	1.18%	599	6101	46830.70	1.88%	35712	59393
$average$		3.17%	66	1407		0.71%	10998	15647

Table B.2: Linear programming relaxation results of the CC+RFV model without the heuristic separation.

	CC+RFV				
Instance	$\mathcal{V}^{LP}$	gap	t_s	#CC	#RFV
burma_1	13.93	0.00%	0	56	97
burma_2	25.66	0.00%	0	68	70
burma_3	11.89	0.00%	0	32	46
bayg_1	5330.17	0.29%	0	363	125
bayg_2	5791.01	0.00%	0	353	81
bayg_3	5544.33	0.00%	0	352	8
att_1	23686.00	0.00%	1	1072	28
att_2	20609.10	0.00%	0	960	0
att_3	9024.58	0.00%	1	763	137
bier_1	33446.00	0.78%	337	3653	540
bier_2	88479.50	0.29%	330	3829	531
bier_3	47504.40	0.46%	1265	646	533
$average$		0.15%	161	1012	183

B.2 Optimal results without the heuristic callback

Tables B.3 and B.4 show the optimal results obtained with the CC model and with the y-separation and 1-separation, respectively, without using the heuristic callback during the B&C algorithm. These tables contain the optimal value ($\mathcal{V}$), the computational time, in seconds, to obtain the optimal value (t_s), the number of B&C subproblems solved (#sub) and the number of added violated CC inequalities (#CC) and RFV inequalities (#RFV). Additionally, the tables also contain, in the last row, the average of the results obtained.

Table B.3: Optimal values of the CC model without the heuristic callback.

	CC			
Instance	$\mathcal{V}$	t_s	#sub	#CC
burma_1	13.93	0	0	70
burma_2	25.66	0	0	36
burma_3	11.89	0	0	52
bayg_1	5345.86	0	0	263
bayg_2	5791.01	1	0	238
bayg_3	5544.33	0	0	207
att_1	23686.00	0	0	440
att_2	20609.10	1	0	757
att_3	9024.58	1	0	450
bier_1	33709.70	37	210	1809
bier_2	88736.40	344	592	10438
bier_3	47726.30	145	271	6056
$average$		44	89	1735

Table B.4: Optimal values of the y-separation and the 1-separation without the heuristic callback.

	y-separation					1-separation				
Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV	$\mathcal{V}$	t_s	#sub	#CC	#RFV
burma_1	13.93	0	0	16	24	13.93	0	0	18	36
burma_2	25.66	0	0	5	27	25.66	0	0	5	35
burma_3	11.89	0	0	8	17	11.89	0	0	6	34
bayg_1	5345.86	1	0	24	215	5345.86	0	1	41	244
bayg_2	5791.01	0	0	13	225	5791.01	1	0	11	211
bayg_3	5544.33	0	0	15	161	5544.33	0	0	15	181
att_1	23686.00	1	0	18	442	23686.00	0	0	15	467
att_2	20609.10	1	0	95	796	20609.10	1	0	95	646
att_3	9024.58	0	0	165	276	9024.58	0	0	172	252
bier_1	337709.70	23	317	292	2125	33709.70	20	185	287	1299
bier_2	88736.40	143	467	1394	8562	88736.40	73	208	867	4086
bier_3	47726.30	132	355	1880	7007	47726.30	366	600	2045	8425
average		27	95	327	1656		38	83	298	1326

B.3 Linear programming relaxation results for instance set 2

Table B.5 shows the LP relaxation results for the instance set 2 obtained using the y-separation algorithm and the 1-separation algorithm. Table B.5 is divided into two parts, each one associated with one of the separation algorithms. Each of those parts shows the LP relaxation value ($\mathcal{V}^{LP}$), the percentage of gap between the LP relaxation value and the optimal value (gap), the computational time, in seconds, to obtain the LP relaxation value (t_s), the number of added violated CC inequalities (#CC) and the number of added violated RFV inequalities (#RFV). Table B.5 shows, in the last row, the average of the results obtained.

Table B.5: Linear programming relaxation results for instance set 2.

	y-separation					1-separation				
Instance	$\mathcal{V}^{LP}$	gap	t_s	#CC	#RFV	$\mathcal{V}^{LP}$	gap	t_s	#CC	#RFV
pr136_1	61088.90	0.58%	24	342	1196	61101.20	0.56%	45	364	1452
pr136_2	43136.70	0.89%	45	1457	1177	43111.50	0.94%	36	1348	1135
pr136_3	80836.20	0.79%	5	53	780	80836.30	0.79%	4	57	651

Continues on the next page

Table B.5

	y-separation					1-separation				
Instance Name	$\mathcal{V}^{LP}$	gap	t_s	#CC	#RFV	$\mathcal{V}^{LP}$	gap	t_s	#CC	#RFV
pr136_4	62917.60	0.52%	13	369	1188	63080.40	0.26%	50	416	1341
gr137_1	43905.70	0.81%	6	167	865	44032.50	0.52%	19	212	1301
gr137_2	36223.30	0.58%	90	712	1371	36212.80	0.61%	75	726	1372
gr137_3	55196.30	1.29%	3	30	666	55220.40	1.25%	5	37	729
gr137_4	46500.00	0.26%	10	155	983	46568.20	0.11%	30	140	1157
pr144_1	45947.70	0.92%	25	314	1335	46319.20	0.12%	958	459	2834
pr144_2	36453.50	0.18%	116	1647	1740	36518.00	0.00%	1029	1902	2639
pr144_3	53974.50	1.21%	9	59	869	53999.00	1.16%	23	72	1038
pr144_4	48405.60	2.02%*	87	951	1936	48412.40	2.01%*	173	1325	2093
kroA150_1	14295.00	0.08%	25	611	1121	14295.00	0.08%	25	705	1179
kroA150_2	9372.01	1.15%	139	2098	955	9402.32	0.83%	96	1905	904
kroA150_3	20556.50	1.55%*	9	72	952	20556.50	1.55%*	8	74	990
kroA150_4	13360.50	0.32%	75	821	1822	13360.50	0.32%	109	1156	2107
kroB150_1	14498.40	0.16%	189	792	1464	14497.20	0.17%	58	745	1125
kroB150_2	9369.97	1.94%	198	1911	907	9333.28	2.32%	49	1581	589
kroB150_3	19880.50	0.22%	6	52	792	19880.50	0.22%	7	55	731
kroB150_4	12525.50	0.05%	67	976	1486	12525.50	0.05%	57	968	1374
pr152_1	51613.50	0.37%	198	713	1772	51487.40	0.61%	65	723	1344
pr152_2	45563.40	0.54%	858	2121	2103	45358.40	0.99%	627	2322	2252
pr152_3	63622.90	1.25%*	17	187	1049	63676.00	1.16%*	105	362	1782
pr152_4	56478.50	1.50%	55	546	1433	56481.00	1.49%	130	647	1922
u159_1	29524.20	1.00%	40	442	1524	29655.50	0.55%	179	618	2282
u159_2	23100.00	1.30%	213	1301	2121	22945.10	1.96%	105	1183	1943
u159_3	36269.60	0.36%	19	112	1191	36267.10	0.36%	15	82	994
u159_4	30313.30	1.72%	74	722	1804	30332.80	1.66%	232	753	2065
rat195_1	1237.47	3.70%	195	1461	1735	1238.25	3.64%	175	1152	1559
rat195_2	903.25	0.96%	380	2387	1409	904.88	0.78%	1151	2521	2026
rat195_3	1762.05	2.86%	43	307	1199	1761.73	2.88%	57	301	1311
rat195_4	1285.15	2.64%	257	912	2040	1286.10	2.57%	246	993	1965
d198_1	10884.50	0.55%	864	1642	3423	10882.00	0.58%	920	2303	3902
d198_2	10142.90	0.68%	1579	3363	3867	10143.20	0.67%	1209	3167	3704
d198_3	13779.70	0.46%	182	238	2481	13768.90	0.54%	109	193	1714
d198_4	12361.40	0.46%	862	858	3629	12362.40	0.45%	1597	891	4128
kroA200_1	16220.00	1.34%*	115	868	1557	16220.00	1.34%*	116	943	1682
kroA200_2	12345.90	0.56%	327	2329	1258	12345.90	0.56%	292	2232	1254
kroA200_3	23941.00	2.17%*	59	75	1551	23941.00	2.17%*	48	103	1383
kroA200_4	16479.00	0.24%	175	894	2059	16481.30	0.22%	209	1060	2198

Continues on the next page

Table B.5

	y-separation					1-separation				
Instance Name	$\mathcal{V}^{LP}$	gap	t_s	#CC	#RFV	$\mathcal{V}^{LP}$	gap	t_s	#CC	#RFV
kroB200_1	17195.40	1.89%	169	958	1808	17177.60	1.99%	176	1096	1810
kroB200_2	12451.60	3.33%	244	2537	828	12428.20	3.52%	320	2557	1038
kroB200_3	23671.00	1.73%*	62	132	1346	23671.20	1.73%*	75	136	1498
kroB200_4	17406.30	1.00%	629	1057	2763	17414.10	0.96%	1653	1001	2845
gr202_1	23215.70	0.76%	457	622	2165	23210.00	0.79%	418	672	2394
gr202_2	14924.50	0.22%	447	3600	1899	14941.20	0.11%	380	2356	1740
gr202_3	34361.90	0.54%	41	123	1396	34353.80	0.56%	34	125	1245
gr202_4	27722.20	1.13%*	474	1224	2906	27735.30	1.08%*	426	1155	3210
pr226_1	51994.50	0.22%	6270	1002	4699	52041.30	0.13%	27672	1154	9000
pr226_2	47456.90	0.27%	23542	5045	6274	47540.60	0.09%	31580	4467	6182
pr226_3	66583.00	0.34%	1893	322	5599	66652.50	0.24%	269	146	1930
pr226_4	51892.00	0.03%	3051	1213	3816	51893.50	0.02%	12860	1143	6431
gr229_1	69553.80	1.68%*	132	710	1841	69864.00	1.24%*	265	873	2405
gr229_2	30923.90	2.30%	1390	3016	2064	30776.40	2.77%	1671	3344	2113
gr229_3	102496.00	0.34%	73	211	1569	102526.00	0.31%	126	275	1883
gr229_4	45899.30	0.72%	330	1238	2332	45816.70	0.90%	385	1252	2267
gil262_1	1497.97	2.03%*	961	1144	3347	1497.25	2.08%*	2202	1122	3452
gil262_2	1059.35	0.90%	1136	2924	2109	1059.59	0.88%	1008	2839	2034
gil262_3	1954.50	2.86%*	222	150	1935	1954.79	2.34%*	373	193	2175
gil262_4	1641.00	7.45%*	1057	909	3757	1641.00	7.45%*	907	960	3566
pr264_1	33785.10	0.35%	18860	2952	7744	33761.90	0.42%	6660	2089	5805
pr264_2	28093.50	2.28%*	2393	3336	4702	28106.50	2.23%*	4770	4981	5108
pr264_3	40521.00	0.45%	778	347	3230	40502.00	0.50%	684	266	3198
pr264_4	34920.20	0.66%	825	1105	3526	34924.60	0.65%	3079	1564	5076
average		1.15%	1142	1109	2132		1.13%	1694	1134	2290

*Gap computed with the best upper bound shown in Table 5.22 instead of the optimal value

The gap values marked with an asterisk (*) were obtained by using the best upper bound obtained with the B&C algorithm after 10800 seconds of computational time instead of the optimal value as we could not obtain it within the time limit.

B.4 Optimal results obtained with the 1-separation

Table B.6 shows the results obtained using the 1-separation for the instances of type $high$ from the instance set 2. This table shows the optimal value ($\mathcal{V}$), the computational time, in seconds, to obtain the optimal value (t_s), the number of B&C subproblems solved (#sub) and the number of added violated CC inequalities (#CC) and RFV inequalities (#RFV). This table also contains, in the last row, the average of the results obtained.

Table B.6: Optimal values for the instances of type $high$ from the instance set 3 with 1-separation.

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
pr136_3	81481	129	1815	211	7713
gr137_3	55919	17	238	78	1902
pr144_3	54635	6272	52386	855	50308
kroA150_3	[20774.00, 20880]	10801	60276	1201	46322
kroB150_3	19925	5	5	27	828
pr152_3	[63933.70, 65347]	10801	13608	7028	96874
u159_3	36399	18	74	151	1464
rat195_3	[1778.29, 1827]	10804	5210	8530	106388
d198_3	13843	146	144	247	4516
kroA200_3	[24048.00, 24320]	10804	10796	4324	120222
kroB200_3	[23768.00, 24297]	10803	10102	2476	102907
gr202_3	34547	803	3734	609	15008
pr226_3	66812	1536	38	370	8906
gr229_3	102840	2229	4771	1281	32477
gil262_3	[1963.51, 1980]	10804	7099	2951	92105
pr264_3	[40646.90, 40774]	10803	1189	2230	94387
$average$		5423	10718	2036	48895

B.5 Linear programming relaxation results for instance set 3

Table B.7 shows the LP relaxation results obtained using the y-separation algorithm and the 1-separation algorithm for the instance set 3 and it is organized in the same manner as Table B.5.

Table B.7: Linear programming relaxation results for the instance set 3.

	y-separation					1-separation				
Instance	$\mathcal{V}^{LP}$	gap	t_s	#CC	#RFV	$\mathcal{V}^{LP}$	gap	t_s	#CC	#RFV
br17_1	31	0.00%	0	17	64	31	0.00%	0	13	99

Continues on the next page

Table B.7

	y-separation					1-separation				
Instance Name	$\mathcal{V}^{LP}$	*gap*	t_s	#CC	#RFV	$\mathcal{V}^{LP}$	*gap*	t_s	#CC	#RFV
br17_2	28	0.00%	0	1	32	28	0.00%	0	1	69
br17_3	39	0.00%	0	4	76	39	0.00%	0	3	77
br17_4	36	0.00%	0	6	110	36	0.00%	0	1	151
ftv33_1	842.00	3.00%	0	15	214	842.00	3.00%	0	20	228
ftv33_2	377.50	5.86%	0	33	88	395.50	1.37%	1	162	264
ftv33_3	1286.00	0.00%	0	0	144	1286.00	0.00%	0	0	144
ftv33_4	827.00	0.24%	0	0	136	827.00	0.24%	0	0	148
ftv35_1	1002.50	0.55%	1	3	344	1002.83	0.51%	0	3	432
ftv35_2	523.00	1.32%	0	21	187	523.00	1.32%	0	15	178
ftv35_3	1203.67	2.30%	0	0	132	1203.67	2.30%	1	0	133
ftv35_4	1000.00	0.79%	0	4	391	1000.00	0.79%	0	4	412
ftv38_1	830.00	0.00%	0	13	161	830.00	0.00%	0	8	151
ftv38_2	388.41	0.92%	0	118	123	392.00	0.00%	1	101	169
ftv38_3	1424.50	1.69%	1	0	167	1424.50	1.69%	0	0	169
ftv38_4	774.00	0.00%	0	18	185	774.00	0.00%	0	18	220
p43_1	5482.50	0.01%	0	56	202	5483.00	0.00%	0	23	209
p43_2	5472.00	0.02%	1	218	268	5472.13	0.02%	3	210	523
p43_3	5530.00	0.00%	0	1	93	5530.00	0.00%	0	1	103
p43_4	5492.00	0.00%	0	1	106	5492.00	0.00%	0	1	139
ftv44_1	979.15	1.69%	0	13	167	991.17	0.49%	0	15	189
ftv44_2	602.94	3.53%	0	149	234	607.46	2.81%	0	146	268
ftv44_3	1325.00	1.34%	0	6	154	1325.00	1.34%	0	6	145
ftv44_4	954.50	4.36%	1	34	282	958.00	4.01%	1	30	304
ftv47_1	1166.42	1.07%	0	41	294	1167.33	0.99%	0	22	264
ftv47_2	698.98	4.12%	0	169	286	701.56	3.76%	0	133	267
ftv47_3	1438.00	2.31%	0	3	172	1438.00	2.31%	0	3	175
ftv47_4	1099.00	0.00%	0	10	159	1099.00	0.00%	0	10	159
ry48p_1	10252.70	0.63%	0	64	330	10252.70	0.63%	0	46	338
ry48p_2	6716.64	1.04%	1	260	294	6716.64	1.04%	1	284	322
ry48p_3	12535.20	1.70%	0	21	204	12535.20	1.70%	0	21	203
ry48p_4	10025.80	1.12%	0	45	379	10049.40	0.88%	0	46	401
ft53_1	3534.25	1.06%	0	46	321	3553.89	0.51%	1	87	468
ft53_2	2715.46	3.67%	1	152	356	2750.85	2.42%	1	182	408
ft53_3	5892.67	1.33%	0	12	167	5892.67	1.33%	0	12	171
ft53_4	4680.84	1.12%	0	49	310	4683.10	1.08%	0	39	307
ftv55_1	564.67	0.94%	1	69	435	564.17	1.02%	1	76	401
ftv55_2	344.84	5.52%	2	432	190	359.63	1.47%	4	585	256

Continues on the next page

Table B.7

	y-separation					1-separation				
Instance Name	$\mathcal{V}^{LP}$	*gap*	t_s	#CC	#RFV	$\mathcal{V}^{LP}$	*gap*	t_s	#CC	#RFV
ftv55_3	1010.75	1.00%	3	47	1193	1016.50	0.44%	1	24	908
ftv55_4	579.00	0.00%	1	95	417	579.00	0.00%	0	69	143
ftv64_1	964.83	1.25%	3	157	723	968.39	0.88%	5	208	933
ftv64_2	640.80	2.91%	2	551	342	638.61	3.24%	2	407	378
ftv64_3	1611.00	0.37%	0	0	220	1611.00	0.37%	0	0	180
ftv64_4	1509.00	0.40%	11	0	1941	1509.00	0.40%	9	0	2437
ft70_1	21198.10	0.13%	1	31	549	21198.10	0.13%	0	23	556
ft70_2	15316.60	0.28%	2	78	799	15321.60	0.25%	3	70	910
ft70_3	29499.70	0.25%	1	19	396	29499.70	0.25%	0	26	373
ft70_4	26749.10	0.25%	0	6	277	26758.50	0.22%	0	0	248
ftv70_1	798.73	5.92%	2	281	646	802.23	5.51%	2	254	648
ftv70_2	629.57	6.87%	2	393	456	631.19	6.63%	2	370	488
ftv70_3	1315.80	1.95%	1	21	414	1315.80	1.95%	0	16	483
ftv70_4	1241.31	1.56%	0	30	530	1241.31	1.56%	0	22	398
kro124p_1	25004.00	0.98%	2	20	844	25004.00	0.98%	4	42	919
kro124p_2	12056.80	2.93%	9	1017	448	12009.50	3.31%	8	964	477
kro124p_3	31829.40	0.61%	1	11	365	31829.40	0.61%	1	11	366
kro124p_4	18148.80	2.17%	3	472	488	18148.80	2.17%	3	512	471
ftv170_1	1593.61	3.53%	307	1036	2721	1592.89	3.58%	1358	1345	4086
ftv170_2	1045.17	5.67%	401	2022	2090	1046.11	5.59%	663	2225	2150
ftv170_3	2184.16	1.39%	30	84	1374	2187.77	1.23%	24	51	1402
ftv170_4	1573.30	2.28%	104	608	2027	1582.40	1.71%	255	854	2756
rbg323_1	336.00	0.30%	25	6	174	336.33	0.20%	630	24	2764
rbg323_2	67.30	3.86%	14	2	53	67.32	3.83%	121	2	1075
rbg323_3	822.00	0.00%	12	2	112	822.00	0.00%	8	1	119
rbg323_4	333.67	0.40%	7	1	59	333.67	0.40%	6	1	59
rbg358_1	206.87	1.02%	18	1	68	206.87	1.02%	30	1	138
rbg358_2	48.75	2.50%	62	2	379	48.75	2.50%	24	1	240
rbg358_3	658.00	0.00%	10	0	64	658.00	0.00%	10	0	64
rbg358_4	408.33	0.16%	17	1	126	408.33	0.16%	13	1	71
rbg403_1	345.00	0.00%	11	4	56	345.00	0.00%	16	4	45
rbg403_2	30.50	4.69%	34	11	154	30.50	4.69%	47	12	159
rbg403_3	1425.00	0.14%	77	1	451	1425.00	0.14%	37	0	222
rbg403_4	486.00	0.00%	9	0	20	486.00	0.00%	11	0	20
rbg443_1	539.00	0.00%	32	0	80	539.00	0.00%	0	0	80
rbg443_2	Out of memory					Out of memory				
rbg443_3	1353.00	0.00%	21	0	40	1353.00	0.00%	20	0	40

Continues on the next page

Table B.7

	y-separation					1-separation				
Instance Name	$\mathcal{V}^{LP}$	gap	t_s	#CC	#RFV	$\mathcal{V}^{LP}$	gap	t_s	#CC	#RFV
rbg443_4	728.00	0.14%	23	0	40	728.00	0.14%	22	0	40
$average$		1.46%	17	122	388		1.24%	45	132	482

B.6 Linear programming relaxation results for instances rbg with SCF model

Table B.8 contains the LP relaxation results obtained with the SCF model for the rbg instances from the instance set 3. Table B.8 shows the LP relaxation value obtained ($\mathcal{V}^{LP}$) and the computational time, in seconds, to obtain the LP relaxation value (t_s). Additionally, the table also contains, in the last row, the average of the results obtained.

Table B.8: Linear programming relaxation results for instances rbg with SCF model.

Instance	$\mathcal{V}^{LP}$	gap	t_s
rbg323_1	336.00	0.30%	167
rbg323_2	67.26	3.91%	102
rbg323_3	822.00	0.00%	48
rbg323_4	333.67	2.28%	74
rbg358_1	206.87	1.02%	102
rbg358_2	48.75	2.50%	123
rbg358_3	658.00	0.00%	44
rbg358_4	408.33	0.16%	61
rbg403_1	345.00	0.00%	188
rbg403_2	30.50	4.69%	341
rbg403_3	1425.00	0.14%	55
rbg403_4	486.00	0.00%	81
rbg443_1	539.00	0.00%	192
rbg443_2	Out of memory		
rbg443_3	1353.00	0.00%	68
rbg443_4	728.00	0.14%	80
average		1.01%	115

B.7 Linear programming relaxation results for instance set 4

Table B.9 shows the LP relaxation results obtained using the y-separation algorithm and the 1-separation algorithm for the instance set 4 and it is organized in a similar manner as Table B.5.

Table B.9: Linear programming relaxation results for the instance set 4.

	y-separation					1-separation				
Instance	$\mathcal{V}^{LP}$	*gap*	t_s	#CC	#RFV	$\mathcal{V}^{LP}$	*gap*	t_s	#CC	#RFV
AsimSingh50_1	437.88	0.03%	0	0	111	437.92	0.02%	2	8	405
AsimSingh50_2	210.39	0.29%	0	0	72	210.50	0.24%	1	2	199
AsimSingh50_3	694.00	0.00%	0	0	29	694.00	0.00%	0	0	32
AsimSingh50_4	366.69	0.09%	1	0	124	366.69	0.09%	0	0	107
AsimSingh100_1	1005	0.00%	1	0	196	1005	0.00%	0	0	55
AsimSingh100_2	465	0.00%	0	12	57	465	0.00%	1	11	198

Continues on the next page

Table B.9

	y-separation					1-separation				
Instance Name	$\mathcal{V}^{LP}$	gap	t_s	#CC	#RFV	$\mathcal{V}^{LP}$	gap	t_s	#CC	#RFV
AsimSingh100_3	1350	0.00%	0	0	60	1350	0.00%	0	0	60
AsimSingh100_4	705	0.00%	1	1	91	705	0.00%	0	15	113
AsimSingh150_1	1185	0.00%	1	0	80	1185	0.00%	1	3	90
AsimSingh150_2	735	0.00%	1	2	31	735	0.00%	3	42	342
AsimSingh150_3	1770	0.00%	0	0	40	1770	0.00%	0	0	40
AsimSingh150_4	885	0.00%	1	4	54	885	0.00%	0	4	54
AsimSingh200_1	1545	0.00%	2	8	70	1545	0.00%	2	4	75
AsimSingh200_2	1050	0.00%	0	0	12	1050	0.00%	0	0	16
AsimSingh200_3	2400	0.00%	2	0	180	2400	0.00%	4	1	318
AsimSingh200_4	1455	0.00%	2	0	54	1455	0.00%	1	0	54
AsimSingh250_1	2145	0.00%	1	0	60	2145	0.00%	3	0	100
AsimSingh250_2	1200	0.00%	9	6	166	1200	0.00%	11	6	169
AsimSingh250_3	3135	0.00%	10	0	301	3135	0.00%	8	0	223
AsimSingh250_4	2445	0.00%	2	0	80	2445	0.00%	2	0	80
AsimSingh300_1	2595	0.00%	9	0	131	2595	0.00%	19	8	142
AsimSingh300_2	1605	0.00%	9	14	89	1605	0.00%	19	27	177
AsimSingh300_3	3825	0.00%	7	3	171	3825	0.00%	5	1	79
AsimSingh300_4	2655	0.00%	12	4	141	2655	0.00%	13	8	153
$average$		0.02%	3	2	100		0.01%	4	6	137

Appendix C

Heuristic algorithms detailed results

C.1 Genetic algorithm

C.1.1 Results obtained with the GA algorithm

Table C.1 shows the results obtained using the GA algorithm with the parameter set 1 and 2. This table contains the best solution obtained with the GA algorithm (ν^B), the percentage of gap between the solution obtained and the best known upper bound presented in Table 6.1 ($gap = 100 \times (\textit{heuristic solution} - \textit{best upper bound})/\textit{best upper bound}$) and the computational time, in seconds, to obtain the best solution with the GA algorithm (t_s).

The parameter set 1 is characterized by the following setting: $\eta^1 = 20$, $\tau^1 = 4$, $\zeta^1 = 16$, $\epsilon^1 = 0.25$, while the parameter set 2 contains the following parameters: $\eta^2 = 120$, $\tau^2 = 10$, $\zeta^2 = 100$, $\epsilon^1 = 0.50$. The results where obtained performing 10000 iterations.

Table C.1: Results obtained with the GA algorithm.

	Parameter set 1			Parameter set 2		
Instance	ν^B	gap	t_s	ν^B	gap	t_s
a_1	5592.09	195.70%	1	4459.99	135.83%	7
a_2	7097.26	318.11%	1	3078.92	81.38%	6
gr_1	10687.2	488.16%	1	9022.13	396.52%	15
gr_2	9198.29	537.42%	1	5777.71	300.38%	15
gr_3	9235.81	567.24%	1	5794.06	318.59%	15
pr_1	1440790	781.42%	2	872291.00	433.64%	22
pr_2	1514560	731.52%	2	995518.00	446.56%	22
pr_3	1723190	1052.97%	2	941466.00	529.93%	20
$average$		584.07%	1		330.35%	15

C.1.2 Results obtained with the GA+NN algorithm

Table C.2 shows the results obtained with the GA+NN algorithm generating 60 individuals and 120 individuals with the random nearest neighbor in the initial population. This table presents the same information as Table C.1 but for the GA+NN algorithm and for the parameter under testing.

The results were obtained with the parameter set 2 and performing 10000 iterations of the GA+NN algorithm.

Table C.2: Results obtained with the GA+NN algorithm.

	Generating 60 individuals			Generating 120 individuals		
Instance	ν^B	gap	t_s	ν^B	gap	t_s
a_1	2215.54	17.15%	7	2213.03	17.02%	9
a_2	1998.53	17.74%	7	1993.66	17.45%	8
gr_1	2110.39	16.14%	25	2089.74	15.01%	34
gr_2	1622.38	12.43%	23	1619.08	12.20%	32
gr_3	1575.05	13.79%	23	1618.34	16.92%	33
pr_1	171618.00	4.99%	81	170553.00	4.34%	81
pr_2	183986.00	1.01%	89	183830.00	0.93%	89
pr_3	159888.00	6.98%	78	159736.00	6.88%	78
$average$		11.28%	31		11.34%	46

C.1.3 Results obtained with the LS algorithm

Table C.3 shows the results obtained by applying the LS algorithm to the best solution found by the GA+NN algorithm. There are results obtained performing 1000 and 5000 iterations of the LS algorithm. This table contains the same information as Table C.1 but for the LS algorithm and for the parameter under testing.

The GA+NN algorithm was applied with the following setting: the parameter set used was the parameter set 2, 10000 iterations were performed and 60 individuals from the initial population were generated with the random nearest neighbor.

Table C.3: Results obtained with the LS algorithm.

	1000 iterations			5000 iterations		
Instance	ν^B	gap	t_s	ν^B	gap	t_s
a_1	1904.56	0.71%	8	1904.56	0.71%	9
a_2	1778.50	4.77%	8	1778.50	4.77%	8
gr_1	1790.88	-1.44%	26	1790.88	-1.44%	32
gr_2	1476.67	2.33%	24	1476.67	2.33%	28
gr_3	1380.57	-0.26%	23	1380.57	-0.26%	28
pr_1	151889.00	-7.08%	57	151889.00	-7.08%	63
pr_2	164433.00	-9.72%	61	164433.00	-9.72%	69
pr_3	135688.00	-9.21%	55	135688.00	-9.21%	60
$average$		-2.49%	33		-2.49%	37

C.1.4 Results obtained with the LS_random algorithm

Table C.4 shows the results obtained when we apply the LS_random algorithm to the best solution found by the GA+NN algorithm. The referred table shows the results obtained performing 1000 iterations of the LS_random algorithm. This table contains the same information as Table C.1 but for the LS_random algorithm.

The GA+NN algorithm was applied with the following setting: the parameter set used was the parameter set 2, 10000 iterations were performed and 60 individuals from the initial population were generated with the random nearest neighbor.

Table C.4: Results obtained with the LS_random algorithm.

Instance	ν^B	gap	t_s
a_1	1947.08	2.96%	9
a_2	1811.01	6.69%	9
gr_1	1815.22	-0.10%	36
gr_2	1502.66	4.13%	33
gr_3	1411.22	1.95%	32
pr_1	154508.00	-5.48%	72
pr_2	166230.00	-8.74%	78
pr_3	141554.00	-5.29%	67
$average$		-0.48%	42

C.1.5 Results obtained with the LS_insertRemove algorithm

Table C.5 shows the results obtained when we apply the LS_insertRemove algorithm to the best solution found by the GA+NN algorithm. The referred table shows the results obtained performing 1000 and 5000 iterations of the LS_insertRemove algorithm. This table contains the same information as Table C.1 but for the LS_insertRemove algorithm and for the parameter under testing.

The GA+NN algorithm was applied with the following setting: the parameter set used was the parameter set 2, 10000 iterations were performed and 60 individuals from the initial population were generated with the random nearest neighbor.

Table C.5: Results obtained with the LS_insertRemove algorithm.

	1000 iterations			5000 iterations		
Instance	ν^B	gap	t_s	ν^B	gap	t_s
a_1	1829.91	-3.24%	9	1829.91	-3.24%	15
a_2	1705.25	0.46%	9	1705.25	0.46%	17
gr_1	1649.19	-9.24%	40	1649.19	-9.24%	100
gr_2	1328.46	-7.94%	39	1328.29	-7.95%	101
gr_3	1297.59	-6.26%	38	1297.59	-6.26%	98
pr_1	145799.00	-10.81%	92	145799.00	-10.81%	277
pr_2	155087.00	-14.85%	97	155087.00	-14.85%	271
pr_3	129137.00	-13.60%	89	129024.00	-13.67%	250
$average$		-8.19%	52		-8.19%	141

C.1.6 Results obtained by applying the LS_insertRemove algorithm to several solutions

Table C.6 shows the results obtained by applying the LS_insertRemove algorithm to twelve different solutions present in the final population of the GA+NN algorithm. These results were obtained by performing 1000 iterations of the LS_insertRemove algorithm. Table C.6 contains the same information as Table C.1 but for the LS_insertRemove algorithm applied to several solutions.

The GA+NN algorithm was applied with the following setting: the parameter set used was the parameter set 2, 10000 iterations were performed and 60 individuals from the initial population were generated with the random nearest neighbor.

Table C.6: Results obtained by applying the LS_insertRemove algorithm to several solutions.

Instance	ν^B	gap	t_s
a_1	1823.20	-3.59%	26
a_2	1662.88	-2.04%	31
gr_1	1615.72	-11.08%	211
gr_2	1314.29	-8.92%	224
gr_3	1252.53	-9.51%	215
pr_1	141679.00	-13.33%	577
pr_2	151939.00	-16.58%	600
pr_3	129137.00	-13.60%	555
$average$		-9.83%	305

C.2 Iterated local search algorithm

C.2.1 Results obtained with the several choosing criteria of the perturbation method

Table C.7 shows the detailed results obtained using the choosing criteria $Mean$, Min and Max in the perturbation method while Table C.8 shows the results obtained with the choosing criteria $Random\_choice$ and $Least\_chosen$. These tables contain the same information as Table C.1 but for the ILS algorithm considering the several choosing criteria.

The results presented in Tables C.7 and C.8 were obtained by using the removal criterion $Greedy$ in the perturbation procedure and by performing 1000 iterations of the ILS algorithm.

Table C.7: Comparison of the choosing criteria $Mean$, Min and Max.

	Criterion $Mean$			Criterion Min			Criterion Max		
Instance	ν^B	gap	t_s	ν^B	gap	t_s	ν^B	gap	t_s
a_1	1842.22	-2.59%	5	1878.17	-0.69%	23	1857.35	-1.79%	5
a_2	1793.08	5.63%	4	1828.65	7.73%	22	1821.94	7.33%	4
gr_1	1773.97	-2.37%	53	1727.07	-4.95%	296	1769.69	-2.61%	58
gr_2	1368.63	-5.16%	51	1314.09	-8.94%	295	1389.77	-3.69%	61
gr_3	1416.86	2.36%	47	1396.45	0.89%	298	1447.73	4.59%	49
pr_1	148721.33	-9.02%	188	148206.79	-9.33%	1130	149540.20	-8.52%	206
pr_2	154039.92	-15.43%	225	152822.84	-16.10%	1135	153805.55	-15.56%	222
pr_3	144288.58	-3.46%	207	138658.71	-7.22%	1134	142751.74	-4.49%	233
$average$		-3.75%	98		-4.83%	542		-3.09%	105

Table C.8: Comparison of the choosing criteria $Random\_choice$ and $Least\_chosen$.

	Criterion $Random$			Criterion $Least\_chosen$		
Instance	ν^B	gap	t_s	ν^B	gap	t_s
a_1	1799.07	-4.87%	7	1791.27	-5.28%	8
a_2	1690.12	-0.43%	7	1654.62	-2.52%	7
gr_1	1643.99	-9.52%	94	1628.81	-10.36%	99
gr_2	1266.80	-12.21%	101	1276.95	-11.51%	106
gr_3	1298.08	-6.22%	106	1299.39	-6.13%	106
pr_1	139740.57	-14.51%	356	139228.33	-14.83%	348
pr_2	146234.26	-19.72%	348	149680.25	-17.82%	338
pr_3	134001.99	-10.34%	398	135351.27	-9.44%	363
$average$		-9.73%	177		-9.74%	172

C.2.2 Results obtained with the removal criterion $Random\_removal$ in the perturbation method

Table C.9 shows the detailed results obtained using as removal criterion in the perturbation method the criterion $Random\_removal$. We experimented considering as choosing criterion a combination of both criteria $Random\_choice$ and $Least\_chosen$. The results were obtained performing 1000 iterations of the ILS algorithm. Table C.9 contains the same information as Table C.1 but for the ILS algorithm considering the removal criterion $Random\_removal$.

The results were obtained performing 1000 iterations of the ILS algorithm.

Table C.9: Results obtained with the removal criterion $Random\_removal$.

	Criterion $Random\_choice$			Criterion $Least\_chosen$		
Instance	ν^B	gap	t_s	ν^B	gap	t_s
a_1	1897.33	0.33%	6	1897.33	0.33%	6
a_2	1790.61	5.49%	5	1760.77	3.73%	6
gr_1	1769.16	-2.64%	96	1812.48	-0.25%	89
gr_2	1453.82	0.75%	94	1453.82	0.75%	88
gr_3	1447.73	4.59%	94	1447.73	4.59%	91
pr_1	152731.61	-6.56%	252	152731.61	-6.56%	290
pr_2	157153.10	-13.72%	242	157153.10	-13.72%	299
pr_3	142850.50	-4.42%	272	144669.54	-3.20%	311
$average$		-2.02%	126		-1.79%	148

C.2.3 Results obtained combining both removal criteria in the perturbation method

Tables C.10, C.11, C.12 and C.13 show the detailed results obtained using as removal criterion the combination of both removal criteria. The removal criterion $Greedy$ is applied, except in every ρ iterations where the criterion used is the criterion $Random\_removal$. Tables C.10 and C.11 use the choosing criterion $Random\_choice$ while tables C.12 and C.13 use the choosing criterion $Least\_chosen$. These tables contain the same information as Table C.1 but for the ILS algorithm considering the several ρ values.

The results were obtained performing 1000 iterations of the ILS algorithm.

Table C.10: Results obtained with the choosing criterion $Random\_choice$ and the combination of both removal criteria with $\rho = 200$, $\rho = 100$ and $\rho = 50$.

	$\rho = 200$			$\rho = 100$			$\rho = 50$		
Instance	ν^B	gap	t_s	ν^B	gap	t_s	ν^B	gap	t_s
a_1	1798.82	-4.88%	8	1784.98	-5.61%	7	1801.04	-4.77%	7
a_2	1650.52	-2.77%	7	1688.36	-0.54%	7	1646.50	-3.00%	7
gr_1	1628.54	-10.38%	93	1626.48	-10.49%	93	1601.93	-11.84%	95
gr_2	1266.39	-12.24%	87	1264.77	-12.35%	89	1271.43	-11.89%	95
gr_3	1295.33	-6.42%	91	1296.50	-6.33%	99	1292.73	-6.61%	99
pr_1	139740.57	-14.51%	291	139212.93	-14.83%	327	138295.20	-15.40%	335
pr_2	145501.06	-20.12%	284	147907.01	-18.80%	328	146533.45	-19.55%	317
pr_3	133953.44	-10.37%	321	136161.33	-8.90%	370	132662.93	-11.24%	367
$average$		-10.21%	148		-9.73%	165		-10.58%	165

Table C.11: Results obtained with the choosing criterion $Random\_choice$ and the combination of both removal criteria with $\rho = 25$, $\rho = 10$ and $\rho = 5$.

	$\rho = 25$			$\rho = 10$			$\rho = 5$		
Instance	ν^B	gap	t_s	ν^B	gap	t_s	ν^B	gap	t_s
a_1	1783.89	-5.67%	7	1774.92	-6.15%	7	1795.37	-5.07%	11
a_2	1632.04	-3.86%	7	1627.51	-4.12%	7	1639.97	-3.39&	11
gr_1	1608.39	-11.48%	96	1607.67	-11.52%	94	1580.58	-13.01%	94
gr_2	1266.25	-12.25%	92	1261.37	-12.59%	94	1271.37	-11.90%	91
gr_3	1289.04	-6.87%	97	1306.78	-5.59%	97	1315.09	-4.99%	96
pr_1	138232.66	-15.43%	339	137055.01	-16.15%	322	139735.04	-14.52%	317
pr_2	146323.81	-19.67%	323	146653.61	-19.48%	320	146201.62	-19.73%	339
pr_3	131444.90	-12.05%	364	132056.01	-11.64%	365	131331.97	-12.13%	375
$average$		-10.91%	166		-10.91%	163		-10.59%	242

Table C.12: Results obtained with the choosing criterion $Least\_chosen$ and the combination of both removal criteria with $\rho = 200$, $\rho = 100$ and $\rho = 50$.

	$\rho = 200$			$\rho = 100$			$\rho = 50$		
Instance	ν^B	gap	t_s	ν^B	gap	t_s	ν^B	gap	t_s
a_1	1786.33	-5.54%	7	1792.52	-5.22%	8	1789.06	-5.40%	8
a_2	1648.81	-2.87%	7	1654.72	-2.52%	8	1651.27	-2.72%	7
gr_1	1626.21	-10.50%	94	1628.89	-10.36%	101	1619.61	-10.87%	90
gr_2	1264.70	-12.36%	100	1266.59	-12.23%	103	1263.79	-12.42%	101
gr_3	1297.96	-6.23%	157	1301.41	-5.98%	109	1296.00	-6.37%	97
pr_1	140041.50	-14.33%	342	141870.34	-13.21%	367	139128.57	-14.89%	326
pr_2	148050.44	-18.72%	303	149161.05	-18.11%	368	149044.17	-18.17%	319
pr_3	134291.54	-10.15%	356	136250.02	-8.84%	419	133980.59	-10.35%	404
$average$		-10.09%	164		-9.56%	185		-10.15%	169

Table C.13: Results obtained with choosing criterion $Least\_chosen$ and the combination of both removal criteria with $\rho = 25$, $\rho = 10$ and $\rho = 5$.

	$\rho = 25$			$\rho = 10$			$\rho = 5$		
Instance	ν^B	gap	t_s	ν^B	gap	t_s	ν^B	gap	t_s
a_1	1792.51	-5.22%	8	1786.90	-5.90%	7	1786.41	-5.65%	8
a_2	1644.00	-3.15%	7	1619.51	-4.59%	7	1641.48	-3.30%	8
gr_1	1605.86	-11.62%	99	1605.29	-11.65%	94	1606.09	-11.61%	104
gr_2	1269.61	-12.02%	103	1266.43	-12.24%	95	1269.21	-12.05%	102
gr_3	1295.80	-6.39%	107	1247.16	-9.90%	103	1272.17	-8.09%	105
pr_1	138700.66	-15.15%	324	138414.81	-15.32%	338	139262.28	-14.80%	350
pr_2	149298.98	-18.03%	286	146312.92	-19.67%	344	145713.54	-20.00%	350
pr_3	133180.85	-10.89%	322	131329.10	-12.13%	423	134516.68	-10.00%	381
$average$		-10.29%	157		-11.38%	176		-10.67%	176

C.2.4 Results obtained with different numbers of iterations in the ILS algorithm

Table C.14 shows the detailed results obtained performing 5000 iterations of the ILS algorithm. The results were obtained using the choosing criterion $Least\_chosen$ and the removal criterion adopted was the combination of both removal criteria, that is, the removal criterion used is the $Greedy$ criterion but once in every $\rho = 10$ iterations we apply the removal criterion $Random\_removal$. This table contains the same information as Table C.1 but for the ILS algorithm.

Table C.14: Results obtained performing 5000 iterations of the ILS algorithm.

Instance	ν^B	gap	t_s
a_1	1783.45	-5.70%	33
a_2	1587.90	-6.46%	33
gr_1	1570.69	-13.56%	443
gr_2	1254.25	-13.08%	452
gr_3	1236.94	-10.64%	465
pr_1	138415.00	-15.32%	1430
pr_2	144621.00	-20.60%	1419
pr_3	131156.00	-12.24%	1603
$average$		-12.20%	735

C.3 Hybrid algorithm

C.3.1 Evaluating the number of visits by using instance $bier$

Table C.15 shows the optimal values and the respective computational times of the variations of instance $bier$ presented in Section 6.3. This table shows the optimal value ($\mathcal{V}$), the computational time, in seconds, to obtain the optimal value with the y-separation (t_s), the number of B&C subproblems solved during the B&C algorithm (#sub) and the number of added violated CC inequalities (#CC) and RFV inequalities (#RFV).

Table C.15: Optimal results for the variations of instance $bier$.

Instance	$\mathcal{V}$	t_s	#sub	#CC	#RFV
$bier^1$	11434.90	147	6	2418	509
$bier^2$	15215.60	33	4	1188	856
$bier^3$	23757.30	252	58	3050	2720
$bier^4$	34636.77	95	41	2521	1765
$bier^5$	41777.20	306	357	4098	6468
$bier^6$	53799.80	45	82	1518	2843
$bier^7$	66322.10	78	548	970	4607
$bier^8$	81382.70	38	432	424	2816
$bier^9$	100433.00	7	35	36	639
$bier^{10}$	118294.00	5	83	3	442
$average$		101	165	1623	2367

C.3.2 Results obtained with the several r^* values

Table C.16 shows the detailed results obtained when testing the parameter r^*. This table contains the same information as Table C.1 but for the constructive phase of the hybrid algorithm and for the parameter under testing.

These results were obtained by using the parameter setting: $\Delta = 180$ and $\Lambda = 70$.

Table C.16: Results obtained with the several r^* values.

	$r^* = 0.70$			$r^* = 0.80$			$r^* = 0.90$		
Instance	ν^B	gap	t_s	ν^B	gap	t_s	ν^B	gap	t_s
a_1	1807.57	-4.42%	19	1838.46	-2.79%	3	2075.80	9.76%	0
a_2	1679.31	-1.07%	4	1731.56	2.01%	0	1761.49	3.77%	0
gr_1	1638.06	-9.85%	46	1788.10	-1.59%	6	1868.98	2.86%	0
gr_2	1441.70	-0.09%	59	1553.02	7.62%	16	1629.61	12.93%	1
gr_3	1339.34	-3.24%	144	1437.48	3.85%	15	1468.84	6.12%	0
pr_1	146935.16	-10.11%	9192	151591.47	-7.26%	408	167540.71	2.50%	2
pr_2	151433.45	-16.86%	6329	158160.59	-13.17%	74	163720.66	-10.11%	3
pr_3	132834.38	-11.12%	112	140399.08	-6.06%	10	143568.22	-3.94%	1
$average$		-7.10%	1988		-2.17%	67		2.98%	1

C.3.3 Results obtained with the several Δ values

Table C.17 and show the detailed results obtained considering the values of Δ of 140, 160 and 200. This table contains the same information as Table C.1 but for the constructive phase of the hybrid algorithm and for the parameter under testing. Note that the detailed results obtained with $\Delta = 180$ are available in Table C.16.

The results of Table C.17 were obtained by using the parameter setting: $r^* = 0.80$ and $\Lambda = 70$.

Table C.17: Results obtained with $\Delta = 140$, $\Delta = 160$ and $\Delta = 200$ values.

	$\Delta = 140$			$\Delta = 160$			$\Delta = 200$		
Instance	ν^B	*gap*	t_s	ν^B	*gap*	t_s	ν^B	*gap*	t_s
a_1	1838.46	-2.79%	3	1838.46	-2.79%	3	1838.46	-2.79%	4
a_2	1731.56	2.01%	0	1731.56	2.01%	0	1731.56	2.01%	0
gr_1	1788.10	-1.59%	6	1788.10	-1.59%	7	1788.10	-1.59%	7
gr_2	1553.02	7.62%	16	1553.02	7.62%	17	1553.02	7.62%	17
gr_3	1437.48	3.85%	15	1437.48	3.85%	16	1437.48	3.85%	16
pr_1	154814.76	-5.29%	29	151591.47	-7.26%	426	152529.95	-6.69%	528
pr_2	161633.63	-11.26%	199	161328.90	-11.43%	16782	163419.02	-10.28%	2514
pr_3	140399.08	-6.06%	11	140399.08	-6.06%	13	140399.08	-6.06%	13
average		-1.69%	35		-1.96%	2158		-1.74%	387

C.3.4 Results obtained with the several Λ values

Table C.18 shows the detailed results obtained considering the values of Λ of 30, 50 and 90. This table contains the same information as Table C.1 but for the constructive phase of the hybrid algorithm and for the parameter under testing. Note that the detailed results obtained with $\Lambda = 180$ are available in Table C.16.

The results of Table C.17 were obtained by using the parameter setting: $r^* = 0.80$ and $\Delta = 180$.

Table C.18: Results obtained with $\Lambda = 30$, $\Lambda = 50$ and $\Lambda = 90$ values.

	$\Lambda = 30$			$\Lambda = 50$			$\Lambda = 90$		
Instance	ν^B	gap	t_s	ν^B	gap	t_s	ν^B	gap	t_s
a_1	1838.46	-2.79%	4	1838.46	-2.79%	4	1838.46	-2.79%	4
a_2	1731.56	2.01%	0	1731.56	2.01%	0	1731.56	2.01%	0
gr_1	1788.10	-1.59%	7	1788.10	-1.59%	6	1788.10	-1.59%	7
gr_2	1553.02	7.62%	17	1553.02	7.62%	17	1553.02	7.62%	17
gr_3	1437.48	3.85%	17	1437.48	3.85%	16	1437.48	3.85%	16
pr_1	151591.47	-7.26%	421	151591.47	-7.26%	437	151591.47	-7.26%	421
pr_2	156104.63	-14.30%	59	156104.63	-14.30%	62	158160.59	-13.17%	77
pr_3	140399.08	-6.06%	12	140399.08	-6.06%	12	140399.08	-6.06%	12
$average$		-2.31%	67		-2.31%	69		-2.17%	69

C.3.5 Results obtained with the hybrid algorithm

Table C.19 shows the detailed results obtained with the hybrid algorithm, that is, with the ILS algorithm applied to the feasible FTSP solution obtained in the constructive phase, performing 1000 and 5000 iterations of the ILS algorithm. This table contains the same information as Table C.1 but for the hybrid algorithm.

These results were obtained with the following parameter setting: $r^* = 0.80$, $\Delta = 180$ and $\Lambda = 30$.

Table C.19: Results obtained with the hybrid algorithm.

	1000 iterations			5000 iterations		
Instance	ν^B	gap	t_s	ν^B	gap	t_s
a_1	1780.21	-5.87%	12	1749.99	-7.46%	36
a_2	1568.56	-7.59%	9	1568.56	-7.59%	33
gr_1	1571.19	-13.53%	108	1551.27	-14.63%	467
gr_2	1279.30	-11.35%	116	1252.33	-13.22%	492
gr_3	1259.08	-9.04%	124	1249.70	-9.72%	485
pr_1	137099.62	-16.13%	782	136092.29	-16.74%	2072
pr_2	147572.96	-18.98%	406	147572.96	-18.98%	1787
pr_3	129624.14	-13.27%	398	128770.54	-13.84%	1858
$average$		-11.97%	244		-12.77%	904

C.3.6 Results obtained with the hybrid algorithm with dual information

Table C.20 shows the detailed results obtained with the hybrid algorithm with the dual information. This table contains the same information as Table C.1 but for the hybrid algorithm.

These results were obtained performing 1000 iterations of the ILS algorithm and with the following parameter setting: $r^* = 0.80$, $\Delta = 180$ and $\Lambda = 30$.

Table C.20: Results obtained with the hybrid algorithm with dual information.

	Taboo first local search			Taboo first 100 iterations		
Instance	ν^B	gap	t_s	ν^B	gap	t_s
a_1	1773.71	-6.21%	13	1806.52	-4.48%	14
a_2	1577.17	-7.09%	7	1609.13	-5.20%	9
gr_1	1571.19	-13.53%	103	1571.19	-13.53%	102
gr_2	1279.30	-11.35%	125	1279.30	-11.35%	116
gr_3	1259.08	-9.04%	135	1259.08	-9.04%	116
pr_1	137993.90	-15.58%	831	138868.00	-15.05%	773
pr_2	147572.96	-18.98%	693	147572.96	-18.98%	749
pr_3	129624.14	-13.27%	356	129624.14	-13.27%	386
$average$		-11.88%	283		-11.36%	283

C.4 Final results with different seeds

Table C.21 contains the results obtained with the ILS algorithm and the hybrid algorithm using the best parameter setting and doing five runs for each instance with unknown optimal value. Besides the instance's name and the seed used, Table C.21 contains the same information as Table C.1 but for the ILS algorithm and for the hybrid algorithm.

These results were obtained performing 5000 iterations of each method. In the ILS algorithm the choosing criterion in the criterion $Least\_chosen$ and the removal criterion is a combination of the removal criterion $Greedy$ and $Random\_removal$, that is, we apply the criterion $Greedy$ but once in every $\rho = 10$ iterations the criterion $Random\_removal$ is applied. The results with the hybrid algorithm were obtained with the following parameter setting: $r^* = 0.80$, $\Delta = 180$ and $\Lambda = 30$.

Table C.21: Results obtained in five runs for the instances with unknown optimal value.

		ILS algorithm			Hybrid algorithm		
Instance	Seed	ν^B	gap	t_s	ν^B	gap	t_s
a_1	1	1799.24	-4.86%	35	1741.14	-7.93%	35
	2	1778.44	-5.96%	33	1769.33	-6.44%	36
	3	1783.45	-5.70%	33	1749.99	-7.46%	36
	4	1771.25	-6.34%	33	1772.71	-6.26%	39
	5	1778.89	-5.94%	33	1777.64	-6.00%	36
a_2	1	1606.37	-5.37%	34	1585.15	-6.62%	32
	2	1623.17	-4.38%	33	1580.75	-6.88%	34
	3	1587.90	-6.46%	33	1568.56	-7.59%	32
	4	1573.27	-7.32%	33	1582.63	-6.77%	33
	5	1619.43	-4.60%	33	1606.33	-5.37%	33
gr_1	1	1559.59	-14.17%	490	1556.70	-14.33%	447
	2	1554.80	-14.43%	444	1576.61	-13.23%	465
	3	1570.69	-13.56%	443	1551.27	-14.63%	460
	4	1565.76	-13.83%	441	1583.01	-12.88%	467
	5	1569.28	-13.64%	442	1597.92	-12.06%	473
gr_2	1	1259.92	-12.69%	497	1236.96	-14.28%	466
	2	1246.21	-13.64%	453	1245.98	-13.66%	486
	3	1254.25	-13.08%	452	1252.33	-13.22%	487
	4	1249.24	-13.43%	451	1245.59	-13.68%	474
	5	1248.27	-13.50%	450	1246.42	-13.63%	493
gr_3	1	1248.74	-9.78%	533	1246.64	-9.94%	477
	2	1227.50	-11.32%	465	1240.23	-10.40%	496
	3	1236.94	-10.64%	465	1249.70	-9.72%	509
	4	1240.73	-10.36%	464	1242.65	-10.22%	493
	5	1234.49	-10.81%	465	1248.71	-9.79%	508
pr_1	1	136231.00	-16.66%	1442	136937.58	-16.23%	1870
	2	134283.00	-17.85%	1442	136539.42	-16.47%	1970
	3	138415.00	-15.32%	1430	136092.29	-16.74%	1953
	4	134353.00	-17.81%	1432	138873.29	-15.04%	1986
	5	135201.00	-17.29%	1435	135848.93	-16.89%	2452
pr_2	1	146383.00	-19.63%	1414	147180.43	-19.20%	1642
	2	147105.00	-19.24%	1414	146596.93	-19.52%	1608

Continues on the next page

Table C.21

		ILS algorithm			Hybrid algorithm		
Instance	Seed	ν^B	gap	t_s	ν^B	gap	t_s
	3	144621.00	-20.60%	1419	147572.96	-18.98%	1509
	4	145503.00	-20.12%	1420	148017.03	-18.74%	1650
	5	145894.00	-19.90%	1416	147745.14	-18.89%	1694
pr_3	1	130854.00	-12.45%	1608	123164.62	-17.59%	1824
	2	128991.00	-13.69%	1607	128138.67	-14.26%	2161
	3	131156.00	-12.24%	1603	128770.54	-13.84%	1671
	4	128562.00	-13.98%	1607	130306.39	-12.81%	1920
	5	131980.00	-11.69%	1613	128127.89	-14.27%	1857
pr144_4	1	49775	0.75%	4	49650	0.50%	4
	2	49918	1.04%	4	49650	0.50%	4
	3	49706	0.61%	4	49651	0.50%	4
	4	49527	0.25%	4	49651	0.50%	4
	5	49568	0.33%	4	49839	0.88%	4
kroA150_3	1	21645	3.66%	4	21038	0.76%	5
	2	21662	3.75%	4	21038	0.76%	5
	3	21645	3.66%	4	21038	0.76%	5
	4	21645	3.66%	4	21038	0.76%	5
	5	21645	3.66%	4	21038	0.76%	5
pr152_3	1	65621	1.86%	5	64921	0.77%	5
	2	65621	1.86%	5	64758	0.52%	5
	3	65578	1.79%	5	64960	0.83%	5
	4	65461	1.61%	5	64718	0.45%	6
	5	65339	1.42%	5	64758	0.52%	6
rat195_1	1	1310	1.95%	12	1314	2.26%	12
	2	1300	1.17%	12	1323	2.96%	12
	3	1327	3.27%	12	1317	2.49%	12
	4	1317	2.49%	12	1314	2.26%	12
	5	1316	2.41%	12	1320	2.72%	12
rat195_3	1	1850	1.98%	9	1840	1.43%	12
	2	1854	2.21%	9	1845	1.71%	11
	3	1849	1.93%	9	1840	1.43%	12
	4	1841	1.49%	9	1860	2.54%	12

Continues on the next page

Table C.21

		ILS algorithm			Hybrid algorithm		
Instance	Seed	ν^B	gap	t_s	ν^B	gap	t_s
	5	1845	1.71%	9	1840	1.43%	12
rat195_4	1	1374	4.09%	10	1331	0.83%	10
	2	1373	4.02%	10	1333	0.98%	10
	3	1374	4.09%	10	1336	1.21%	11
	4	1373	4.02%	10	1334	1.06%	11
	5	1359	2.95%	10	1335	1.14%	11
kroA200_1	1	17318	5.33%	12	17223	4.76%	12
	2	17647	7.34%	12	17331	5.41%	12
	3	17540	6.68%	12	17321	5.35%	12
	4	17380	5.71%	12	17099	4.00%	12
	5	17760	8.02%	12	17159	4.37%	12
kroA200_3	1	25590	4.57%	9	24580	0.45%	6886
	2	25592	4.58%	9	24603	0.54%	4665
	3	25586	4.56%	9	24580	0.45%	4643
	4	25621	4.70%	9	24580	0.45%	4649
	5	25487	4.15%	9	24580	0.45%	4916
kroB200_3	1	25410	5.49%	8	24542	1.88%	190
	2	25437	5.60%	8	24542	1.88%	207
	3	25600	6.28%	8	24542	1.88%	182
	4	25498	5.85%	8	24542	1.88%	202
	5	25440	5.61%	8	24542	1.88%	199
gr202_4	1	29425	4.94%	11	28290	0.90%	12
	2	29447	5.02%	11	28339	1.07%	12
	3	29374	4.76%	11	28504	1.66%	12
	4	29186	4.09%	11	28117	0.28%	13
	5	28766	2.59%	11	28318	1.00%	12
gr229_1	1	72238	2.12%	19	71667	1.31%	19
	2	72456	2.42%	19	72455	2.42%	20
	3	72477	2.45%	19	71796	1.49%	19
	4	72603	2.63%	19	71752	1.43%	20
	5	72388	2.33%	19	72329	2.24%	19
gil262_1	1	1567	2.49%	23	1577	3.14%	24

Continues on the next page

Table C.21

		ILS algorithm			Hybrid algorithm		
Instance	Seed	ν^B	gap	t_s	ν^B	gap	t_s
	2	1610	5.30%	24	1577	3.14%	24
	3	1592	4.12%	23	1577	3.14%	24
	4	1590	3.99%	23	1577	3.14%	24
	5	1589	3.92%	23	1577	3.14%	25
gil262_3	1	2069	2.83%	16	2000	-0.60%	8886
	2	2073	3.03%	16	2006	-0.30%	7833
	3	2071	2.93%	16	1987	-1.24%	7851
	4	2072	2.98%	16	2000	-0.60%	7846
	5	2083	3.53%	16	2000	-0.60%	8562
gil262_4	1	1753	-1.13%	20	1712	-3.44%	89
	2	1738	-1.97%	20	1723	-2.82%	63
	3	1746	-1.52%	20	1722	-2.88%	62
	4	1737	-2.03%	20	1750	-1.30%	68
	5	1754	-1.07%	20	1724	-2.76%	38
pr264_2	1	29032	0.99%	25	28865	0.41%	37
	2	29036	1.00%	25	29049	1.05%	26
	3	28918	0.59%	25	28863	0.40%	25
	4	28892	0.50%	25	29097	1.21%	26
	5	29019	0.94%	25	29019	0.94%	26
rbg443_2	1	418	-29.87%	130	415	-30.37%	130
	2	400	-32.89%	131	399	-33.05%	133
	3	432	-27.52%	131	399	-33.05%	131
	4	421	-29.36%	131	424	-28.86%	133
	5	421	-29.36%	130	412	-30.87%	131

Appendix D

RFTSP detailed results

D.1 Linear programming relaxation results obtained with the adapted formulations for the RFTSP

Tables D.1 and D.2 show the LP relaxation value obtained with the adapted formulations, namely the FCF model, the CC model, the RFV model and the CC+RFV model, for the instances from the instance set 1 and 3, respectively. Each table is divided into four parts, each part devoted to a different formulation. Each part shows the LP relaxation value ($\mathcal{V}^{LP}$), the percentage of gap between the LP relaxation value and the optimal value ($gap = 100\times$ *(optimal value - LP relaxation value)/optimal value*) and the computational time, in seconds, to obtain the LP relaxation value (t_s). The tables also contain, in the last row, the average of the results obtained.

Table D.1: Linear programming relaxation results obtained with adapted formulations for instance set 1.

	FCF			CC			RFV			CC+RFV		
Instance	$\mathcal{V}^{LP}$	gap	t_s	$\mathcal{V}^{LP}$	gap	t_s	$\mathcal{V}^{LP}$	gap	t_s	$\mathcal{V}^{LP}$	gap	t_s
burma_1	12.37	15.89%	0	12.35	16.02%	0	14.70	0.00%	1	14.70	0.00%	0
burma_2	26.78	0.00%	0	26.78	0.00%	0	26.78	0.00%	0	26.78	0.00%	0
burma_3	10.22	13.98%	0	9.93	16.47%	0	11.89	0.00%	0	11.89	0.00%	0
bayg_1	7028.89	3.84%	0	7300.47	0.12%	0	7262.22	0.64%	0	7309.21	0.00%	0
bayg_2	5773.85	24.41%	0	7483.84	2.02%	1	7638.47	0.00%	0	7638.47	0.00%	1
bayg_3	7066.07	13.51%	0	8101.96	0.83%	0	8157.82	0.15%	0	8157.82	0.15%	0
att_1	40021.10	17.03%	2	45623.80	5.42%	0	45561.10	5.55%	5	46876.00	2.91%	0
att_2	25198.00	12.51%	2	28554.70	0.85%	1	28800.10	0.00%	53	28800.10	0.00%	0
att_3	11122.70	7.26%	1	11988.40	0.04%	0	11990.50	0.02%	39	11992.80	0.00%	1
$average$		12.05%	1		4.64%	0		0.71%	11		0.34%	0

Table D.2: Linear programming relaxation results obtained with adapted formulations for instance set 3.

	FCF			CC			RFV			CC+RFV		
Instance	$\mathcal{V}^{LP}$	gap	t_s	$\mathcal{V}^{LP}$	gap	t_s	$\mathcal{V}^{LP}$	gap	t_s	$\mathcal{V}^{LP}$	gap	t_s
br17_1	36	0.00%	1	36	0.00%	1	36	0.00%	0	36	0.00%	0
br17_2	28	0.00%	0	28	0.00%	0	28	0.00%	0	28	0.00%	0
br17_3	58.83	24.57%	0	76.33	2.14%	0	78	0.00%	0	78	0.00%	0
br17_4	58.00	17.14%	0	70	0.00%	1	70	0.00%	0	70	0.00%	0
ftv33_1	910.14	5.49%	0	953.25	1.01%	0	953.50	0.99%	0	953.50	1.00%	0
ftv33_2	416.50	6.19%	1	411.50	7.32%	0	442.00	0.45%	1	442	0.45%	0
ftv33_3	1654.88	8.27%	0	1778.25	1.43%	0	1778.25	1.43%	0	1778.25	1.45%	0
ftv33_4	916.50	4.13%	0	956	0.00%	0	956	0.00%	0	956	0.00%	1
ftv35_1	1027.33	7.53%	1	1084.30	2.40%	0	1111	0.00%	0	1111	0.00%	0
ftv35_2	507.63	9.35%	0	537.50	4.02%	1	560	0.00%	0	560	0.00%	0
ftv35_3	1332.50	6.43%	0	1423.33	0.05%	0	1424	0.00%	0	1424	0.00%	0
ftv35_4	1012.25	4.32%	1	1058	0.00%	1	1058	0.00%	0	1058	0.00%	0
$average$		7.79%	0		1.53%	0		0.45%	5		0.24%	0

D.2 Linear programming relaxation results obtained with the y-separation for the RFTSP

Tables D.3 and D.4 show the LP relaxation results obtained with the P-CC+RV model for the instances from the instance sets 1 and 3, respectively. Each table contains the LP relaxation value ($\mathcal{V}^{LP}$), the percentage of gap between the LP relaxation value and the optimal value (gap) and the computational time, in seconds, to obtain the LP relaxation value (t_s). The tables also contain, in the last row, the average of the results obtained.

Table D.3: Linear programming relaxation results obtained with the P-CC+RV model for the instance set 1.

Instance	$\mathcal{V}^{LP}$	gap	t_s
burma_1	14.70	0.00%	0
burma_2	26.78	0.00%	0
burma_3	11.89	0.00%	0
bayg_1	7309.21	0.00%	0
bayg_2	7638.47	0.00%	1
bayg_3	8157.82	0.15%	0
att_1	46937.70	2.70%	4
att_2	28800.10	0.00%	1
att_3	11992.80	0.00%	1
$average$		0.32%	1

Table D.4: Linear programming relaxation results obtained with the P-CC+RV model for the instance set 3.

Instance	$\mathcal{V}^{LP}$	gap	t_s
br17_1	36	0.00%	0
br17_2	28	0.00%	1
br17_3	78	0.00%	1
br17_4	70	0.00%	0
ftv33_1	956.50	0.67%	0
ftv33_2	442.00	0.45%	0
ftv33_3	1801.00	0.17%	1
ftv33_4	956	0.00%	0
ftv35_1	1111	0.00%	0
ftv35_2	560	0.00%	0
ftv35_3	1424	0.00%	1
ftv35_4	1058	0.00%	0
$average$		0.11%	0

D.3 Linear programming relaxation results obtained with y-separation for the RFTSP

Tables D.5 and D.6 show the LP relaxation results obtained with CC+RFV model using the y-separation for the instances from the instance sets 1 and 3, respectively. Each table contains the LP relaxation value ($\mathcal{V}^{LP}$), the percentage of gap between the LP relaxation value and the optimal value (gap) and the computational time, in seconds, to obtain the LP relaxation value (t_s). The tables also contain, in the last row, the average of the results obtained.

Table D.5: Linear programming relaxation results obtained with the y-separation for the instance set 1.

Instance	$\mathcal{V}^{LP}$	gap	t_s
burma_1	14.70	0.00%	0
burma_2	26.78	0.00%	0
burma_3	11.89	0.00%	0
bayg_1	7309.21	0.00%	0
bayg_2	7638.47	0.00%	1
bayg_3	8157.82	0.15%	0
att_1	46756.10	3.07%	0
att_2	28800.10	0.00%	0
att_3	11992.80	0.00%	1
$average$		0.36%	1

Table D.6: Linear programming relaxation results obtained with the y-separation for the instance set 3.

Instance	$\mathcal{V}^{LP}$	gap	t_s
br17_1	36	0.00%	0
br17_2	28	0.00%	1
br17_3	78	0.00%	1
br17_4	70	0.00%	0
ftv33_1	953.50	0.99%	0
ftv33_2	415.50	6.42%	0
ftv33_3	1778.25	1.43%	0
ftv33_4	956	0.00%	1
ftv35_1	1087.20	2.14%	0
ftv35_2	560	0.00%	0
ftv35_3	1423.33	0.05%	0
ftv35_4	1058	0.00%	0
$average$		0.92%	0

D.4 Final heuristic results with different seeds for the RFTSP

Table D.7 contains the results obtained with the ILS algorithm and the hybrid algorithm using the best parameter setting and doing five runs for each instance with unknown optimal value. The referred table contains the instance's name, the seed used, the value of the solution obtained (ν^B), the gap between the solution obtained and the reference solution ($gap = 100 \times ($*heuristic solution* $-$ *reference solution*$)/$*reference solution*) and the computational time, in seconds, to obtain the solution presented (t_s).

These results were obtained performing 5000 iterations of each method. In the ILS algorithm the choosing criterion is the criterion $Least\_chosen$ and the removal criterion is a combination of the removal criterion $Greedy$ and $Random\_removal$, that is, we apply the criterion $Greedy$ but once in every $\rho = 10$ iterations the criterion $Random\_removal$ is applied. The results with the hybrid algorithm were obtained with the following parameter setting: $r^* = 0.80$, $\Delta = 180$ and $\Lambda = 30$.

Table D.7: Results obtained in five runs for the instances with unknown optimal value for the RFTSP.

		ILS algorithm			Hybrid algorithm		
Instance	Seed	ν^B	gap	t_s	ν^B	gap	t_s
a_1	1	1693.35	2.32%	33	1676.60	1.31%	33
	2	1692.82	2.29%	33	1664.49	0.58%	32
	3	1662.03	0.43%	33	1673.99	1.15%	35
	4	1698.48	2.63%	32	1680.20	1.53%	31
	5	1662.30	0.44%	32	1662.48	0.45%	31
a_2	1	1652.80	10.36%	29	1521.15	1.57%	30
	2	1650.57	10.21%	29	1520.93	1.55%	28
	3	1644.12	9.78%	29	1518.09	1.36%	32
	4	1649.93	10.16%	28	1520.03	1.49%	29
	5	1647.74	10.02%	29	1519.42	1.45%	28
gr_1	1	1902.11	3.25%	456	1851.65	0.51%	470
	2	1893.18	2.76%	431	1861.05	1.02%	452
	3	1892.55	2.73%	428	1842.76	0.03%	489
	4	1897.39	2.99%	429	1864.82	1.23%	442
	5	1913.58	3.87%	430	1842.25	0.00%	423

Continues on the next page

Table D.7

		ILS algorithm			Hybrid algorithm		
Instance	Seed	ν^B	gap	t_s	ν^B	gap	t_s
gr_2	1	1352.94	0.80%	450	1347.04	0.36%	457
	2	1358.60	1.22%	438	1347.49	0.40%	458
	3	1346.30	0.31%	437	1342.17	0.00%	460
	4	1344.53	0.18%	437	1348.56	0.48%	468
	5	1349.10	0.52%	437	1357.40	1.13%	438
gr_3	1	1349.55	1.68%	481	1327.26	0.00%	522
	2	1347.74	1.54%	454	1340.02	0.96%	506
	3	1338.52	0.85%	454	1346.17	1.42%	514
	4	1381.97	4.12%	454	1339.85	0.95%	498
	5	1345.75	1.39%	453	1350.65	1.76%	491
pr_1	1	159542	0.74%	1550	158805.14	0.28%	1424
	2	158433	0.04%	1403	159190.03	0.52%	1409
	3	159233	0.55%	1407	158467.60	0.07%	1399
	4	159963	1.01%	1407	158362.92	0.00%	1399
	5	160933	1.62%	1428	159216.97	0.54%	1397
pr_2	1	183236	4.06%	1514	179713.68	2.06%	1369
	2	178497	1.37%	1398	176086.63	0.00%	1373
	3	179516	1.95%	1400	179130.72	1.73%	1367
	4	176788	0.40%	1405	183864.31	4.42%	1363
	5	181719	3.20%	1400	181448.97	3.05%	1367
pr_3	1	134683	4.05%	1861	131090.75	1.28%	1526
	2	135549	4.72%	1572	130486.67	0.81%	1526
	3	135549	4.94%	1572	133357.48	3.03%	1518
	4	132036	2.01%	1573	134614.10	4.00%	1521
	5	131433	1.54%	1570	129436.70	0.00%	1516
pr136_1	1	94055	1.91%	5	93717	1.54%	4
	2	93808	1.64%	4	93485	1.29%	4
	3	94805	2.72%	4	93485	1.29%	4
	4	95484	3.46%	4	94289	2.16%	4
	5	92581	0.31%	4	93478	1.28%	4

Continues on the next page

Table D.7

		ILS algorithm			Hybrid algorithm		
Instance	Seed	ν^B	gap	t_s	ν^B	gap	t_s
pr144_1	1	58125	8.16%	4	53769	0.06%	5
	2	58125	8.16%	4	53739	0.00%	4
	3	58191	8.28%	4	53769	0.06%	4
	4	58125	8.16%	4	53769	0.06%	4
	5	58191	8.28%	4	54067	0.61%	4
rat195_1	1	1544	-0.26%	12	1522	-1.68%	13
	2	1551	0.19%	12	1530	-1.16%	12
	3	1546	-0.13%	12	1546	-0.13%	12
	4	1552	0.26%	12	1550	0.13%	12
	5	1544	-0.26%	12	1563	0.97%	12
d198_1	1	11688	0.70%	14	11716	0.94%	13
	2	11632	0.22%	12	11663	0.48%	13
	3	11776	1.46%	12	11640	0.28%	12
	4	11728	1.04%	12	11649	0.36%	12
	5	11657	0.43%	12	11690	0.72%	13
kroA200_2	1	32417	0.42%	13	32864	1.80%	13
	2	32541	0.80%	12	32720	1.36%	12
	3	32129	-0.47%	12	32640	1.11%	12
	4	31824	-1.42%	12	32892	1.89%	12
	5	32331	0.15%	12	32999	2.22%	12
kroB200_2	1	31711	3.35%	14	32155	4.79%	13
	2	31503	2.67%	12	32031	4.39%	12
	3	31425	2.41%	12	32529	6.01%	12
	4	31483	2.60%	12	32299	5.26%	12
	5	31656	3.17%	12	32117	4.67%	12
gr202_4	1	32610	4.51%	11	31303	0.32%	11
	2	32741	4.93%	11	31303	0.32%	10
	3	32610	4.51%	11	31303	0.32%	11
	4	32591	4.45%	11	31355	0.49%	11
	5	32142	3.01%	11	31303	0.32%	11

Continues on the next page

Table D.7

		ILS algorithm			Hybrid algorithm		
Instance	Seed	ν^B	gap	t_s	ν^B	gap	t_s
pr226_1	1	63178	-3.09%	20	63254	-2.98%	19
	2	63315	-2.88%	19	63178	-3.09%	18
	3	63337	-2.85%	19	63269	-2.95%	18
	4	63337	-2.85%	18	63382	-2.78%	18
	5	63296	-2.91%	18	63280	-2.94%	18
pr226_3	1	92766	3.44%	11	90671	1.10%	20
	2	92766	3.44%	10	89524	-0.18%	18
	3	92766	3.44%	10	89524	-0.18%	19
	4	92715	3.38%	10	90671	1.10%	19
	5	92715	3.38%	10	89524	-0.18%	19
gil262_2	1	3067	3.44%	29	3078	3.81%	27
	2	3023	1.96%	26	3077	3.78%	27
	3	3014	1.65%	26	3095	4.38%	26
	4	3028	2.12%	26	3096	4.42%	26
	5	3043	2.63%	26	3053	2.97%	26
gil262_4	1	7003	4.58%	20	6975	4.17%	25
	2	6959	3.93%	20	7012	4.72%	24
	3	6979	4.23%	20	6941	3.66%	24
	4	6968	4.06%	20	6921	3.36%	24
	5	6918	3.32%	20	6963	3.99%	24
pr264_2	1	31896	-5.14%	28	30927	-8.03%	25
	2	31982	-4.89%	26	30614	-8.96%	24
	3	32598	-3.06%	26	30836	-8.30%	24
	4	30670	-8.79%	25	31108	-7.49%	24
	5	32312	-3.91%	26	30925	-8.03%	24
pr264_3	1	56348	0.36%	18	55533	-1.09%	22972
	2	55914	-0.41%	17	55556	-1.05%	23208
	3	56219	0.13%	17	55540	-1.08%	23166
	4	56139	-0.01%	17	55493	-1.16%	23148
	5	56137	-0.01%	17	55516	-1.12%	23196

Continues on the next page

Table D.7

		ILS algorithm			Hybrid algorithm		
Instance	Seed	ν^B	gap	t_s	ν^B	gap	t_s
pr264_4	1	45342	-0.86%	24	44828	-1.99%	73
	2	45497	-0.52%	23	44790	-2.07%	62
	3	45526	-0.46%	23	44820	-2.00%	70
	4	45505	-0.51%	23	44820	-2.00%	69
	5	45349	-0.85%	23	44758	-2.14%	69
ftv170_2	1	1727	7.13%	7	1764	9.43%	7
	2	1883	16.81%	7	1774	10.05%	7
	3	1824	13.15%	7	1906	18.24%	7
	4	1767	9.62%	7	1715	6.39%	7
	5	1956	21.34%	7	1875	16.32%	7
ftv170_4	1	2798	10.94%	7	2796	10.86%	9
	2	2702	7.14%	8	2838	12.53%	8
	3	2650	5.08%	8	2726	8.09%	8
	4	2778	10.15%	8	2741	8.68%	8
	5	2716	7.69%	8	2825	12.01%	8
rbg323_1	1	1663	-0.30%	45	1668	0.00%	48
	2	1653	-0.90%	46	1659	-0.54%	46
	3	1677	0.54%	46	1673	0.30%	46
	4	1644	-1.44%	46	1659	-0.54%	46
	5	1663	-0.30%	46	1654	-0.84%	46
rbg323_2	1	787	-5.86%	50	778	-6.94%	51
	2	774	-7.42%	50	792	-5.26%	48
	3	767	-8.25%	50	788	-5.74%	49
	4	768	-8.13%	50	784	-6.22%	49
	5	763	-8.73%	49	790	-5.50%	48
rbg323_3	1	2832	-1.91%	29	2748	-4.81%	39
	2	2839	-1.66%	29	2736	-5.23%	36
	3	2804	-2.87%	29	2748	-4.81%	36
	4	2870	-0.59%	30	2726	-5.58%	36
	5	2858	-1.00%	29	2733	-5.33%	36

Continues on the next page

Table D.7

		ILS algorithm			Hybrid algorithm		
Instance	Seed	ν^B	gap	t_s	ν^B	gap	t_s
rbg323_4	1	1571	-0.06%	41	1583	0.70%	42
	2	1590	1.15%	41	1566	-0.38%	40
	3	1590	1.15%	41	1561	-0.70%	40
	4	1593	1.34%	41	1568	-0.25%	40
	5	1577	0.32%	41	1547	-1.59%	40
rbg358_1	1	1814	-8.11%	69	1831	-7.24%	70
	2	1833	-7.14%	68	1829	-7.35%	66
	3	1821	-7.75%	68	1827	-7.45%	67
	4	1840	-6.79%	68	1819	-7.85%	67
	5	1833	-7.14%	69	1845	-6.53%	68
rbg358_2	1	811	-10.68%	68	847	-6.72%	70
	2	764	-15.86%	68	809	-10.90%	66
	3	819	-9.80%	67	838	-7.71%	66
	4	857	-5.62%	67	834	-8.15%	66
	5	787	-13.33%	68	825	-9.14%	70
rbg403_1	1	1698	-13.63%	99	1695	-13.78%	101
	2	1705	-13.28%	99	1694	-13.84%	95
	3	1696	-13.73%	98	1682	-14.45%	96
	4	1682	-14.45%	99	1682	-14.45%	96
	5	1714	-12.82%	98	1696	-13.73%	103
rbg403_2	1	782	-24.74%	99	796	-23.39%	102
	2	819	-21.17%	99	821	-20.98%	96
	3	790	-23.97%	99	789	-24.06%	98
	4	793	-23.68%	99	792	-23.77%	97
	5	785	-24.45%	100	778	-25.12%	102
rbg403_3	1	3373	-6.80%	56	3287	-9.17%	117
	2	3377	-6.69%	56	3262	-9.86%	102
	3	3455	-4.53%	57	3214	-11.19%	103
	4	3380	-6.60%	56	3270	-9.64%	103
	5	3420	-5.50%	56	3259	-9.95%	112

Continues on the next page

Table D.7

		ILS algorithm			Hybrid algorithm		
Instance	Seed	ν^B	gap	t_s	ν^B	gap	t_s
rbg403_4	1	2016	-4.59%	77	1921	-9.09%	86
	2	1965	-7.00%	77	1908	-9.70%	87
	3	1933	-8.52%	79	1927	-8.80%	81
	4	1972	-6.67%	77	1936	-8.38%	81
	5	1946	-7.90%	77	1923	-8.99%	86
rbg443_1	1	2270	-9.60%	103	2282	-9.12%	111
	2	2253	-10.27%	102	2271	-9.56%	119
	3	2289	-8.84%	103	2243	-10.67%	104
	4	2298	-8.48%	103	2262	-9.92%	104
	5	2222	-11.51%	102	2260	-10.00%	110
rbg443_2	1	1282	-7.90%	120	1272	-8.62%	124
	2	1290	-7.33%	119	1260	-9.48%	135
	3	1262	-9.34%	120	1269	-8.84%	118
	4	1293	-7.11%	120	1262	-9.34%	117
	5	1314	-5.60%	119	1263	-9.27%	124
rbg443_4	1	2446	-0.12%	83	2411	-1.55%	92
	2	2457	0.33%	85	2379	-2.86%	101
	3	2437	-0.49%	83	2389	-2.45%	87
	4	2469	0.82%	84	2380	-2.82%	87
	5	2470	0.86%	84	2389	-2.45%	92
AsimSingh150_1	1	1297	0.93%	6	1293	0.62%	6
	2	1298	1.01%	6	1292	0.54%	6
	3	1302	1.32%	6	1294	0.70%	5
	4	1299	1.09%	6	1296	0.86%	5
	5	1296	0.86%	6	1294	0.70%	6
AsimSingh150_4	1	977	2.73%	5	965	1.47%	5
	2	974	2.42%	5	970	2.00%	5
	3	983	3.36%	5	968	1.79%	5
	4	980	3.05%	5	967	1.68%	5
	5	973	2.31%	5	968	1.79%	5

Continues on the next page

Table D.7

		ILS algorithm			Hybrid algorithm		
Instance	Seed	ν^B	gap	t_s	ν^B	gap	t_s
AsimSingh300_1	1	2803	2.60%	41	2781	1.79%	52
	2	2804	2.64%	40	2786	1.98%	56
	3	2804	2.64%	40	2794	2.27%	50
	4	2807	2.75%	40	2776	1.61%	50
	5	2798	2.42%	40	2788	2.05%	53
AsimSingh300_3	1	4244	2.39%	24	4172	0.65%	105
	2	4245	2.41%	24	4164	0.46%	124
	3	4240	2.29%	24	4172	0.65%	105
	4	4243	2.36%	24	4157	0.29%	104
	5	4236	2.20%	24	4172	0.65%	112
AsimSingh300_4	1	2904	2.07%	31	2888	1.51%	33
	2	2904	2.07%	32	2883	1.34%	37
	3	2882	1.30%	31	2883	1.34%	33
	4	2905	2.11%	32	2883	1.34%	33
	5	2897	1.83%	31	2883	1.34%	35

Appendix E

The Branch-and-Cut Algorithm for the Inter- and Intrafamily Formulations

In the inter- and intrafamily formulations there are some constraints that are in exponential number, namely the subtour elimination constraints in the interfamily subproblem (7.30), the P-CC inequalities (7.25) and the P-RV inequalities (7.26). The subtour elimination constraints for the interfamily subproblem are present in all the proposed inter- and intrafamily formulations whereas the P-CC inequalities and the P-RV inequalities are only part of the P-CC model and of the P-RV model, respectively. Therefore, in order to use the inter- and intrafamily formulations to solve the RFTSP we must resort to a B&C algorithm. We designed a B&C algorithm to solve the several proposed inter- and intrafamily formulations, which are the P-SCF model, the P-MCF model, the P-CC model and the P-RV model. The outline of the B&C algorithm for these formulations is similar to the outline of the B&C algorithm for the FTSP presented in Section 5.1 since the constraints that are in exponential number are added by using the cutting plane algorithm presented in Algorithm 2.1 of Section 2.3. As we mentioned previously, subtour elimination constraints in the interfamily subproblem (7.30) must always be separated while the P-CC inequalities (7.25) and the P-RV inequalities (7.26) are only separated if we want to use the P-CC or the P-RV models, respectively, to solve the RFTSP.

In Section E.1 we present both the user cut callback and the lazy constraint callback used in the B&C algorithm for the inter- and intrafamily formulations.

E.1 The separation algorithms

Throughout this section we only present the separation algorithms for sets of constraints that are in exponential number in the inter- and intrafamily formulations, namely the subtour elimination constraints in the interfamily subproblem (7.30), the P-CC inequalities (7.25) and the P-RV inequalities (7.26) since, even though the separation algorithms are similar to the separation algorithm for the CC and the RFV inequalities, the adaptation is not straightforward. In Sections E.1.1, E.1.2 and E.1.3 we present the separation algorithms for the subtour elimination constraints in the interfamily subproblem (7.30), for the P-CC inequalities (7.25) and for the P-RV inequalities (7.26), respectively.

E.1.1 Separating the subtour elimination constraints in the interfamily subproblem

The subtour elimination constraints for the interfamily subproblem $x(S', S) \geq 1, \forall S \subseteq N : \exists l \in \mathcal{L} : F_l \subseteq S$ (7.30) were introduced as being similar to the connectivity cuts for a TSP in which the nodes are the families, that is, the nodes are the sets F_l, with $l \in \mathcal{L}$. Therefore, we may use a separation algorithm based on max-flow/min-cut computations as long as we ensure that there exists at least one family $l \in \mathcal{L}$ in the set S. Recall that in order to separate the RFV inequalities (4.35) presented in Section 4.2.2.3 we also had to ensure that certain nodes were in the set S, therefore, we will use an algorithm similar to the Separation Algorithm 5.4 for the RFV inequalities presented in Section 5.2.2 to separate the subtour elimination constraints in the interfamily subproblem.

Let $(x^*, w^*, p^*, q^*, u^*)$ be a fractional solution that satisfies the constraints in the inter- and intrafamily formulation that are not in exponential number, that is, constraints (7.4)-(7.11), (4.2), (7.27)-(7.34). Additionally, consider a capacitated graph $G^* = ((0 \cup N) \cup t, A \cup A^t)$ where t is a fictitious node and A_t is the set of arcs from N to t, that is, $A^t = \{(i, t) : i \in N\}$, similarly to what we defined in Section 5.2.2. The capacities of the arc $(i, j) \in N$ are $l_{ij} = 0$ and $u_{ij} = x^*_{ij}$ and, for now, the capacities of the arc $(i, t) \in A^t$ are both zero ($l_{it} = u_{it} = 0$). In order to separate the subtour elimination constraints in the interfamily subproblem (7.30) we just need to fix each family $l \in \mathcal{L}$ to t by setting the upper capacity of the arcs (i, t) with $i \in F_l$ to infinity and compute the maximum flow from the depot to t. Algorithm E.1 shows the pseudocode for the separation algorithm for the subtour elimination constraints in the interfamily subproblem (7.30).

Similarly to what we did in the separation algorithms presented in Section 5.2, we decided to limit the number of violated inequalities found before re-solving the B&C subproblem and to compute the maximum flow from the depot to the families by using a permutation of the families instead of its lexicographic order.

Since to separate all the subtour elimination constraints in the interfamily subproblem we only need to compute L maximum-flows, one for each family, we decided not to develop a heuristic separation for these constraints.

Algorithm E.1 Separation algorithm for the subtour elimination constraints in the interfamily subproblem.

```
Require: A fractional solution $(x^*, w^*, p^*, q^*, u^*)$ that satisfies the equation system (7.4)-(7.11), (4.2), (7.27)-(7.34).
 1: Create a new complete graph $G^* = (0 \cup N \cup t, A \cup A_t)$.
 2: for all $(i, j) \in A$ do
 3:     Set its capacities to $l_{ij} = 0$ and $u_{ij} = x^*_{ij}$.
 4: end for
 5: for all $(i, t) \in A^t$ do
 6:     Set its capacities to $l_{ij} = u_{ij} = 0$.
 7: end for
 8: for all $l \in \mathcal{L}$ do
 9:     for all $(i, t) \in A_t$ with $i \in F_l$ do
10:         Set capacities to $l_{ij} = 0$ and $u_{ij} = +\infty$.
11:     end for
12:     Determine the max-flow between $0$ and $t$ in $G^*$. Let $\nu$ be the value of the max-flow.
13:     if $\nu < 1$ then
14:         Determine the sets $S$ and $S' = (N \cup 0) \setminus S$ such that $S'$ is the set of nodes reachable from
            node $0$ in the residual network associated with the max-flow.
15:         Add the violated inequality $x(S', S) \geq 1$.
16:     end if
17:     for all $(i, t) \in A_t$ with $i \in F_l$ do
18:         Set its capacities to $l_{ij} = u_{ij} = 0$.
19:     end for
20: end for
```

Separation algorithm for integer solutions

When the solution $(x^*, w^*, p^*, q^*, u^*)$ is integer we use the same separation algorithm, that is, Algorithm E.1 but we only add a maximum of one violated inequality. Therefore, Algorithm E.1 stops either when it computes the maximum flow from the depot to every family $l \in \mathcal{L}$ or when it finds one violated subtour elimination constraint in the interfamily subproblem.

E.1.2 Separating the P-CC inequalities

The P-CC constraints $w^k(R', R) \geq q^k_m, \ \forall l \in \mathcal{M}, \ \forall R \subseteq F_l, \ \forall k \in R', \ \forall m \in R$ (7.25) are similar to the CC inequalities (4.28) for the FTSP presented in Section 4.2.2.1 with the following differences: (i) the initial and the final nodes from the arcs in the cut-set $[R', R]$ belong to the same family; and (ii) the initial node of the path is not predetermined. Consequently, to address (i) the separation algorithm for the P-CC inequalities for a family $l \in \mathcal{M}$ uses a network which only contains the nodes in F_l and the set of arcs A_l and, to address (ii), we must consider that any node in F_l can be the initial node of the path.

Let $(x^*, w^*, p^*, q^*, u^*)$ be a fractional solution that satisfies the constraints in the inter- and intrafamily formulation that are not in exponential number, that is, constraints (7.4)-(7.11), (4.2), (7.27)-(7.34). Since the initial node of the path is not predetermined, the capacitated graph in which we will compute the maximum flow depends on the initial node of the path, thus, let $l \in \mathcal{M}$ be a multi-visit family and $k \in F_l$ be the initial node of the path. Consider a capacitated graph $G^{k*} = (F_l, A_l)$ in which the capacities of the arc $(i, j) \in A_l$ are $l_{ij} = 0$ and $u_{ij} = w^{k*}_{ij}$. In order to check whether or not there are violated P-CC inequalities in the case in which $k \in F_l$ is the initial node of the path we must compute the value of the maximum flow from k to every node $m \in F_l : m \neq k$ in the capacitated network presented previously and compare its value to q^{k*}_m. Note that the P-CC inequalities are redundant when either p^{k*} or q^{k*}_m are zero. The pseudocode for the separation algorithm for the P-CC inequalities is available in Algorithm E.2.

Similarly to what we did in the separation algorithm for the CC inequalities presented in Algorithm 5.2, we decided to set a limit to the maximum number of added violated P-CC inequalities per iteration. However, the maximum flow is sent from k to m according to a lexicographic ordering of the nodes.

Heuristic separation of the P-CC inequalities

In order to separate the P-CC inequalities for a family $l \in \mathcal{M}$ we may have to compute $n_l \times (n_l - 1)$ maximum flows, therefore we developed a heuristic separation for the P-CC inequalities similar to

Algorithm E.2 Separation algorithm for the P-CC inequalities.

```
Require: A fractional solution $(x^*, w^*, p^*, q^*, u^*)$ that satisfies the equation system (7.4)-(7.11),
    (4.2), (7.27)-(7.34).
 1: for all $l \in \mathcal{M}$ do
 2:   for all $k \in F_l$ such that $p^{k*} > 0$ do
 3:     Create a new complete graph $G^{k*} = (F_l, A_l)$.
 4:     for all $(i, j) \in A_l$ do
 5:       Set its capacities to $l_{ij} = 0$ and $u_{ij} = w_{ij}^{k*}$.
 6:     end for
 7:     for all $m \in F_l$ such that $m \neq k$ and $q_m^{k*} > 0$ do
 8:       Determine the max-flow between $k$ and $m$ in $G^{k*}$. Let $\nu$ be the value of the max-flow.
 9:       if $\nu < q_m^{k*}$ then
10:         Determine the sets $R$ and $R' = F_l \setminus R$ such that $R'$ is the set of nodes reachable from
            node $k$ in the residual network associated with the max-flow.
11:         Add the violated inequality $w^k(R', R) \geq q_m^k$.
12:       end if
13:     end for
14:   end for
15: end for
```

the heuristic separation for the CC inequalities presented in Section 5.2.1 and which is based on the connected components induced by the fractional solution $(x^*, w^*, p^*, q^*, u^*)$. However, in the case of the P-CC inequalities, the connected components depend on the initial node of the path. Consequently, for a multi-visit family $l \in \mathcal{M}$ and the same fractional solution we may have n_l different connected components. Consider that $k \in F_l$ is the initial node of the path of family l. We assume that nodes $i \in F_l$ and $j \in F_l$ belong to the same connected component if $w_{ij}^{k*} > 0$. Let $\{C_0^k, C_1^k, \ldots, C_p^k\}$ be the set of connected components associated with the fractional solution $(x^*, w^*, p^*, q^*, u^*)$ and $\overline{R} = \{m \in F_l : q_m^{k*} = 0\}$ be the set of non-visited nodes in the path that has k as its initial node. The component C_0^k contains the node k. Like in the heuristic separation of the CC inequalities, we construct a maximum of $p+1$ violated P-CC inequalities by using the $p+1$ connected components. The construction process of the sets R' and R is also similar to the one presented for the case of the CC inequalities in Section 5.2.1 which is: in the first inequality we define $R' = C_0^k$ and $R = C_1^k \cup \ldots \cup C_p^k \cup \overline{R}$ and for ith-inequality, with $i = 1, \ldots, p$, of the remaining p inequalities we define $R = C_i^k$ and $R' = F_l \setminus C_i^k$. Notice that for every R' and R defined previously: (i) $k \in R'$, (ii) set R contains visited nodes, that is, nodes $m \in F_l$ such that $q_m^{k*} > 0$, and (iii) $w^{k*}(R', R) = 0$, therefore all these sets originate violated P-CC inequalities. As $w^{k*}(R', R) = 0$, choosing for the right-hand side any node $m \in R$ such that $q_m^{k*} > 0$ originates a violated P-CC inequality. However, similarly to what we did for the CC inequalities, we decided to choose the node with the highest q_m^{k*} value, that is, the node $m \in R$ such that $q_m^{k*} \geq q_i^{k*}, \forall i \in R$. The pseudocode for the heuristic separation of the P-CC inequalities is presented in Algorithm E.3. Once again, we decided to put a limit to the number of added violated inequalities. Therefore, the heuristic separation algorithm stops either when it has separated all the P-CC inequalities induced by the connected components or when the maximum number of added violated inequalities was reached.

In the user cut callback for the P-CC inequalities, we start by applying the heuristic separation algorithm presented in Algorithm E.3 and in the end, when the heuristic procedure is not able to provide new violated inequalities, we use the exact separation algorithm, presented in Algorithm E.2, either to find more violated inequalities or to conclude that there are no more violated inequalities. By using this procedure we ensure that the solution obtained is the optimal solution for LP relaxation of the P-CC model.

Separation algorithm for the integer solutions

When the solution $(x^*, w^*, p^*, q^*, u^*)$ is integer we use the exact separation algorithm for the P-CC inequalities, that is, Algorithm E.2 but we only add a maximum of one violated inequality per

Algorithm E.3 Heuristic separation of the P-CC inequalities.

```
Require: A fractional solution $(x^*, w^*, p^*, q^*, u^*)$ that satisfies the equation system (7.4)-(7.11), (4.2), (7.27)-(7.34).
 1: for all $l \in \mathcal{M}$ do
 2:   for all $k \in F_l$ such that $p^{k*} > 0$ do
 3:     Determine the connected components $\{C_0^k, \dots, C_p^k\}$ and the set of non-visited nodes $\overline{R}$ induced by the fractional solution.
 4:     $iter = 0$.
 5:     while $iter < p + 1$ do
 6:       if $iter = 0$ then
 7:         Set $R' = C_0^k$ and $R = C_1^k \cup C_2^k \cup \dots \cup C_p^k \cup \overline{R}$.
 8:       else
 9:         Set $R = C_{iter}^k$ and $R' = F_l \setminus R$.
10:       end if
11:       Determine $m \in R$ such that $q_m^{k*} \geq q_i^{k*}, \forall i \in R$.
12:       Add the violated inequality $w^k(R', R) \geq q_m^k$.
13:       $iter = iter + 1$.
14:     end while
15:   end for
16: end for
```

iteration.

E.1.3 Separating the P-RV inequalities

The P-RV inequalities $w^k(R', R) \geq 1,\ \forall l \in \mathcal{M},\ \forall R \subseteq F_l : |R| \geq n_l - v_l + 1,\ \forall k \in R'$ (7.26) are only defined for sets R with a certain cardinality, more precisely, the set R must have at least $n_l - v_l + 1$ nodes. Therefore, the separation algorithm for the P-RV inequalities is similar to the separation algorithm for the RFV inequalities (4.35) presented in Section 5.2.2. In addition, the cut-set $[R', R]$ only contains arcs which have their initial and final nodes from the same family and that depend on the initial node of the path, consequently, the P-RV inequalities are separated in a network similar to the one introduced in Section E.1.2 to separate the P-CC inequalities.

Before presenting the separation algorithm for the P-RV inequalities, recall the set $\mathbb{S}_l^{=} = \{S \subseteq F_l : |S| = n_l - v_l + 1\}$ which is the set of subsets of F_l, with $l \in \mathcal{M}$, for which the P-RV inequalities are defined. Let $(x^*, w^*, p^*, q^*, u^*)$ be a fractional solution that satisfies the constraints in the inter- and intrafamily formulation that are not in exponential number, that is, constraints (7.4)-(7.11), (4.2), (7.27)-(7.34). Consider that for a family $l \in \mathcal{M}$ the initial node of the path is $k \in F_l$. In addition, consider the capacitated network $G^{k*} = (F_l \cup t, A_l \cup A^t)$ where t is a fictitious node and A_t is the set of arcs from F_l to t, that is, $A^t = \{(i, t) : i \in F_l\}$. The capacities of the arcs $(i, j) \in A_l$ are $l_{ij} = 0$ and $u_{ij} = w_{ij}^{k*}$ while the capacities of the arcs $(i, t) \in A^t$ are both zero ($l_{it} = u_{it} = 0$), for now. In order to separate all the P-RV inequalities (7.26) we must fix every set $R_l \in \mathbb{S}_l^{=}$ to R, which is done by setting the capacity of the arcs $(i, t) \in A^t$ such that $i \in R_l$ to infinity and computing the maximum flow from k to t. The pseudocode for the separation algorithm for the P-RV inequalities is available in Algorithm E.4.

Similarly to what we did with the other separation algorithms, we limited the number of added violated inequalities per iteration. Therefore, Algorithm E.4 stops either when it cannot find any violated P-RV inequality or when the maximum number of added violated inequalities was reached.

Heuristic separation for the P-RV inequalities

We verified in Section 5.2.2 that the exact separation of the RFV inequalities is very time consuming. As the separation algorithm of the P-RV inequalities is similar to the separation algorithm of the RFV inequalities it is very time consuming. Therefore, we developed a heuristic separation algorithm for the P-RV inequalities, which is similar to the heuristic separation algorithm for the P-CC inequalities presented in Algorithm E.3 with the difference that after determining the sets R' and R we need to verify if $R \in \mathbb{S}_l^{=}$. If it is, then we add the violated P-RV inequality found. Algorithm E.5 shows the pseudocode for the heuristic separation of the P-RV inequalities.

Algorithm E.4 Separation algorithm for the P-RV inequalities.

```
Require: A fractional solution $(x^*, w^*, p^*, q^*, u^*)$ that satisfies the equation system (7.4)-(7.11),
    (4.2), (7.27)-(7.34).
 1: for all $l \in \mathcal{M}$ do
 2:   for all $k \in F_l$ such that $p^{k*} > 0$ do
 3:     Create a new complete graph $G^{k*} = (F_l, A_l \cup A^t)$.
 4:     for all $(i,j) \in A_l$ do
 5:       Set its capacities to $l_{ij} = 0$ and $u_{ij} = w_{ij}^{k*}$.
 6:     end for
 7:     Compute all the sets in $\mathbb{S}_l^=$ (composed of subsets of $F_l$).
 8:     while There are sets $R_l \in \mathbb{S}_l^=$ to consider do
 9:       for all $(i,t) \in A_t$ do
10:         Set its capacities to $l_{ij} = u_{ij} = 0$.
11:       end for
12:       Consider $R_l \in \mathbb{S}_l^=$ and set the capacities of arcs $(i,t) \in A_t$, with $i \in R_l$, to $u_{it} = +\infty$
13:       Determine the max-flow between $k$ and $t$ in $G^{k*}$. Let $\nu$ be the value of the max-flow.
14:       if $\nu < 1$ then
15:         Determine the sets $R$ and $R' = F_l \setminus R$ such that $R'$ is the set of nodes reachable from
            node $k$ in the residual network associated with the max-flow.
16:         Add the violated inequality $w^k(R', R) \geq p^k$.
17:       end if
18:     end while
19:   end for
20: end for
```

Algorithm E.5 Heuristic separation of the P-RV inequalities.

```
Require: A fractional solution $(x^*, w^*, p^*, q^*, u^*)$ that satisfies the equation system (7.4)-(7.11), (4.2), (7.27)-(7.34).
 1: for all $l \in \mathcal{M}$ do
 2:   for all $k \in F_l$ such that $p^{k*} > 0$ do
 3:     Determine the connected components $\{C_0^k, \ldots, C_p^k\}$ and the set of non-visited nodes $\overline{R}$ induced by the fractional solution.
 4:     $iter = 0$.
 5:     while $iter < p + 1$ do
 6:       if $iter = 0$ then
 7:         Set $R' = C_0^k$ and $R = C_1^k \cup C_2^k \cup \ldots \cup C_p^k \cup \overline{R}$.
 8:       else
 9:         Set $R = C_{iter}^k$ and $R' = F_l \setminus R$.
10:       end if
11:       if $R \in \mathbb{S}^{FV}$ then
12:         Add the violated inequality $w^k(R', R) \geq p^k$.
13:       end if
14:       $iter = iter + 1$.
15:     end while
16:   end for
17: end for
```

We start by separating the P-RV inequalities by using the heuristic separation algorithm presented in Algorithm E.5 and then, when the heuristic separation algorithm is not able to provide more violated P-RV inequalities, we apply the exact separation presented in Algorithm E.3. With this approach we ensure that the solution obtained does not violate any P-RV inequality.

Separation algorithm for integer solutions

When the solution $(x^*, w^*, p^*, q^*, u^*)$ is integer we use a different separation algorithm, which is based on the idea that if an integer solution is unfeasible then it contains subtours. Recall that, in this case, the subtours depend on the initial node of the path. Consider a multi-family $l \in \mathcal{M}$ and a node $k \in F_l$ which is the initial node of the path associated with family l. Let $\mathbb{T}_l^{k*}$ be the set of subtours associated with the solution $(x^*, w^*, p^*, q^*, u^*)$ and with the family l that has k as the initial node of the path. Consider that the subtour $T_0 \in \mathbb{T}_l^{k*}$ is the one that contains node $k \in F_l$. Additionally, let $\overline{R} = \{i \in F_l : q_i^{k*} = 0\}$ be the set of non-visited nodes in the path that has k as the initial node. We define R as a subset of F_l that has at least one subtour, except for subtour T_0, and the set $\overline{R}$. Since: (i) $k \in R'$; and (ii) $|R| \geq n_l - v_l + 1$ as there are visited nodes in R and, consequently, there are not enough nodes in R' to fulfill the family visits, then the set R is in the conditions of the P-RV inequalities. Algorithm E.6 shows the pseudocode for the lazy constraint callback used for the P-RV inequalities.

Algorithm E.6 Separation algorithm for the P-RV inequalities for integer solutions.

```
Require: An integer solution $(x^*, w^*, p^*, q^*, u^*)$ that satisfies the equation system (7.4)-(7.11),
    (4.2), (4.4) for $i = 0$, (7.28)-(7.34).
 1: for all $l \in \mathcal{M}$ do
 2:     for all $k \in F_l$ such that $p^{k*} > 0$ do
 3:         Determine $\mathbb{T}_l^{k*}$ and $\overline{R}$. Let $T_i$ be the $i$-th subtour in $\mathbb{T}_l^{k*}$.
 4:         Set $j = 1$.
 5:         while $j \leq |\mathbb{T}_l^{k*}|$ do
 6:             Set $R = \cup_{i=j}^{|\mathbb{T}_l^{k*}|} T_i \cup \overline{R}$ and $R' = F_l \setminus R$.
 7:             Add the violated inequality $w^k(R', R) \geq p^k$.
 8:             Set $j = j + 1$.
 9:         end while
10:     end for
11: end for
```

In the B&C algorithm for the inter- and intrafamily formulations we used the heuristic callback described in Section 7.4.1

Appendix F

Dimension proof

In this appendix we show how we obtained the dimension (8.1). Recall that

$$dim(P_{FTSP}) = |N|^2 - 1 - L - \sum_{l \in \mathcal{U}} n_l \times (n_l - 1) - \sum_{l \in \mathcal{W}} (n_l - 1),$$

where P_{FTSP} is the convex hull of the integer points that are feasible solutions for the FTSP. We start by formally defining the polytope P_{FTSP} and then we show how we determined its dimension (8.1).

When we presented the generic model in Section 4.1 we mentioned that the y variables are auxiliary. Thus, polytope P_{FTSP} can be defined in the space of the x variables. Therefore, for simplification, we present the generic model for the FTSP only defined with the x variables:

$$\textit{Minimize} \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{4.1}$$

Subject to:

$$\sum_{j \in N} x_{0j} = 1 \tag{4.2}$$

$$\sum_{j \in 0 \cup N} x_{ji} - \sum_{j \in 0 \cup N} x_{ij} = 0 \qquad \forall i \in 0 \cup N \tag{4.4}$$

$$\sum_{i \in F_l} \sum_{j \in 0 \cup N} x_{ij} = v_l \qquad \forall l \in \mathcal{L} \tag{F.1}$$

$$\{(i,j) \in A : x_{ij} = 1\} \textit{ is a single connected circuit} \tag{4.6}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A, \tag{4.7}$$

The only difference between this model and the one presented in Section 4.1 is that in constraints (4.5) we replaced the y variables with the x variables using the equalities (4.3), which originated constraints (F.1).

The polytope P_{FTSP} is formally defined as follows:

$$P_{FTSP} = conv(\{x \in \mathbb{R}^{|A|} : x \textit{ satisfies } (4.2), (4.4), (F.1), (4.6) - (4.7)\}).$$

Note that the polytope P_{FTSP} is defined with the generic subtour elimination constraints (4.6) because we will not need to explicitly define them in order to obtain the dimension (8.1). In addition, recall that we assume that G is a complete graph, thus, $|A| = (|N| + 1) \times |N|$.

We start by determining the $rank(A^=, b^=)$, where $(A^=, b^=)$ is the equality matrix associated with the polytope P_{FTSP}. The number of linearly independent equalities presented in the generic formulation showed previously is $|N|+1+L$, which correspond to one equality (4.2), $|N|$ equalities (4.4) and L equalities (F.1). Note that, even though there are $|N| + 1$ equalities (4.4), one of them is not linearly independent from the others.

Consider a single-family $l \in \mathcal{U}$. As we mentioned in Section 4.1, there is a set of valid inequalities for family l that results from the observation that if $i, j \in F_l$, then the arc (i, j) will never be used in a feasible solution for the FTSP, since using these arcs would imply visiting two nodes from a single-visit family. Therefore, $x_{ij} = 0,\ \forall (i, j) \in A : i, j \in F_l$ and there are an additional $n_l \times (n_l - 1)$ equalities for the single-visit families.

Consider now a family $l \in \mathcal{W}$. Since we are required to visit every node from family l, the visit constraints (F.1) associated with family l may be replaced with $\sum_{j \in 0 \cup N} x_{ij} = 1, \forall i \in F_l$ and, thus, there are an additional n_l equalities. However, in the presence of constraints (F.1) one of the previous equalities is not linearly independent from the others. Therefore, for $l \in \mathcal{W}$, we have an additional $n_l - 1$ linearly independent equalities.

The number of equalities presented is a lower bound for the number of linearly independent rows in $(A^=, b^=)$, consequently and according to the definition of rank (Definition 7) we have:

$$rank(A^=, b^=) \geq |N| + 1 + L + \sum_{l \in \mathcal{U}} n_l \times (n_l - 1) + \sum_{l \in \mathcal{W}} (n_l - 1)$$

From Proposition 1 presented in Section 2.2 and due to the polytope P_{FTSP} being contained in $\mathbb{R}^{|A|}$ it follows that:

$$\begin{aligned} & dim(P_{FTSP}) = |A| - rank(A^=, b^=) \\ \Longrightarrow\ & dim(P_{FTSP}) \leq |A| - [|N| + 1 + L + \sum_{l \in \mathcal{U}} n_l \times (n_l - 1) + \sum_{l \in \mathcal{A}} (n_l - 1)] \\ \Longrightarrow\ & dim(P_{FTSP}) \leq |N|^2 - 1 - L - \sum_{l \in \mathcal{U}} n_l \times (n_l - 1) - \sum_{l \in \mathcal{W}} (n_l - 1). \end{aligned} \quad \text{(F.2)}$$

From Definition 9 of Section 2.2 we know that the dimension of a polytope is the maximum number of affinely independent points that belong to the polytope minus 1. Hence, in order for the inequality (F.2) to be satisfied with equality we must find $|N|^2 - L - \sum_{l \in \mathcal{U}} n_l \times (n_l - 1) - \sum_{l \in \mathcal{W}}(n_l - 1)$ affinely independent points that belong to the polytope P_{FTSP}.

The underlying idea for the construction of the $|N|^2 - L - \sum_{l \in \mathcal{U}} n_l \times (n_l - 1) - \sum_{l \in \mathcal{W}}(n_l - 1)$ affinely independent points is to consider a TSP instance that contains the depot and v_l nodes from each family $l \in \mathcal{L}$ and use the affinely independent points of the TSP polytope. Then, the other affinely independent points are obtained by replacing visited nodes with non-visited nodes in the TSP instance.

Consider the set $\mathcal{T}$ which contains the depot and the first v_l nodes, according to a lexicographic ordering of the nodes, of each family l, with $l \in \mathcal{L}$, that is, $\mathcal{T} = \{0, 1, \ldots, v_1, n_1 + 1, \ldots, n_1 + v_2, \ldots\}$. Note that $|\mathcal{T}| = \sum_{l \in \mathcal{L}} v_l + 1 = V + 1$, thus, solving the FTSP considering the nodes in $\mathcal{T}$ is equivalent to solving a TSP with $V + 1$ nodes. Since the dimension of the polytope associated with the TSP with $V + 1$ nodes is $(V + 1) \times V - 2(V + 1) + 1$ (see, e.g., Lawler et al., 1985), we know that, from the definition of dimension, the polytope associated with the TSP contains $(V + 1) \times V - 2(V + 1) + 1 + 1$ affinely independent points. Due to the set $\mathcal{T}$ containing v_l nodes from each family $l \in \mathcal{L}$, these $(V + 1) \times V - 2(V + 1) + 1 + 1$ affinely independent points are feasible FTSP solutions and, thus, belong to the polytope P_{FTSP}. Therefore, we have

$$(V + 1) \times V - 2(V + 1) + 1 + 1 = V \times (V - 1) \tag{F.3}$$

affinely independent points in P_{FTSP}.

The remaining affinely independent points are obtained by replacing nodes in $\mathcal{T}$ with nodes in $N \setminus \mathcal{T}$. There are three distinct cases: (i) the nodes in $N \setminus \mathcal{T}$ belong to a family $l \in \mathcal{L} \setminus (\mathcal{U} \cup \mathcal{W})$; (ii) the nodes in $N \setminus \mathcal{T}$ belong to a family $l \in \mathcal{U}$; and (iii) the arcs used link nodes in $N \setminus \mathcal{T}$ that belong to different families. Note that we do not consider the case in which the nodes in $N \setminus \mathcal{T}$ belong to a family $l \in \mathcal{W}$ since in these families we must visit every family node and, consequently, there are no nodes in $N \setminus \mathcal{T}$ that belong to family l. We start by considering nodes in the case (i), then nodes in the case (ii) and, finally, arcs in the case (iii).

For all the affinely independent points, or feasible FTSP solutions, presented henceforth, we only explicitly show some of its arcs, which correspond to the ones that ensure their affinely independence. The rest of the solution is completed with any nodes from the set $\mathcal{T}$.

Let α be a node in the case (i), that is, consider family $l \in \mathcal{L} \setminus (\mathcal{U} \cup \mathcal{W})$ and $\alpha \in F_l : \alpha \notin \mathcal{T}$. Note that α exists since we are not required to visit every node from family l. Consider the new solutions constructed by using the following sequence of arcs:

$$\{(0, \alpha), (\alpha, i), \ldots\} \qquad \forall i \in \mathcal{T} \setminus 0 \tag{F.4}$$

$$\{(0, i), (i, \alpha), \ldots\} \qquad \forall i \in \mathcal{T} \setminus 0. \tag{F.5}$$

This construction originates $2\times|\mathcal{T}\backslash 0| = 2V$ new solutions, which are affinely independent between them and from the solutions presented previously since these are the only solutions that use the arcs (i,α) and (α,i), with $i \in \mathcal{T}\setminus 0$. We can use this construction for all nodes $\alpha \in F_l \setminus \mathcal{T}$. Therefore, for each family $l \in \mathcal{L}\setminus(\mathcal{U}\cup\mathcal{W})$ we can construct $(n_l - v_l)\times 2V$ affinely independent points by using constructions (F.4) and (F.5).

Consider now that β is the n_l-th node from family l. By noting that $\beta \notin \mathcal{T}$ due to the lexicographic ordering chosen, we can construct more affinely independent points by using the following sequence of arcs:

$$\{(0,i),\ldots,(\beta,0)\} \qquad\qquad \forall i \in F_l \cap \mathcal{T} \tag{F.6}$$

These points are the only ones defined in the hyperplane characterized by $x_{0i} = 1$ and $x_{\beta 0} = 1$, with $i \in F_l \cap \mathcal{T}$. Therefore, we obtain an additional v_l points for each family $l \in \mathcal{L}\setminus(\mathcal{U}\cup\mathcal{W})$ by using construction (F.6).

Finally, considering the nodes $\gamma \in F_l \setminus \{\mathcal{T}\cup\beta\}$ we can construct more points by using the following construction:

$$\{\ldots,(\gamma,0)\} \qquad\qquad \gamma \in F_l \setminus \{\mathcal{T}\cup\beta\}, \tag{F.7}$$

which are affinely independent from the ones previously presented since they are the only that use the arc $(\gamma,0)$ with $\gamma \in F_l \setminus \{\mathcal{T}\cup\beta\}$.

Summarizing, with the constructions (F.4), (F.5), (F.6) and (F.7) we can obtain:

$$\sum_{l\in\mathcal{L}\setminus(\mathcal{U}\cup\mathcal{W})} [2V\times(n_l - v_l) + v_l + n_l - v_l - 1] = \sum_{l\in\mathcal{L}\setminus(\mathcal{U}\cup\mathcal{W})} [2V\times(n_l - v_l) + n_l - 1] \tag{F.8}$$

affinely independent points.

Consider now a non-visited node in case (ii), that is, consider a single-visit family $l \in \mathcal{U}$ and α a non-visited node from family l. Additionally, we define β as the visited node from family l, that is, $\beta \in F_l \cap \mathcal{T}$. By using a construction similar to constructions (F.4) and (F.5) presented previously and noting that these constructions may only be applied to $i \in \mathcal{T}\setminus\{0,\beta\}$ since we cannot use the arcs (α,β) and (β,α), we can construct $2(V-1)$ new affinely independent solutions with α, which are affinely independent for the same reason as before, that is, they use arcs which have never been used in any solution. We can also use construction (F.7) for the single-visit families, therefore there is an additional affinely independent point that we can obtain by using the arc $(\alpha,0)$. Consequently, and since the constructions presented are valid for every $\alpha \in F_l \setminus \mathcal{T}$, the total number of affinely independent solutions constructed considering single-visit families is:

$$\sum_{l\in\mathcal{U}} [2(V-1)+1]\times(n_l - v_l) = \sum_{l\in\mathcal{U}} (2V-1)\times(n_l - v_l) \tag{F.9}$$

Finally, we consider solutions that use arcs (i, j) between nodes in the case (iii), that is, such that $i, j \in N \setminus \mathcal{T}$. Therefore, let $\alpha, \beta \notin \mathcal{T} : \alpha \neq \beta$. We can construct solutions that are affinely independent from the ones presented previously since they are the only ones that use the following arcs:

$$\{(0, \alpha), (\alpha, \beta), \ldots\} \qquad \forall \alpha, \beta \in N \setminus \mathcal{T} : \alpha \neq \beta \tag{F.10}$$

$$\{(0, \beta), (\beta, \alpha), \ldots\} \qquad \forall \alpha, \beta \in N \setminus \mathcal{T} : \alpha \neq \beta \tag{F.11}$$

By using constructions (F.10) and (F.11) we obtain a maximum of $(|N| - V) \times (|N| - V - 1)$ additional affinely independent solutions. Observe that when the nodes α and β belong to a single-visit family $l \in \mathcal{U}$, the arcs (α, β) and (β, α) cannot be used to construct points of P_{FTSP}. Therefore, for each family $l \in \mathcal{U}$ we must subtract $(n_l - v_l) \times (n_l - v_l - 1)$ arcs. Summarizing, the total number of affinely independent points constructed that use arcs between nodes $\alpha, \beta \in N \setminus \mathcal{T} : \alpha \neq \beta$ is:

$$(|N| - V)(|N| - V - 1) - \sum_{l \in \mathcal{U}} (n_l - v_l)(n_l - v_l - 1) \tag{F.12}$$

By adding all the affinely independent points constructed so far, that is, (F.3), (F.8), (F.9) and (F.12) we obtain the expression:

$$\begin{aligned} & V \times (V - 1) + (|N| - V) \times (|N| - V - 1) + \sum_{l \in \mathcal{L}} [2V \times (n_l - v_l) + n_l - 1] \\ - \quad & \sum_{l \in \mathcal{U}} [2V \times (n_l - v_l) + n_l - 1] - \sum_{l \in \mathcal{W}} [2V \times (n_l - v_l) + n_l - 1] \\ + \quad & \sum_{l \in \mathcal{U}} (2V - 1) \times (n_l - v_l) - \sum_{l \in \mathcal{U}} (n_l - v_l)(n_l - v_l - 1) \end{aligned} \tag{F.13}$$

By noting that when $l \in \mathcal{W}$ we have $n_l = v_l$ and that when $l \in \mathcal{U}$ we have $v_l = 1$, expression (F.13) is equal to:

$$\begin{aligned} & V \times (V - 1) + (|N| - V) \times (|N| - V - 1) + \sum_{l \in \mathcal{L}} 2V \times (n_l - v_l) + \sum_{l \in \mathcal{L}} n_l - |\mathcal{L}| \\ - \quad & \sum_{l \in \mathcal{U}} 2V \times (n_l - 1) - \sum_{l \in \mathcal{U}} (n_l - 1) - \sum_{l \in \mathcal{W}} (n_l - 1) \\ + \quad & \sum_{l \in \mathcal{U}} 2V \times (n_l - 1) - \sum_{l \in \mathcal{U}} (n_l - 1) - \sum_{l \in \mathcal{U}} (n_l - 1)(n_l - 2) \end{aligned} \tag{F.14}$$

By using the notation presented in Chapter 3, we have $\sum_{l \in \mathcal{L}} n_l = |N|$ and $\sum_{l \in \mathcal{L}} v_l = V$. In addition, note that the term $\sum_{l \in \mathcal{U}} 2V \times (n_l - 1)$ appears in the previous expression with a positive and negative sign, therefore expression (F.14) is equal to:

$$\begin{aligned} & V \times (V-1) + (|N|-V) \times (|N|-V-1) + 2V|N| - 2V^2 + |N| - |\mathcal{L}| \\ - & \sum_{l \in \mathcal{W}} (n_l - 1) - \sum_{l \in \mathcal{U}} (n_l - 1)(n_l - 2 + 1 + 1) \quad \text{(F.15)} \\ = & |N|^2 - L - \sum_{l \in \mathcal{W}} (n_l - 1) - \sum_{l \in \mathcal{U}} n_l \times (n_l - 1) \quad \text{(F.16)} \end{aligned}$$

As it was already mentioned, the dimension of a polytope is equal to maximum number of affinely independent points minus one. Therefore,

$$dim(P_{FTSP}) \geq |N|^2 - 1 - L - \sum_{l \in \mathcal{W}} (n_l - 1) - \sum_{l \in \mathcal{U}} n_l \times (n_l - 1) \quad \text{(F.17)}$$

From (F.2) and (F.17) we obtain the dimension (8.1).

Example 34. The dimension of the polytope associated to the FTSP instance presented in Figure 3.1 is $5^2 - 1 - 2 - 0 - 2 \times 1 = 20$, therefore, there are 21 affinely independent points in the polytope associated to this instance. Table F.1 shows the affinely independent points obtained using the constructions presented previously.

Table F.1: Affinely independent points considering the FTSP instance presented in Figure 3.1.

TSP points considering $\mathcal{T} = \{0, 1, 3, 4\}$ (F.3)	
$\{(0,4),(4,1),(1,3),(3,0)\}$	$\{(0,3),(3,1),(1,4),(4,0)\}$
$\{(0,1),(1,4),(4,3),(3,0)\}$	$\{(0,3),(3,4),(4,1),(1,0)\}$
$\{(0,1),(1,3),(3,4),(4,0)\}$	$\{(0,4),(4,3),(3,1),(1,0)\}$
Points using construction (F.4)	Points using construction (F.5)
$\{(0,5),(5,1),(1,3),(3,0)\}$	$\{(0,1),(1,5),(5,3),(3,0)\}$
$\{(0,5),(5,3),(3,1),(1,0)\}$	$\{(0,3),(3,5),(5,1),(1,0)\}$
$\{(0,5),(5,4),(4,1),(1,0)\}$	$\{(0,4),(4,5),(5,1),(1,0)\}$
Points using construction (F.6)	
$\{(0,3),(3,1),(1,5),(5,0)\}$	
$\{(0,4),(4,1),(1,5),(5,0)\}$	
Points using construction (F.4) for single-visit families	Points using construction (F.5) for single-visit families
$\{(0,2),(2,3),(3,4),(4,0)\}$	$\{(0,3),(3,2),(2,4),(4,0)\}$
$\{(0,2),(2,4),(4,3),(3,0)\}$	$\{(0,4),(4,2),(2,3),(3,0)\}$
Points using construction (F.7) for single-visit families	
$\{(0,3),(3,4),(4,2),(2,0)\}$	
Points using construction (F.10)	Points using construction (F.11)
$\{(0,2),(2,5),(5,3),(3,0)\}$	$\{(0,5),(5,2),(2,3),(3,0)\}$

Note that dimension (8.1) may not be valid for instances with a smaller size than the instance presented in Figure 3.1 due to the reduced number of nodes.