Titre: Title:	Using Information from Solution Densities of Relaxations in Solving Variants of the Traveling Salesman Problem
Auteur: Author:	Garance Cordonnier Martin De Gibergues
Date:	2021
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Cordonnier Martin De Gibergues, G. (2021). Using Information from Solution Densities of Relaxations in Solving Variants of the Traveling Salesman Problem [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/6568/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/6568/
Directeurs de recherche: Advisors:	Gilles Pesant, & Andrea Lodi
Programme: Program:	Génie informatique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Using Information from Solution Densities of Relaxations in Solving Variants of the Traveling Salesman Problem

GARANCE CORDONNIER MARTIN DE GIBERGUES
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Mai 2021

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Using Information from Solution Densities of Relaxations in Solving Variants of the Traveling Salesman Problem

présenté par **Garance CORDONNIER MARTIN DE GIBERGUES**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Quentin CAPPART, président
Gilles PESANT, membre et directeur de recherche
Andrea LODI, membre et codirecteur de recherche
Michel GENDREAU, membre

DEDICATION

To Quebec and the city of Montreal,
for the great experience that these two years have been
and to both Polytechnique(s) for making it possible

ACKNOWLEDGEMENTS

First, I would like to give my gratitude to both my research directors, Dr. Andrea Lodi and Dr. Gilles Pesant, for their supervision, their availability and numerous advice. For a first research experience, I was very lucky to be able to count on their expertise and patience. A special thank you goes to the continued supervision with online communication tools after the start of the pandemic.

I want to show my appreciation to Nicolas Isoart and Jean-Charles Régin for sending me their implementation of Talos and answering my questions, and to Quentin Cappart and Michel Gendreau who agreed to be a part of my jury.

I am thankful to my parents for everything, and even accepting out of love to read a n pages document about computer science.

In these particular times we are living in, I would like to thank everyone around me for the moral support, and especially the group of friends that went through the holidays together without our families; you are a ray of sunshine. A giant acknowledgement to Nicolas who kindly endured my confined self for a year.

RÉSUMÉ

Le sujet de cette maîtrise est la famille de problèmes liée au Voyageur de Commerce. Ce problème NP-difficile, rendu célèbre pour ses nombreuses applications et pour la variété des améliorations dans le domaine de l'optimisation apparues en l'étudiant, vise à déterminer un cycle hamiltonien de poids minimum dans un graphe. Plusieurs variantes existent, qui incluent des contraintes additionnelles. En particulier, ce sujet se concentre sur l'introduction d'une asymmétrie des coûts et l'ajout de contraintes temporelles.

Au cours de ce projet, nous cherchons à développer des méthodes d'aide à la résolution de ces variantes, basées sur le concept de *densités de solutions*. Nous utilisons l'information obtenue en résolvant des problèmes similaires et plus faciles. Pour ce faire, plusieurs relaxations sont présentées ainsi que la façon de calculer les densités correspondantes.

L'objectif est d'accélérer la résolution par des modèles de Programmation par Contraintes et de Programmation en Nombres Entiers, deux paradigmes de modélisation et résolution de problèmes. Dans ces modèles, nos densités de solutions peuvent être interprétées comme des probabilités qu'une valeur particulière soit affectée à une variable.

Ce mémoire décrit les résultats expérimentaux obtenus à l'aide de plusieurs solveurs plus ou moins spécialisés dans la résolution du Problème du Voyageur de Commerce. Nous montrons que les densités de solutions peuvent être utilisées pour éliminer des arêtes peu prometteuses, pré-sélectionner des variables ou pré-déterminer des précédences dans un tour.

ABSTRACT

The traveling salesman problem (TSP) is a very widely studied routing problem that aims at finding a minimum length Hamiltonian tour in a graph. The problem in itself is NP-hard, and often used as a research focus in the development of optimization techniques. Several variants exist for the problem, adding time constraints or asymmetric distances in the graph. The goal of this subject is to develop reduction methods to simplify the resolution of these TSP-like problems.

Our main focus is using relaxations of the problems in order to get relevant information on good variable-value assignments. We present several relaxations, i.e., easier related problems, and the corresponding information, called *solution density*. This document compiles computation methods and less time-consuming approximation for these solution densities.

We introduce methods to use this information to simplify the resolution: how to select good arcs, reduce the domains of the time variables and infer partial orders. We test these approaches on several solvers from two different solving paradigms, namely Constraint Programming and Mixed-Integer Programming and report experimental results on usual benchmarks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

TSP	Traveling Salesman Problem
ATSP	Asymmetric Traveling Salesman Problem
TSPTW	Traveling Salesman Problem with Time Windows
LP	Linear Programming
MIP	Mixed-Integer Programming
CP	Constraint Programming
CSP	Constraint Satisfaction Programs
COP	Contsraint Optimization Programs
SD	Solution Density
VRP	Vehicle Routing Problem
MTZ	Miller-Tucker-Zemlin
WCC	Weighted-Circuit Constraint
CBS	Counting-Based Search
(MW)VCP	(Minimum Weighted) Vertex-Cover Problem
FPT	Fixed-Parameter Tractable

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

1.1 The Traveling Salesman Problem

Even before the rise of online shopping, worldwide trade of goods was made possible ever since the Middle Ages by traveling commercials, walking the roads to distribute products from town to town. The traveling salesman (or salesperson) became a literature figure, from the character by Honoré de Balzac in the XIXth century [1] to the american play [2] "Death of a Salesman" in 1949, adapted in several countries' theaters and as movies. Simultaneously a stereotype of materialistic middle-class, a window open on the world and exotic travels, and an occupation affected by the evolution of industry and trade, the traveling salesman is studied as an historical figure [3–5]. In 1930, the Traveling Salesman gives birth to an eponym problem that studies the optimization of the salesman's trajectory: how can he choose a road map that will help him save some traveling time?

The problem has been intensively studied in the past century. It is a central problem in constrained optimization, which makes it a very interesting challenge, especially to test generic solving approaches, since a lot of benchmark data exist to be compared with. The book [6] is a TSP-specific bibliography that argues that the success of the TSP as a central problem is a consequence of its help in designing generic techniques in applied mathematics, especially in developing the field of Mixed-Integer Programming, Branch and Bound methods and heuristic search. The idea that started this research is to use the TSP as an entry point for exploring Counting-Based search (CBS) and the use of relaxations' solution densities in optimization problems: in the original CBS article for optimization [7], the TSP is already used as a first example and the project [8] was a logical next step in the same direction.

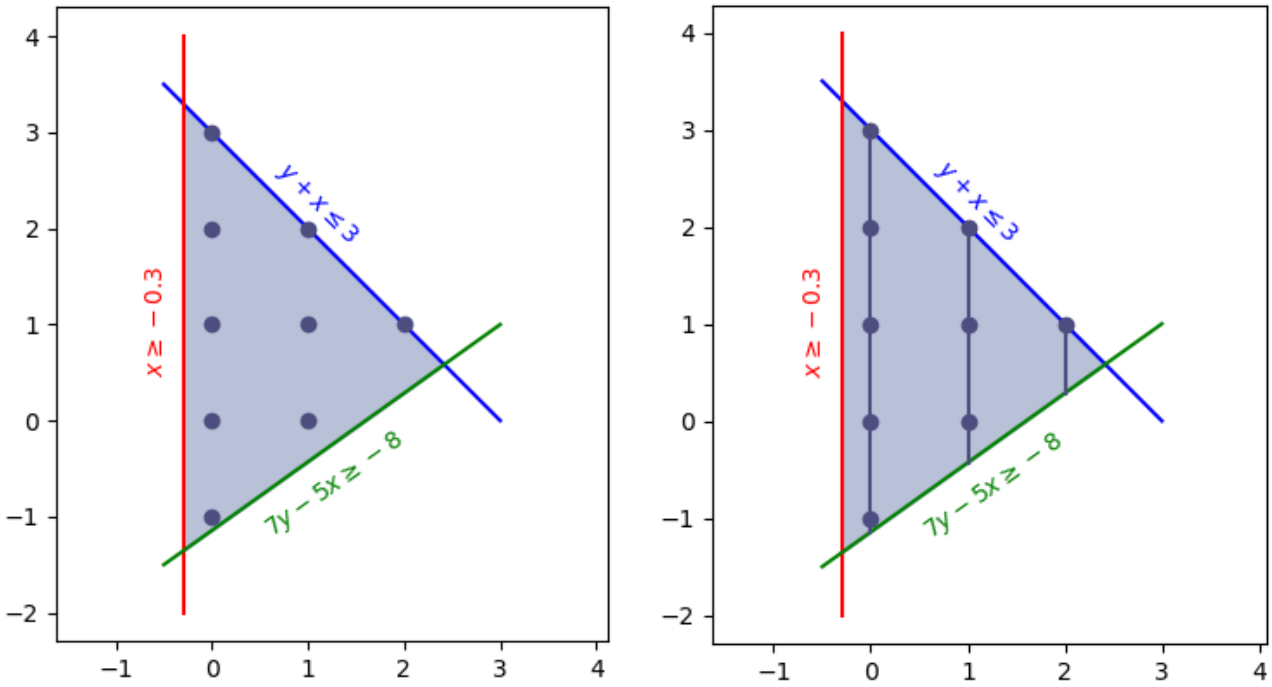
Besides its theoretical interests, the TSP also has a lot of practical applications, not only in planning and scheduling [9] where the problem first appeared but also in other domains such as DNA sequencing [10].

1.2 Mixed-Integer Programming

Mixed Integer Linear Programming (MILP) is a term that encapsulates a class of problems described by some linear inequalities over a set of variables among which a non-empty subset

must take integral values. It is the most common subclass of all MIP[1] problems, which can feature quadratic objectives or constraints, for instance. Geometrically, a linear inequality can be understood as a half-space (understand half-plane when there are two variables) in which the vector of variables needs to be. When there are several such constraints, the intersection of half-spaces defines a polytope (polygon in two dimensions) with possibly infinite bounds in some directions. The integrality constraints on some variables define points of integral coordinates in this polytope.

In Figure 1.1, we show the geometric representation of a 2-variable linear program. The



(a) (b)

Figure 1.1 An illustration of Linear Programming and (Mixed-) Integer Linear Programming

number of variables corresponds to the dimension of the space, so a 2-variable program can be represented in a plane. We call the two variables x and y, that are featured respectively in the horizontal and vertical axes. The colored area represents the set of all admissible points when we consider only the linear inequalities. By adding integrality constraints on both variables, we obtain the points of Figure 1.1a: instead of an infinity of solutions defined by every point of the area, only 9 admissible points remain, featured in bold in the figure.

[1]In this project, we only study linear programs, the rest of the paragraph focuses on programs with only linear constraints and objectives

In Figure 1.1b, we look at a mixed-integer linear program, where only x has to take integral values: the set of admissible solutions is defined by the three vertical lines. An optimization program looks at maximizing (or minimizing) a linear function over this set.
Several algorithms exist to solve continuous Linear Programming. The most famous one is the Simplex Algorithm, that has an exponential complexity in the worst case. On the other hand, its polynomial complexity in average [11] makes it a useful tool in practice. Algorithms that are polynomial in the worst case also exist: the polynomial complexity of the problem was proven in 1979 [12]. Most solvers implement the Simplex Algorithm rather than alternatives, even though some polynomial Interior Point Methods like the one in [13] are competitive with the Simplex Algorithm.
While Linear Programming on continuous variables can be solved in polynomial time and is therefore considered an easy problem, the introduction of integrality constraints makes the general resolution of MIP problems difficult. However, Linear Programming takes an important place in the resolution of general MIP problems. Algorithms such as branch-and-bound and branch-and-cut are widely studied to make these problems solvable in practice. Branch-and-cut algorithms consist of two different steps.

- Finding and adding cutting planes that can be visualized with the black line in Figure 1.2. A cutting plane is an additional linear constraint that removes a subset of the admissible polytope containing no integer point and thus helps refining the set of admissible points.

- Branching on subsets of the polytope: in Figure 1.2, the colored figure is separated into two separate subsets. In each of these subsets, a lower bound can be obtained by solving the linear relaxation; and a classical branch-and-bound exploration allows to recursively look for integral solutions.

It is of crucial interest to find the best model for any problem and reduce it as much as possible, as for exponential algorithms a small reduction of the problem can dramatically reduce the resolution time.
A generic formulation of a MIP problem is the following: we are given a matrix $A \in M_{m,n}(\mathbb{R})$ and two vectors $b \in \mathbb{R}^m, c \in \mathbb{R}^n$. The MIP in its general form can be written as

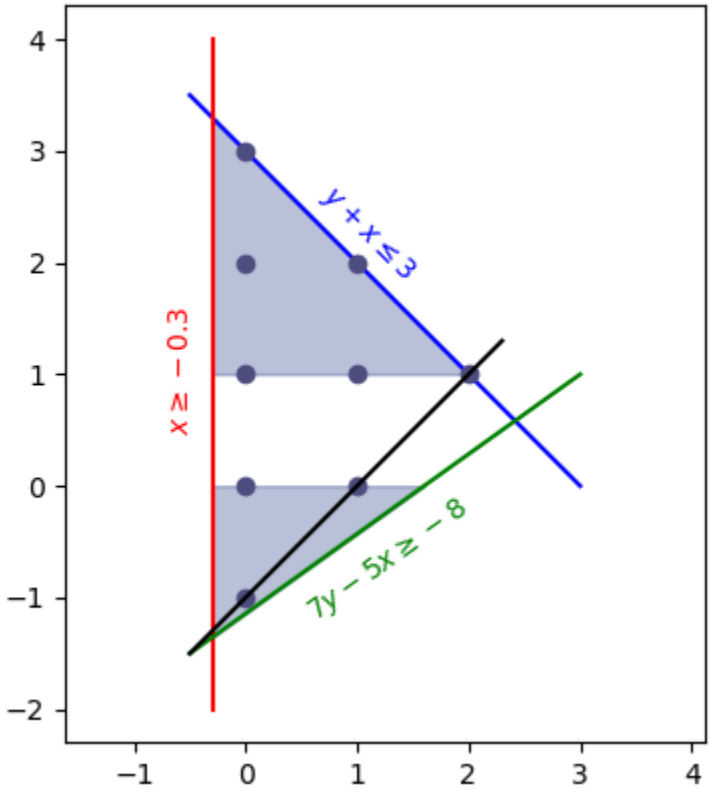


Figure 1.2 Branch-and-cut algorithms on a simplified model

$$\min_{x \in E} c^T x \tag{1.1}$$

$$s.t.\ Ax \geq b \tag{1.2}$$

$$x \in E \tag{1.3}$$

$$\tag{1.4}$$

where $E := \mathbb{Z}^k \times \mathbb{R}^l, k+l = n$. The inequality between vectors is to be understood coordinate by coordinate, and k represents the number of integral variables. We ordered the variables in the formulation to put all the integral variables first. Without loss of generality, we defined the objective as a minimization, but it can also be written as a maximization by changing the sign of the objective function.

1.3 Constraint Programming

Unlike Mixed Integer Programming, Constraint Programming is a very generic paradigm that encapsulates non-linear constraints and allows declarative programming with a wide variety of constraints. That means that the models are written in a high-level, easily understandable form without giving the details of the resolution method. We usually distinguish between two forms of Constraint Programs: Constraint Satisfaction Problems (CSP) [14] and Constraint Optimization Problems (COP) that include an objective function. COPs in general can be solved as a series of CSPs [15], which make the two problems strongly connected. The intuition is that a Constraint Program is made of a set of variables and several constraints on those that describe what a solution should look like, and a solution is simply a choice of value for the corresponding variables.
Formally, a constraint satisfaction problem is defined by a set of variables $V = (v_1, \dots, v_n)$ with respective domains D_i and a set of constraints $C = (c_1, \dots, c_m)$ that are subsets of the domains: $\forall j \in [|1, m|], c_j \subseteq D_1 \times \dots \times D_n$. Solving a constraint satisfaction problem is finding a variable-value assignment

$$f : V \to D \tag{1.5}$$

$$v_i \mapsto x_i \in D_i \tag{1.6}$$

such that $(f(v_1), \dots, f(v_n)) \in \bigcap_{1 \leq j \leq m} c_j$. A constraint optimization problem adds an objective function $g(x_1, \dots, x_n)$ to be minimized (resp. maximized): it can be seen as a series of constraint satisfaction problems with an additional constraint $g(f(v_1), \dots, f(v_n)) < Z$ (resp. $> Z$) where Z takes the successive values of found solutions. When the additional constraint makes the problem infeasible, the optimum value has been found.
A common example of everyday life problem easily modelled in Constraint Programming is the resolution of Sudoku grids. The objective of the player is to find a value to put in each cell of the grid. Each cell can be understood as a variable with domain $[|1, 9|]$. Abstract relationships on the cells make some assignments impossible, for instance putting two 1s in the same line. Each line, column and 3×3 cell corresponds to a constraint, and each constraint can be understood as encapsulating a different aspect of the problem's structure. The constraints in the problem are global constraints commonly known as `AllDifferent` constraints.
The two main aspects of solving Constraint Programs are constraint propagation [16], which is the incremental reduction of the domains often based on logical inference, and search.

When solving an easy Sudoku grid, you can usually remove some values from each cell's possibilities by looking at neighbouring cells: this is called propagation. However, it happens that for more difficult instances, you have to make a choice and see if it leads to an impossible situation. This is the search part, which needs a search heuristic to select the first affectation that will be tried in each situation. In the following, we will mainly focus on search, and especially finding variable and value selection heuristics, rather than propagation.

1.4 Relaxations and solution densities

The satisfaction problem associated with the Traveling Salesman Problem, i.e., the problem of answering "for a given weight k, is there a tour of weight $\leq k$?" was shown to be NP-complete in 1972 [17], which means there is to this day no exact algorithm able to solve the problem in polynomial time. In fact, it is believed that no such algorithm will ever exist. For a more precise (and fun) insight into what computer scientists think of this (unresolved) question, see [18].

Nonetheless, some related problems, that also look at optimal routes and travels, are easily solved in polynomial time. The difficulty in solving the TSP is that it combines several constraints. We must consider simultaneously that an edge is either included or not included in a solution and cannot be partially included (integrality constraint), we look at constructing one single cycle with no subtour, all of the nodes must be visited, etc. A relaxation of a problem is a related problem obtained by removing, or simplifying, one or several of these constraints. For instance: what happens if we allow several disjoint subtours instead of a single one? What happens if we allow our cycle to go through only a subset of the nodes?

The methods we look at in this research project aim at using relaxations of the problems to develop heuristic reductions and exact guiding of the resolution. The intuition is the following: is it reasonable to think that an easier but similar problem can give us a good idea of what (good) solutions look like?

In practice, the idea is to look separately at different aspects of the problem and see what itineraries are interesting for these easier subproblems. If the problem is easy enough, we can list all of the feasible solutions. Then, the predominance of some routes among the solutions set help us define a score we call Solution Density (SD) of an edge, that will be described more formally in Section 2.3.

1.5 Research objectives

This research aims at following in the footsteps of Pierre Coste in a previous Master's project [8]. We are looking to understand how a relaxation can inform a bigger problem's solutions. While the author of [8] focuses on the symmetric Traveling Salesman Problem, this thesis extends the research to variants of the TSP, namely the asymmetric and time-constrained variants. We adapt the relaxations and solution densities introduced in this former thesis, and investigate how its results can be applied to generalizations. We also aim at developing complementary solution densities for the additional constraints. One of the challenges of this adaptation is to make several relaxations work together and learn how to combine them in an effective way. This project will analyze different ways of combining solution densities for pre-processing and reducing the computational effort of solving MIP formulations, and will continue to use Counting-Based Search for branching heuristics in CP.

1.6 Thesis outline

In this document, we describe in Chapter 2 the literature around the problems and their formulations, as well as the state-of-the-art and the work previously done on relevant solution densities. In Chapter 3 we study the traveling salesman problem and its asymmetric variant; and then in Chapter 4 the time-constrained variant that represents the biggest contribution of this Master's. In each of these problems, we describe the method(s) used on several benchmarks and models, as well as the experimental results obtained. We finally conclude with a synthesis and future leads to continue this research topic.

CHAPTER 2 LITERATURE REVIEW

2.1 The Traveling Salesman Problem and its variants

2.1.1 Routing problems

Vehicle routing is a generic name for a family of problems (VRP) featuring *transportation requests* and *vehicles* that are thoroughly studied in computer science, both for their difficulty and for the gain in global transportation costs due to good resolution algorithms [19]. Vehicle Routing Problems were introduced in [20] in 1954 in the context of scheduling problems for fuel oil tankers.

Variants of the VRPs include restricting the number of available vehicles to 1 or a bounded number, limiting loading capacities for the vehicles, having variable transporting times as a function of the time of the day, adding time windows for the tasks, pickup-and-delivery constraints that force precedence between some tasks, or even more complicated constraints such as the break time needed for employees driving the vehicles in some more specific cases. These additional features usually come from the modelization of real-life transportation problems.

In this thesis, we focus on TSP-like problems, that allow only for one vehicle with no capacity constraints. The Traveling Salesman Problem in itself can be described as finding an optimal Hamiltonian cycle in a graph, i.e., a tour going through each vertex once while minimizing the sum of edge costs. If we suppose that costs in the graph respect the triangle inequality, which means that for all vertices i, j, k we have $d(i, k) \leq d(i, j) + d(j, k)$ where $d(i, j)$ is the weight of the edge from i to j, we can simply see it as a tour going exactly once through each node, or a permutation of the nodes. The triangle inequality also implies that all connected graphs are complete.

We will look more specifically at the Asymmetric Traveling Salesman Problem, in which weights from i to j and from j to i are allowed to have different values. It is the case for instance of networks with one-way roads. The TSP can be seen as a specific case of the ATSP where opposite edges both have the same weight. A transformation from the ATSP into symmetric TSP was introduced in [21] that proves that the two problems are in fact exactly equivalent. However, the transformation doubles the number of vertices, which means there is an interest for solving the ATSP in a direct manner.

The second variant that we will consider in this thesis is the Traveling Salesman Problem with Time Windows. In contrast with the straightforward TSP, time windows $[R_i, D_i]$ are added to each node with the additional constraint that each node must be visited during that

time interval . The Traveling Salesman Problem with Time Windows is in itself asymmetric, as the reversed cycle may not always be a feasible solution, but weights can be symmetric or not.

In general, for all of these problems, the distances $d(i, j)$ between two nodes i and j are arbitrary real numbers, but additional constraints on the distance function can give interesting information on the solutions. A few theorems are described in [22] for symmetric and Euclidean distances, as well as distance functions that do not have negative-weight cycles or respect the triangle inequality.

2.1.2 MIP formulations

Several linear formulations can be used to define the Traveling Salesman Problem and its variants. We will focus on two MIP models for the ATSP, but the survey [22] describes several other models for which thorough computational results were not reported.

Dantzig-Fulkerson-Johnson

Given a directed graph (V,A) and costs $C = (c_{ij})$ associated to the arcs $(i, j) \in E$ of the graph, we define a first MIP model [23]. We define a set of binary variables $(x_{ij})_{i,j \in V}$ representing each of the arcs included in the model. The constraints are the following:

$$\min \sum_{i,j} c_{ij} x_{ij} \tag{2.1}$$

$$s.t. \sum_{j \in \delta^+(i)} x_{ij} = 1, \quad i \in V \tag{2.2}$$

$$\sum_{j \in \delta^-(i)} x_{ji} = 1, \quad i \in V \tag{2.3}$$

$$\sum_{(i,j) \in A(S)} x_{ij} \leq |S| - 1, \quad S \subseteq V, \quad 2 \leq |S| \leq |V| - 2 \tag{2.4}$$

$$x_{ij} \in \{0, 1\}, \quad (i, j) \in A \tag{2.5}$$

where $A(S)$ is the set of arcs whose two endpoints are in the subset S. Here, $\delta^+(i)$ is the set of arcs going out of i and $\delta^-(i)$ the set of arcs going into i.

Constraints (2.2) and (2.3) are known as the degree constraints, since they can be interpreted in a graph as the imposition for each node to have two incident arcs, one going in and one out.

Constraints (2.4) are usually known as subtour elimination constraints. With only the other

constraints, several disjoint tours on subsets of the vertices would make an acceptable solution. These constraints remove all cycles that do not include all vertices, since for the subset S of vertices included in a cycle, there is at least $|S|$ arcs in $A(s)$.
The main drawback of the DFJ model is that there is an exponential number of subtour elimination constraints (an order of $\mathcal{O}(2^n)$). The model that follows is interesting because it does not have this weakness.

Miller-Tucker-Zemlin

The first constraints of the DFJ model remain in this second model [24], i.e.,

$$\min \sum_{i,j} c_{ij} x_{ij} \tag{2.6}$$

$$s.t. \sum_{j \in \delta^+(i)} x_{ij} = 1, \quad i \in V \tag{2.7}$$

$$\sum_{j \in \delta^-(i)} x_{ji} = 1, \quad i \in V \tag{2.8}$$

$$x_{ij} \in \{0, 1\}, \quad (i, j) \in E \tag{2.9}$$

The subtour elimination constraints are replaced by the following. Variables $u_i, i \in \{2, \ldots, n\}$ are introduced, as well as the constraints:

$$u_i - u_j + (n-1)x_{ij} \leq n - 2, \quad i \neq j \in \{2, \ldots, n\} \tag{2.10}$$

$$1 \leq u_i \leq n - 1, \quad i = 2, \ldots, n \tag{2.11}$$

Here, there is a linear number of subtour elimination constraints. If a tour $(v_1, \ldots, v_k, v_1)$ existed in the graph, then by adding up all of the constraints (2.10) relative to arcs (v_i, v_{i+1}) of the cycle, we would obtain a sum on all variables u_i where consecutive terms cancel each other. This would result in having the constraint $n - 1 \leq n - 2$, which is a contradiction.
This model is simpler to write and includes less constraints as the previous one, but it has a weaker linear relaxation. An analytical proof is available in [25]. By weaker linear relaxation, we mean that the set of feasible solutions contains the set of feasible solutions of the DFJ models, hence a higher value for the lower bound provided by the relaxation. That will make the branch-and-bround process less effective.

TSPTW: Big-M formulation

One possible model for the traveling salesman problem with time windows is a "Big-M" formulation [26,27] , which means it uses a variable M representing an infinite value. The model builds onto the MTZ formulation described above and replaces the subtour elimination constraints by time constraints.
Given a graph (V, A), costs $C = (c_{ij})$ as well as travel times $\Theta = (\theta_{ij})$ on the arcs $(i, j) \in A$ (we often have $\Theta = C$ if the cost we look to minimize is the traveling time), time windows $[R_i, D_i]$ on each request $i \in V$, we have two additional nodes p and q so that a cycle is mapped to a path from p to q.
We add binary variables x_{ij} representing arcs in the cycle, and variables s_i representing the starting time of the vertex $i \in V$ in the cycle. The Big-M formulation is defined as such:

$$\min \sum_{i,j \in A} c_{ij} x_{ij} \tag{2.12}$$

$$s.t. \sum_{j \in \delta^+(i)} x_{ij} = 1 \quad \forall i \in V \backslash \{q\} \tag{2.13}$$

$$\sum_{j \in \delta^-(i)} x_{ji} = 1 \quad \forall i \in V \backslash \{p\} \tag{2.14}$$

$$s_i + \theta_{ij} - (1 - x_{ij}) M_{ij} \leq s_j \quad \forall (i, j) \in A \tag{2.15}$$

$$R_i \leq s_i \leq D_i \quad \forall i \in V \tag{2.16}$$

$$x_{ij} \in \{0, 1\} \tag{2.17}$$

where $M_{ij} = D_i - R_j + \theta_{ij}$. This construction ensures that the constraints (2.15) do not restrain the domain for arcs (i, j) with $x_{ij} = 0$. It is thus an "infinite" value in the sense that it is sufficiently large compared to the magnitudes of other time variables. When $x_{ij} = 1$, the constraint translates to $s_i + \theta_{ij} \leq s_j$, which means that the times needed to travel must be respected between the visiting times. It also serves as a cycle elimination constraint: since we are looking for a simple path from p to q, the constraints must eliminate cycles of any length. Suppose there is a cycle $i_1 \to \ldots \to i_k \to i_1$, then the constraints (2.15) with indexes i_j, i_{j+1} for all $j \leq k$ sum to

$$\sum_{i=1}^{k} \theta_{j,j+1} \leq \sum_{j=1}^{k} (s_{j+1} - s_j) = s_{k+1} - s_1 = 0 \tag{2.18}$$

which is only possible with a cycle of negative weight, discarded from the model since it is incompatible with the concept of optimal tour.

We also see that the inequality in the constraints (2.15) instead of an equality means we allow "waiting", i.e., the arrival time s_i at a node i could be anterior to the release time R_i. The waiting period $R_i - s_i$ does not appear in the cost of the tour. The model does not include processing times, as we can adapt any model to have zero processing times by including them into the traveling costs.

Other MIP models for the TSP with Time Windows

The previous model is described in [26] as a first simple model in the construction of a more complicated one called a Time-Bucket Formulation, that builds on the time-indexed formulation from [28]. Indeed, a lot of work has been done on constructing more compact formulations or MIP models that have a stronger linear relaxation (meaning a tighter lower bound).
In this thesis, we focused on the Big-M model from Section 2.1.2, as the goal is not to select an optimal MIP model but to work on reducing the effort of solving the problems by using specific tools. However, we can cite other formulations that can be of interest for comparison in future works, since the results reported here are dependent on the model used for solving. Besides our model of interest, Ascheuer et al. [27] report another model presented in the articles [29, 30]. Instead of Big-M constraints, this second model includes additional integral variables on arc y_{ij} where $y_{ij} = 0$ when $x_{ij} = 0$ and y_{ij} represents the start time of processing of node i otherwise. Constraints (2.15) and (2.16) are replaced with

$$\sum_{i=1;i\neq j}^{n} y_{ij} + \sum_{i=0;i\neq j}^{n} \theta_{ij} x_{ij} \leq \sum_{k=0;k\neq j}^{n} y_{jk}, \qquad \forall j \in V \tag{2.19}$$

$$R_i x_{ij} \leq y_{ij} \leq D_i x_{ij}, \qquad \forall i \neq j, i \neq 0 \tag{2.20}$$

In the same way that the previous model did, the constraints (2.19) and (2.20) are a way of imposing that the travel times and the time windows are respected with a linear constraint. Since only one x_{ij} and the corresponding y_{ij} are non-zero for each j, the constraint (2.19) translate to

$$x_{ij} = 1, x_{jk} = 1 \rightarrow y_{ij} + \theta_{ij} x_{ij} \leq y_{jk}$$

and Equation (2.20) to y_{ij} being in the right time window.

2.2 State of the art

2.2.1 State of the art for the TSP

The generally admitted state of the art for the Traveling Salesman Problem is Concorde [6,31], a branch-and-cut-based software that very effectively solves Euclidean symmetric instances of 10000s of nodes, and outperforms other solvers on all symmetric instances. While the underlying engine behind Concorde is Integer Programming, it works in practice as a black box, and cannot accept additional constraints. It also requires a specific structure for the problems by accepting only full matrix inputs, which correspond to complete graphs. For other sets of instances, namely variants or sparse graphs, Constraint Programming and MIP models become of interest because of their adaptability. Local search techniques have also proven their efficiency, for instance the Lin-Kernighan heuristic [32].

2.2.2 State of the art in Constraint Programming

Weighted Circuit Constraint

The state of the art for the TSP in Constraint Programming, as described in [33], is encapsulated in the Weighted Circuit Constraint (WCC) defined over a weighted graph with an upper bound Z as input. The WCC ensures that the selected edges form a cycle of weight bounded by a given value. In general, the cost is expressed as a linear function of the variables, which complicates the filtering the domains, as the only upper bound on each variable is the whole cost. Including the upper bound in the cycle constraint allows to repair this issue.

The circuit is represented as a variable S in the form of a set containing edges [34], with a lower and upper bound on the set updated during the propagation. The lower and upper bounds on the set are respectively the set M of mandatory edges that need to be included in any solution, and the whole set without a set I of impossible edges: $M \subseteq S \subseteq E \backslash I$ if E is the complete set of edges in the graph.

The method used to update M and I, or **filtering** relative to this constraint, is inspired by the Lagrangian Relaxation of the degree constraints introduced in [35]. The degree constraints are removed and added into the objective, reducing the problem to the easier task of finding an optimal 1-tree, structure that will be defined more thoroughly in Section 2.3. The principle of the Lagrangian Relaxation, often used in Integer Programming [36], is to divide the problem into two sets of constraints, one of which is included in the objective with a set Λ of multipliers: the problem

$$\min_x f(x) \tag{2.21}$$

$$s.t.\ x \in C_1 \tag{2.22}$$

$$x \in \bigcap_{C_i \in C_2} C_i \tag{2.23}$$

$$\tag{2.24}$$

becomes

$$\min_x f(x) + \sum_{C_i \in C_2} \lambda_i d(x, C_i) \tag{2.25}$$

$$s.t.\ x \in C_1 \tag{2.26}$$

$$\tag{2.27}$$

where C_2 is the subset of relaxed constraints, for instance the degree constraints in our specific case. You can see from the difference between the two models that the constraints $C_i \in C_2$ do not appear as constraints anymore. A value of x respecting those constraints is a solution of the Lagrangian Relaxation with the same value for the objective as in the original problem. The values λ_i represent the cost paid to be able to dismiss the corresponding constraints. This structure allows the authors to define marginal costs on edges, i.e., the minimum increase in weight in order to obtain a 1-tree where a specific edge is selected (resp. is removed). The upper bound Z is used to include edges whose marginal cost for selection (resp. for deletion) is too high into the set I of impossible (resp. in M) edges. While this algorithm can theoretically be adapted to work on asymmetric instances, the article [34] mentions that using the transformation to the symmetric TSP [21] is actually more effective.

Search strategies

On top of propagation, constraint programming uses a search tree: one value is selected for a variable (more generally, we can select an additional constraint to add to the model) until a solution or a contradiction is found. The search then backtracks and explores a different possibility. Each investigated possibility is a branch of the search tree. We call search strategy the choice of the value that will be chosen at each step. Search strategies can usually be separated into:

- What variable do we branch on?
- What value do we select for it?

The state of the art consists of, in combination to the above formulation of the constraints, a very effective selection strategy: the LCFirst strategy described in [37]. This heuristic branches on one of the endpoints of the last selected edge, thus adding edges incrementally to a path.
Several value selection strategies are cited as good search strategies, while none of them seems to be significantly better than the others.

k-Cutsets

An additional filtering relying on the structure of the Hamiltonian cycle is introduced in [38]. Isoart et al. make the observation that a bridge, i.e., a single edge disconnecting the graph, makes it impossible for any tour to exist in the graph. Similarly, a pair of edges disconnecting the graph into two subsets A and B need to be in any tour, since any cycle needs to go from A to B and back from B to A. Those simple observations give way to a few elimination and selection techniques, relying on k-cutsets, those sets of k edges disconnecting the graph. More generally, any cutset includes at least 2 edges of any solution, and always in an even number: for instance, exactly two edges of any 3-cutset must be selected. In practice, the propagator of [38] only uses cutsets of size $k = 1, 2, 3$ that can be found using Tsin's algorithm [39]. Tsin's algorithm finds all 2-cutsets in $\mathcal{O}(m+n)$. For each edge e, a 2-cutset (e_1, e_2) in $G\backslash\{e\}$ can be mapped to a 3-cutset (e, e_1, e_2) in G, which makes it possible to compute all 2 and 3 cutsets in $\mathcal{O}(m(n+m))$ by iterating on the choice of e. Not all sizes of cutsets can be used as the complexity of the enumeration relies heavily on the value of k [40].
The k-cutsets properties can easily be translated for asymmetric TSP instances, and the same algorithm without modifications can be used to find cutsets in an oriented graph. However, the solver developed by Isoart & Régin [38] relies on the Weighted Circuit Constraint and does not yet take the asymmetric case as direct input.

2.2.3 State-of-the-art with Time Windows

The report [41] compiles an overview of the state-of-the-art for the general Vehicle Routing Problem with Time Windows in 2010. Not focused on Integer Programming, it includes mostly heuristic and metaheuristic methods based on local search and large neighbourhood, quoted as the current best option for solving large instances. In 2020, the state-of-the-art for

solving the Traveling Salesman Problem with Time Windows with an exact algorithm seems to be [42], a method that combines dynamic programming and column generation: ever since the 1980s, column generation has been used to solve vehicle routing problems [43].

2.3 Computing solution densities

2.3.1 Relaxations and solution densities

A relaxation, in its general form, is a space of feasible solutions containing the problem that we look at. It generally consists of removing some constraints of the problem. One of the main interests of a relaxation is to remove problematic constraints that make the computation hard, in order to obtain a simpler (polynomial) problem and thus having a lower (or upper if we are maximizing) bound on the value of the objective. We gave in Section 2.2.2 the case of Lagrangian relaxations, where the removed constraints are included in the objective with a multiplicator. An infinity of relaxations are created this way that are parameterized by the vector λ of Lagrangian multipliers. Another usual family of relaxation are Linear Relaxations, where we relax the integrality constraints. For instance, replacing binary domains $\{0, 1\}$ by real intervals like $[0, 1]$[1].
Lower and upper bounds can be used in branch-and-bound algorithms [44], for filtering in constraint propagation [34, 45], or even for theoretical bounds on an approximating algorithm. Counting-Based Search, a paradigm of search heuristics described below, aims at using relaxations to guide the search of solutions in constraint programming.
A family of usual relaxations are generally used in algorithms for the Traveling Salesman Problem, including the 1-tree and 2-matching relaxations described below, or the n-path relaxation [46].

2.3.2 Counting-Based Search

Counting-based search strategies, as introduced in [47], are especially defined for satisfaction problems. They focus on guiding the search by counting feasible solutions, using the probability for one variable-value assignment to feature in a solution of a unique constraint. We call these probabilities *solution densities* (SD).
For each constraint $c(x_1, \ldots, x_k)$ with a number of solutions $\#c(x_1, \ldots, x_k)$, the solution density for the assignment $x_i = d$ is defined as

[1] This is an interval, not a citation, even though they look alike

$$\sigma(c, x_i, d) = \frac{\#c(x_1, \ldots, x_{i-1}, d, x_{i+1}, \ldots, x_k)}{\#c(x_1, \ldots, x_k)} \tag{2.28}$$

The solution density is thus exactly the proportion of solutions where x_i takes the value d. For optimization problems, the concept is generalized to that of *cost-based solution densities* [7] that take into account the cost function f we look to minimize. Since we are working with optimization problems only, in this work Solution Density (SD) will be used to designate Cost-Based Solution Density. With $\varepsilon > 0$ a small real number and $\#c_\varepsilon(x_1, \ldots, x_k, z, f)$ the number of solutions of cost $z \leq (1 + \varepsilon) \min f$, the formalism becomes

$$\sigma(c, x_i, d, \varepsilon) = \frac{\#c_\varepsilon(x_1, \ldots, x_{i-1}, d, x_{i+1}, \ldots, x_k, z, f)}{\#c_\varepsilon(x_1, \ldots, x_k, z, f)} \tag{2.29}$$

When $\varepsilon = 0$, it corresponds exactly to counting only optimal solutions. Adding the parameter allows to look at near-optimal solutions that can be interesting to consider when we describe a subspace that contains "good" solutions.

Since these solution densities are defined with respect to a single constraint or a subset of the constraints, enumerating the solutions can be a lot less time-consuming than finding solutions for the whole problem: the idea is to look at relaxations that are easy to compute. However, for specific constraints, we can find faster computation methods that do not need to effectively enumerate all the solutions to count them. We can also simply use approximations of the number of solutions.

2.3.3 1-tree relaxation

Structure

A 1-tree is a sub-graph (V', E') of (V, E) such that $V' = V$ and E' is a spanning tree over $V \backslash \{1\}$ plus two edges incident to 1. From this construction, we see that a hamiltonian cycle is always a 1-tree, which makes the weight of an optimal 1-tree a lower bound on the solution of the TSP. We show a representation of one solution of the 1-tree relaxation in Figure 2.1, where node 1 is the closest point to the top of the figure. The 1-tree relaxation coincides with relaxing all the degree constraints except for the vertex 1, but keeping the less restrictive constraint of the total number of edges. Just like a Hamiltonian cycle, a 1-tree structure contains exactly n edges.

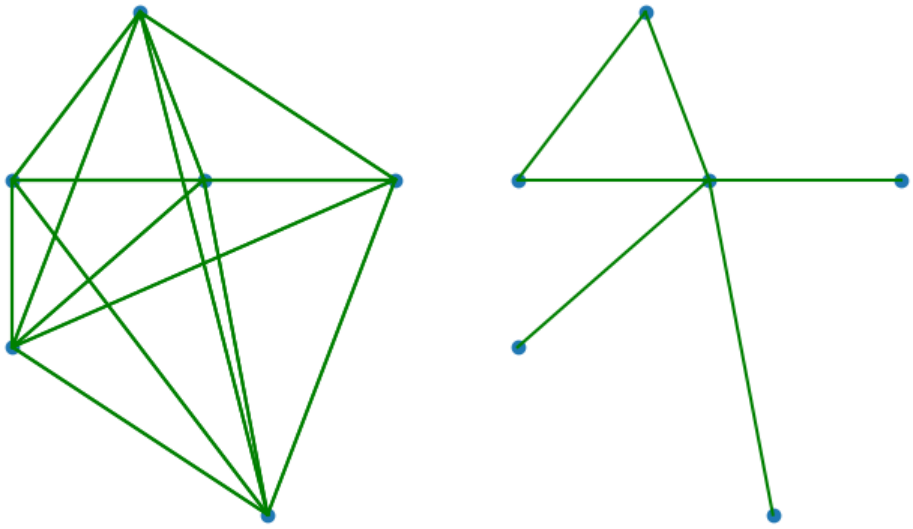


Figure 2.1 Complete graph vs. a solution of the 1-tree relaxation. On the left we see a complete graph on 6 vertices, where vertex 1 is the vertex closest to the top. On the right, we have a solution of the 1-tree relaxation: this particular vertex is of degree 2 and the rest form a spanning tree. We see that there are exactly 6 edges in the right figure.

Counting with the Matrix-Tree theorem

In the thesis [8], the computation method for counting the number of 1-trees over a graph relies in fact on counting regular spanning trees over a graph of vertices $V \cup \{v\}$ where v is a copy of Vertex 1. This structure is also a valid relaxation of Hamiltonian cycles, and it is almost a 1-tree under the assumption that vertices v and 1 have a degree of one in the spanning tree, and are not connected to the same vertex or to each other. In practice, this can generally be achieved by selecting a vertex that maximizes its distance to other vertices. This slight modification allows to effectively count the solutions to the relaxation since the Matrix-Tree theorem [48] gives an exact expression for the number of spanning trees.

The Matrix-Tree theorem states that the number of spanning trees can be obtained in computing the determinant of a matrix in $M_n(X)$. In practice, we will write it as a matrix in $M_n(\mathbb{R})$ by identifying the indeterminate with a value in the interval $[0, 1]$.

More precisely, if we call $w(i, j)$ the weight of the edge (i, j) in the graph, let us denote L the Special Laplacian matrix in $M_n(X)$ such that

$$L_{ij} = \begin{cases} -X^{w(i,j)} & \text{if } i \neq j \\ \sum\limits_{k \neq i \in [1,n]} X^{w(i,k)} & \text{if } i = j \end{cases} \tag{2.30}$$

and $L^{(i)}$ the submatrix of L where the line and column of index i have been removed. Then, for all i, the theorem states that $\det(L^{(i)}) = \sum_k a_k X^k$ where a_k is the number of spanning trees of weight k. It means that for all i between 1 and n, the value of $\det(L^{(i)})$ is the same. By replacing the indeterminate X by a real value between 0 and 1, we control the decay applied to bigger weights, since with $x = 1$ the equation gives the total number of trees. When computing, we replace X by a real value from the start and not only by evaluating the polynomial obtained in computing the determinant.
The next step is to compute the 1-tree density corresponding to each edge e, which is by definition the proportion of 1-trees containing that specific edge. We can write

$$d(e) = 1 - \frac{\det(L_e^{(i)})}{\det(L^{(i)})} \tag{2.31}$$

where L_e is the Special Laplacian Matrix associated with the graph $(V \backslash \{e\}, E)$
Since L_e differs from L by only four coefficients, we can use the Sherman-Morrison formula to simplify this expression: by choosing for each edge i as one of its incident vertices, the two sub-matrices only differ by one coefficient. We recall here the Sherman-Morrison formula [49] for a matrix A and two vectors u, v

$$\det(A + uv^T) = (1 + v^T A^{-1} u) \det(A) \tag{2.32}$$

The final expression for $e = (i, j)$ is

$$d(e) = 1 - {L^{(i)}}^{-1}_{jj} x^{w(e)} \tag{2.33}$$

In this final equation and in the adaptation we will see in the next Chapter, the indexes in the submatrix are adapted with the removal of a row and column. For instance, here, if $i < j$, the coordinates of the coefficient in ${L^{(i)}}^{-1}$ is $(j-1, j-1)$ so that it corresponds to (j, j) in the original matrix. These transformations are omitted from the expression to keep things simple, but should be taken into account when implementing it.
We see that we only need one matrix inversion for each edge. In this case, computing the densities for a subset of the edges only demands the inversion of the matrices of a vertex cover of these edges: in particular, updating the edges going into one single node can be done in $\mathcal{O}(n^3)$ by inverting a single matrix. A non-optimal vertex cover can easily be found with a greedy algorithm.

2.3.4 2-matching relaxation

Definition

The 2-matching relaxation is the relaxation of the subtour elimination constraints (Equations (2.4), (2.10) and (2.11) in our models). By keeping only degree constraints, we can obtain any number of disjoint cycles in the graph. The problem of finding a 2-matching structure can be understood as an assignment problem if we see it as finding a successor/predecessor for each node. This version of the problem will be useful in the following. In Figure 2.2, we represent a solution of the 2-matching relaxation on a graph of 6 vertices: without the subtour elimination constraints, we can obtain a graph of two separate subtours without connectivity.

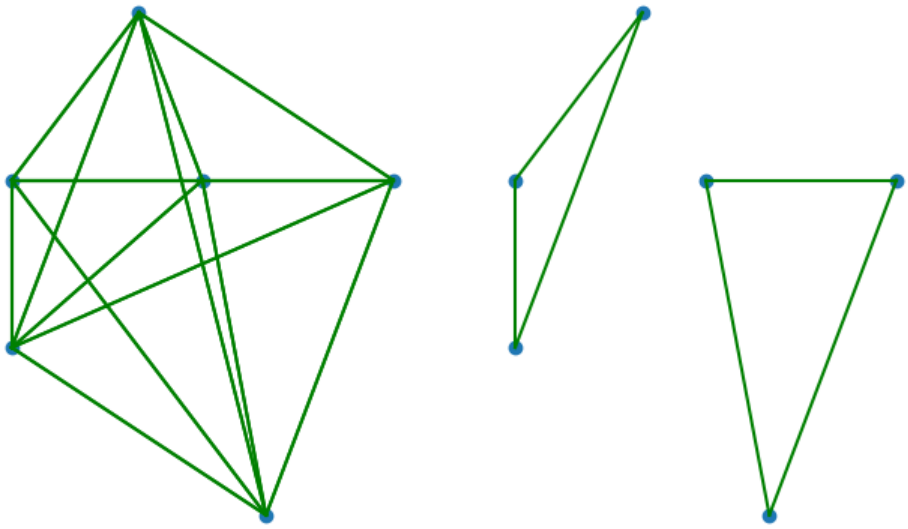


Figure 2.2 Complete graph vs. a solution of the 2-matching relaxation. The figure in the left represents a complete graph on 6 vertices, and the right figure represents a solution of the 2-matching relaxation in this graph. Since 2-matching is the relaxation of the subtour elimination constraints, we see that we obtain a solution with two separate subtours on 3 vertices, but the degree constraints are respected.

Counting as Permanent of a matrix

The adjacency matrix $A = (a_{ij})_{i,j \leq n}$ of a graph is defined by $a_{ij} = 1$ iff an edge (i, j) exists in the graph and $a_{ij} = 0$ otherwise. A matching in such a graph is a permutation $\sigma : [1, n] \to [1, n]$, since there are as many variables as values and all values must be different.

The value

$$\prod_{i=1}^{n} a_{i\sigma(i)} \tag{2.34}$$

is equal to 1 if and only if every $a_{i\sigma(i)}$ is equal to 1, which means that the permutation respects the existence of the edges. Starting from that, if we call P the set of permutations that correspond to acceptable assignments, we have

$$|P| = \sum_{\sigma \in S([1,n])} \mathbb{1}_{\sigma \in P} = \sum_{\sigma \in S([1,n])} \prod_{i=1}^{n} a_{i\sigma(i)} \tag{2.35}$$

where $S([1, n])$ is the set of all permutations of $[1, n]$.
By definition, this value is the permanent of the matrix A. While the definition of the permanent is very close to the determinant of a matrix, the algorithms that allow a $\mathcal{O}(n^3)$ complexity for computing the determinant do not extend to the permanent due to the absence of alternating signs. Valiant [50] shows that computing the permanent of a matrix is in fact #P-complete, which means it is at least as hard as any problem of the class #P of problems that consist in counting the number of solutions for a NP problem.
Pesant [51] introduces a method for computing the cost-aware solution densities of this constraint. An approximation is made by using several usual upper bounds on the permanent of matrices with non-negative values. The algorithm relies on computing these upper bounds for a modified matrix defined with the reduced costs obtained in the Hungarian Algorithm [52], which can be interpreted as the amount we are willing to gain on the objective in order to select a variable-value assignment.

Acceleration

On top of the approximations based on upper bounds, [8] mentions a high correlation between high-ranking edges for the 2-matching solution density and edges with a reduced cost of 0. The reduced costs being much faster to compute than the approximated solution densities by several orders of magnitude, it can be interesting to replace the SD by reduced costs for pre-computation. Since the same edges have good rankings for the 2-matching SD and for reduced costs, we can use reduced costs to sparsify our instances.

2.3.5 Ordering time windows

In the time-constrained TSP, we often have two disjoint time windows that impose a partial order between two nodes. If we have two nodes i and j with respective windows $[a_i, b_i]$ and $[a_j, b_j]$, and that we have the additional condition

$$a_i + d(i, j) > b_j$$

we know for sure that any path going through i during the corresponding time window will arrive at j after the end of the deadline. Consequently any tour that respects the time constraints needs to go through j before i: we denote that as $j \prec i$ and we can see this set of constraints as an order relation on vertices $\mathcal{R} = \{(i, j) | i \prec j\}$
Cherkaoui [53] shows how linear extensions of partial orders, i.e., complete orders on the set that respect the partial orders, can be counted. He especially develops an approximation algorithm to compute the probability that an ordering constraint $i \prec j$ is found in the different extensions of the order, based on the importance of the reconfiguration if the constraint is added to $\mathcal{R}$. Formally, the probability p_{ij} that $i \prec j$ is computed with the following equation.

$$p_{i^*j^*} = 1 - \frac{|\mathcal{R} \cup (i^*, j^*)| - |R|}{2(|\mathcal{R} \cup (j^*, i^*)| - |R|)} \tag{2.36}$$

Here, R is the set of precedence constraints, and $R \cup (i, j)$ is the set where we add the precedence (i, j) but also all of the constraints that are added by transitivity. We write $L[i]$ the set of elements that are lower or equal to i in the partial order, and $U[j]$ the elements greater or equal than j. We compute $|\mathcal{R} \cup (i^*, j^*)| - |R|$ by observing that

$$|\mathcal{R} \cup (i^*, j^*)| - |R| = |L[i^*] \times U[j^*] \cap \overline{\mathcal{R}}| \tag{2.37}$$

We call (i^*, j^*) the tuple among (i, j) and (j, i) that minimizes this quantity, hence implying the lowest amount of change in the partial order, and compute the reverse with $p_{j^*i^*} = 1 - p_{i^*j^*}$: in a total order, we must have $i \prec j$ or $j \prec i$ and the two are incompatible, so the sum of probabilities must be equal to 1. These probabilities p_{ij} can only be equal to 1 if we already have $i \prec j$ and to 0 if we already have $j \prec i$ in the partial order.
An exact algorithm using dynamic programming that can be exponential in the size of the largest antichain (set of incomparable elements) is also described: even though the thesis shows an empirical correlation between the exact and the approximate algorithms, we will eventually experiment with the exact algorithm on smaller instances to evaluate the quality of the approximation.

CHAPTER 3 USING SOLUTION DENSITIES TO SOLVE THE SYMMETRIC AND ASYMMETRIC TSP

In this chapter we study the adaptations from the TSP to the asymmetric TSP. We will first describe how we adapt and how we compute several solution densities. We will describe the models that we use in our experiments and then the methods that we introduce to help solve the (A)TSP with solution densities. Finally, we report the experimental results for these different methods.

3.1 Computing the asymmetric Solution Densities

3.1.1 Adaptation of the symmetric 1-tree Solution Density

In Section 2.3.3, we defined the solution density relative to symmetric instances. The same Matrix-Tree Theorem can be applied to the directed case.
In the literature, we often call 1-arborescence a 1-tree (see Section 2.3.3) in a directed graph: the difference is that the orientation of the arcs matters, with one vertex being the root of the arborescence. Instead of having $\det(L^{(i)})$ constant over all $i = 1, \ldots, n$, we still have $\det(L^{(i)}) = \sum_k a_k X^k$, but here a_k is the number of spanning trees rooted at i. In the case of the ATSP, we will simply count the number of spanning trees rooted at each vertex and add them together to have the total number of rooted spanning trees.
The total number of spanning trees is simply $\sum_{i\in[1,n]} \det(L^{(i)})$.
This means we can re-write Equation (2.31) as

$$d(e) = 1 - \frac{\sum_{i\in[1,n]} \det(L_e^{(i)})}{\sum_{i\in[1,n]} \det(L^{(i)})} \tag{3.1}$$

Since, in an asymmetrical instance, removing an arc (i, j) does not affect the coefficients of the arc (j, i), only two coefficients change in the overall matrix between L and L_e. With only one coefficient changing, we can use the Sherman-Morrison formula again. The final computation is

$$d(e = (i, j)) = 1 - \frac{\sum_{k\leq n} \det(L^{(k)})(1 + x^{w(e)}\delta(i, j, k))}{\sum_{k\leq n} \det(L^{(k)})} \tag{3.2}$$

We call $\delta(i, j, k)$ in the equation the value

$$\delta(i,j,k) = \begin{cases} {L^{(k)}}^{-1}_{ji} - {L^{(k)}}^{-1}_{jj} & \text{if } i \neq j \neq k \\ -{L^{(k)}}^{-1}_{jj} & \text{if } i = k \neq j \\ 0 & \text{if } j = k \end{cases} \tag{3.3}$$

where ${L^{(k)}}^{-1}$ is the inverse matrix of $L^{(k)}$. We recall that the indexations are relative to the coordinates in the original matrix L, meaning that if $k < i$ or $k < j$ the corresponding indices are shifted by one.
With this equation, we see that in order to compute all of the densities, we need to compute the inverse of n matrices of size n, which has a total complexity of $\mathcal{O}(n^4)$.
All of the densities depend on all of those inverses. Here, computing all of the densities or just a subset always involves n matrix inversions, whereas in the symmetric case, updating the densities of all of the arcs incident to a vertex can be done in $\mathcal{O}(n^3)$.

3.1.2 Normalizing weights

There are two issues for the computation of the 1-tree SD, with both high and low weights. Indeed, since the computation results in evaluating a polynomial of high degree in a value between 0 and 1, low weights may result in high values in the terms of the sum, while high weights may cause degenerate values and instability in the computation. Since the solutions do not change with an affine transformation of the weights, a normalization of weights can be done by applying

$$\begin{aligned} f : [min_{old}, max_{old}] &\to [min_{new}, max_{new}] \\ w &\mapsto \frac{max_{new} - min_{new}}{max_{old} - min_{old}}(w - min_{old}) + min_{new} \end{aligned}$$

Empirically, a weight interval of $[8, 10]$ is practical for correctly computing solution densities for instances of size 300 which is the maximum size in our benchmark. The operation of transforming the weights is linear in the number of arcs, or quadratic in the number of vertices, which is of negligible cost compared to the rest of the SD computation.

3.1.3 Choice of x for the 1-tree SD

As described in the section on the 1-tree density, the determinant of the Special Laplacian Matrix is linked to the weight of spanning trees with the relation

$$\det(L^{(i)}) = \sum_{k \in \mathbb{N}} a_k X^k \tag{3.4}$$

where a_k is the number of spanning trees rooted in i of weight k. However, we do not work with matrices of indeterminates but simply with float matrices by evaluating X in a number x between 0 and 1. With $x = 1$, the equation becomes

$$\det(L^{(i)})(x = 1) = \sum_{k \in \mathbb{N}} a_k \tag{3.5}$$

the total number of spanning trees. Otherwise, the polynomial expression can be factored in

$$\det(L^{(i)})(x) = x^{k_{min}} \sum_{k \in \mathbb{N}} a_k (x^{k - k_{min}}) \tag{3.6}$$

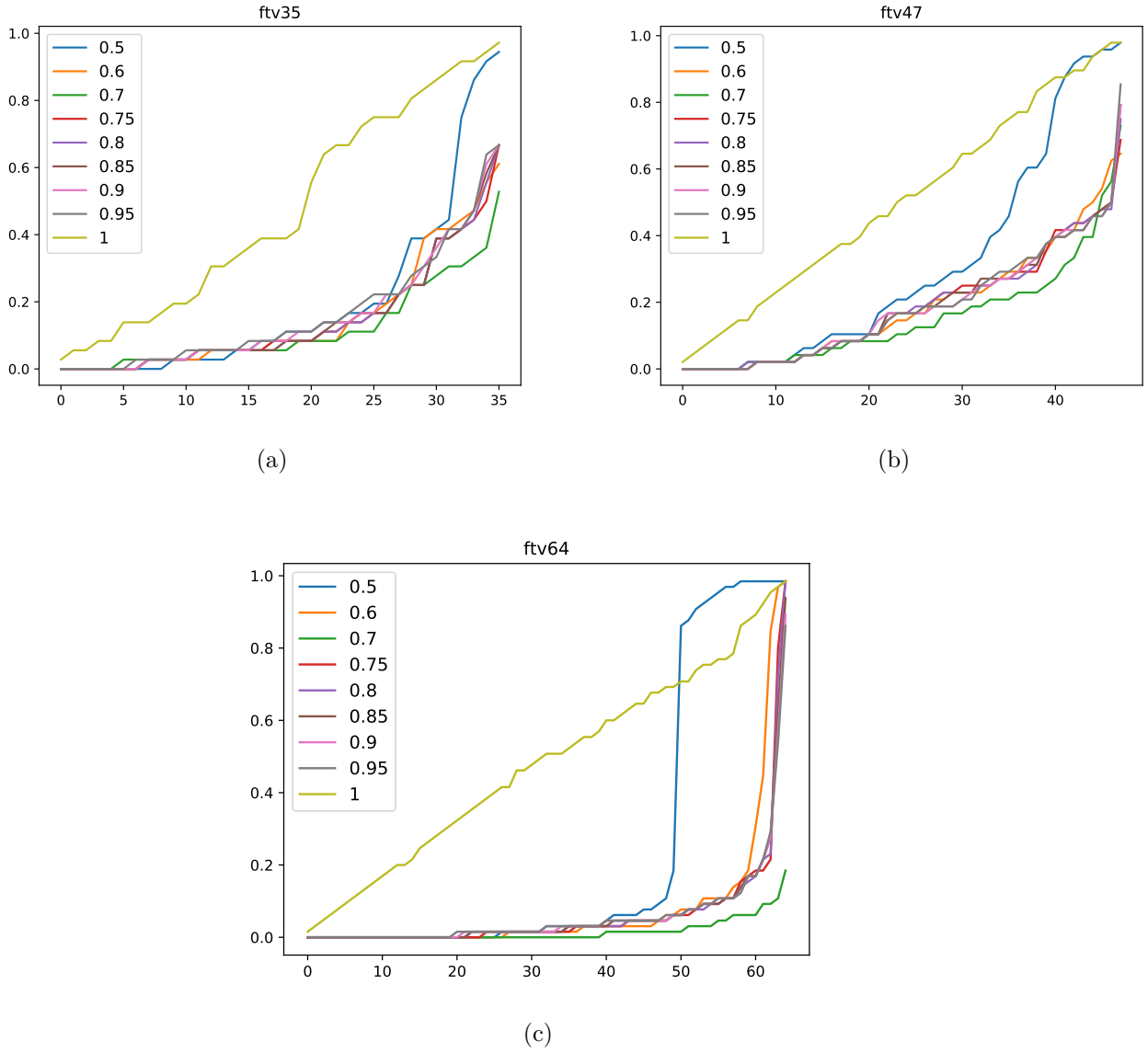
When $x \to 0$, we have

$$\frac{\det(L^{(i)})(x)}{x^{k_{min}}} \xrightarrow[x \to 0]{} a_{k_{min}} \tag{3.7}$$

The value of x can thus be understood as a parameter controlling the importance of the weight in the solution density. The closer x is to 0, the more trees of high weight are discounted. The instances that are found in usual benchmarks are complete graphs, which makes the value $x = 1$ uniform among arcs. We need a value $x < 1$ to make our SD interesting. In Figure 3.1, we look at a few instances from the `TSPLib` ATSP benchmark and observe the distribution of the ranks of the arcs in an optimal solution when the value of x varies.

In all three graphs, what we observe is the rank of the arcs of one optimal solution for the corresponding instances: the ranks are sorted in increasing order to observe the distribution more clearly, and normalized between 0 and 1. We see that one line corresponding to the extreme value of $x = 1$ stands out of the rest and approximates (even more so when the number of nodes grows) the diagonal of the graph. This diagonal corresponds to the fact that the instances are complete, which makes the value of the solution density uniform, and the ranks uniformly distributed.

The goal is for the rank of our optimal arcs to be as close to 0 as possible: we look at the parameter corresponding to the lowest curve in the figure. Here, we see that the graph of $x = 0.7$ seems to be achieve good performances for these instances. If the weights were not normalized before the computation, any value $x < 1$ would result in a quick decay towards 0 with high weights. With these high values, the optimal curve is obtained with the parameters close to $x = 1$. Indeed, a parameter close to 1 allows more variety between the different values,



(a) (b)

(c)

Figure 3.1 Distribution of ranks for arcs of an optimal tour for different values of x between 0.5 and 1. (a) is the instance `ftv35`, (b) is the instance `ftv47` and (c) is the instance `ftv64`. For each vertex, we look at the outgoing arc in an optimal solution and plot its normalized rank as ordinate. A value of 0 means the optimal arc was the best ranked among those going out of the same vertex. We organize the ranks in increasing order instead of by index of the vertex to make the graphs more readable. An optimal configuration is when the curve is as low as possible.

since they are not all equal to 0. To be able to find a real value for our parameter, we need to normalize the weights. The optimal value of x will depend on the chosen normalization. However, a value of 0.7 to 0.8 often gives good results. We will use $x = 0.7$ unless specified

otherwise.

3.1.4 Choice of ε for the 2-matching heuristic

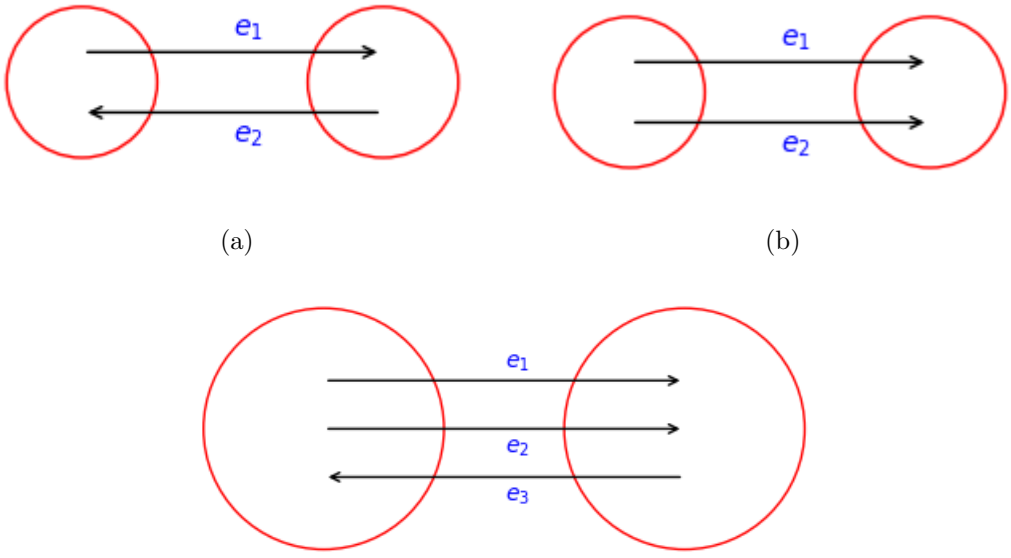
A parameter ε serves approximately the same purpose as the x parameter for the 1-tree heuristic: we want to discard solutions that have a weight of ε above the optimal ($z > (1 + \varepsilon)z*$). In the same way that x controls the decay of higher weight trees, ε controls the limit we allow for higher assignments. The thesis [8] analyzes the effect of the choice of value for ε going from 0.01 to 0.5, showing that higher values of ε perform better on small instances and lower values on bigger instances. Since the 2-matching solution density needs no adjustments from the TSP to the asymmetric version, we will not look at the choice of ε in all of our experiments. I chose $\varepsilon = 0.5$ in the experiments reported in this thesis.

3.1.5 Using cutsets to define a solution density

We defined cutsets of a graph in Section 2.2.2. The inference is the following: given a cutset of three arcs e_1, e_2, e_3 with e_1 and e_2 mandatory arcs, we can remove arc e_3. In the case of oriented graphs, the adaptation is quite simple: let us look at Figure 3.2. The circles represent subgraphs whose only arcs from one to the other are represented by the black arrows. arcs inside the subgraphs are omitted. In Figure 3.2a, arcs e_1 and e_2 are both mandatory in any tour, just like the undirected case. In Figure 3.2b, on the other hand, no tour can exist and the search must backtrack. In Figure 3.2c, e_3 is mandatory and if e_1 is mandatory, e_2 can be removed. If none of the three is mandatory, or if only e_3 is, we cannot infer anything. For undirected graphs, we need any two of the three arcs to be mandatory. However, with a single mandatory arc e_1, or in the oriented case with only e_3, we know that for each solution, one of the two other arcs needs to be chosen, and both arcs cannot be simultaneously in a tour. This information is not used in the filtering introduced in [38]. We try below to define a way to use the information based on cutsets and the Vertex Cover Problem, in the same way that we used other relaxations before.

We can see as a relaxation of the problem the simple fact of looking for a selection of arcs that respect all of the cutsets information. For instance, given two cutsets e_1, e_2, e_3 and e_1, e_2, e_4 with a mandatory arc e_1, the only two assignments for these arcs are $\{e_1, e_2\}$ or $\{e_1, e_3, e_4\}$. We could define this way a solution density that affects a value of 1 to the arc e_1 since it appears in both these solutions, and a value of 0.5 to the three other arcs that appear in only one.

Formally, given a collection of sets $(E_i)_{i\in[|1,n|]}$ with $\forall i \in [|1, n|], E_i \subseteq E$, we look for a subset



(a) (b)

(c)

Figure 3.2 Visualization of 2-cutsets and 3-cutsets in a schematized graph. The red circles represent subgraphs that are only connected through the black arcs. arcs inside the subgraphs are omitted. We added orientation to the symmetric case described in [38]. The orientation changes what we can infer from the cutsets: in (a) both arcs are mandatory whereas in (b) there is no possible tour.

$$S \subseteq E \tag{3.8}$$

$$s.t\ \forall i \in [|1, n|], |S \cap E_i| = 1 \tag{3.9}$$

In this case, we also have the property that each E_i is of size exactly 2 (the two non-mandatory arcs of a 3-cutset), which makes the problem look a lot like the Vertex Cover Problem where we relax Constraint (3.9) into

$$|S \cap E_i| \geq 1 \tag{3.10}$$

Unfortunately, the Vertex Cover Problem is NP-hard, which means using it as a relaxation to guide search or filter the instance is unrealistic, especially if a lot of arcs appear in several cutsets at the same time. Nevertheless, we can reduce an instance $\{e_1, e_2\}, E_2, \ldots, E_n$ with e_1 and e_2 not appearing in any of the E_i by assigning a density 0.5 to e_1 and e_2 and looking at the reduced problem $E_2, \ldots, E_n$.

Even though the problem itself is difficult, it is a very famous and highly studied topic that can be approximated by a $2-\Theta(\frac{1}{\sqrt{n}})$ factor [54]. It is also Fixed-Parameter Tractable: several

kernelization methods[1] exist [55] that make the question of finding a vertex cover of a graph consisting of k or fewer vertices tractable in $\mathcal{O}(kn + 1.2852^k)$ in a graph of n nodes. Such a complexity is called "fixed-parameter tractable"(FPT) as the complexity for a fixed value of k is polynomial in n. This means that for small values of k, i.e., if we have only a small number of cutsets with shared arcs, the problem may not be too computationally heavy.
Since we look at computing solution densities especially, we note that even enumerating solutions to the vertex cover problem is FPT with a complexity of $\mathcal{O}(k^2 2^k + kn)$, which is still linear in n for fixed values of k.

3.2 The models

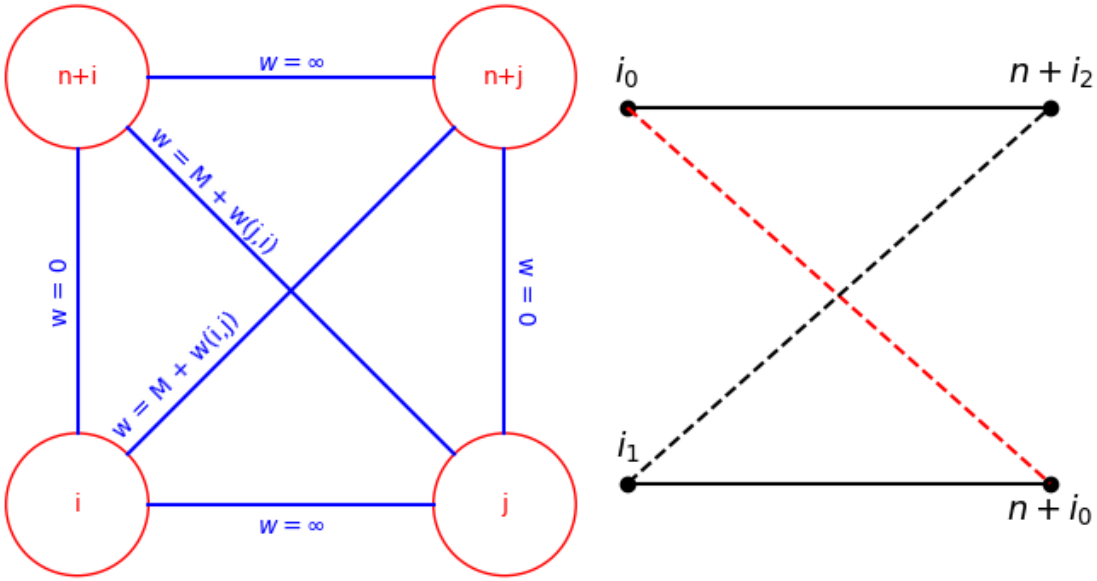
This project aims at analyzing the improvements we can gain by using solution densities in both Mixed-Integer Programming and Constraint Programming. In order to have a generic overview of what it can be used for, we tested our methods in combination with several pre-existing models from the MIP and CP sides. We detail here the models we used with different solvers and structures.

3.2.1 Concorde

Transformation into symmetric instances

Concorde is a very powerful tool for solving TSP instances of any size. It is interesting to see if our filtering methods can speed up even more the resolution. However, since we are focusing on the asymmetric TSP and Concorde only accepts symmetric instances with complete matrices, a transformation [21] has to be performed beforehand. Figure 3.3a illustrates how the transformation of ATSP instances into symmetric instances works: n additional nodes are added as copies of the original n, the node indexed by $n + i$ being a copy of node i.
Node $n+i$ will serve as the outgoing version of node i while i will serve as the incoming node. The arcs between two nodes i, j or $n + i, n + j$ are removed, and weights between i and its copy are set to 0. The weight between a node i and a node $n + j$ is set to $w(i, j) + M$, where M is a value set to be bigger than all of the initial weights. We see that the two arcs $(i, n+j)$ and $(j, n+i)$ serve for the information of the two different weights for the orientations of the initial arc (i, j). Any tour going into i must then go either into $n + i$ and then into a j or

[1] Kernelization is the reduction of a problem by eliminating vertices until the remaining "kernel" of candidate vertices has a size dependent on k and not n: for instance, any vertex of degree strictly higher than k must be in all solutions of size k or less.



(a) Transformation of two nodes (b) Proof of structure of the optimal solution

Figure 3.3 Illustration of the asymmetric to symmetric transformation for two nodes i and j

into another $n + j$, which means that the tour alternates between original nodes and copies: $i_0 \to n + i_1 \to i_2 \to \ldots \to n + i_k \to i_0$

Let us imagine that an optimal tour of the transformed instance does not include an arc $(i_0, n + i_0)$. In Figure 3.3b, we represent the arcs of a portion of this tour in solid lines. By replacing these arcs by the dashed arcs, we obtain a new tour going through all of the nodes that contains the arc $(i_0, n + i_0)$. Each node keeps a degree of 2 and since M is bigger than all of the initial weights, the weight of the two new arcs is necessarily lower than that of the initial arcs. Since we can choose i_1 among the two original neighbours of $n + i_0$, we can choose to transform

$$i_0 \to n + i_2 \to \overset{(c_1)}{\dot{?}\dot{?}} \to n + i_0 \to i_1 \to \overset{(c_2)}{\dot{?}\dot{?}} \to i_0$$

into

$$i_0 \to n + i_0 \to \overset{(\tilde{c_1})}{\dot{?}\dot{?}} \to n + i_2 \to i_1 \to \overset{(c_2)}{\dot{?}\dot{?}} \to i_0$$

where $(\tilde{c_1})$ is the reverse of (c_1), which exists in this symmetric instance. In this way, we are sure that we keep it a cycle. Since we found a lower cost solution, we proved that all of the $(i, n + i)$ arcs are in all of the solutions and n arcs are of the form $M + w(i, j)$ which means we can obtain the optimal weight from the asymmetric instance by removing $n \times M$ from

the objective.
In practice we took a value of $M = 20000$ for our experiments, a value bigger than the weights of the arcs in all our instances.

Edge elimination

Even if edge elimination before solving with Concorde has already been done successfully [56], the complete-matrix restriction makes the elimination less promising: the only tool we have to discard an edge is changing its weight to an excessively crippling value. Since the amount of information given is the same, there is no hope to gain time in the processing of the instance and the edges we discard because of their less promising characteristics could already be removed in the pre-processing of Concorde itself. In practice, we set the weights of the discarded edges to a value of 99999. Those edges hence have too big a weight to be included in any optimal solution – everything happens as if they did not exist at all.

3.2.2 MIP models

The MTZ and Big-M models described in 2.1.2 were implemented in Gurobi respectively for the ATSP and TSPTW problems. Gurobi is a generic solver that accepts MIP models as well as other optimization models and features a Python interface. We used more specifically the API included in the Python module `gurobipy` available at [57].

3.2.3 CP models

Basic model

Since this project is in continuation of [8], we experimented on the same CP model implemented with IBM ILOG CP Solver 1.6. adapted for the Asymmetric Traveling Salesman Problem. The base model includes successor variables `next[i]` associated with each node with a `IloAllDiff` constraint implementing the `AllDifferent` constraint:

$$\forall i, j, i \neq j \Rightarrow \texttt{next[i]} \neq \texttt{next[j]}$$

A `IloNoCycle` constraint ensures that the model is in fact a tour and does not contain subtours.
Constraints are able to provide solution densities by inheriting from the Constraint class

`IlcCountableConstraint` that has a method `recount()` as well as a method `getDensity(i, val)` or `getCostDensity(i,val)` for the weighted case. These methods allow to obtain the density associated with a variable-value assignment for a constraint, as in Equations (2.28) and (2.29).
In our case, we can add the countable constraints `IlcCostAllDiffCounting` implementing the `AllDiff` constraint as well as the 2-matching relaxation, and the `IlcSpanningT` constraint detailed in the thesis [8] implementing the 1-tree relaxation, whose computation method for the solution density needs adapting to the asymmetric case with the equation (3.2) described earlier.

Talos: State-of-the-art model

Since the previous model is very basic and cannot effectively solve bigger TSP instances, we also experimented with the State-of-the-art CP model [38] written in Java with the autonomous solver Talos for the problem. This model only allows symmetric instances so we experimented on the Symmetric TSP.
This model implements the WCC constraint [34], that needs an initial upper bound to perform the filtering. The upper bound is given as input, usually by using already found solutions in the literature and greatly affects the performance (see more detail on the effect of the chosen value in Section 3.5.4). Setting the upper bound very close to the optimum transforms the model into more of a proof of optimality so we also experimented with various upper bounds, even if a close-to-optimal solution could be found heuristically. We see in the results later on that we use an upper bound of 5% above the optimal value in some experiments in order to have a more realistic configuration. However, there is no polynomial approximation algorithm with a constant ratio $\varepsilon > 0$ for the general TSP, so we cannot ensure that a gap of 5% can be achieved in all instances [58].
The paper describing the filtering for the constraints specifies how to adapt it for an asymmetric case but also stipulates that they had better results using the symmetric case with the ATSP-to-TSP transformation, which makes the asymmetric case less interesting to investigate for us.
Since Talos is implemented in Java and our basic CP model in C++, we could not add to this model the computation of the 2-matching SD and all its variants based on different upper bounds. In what follows, the search guided by our solution densities only uses the 1-tree SD. For sparsification however, we were able to use both SDs as it only involves the preprocessing.

3.3 Guiding search

With the two Constraint Programming models, we used the solution densities to guide search: 1-tree density for both CP models and also the 2-matching solution density for the ILOG model. The 2-matching density was implemented with all the variants of upper bounds in C++, so we only use 1-tree for Talos. However, we use the 2-matching implementation to sparsify the instances before solving with Talos, since it does not need to be implemented in the same program. We use the `costMaxSD` search heuristic, which selects the variable-value assignment that has the highest solution density. With Talos, the LCFirst heuristic is used to select a vertex, which means we usually select the best arc incident to the last connected vertex, so we use `costMaxSD` only to select an arc among those that are incident to the selected vertex. Since the implementation of the ILOG model takes all variables into account at each step, we select among any arc available.

For guiding search, we need to update the densities at each step, since adding and removing arcs changes the structure of the graph. Inspired by [59], we decided to update the densities in the Talos solver only from time to time instead of at every node of the search tree. The idea is the following: we select a parameter ρ and only update the densities when the number of optional arcs $|O|$ changes by a factor ρ:

$$|O|^{new} < \rho \times |O|^{old} \text{ or } |O|^{new} > |O|^{old}$$

We observe the distribution of these variations of domains, and how the size of the search tree as well as the solve time are affected by the value of ρ, which is a compromise between heavy computation and approximate solution densities. We also update the densities only for the arcs taken into account at each node, since the vertex selection comes first.

In Figure 3.4, we observe for instance `kroA100` the ratios at each search-tree node between the present and the former domain sizes: a ratio $r < 1$ means the number of optional arcs decreased since the last search node was visited. The first plot represents the histogram of these ratios, where we see that the majority of ratios are between 0 and 1, with very few values going up to a maximum of 8: in general, we go down the search tree and reduce the number of possible choices.

In order to see more clearly the gain we can have by recomputing only when the ratios are below a certain value ρ ($r < \rho$ or $r > 1$ to compute after each backtrack), we look at cumulative densities for the next three plots. We use a logarithmic scale to have a better visual accuracy of the reading. The third and fourth plots are simply zooms of the second, to see the region of interest. They can be read this way: for example in the subplot (d), we see that the curve reaches 10^{-1} for $r \simeq 0.63$. This means that by choosing $\rho = 0.63$, we can

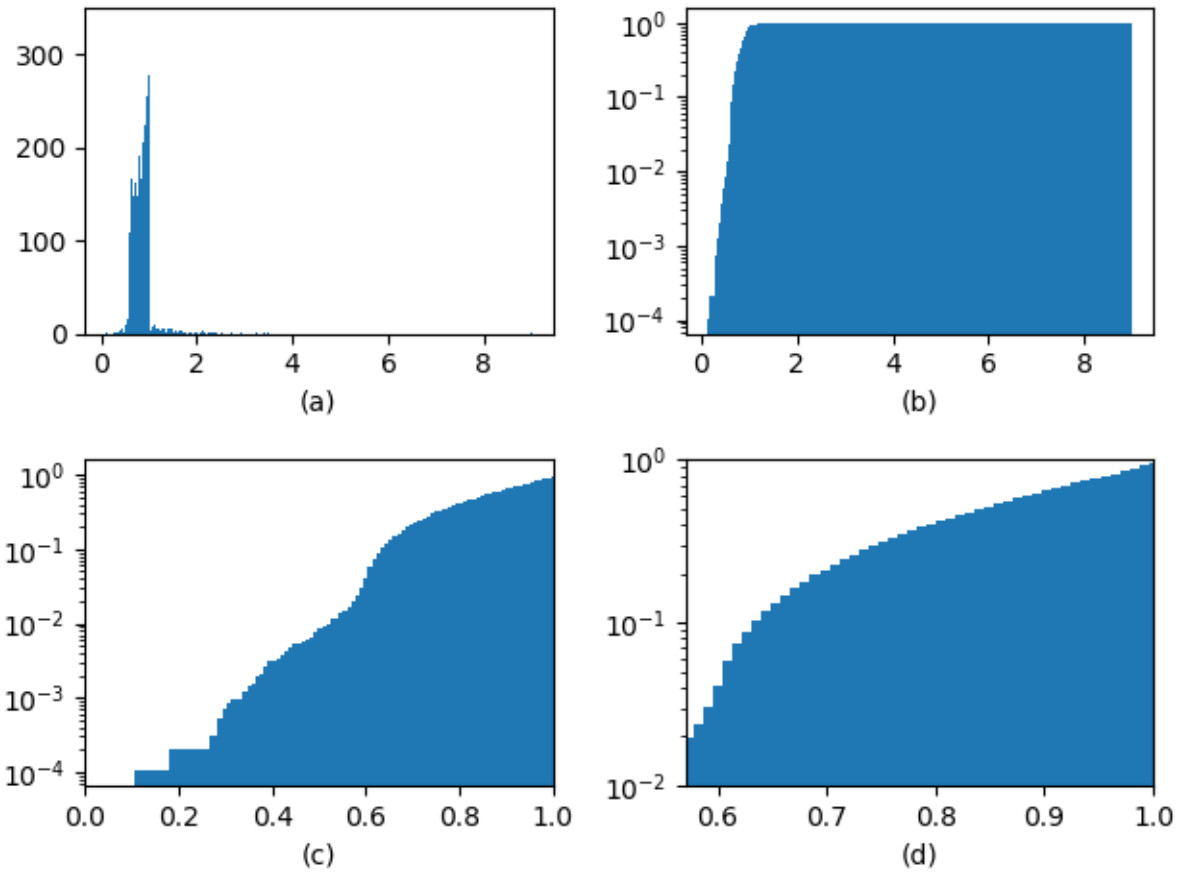


Figure 3.4 Distribution of the variations of domain size at each search-tree node on the symmetric instance `kroA100`. The four graphs show the same data with different visualization. (a) is a histogram of the ratios. (b) is the same but cumulative (the values are added to the ones before) and normalized to end at 1. (c) and (d) are zooms of (b) around an interesting region.

expect to re-compute the densities for 10% of the nodes.

3.4 Filtering

Another possible use of solution densities is an initial filtering of the arcs. We select promising arcs in the instance and remove the variables corresponding to the rest of the arcs, thus reducing the number of arcs. It creates a sparser instance that needs fewer branching decisions to be solved as there is a smaller number of variables.

3.4.1 Selecting the arcs

In general, to keep consistency in the structure of the graph, we select a percentage of arcs going into and from each node, thus keeping at least a sufficient degree for each node. An additional method to keep at least one feasible solution for the ATSP is to add the "trivial" cycle $(1, 2), \ldots, (n, 1)$ which adds at most n arcs, which is asymptotically negligible compared to $\frac{kn(n-1)}{100}$ (k % of all arcs). This cycle can always be included because we only have complete instances in our benchmark.

For the TSPTW that we will detail in the following chapter, there is no such easy way to select one feasible solution, even if the graph is complete. We report in Table 3.1 the percentage of the total of selected arcs over the total of all arcs for several instances of different sizes. We keep a percentage of the arcs incident to each node for each solution density. The total percentage of kept arcs can be higher than this parameter. The columns in Table 3.1 represent the fraction of arcs we keep for each SD and each node.

The values are the percentages of all the arcs, i.e., $100 \times \frac{|E|}{n(n-1)}$ where E is the set of selected arcs.

	5	10	15	20	30	40	50	60
br17	6.25	13.2	21.7	26.5	46.7	56.6	72.1	85.3
ftv44	6.82	12.1	17.0	25.2	35.6	48.1	58.5	71.0
ftv70	6.26	12.9	18.0	24.9	36.6	48.3	58.9	69.6
ft70	6.75	12.9	17.7	24.0	35.3	46.9	58.2	69.2
ftv170	6.05	12.3	17.6	23.5	35.00	46.1	57.0	67.9
rbg323	7.57	14.6	21.1	27.1	39.0	49.7	60.4	71.0
rbg443	8.47	17.2	25.7	33.9	48.7	61.9	71.7	79.2

Table 3.1 Real fraction of kept arcs vs. percentage parameter. We compare each "theoretical" percentage with the percentage of arcs that are kept in total for a few illustrative instances - the results for each instance are not relevant.

3.4.2 Advantages

Sparsifying an instance before solving has the advantage on the previous method that solution densities need to be computed only once, which makes the method's complexity less reliant on the efficiency of the SDs' computation. The two methods can of course be combined together in CP models, but sparsified instances can be used with any solver as it does not need to change anything in the solving itself. We used different solvers to compare the results of sparsified and basic instances.

On the other hand, sparsifying from the start means we possibly lose some solution. What

we describe here is a heuristic, and does not prove the optimality of the solutions found. We keep feasible but possibly suboptimal solutions. We will report the distance to the optimal solution in our experimental results.

3.5 Computational Results

3.5.1 Benchmark

For all of the experiments below, the symmetric and asymmetric instances are taken respectively from the classic libraries for the Traveling Salesman `TSPLIB95/TSP` and `TSPLIB95/ATSP` [60]. There are 19 asymmetric instances studied from sizes 17 to 443. There are 112 symmetric instances, but we will only report results on a few of them. Instances and best-known solutions can be found at [61]. The times needed to compute the solution densities can be found in Section 4.3.2. Those use instances from benchmarks with time windows that present the advantage of being of different sizes.
We symmetrize asymmetric instances for Concorde and use symmetric instances only for Talos, to compare with the previous results obtained with Talos in [38].

3.5.2 Concorde

In Table 3.2, we report the time in seconds needed to solve each instance with only $k\%$ of selected arcs: the columns represent the sparsification rate. The instances are symmetrized versions of the ATSP benchmark, since Concorde only accepts symmetric instances. We use QSopt for Red Hat Linux 64bit version as the underlying linear solver.
We observe that more than half of the instances are solved to optimality with 10% of arcs and almost all of them are solved with 20%. However, there is no general tendency and on the contrary, the average time seems to be rather smaller with a high number of arcs. Furthermore, the solving is really fast and there is little hope that the time we can gain is comparable to the time needed for pre-computation.
Since for almost all of the instances and parameters the optimal solution is found, we do not report gaps to optimality for the rest as is done in some other results tables. We simply report Xs when the optimal solution is not found. In general, we want to compare the time we gain by sparsifying with the margin we are willing to lose on the optimal solution. Here, we see that we do not gain any time in average, so there is little meaning in comparing the quality of the solutions.

Instance	5 %	10 %	15 %	20%	30 %	40%	50%	60 %	100%
br17	X	X	X	X	X	0.04	0.05	0.05	0.05
ft53	X	X	X	0.14	0.11	0.10	0.09	0.09	0.09
ft70	X	0.49	0.49	0.31	0.42	0.35	0.37	0.44	0.30
ftv170	5.73	4.96	2.03	4.49	2.27	4.95	2.77	3.85	1.82
ftv33	X	X	X	0.04	0.04	0.04	0.04	0.04	0.05
ftv35	X	X	X	1.25	0.86	1.22	1.00	3.53	1.24
ftv38	X	X	X	2.08	2.69	1.45	1.93	1.18	2.35
ftv44	X	0.59	1.43	1.31	0.90	0.97	1.17	0.84	1.34
ftv47	X	1.22	1.50	4.08	1.80	3.30	1.18	2.52	1.41
ftv55	X	1.09	1.03	0.91	3.74	0.91	1.64	1.85	1.56
ftv64	X	3.11	6.18	3.78	3.23	3.98	5.11	5.77	6.31
ftv70	X	1.73	1.69	2.16	1.71	2.69	1.18	2.21	1.15
kro124p	X	X	0.93	0.69	0.76	1.13	0.50	0.77	0.64
geometric mean	5.73	1.39	1.46	0.95	0.89	0.73	0.63	0.79	0.66
arithmetic mean	5.73	1.88	1.96	1.77	1.54	1.63	1.31	1.78	1.41

Table 3.2 Results of sparsified asymmetric instances with Concorde. The times are in seconds, and the percentages are the number of arcs we keep. A lower percentage means an instance we reduced aggressively.

3.5.3 ILOG CP model

The results reported below are for our basic CP model, described in Section 3.2.3.

Using SDs as search strategy

We report in Appendix in Table A.1 the best solutions found in 3600s for two different search heuristics.

We compare our search heuristic `costMaxSD` in which we select at each step the arc with higher solution density among 1-tree and 2-matching with the search heuristic `iloMinSizeInt`. The latter selects at each step the variable `next[i]` with the minimum domain (minimum degree for the vertex) and the arc among those lexicographically.

It is difficult from only the times to compare the results since all of the instances time out. However, we can see that for almost all of the instances, our heuristic finds a better solution than `iloMinSizeInt`. On a few instances, we will see in Figure 3.5 how the best solution evolves over time.

We see from Figure 3.5 that our search heuristic almost always outperforms `iloMinSizeInt`, and always finds a better solution in the early stages. In the only instance in which `iloMinSizeInt` finds a better solution in the end, `ftv33`, the shift only appears after over 2000s of solving.

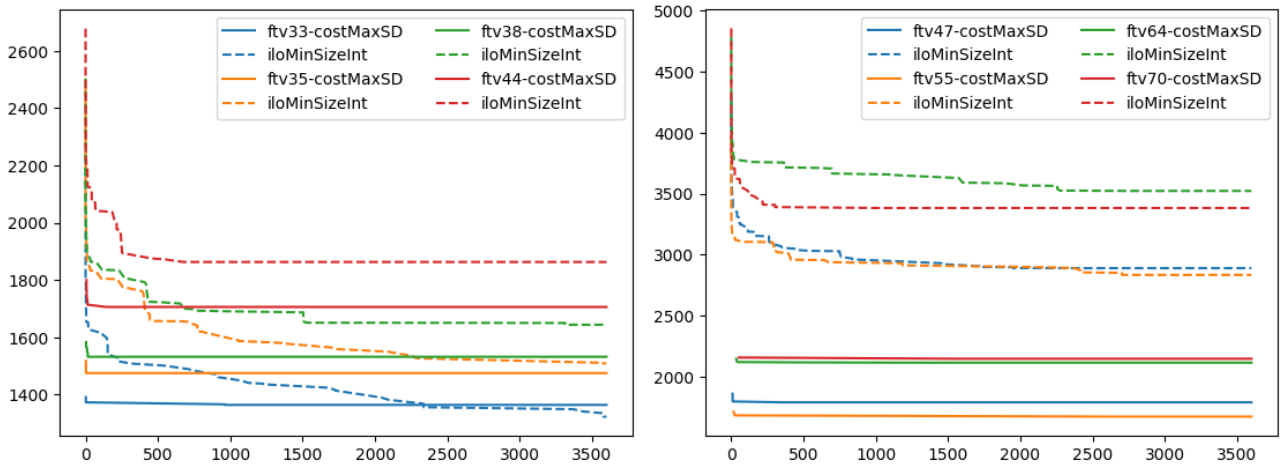


Figure 3.5 Evolution of best solution over time for two different search heuristics in the ILOG CP Model. The horizontal axis is the time in seconds. We compare 8 instances.

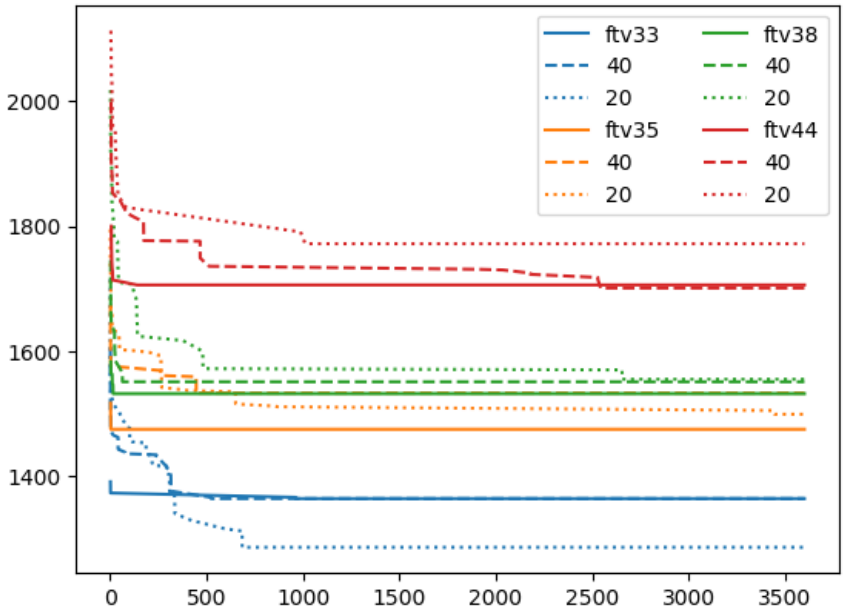
It is difficult to know what would happen if the experiments were conducted until the end, since even very small instances like these take at least one hour.

Sparsification

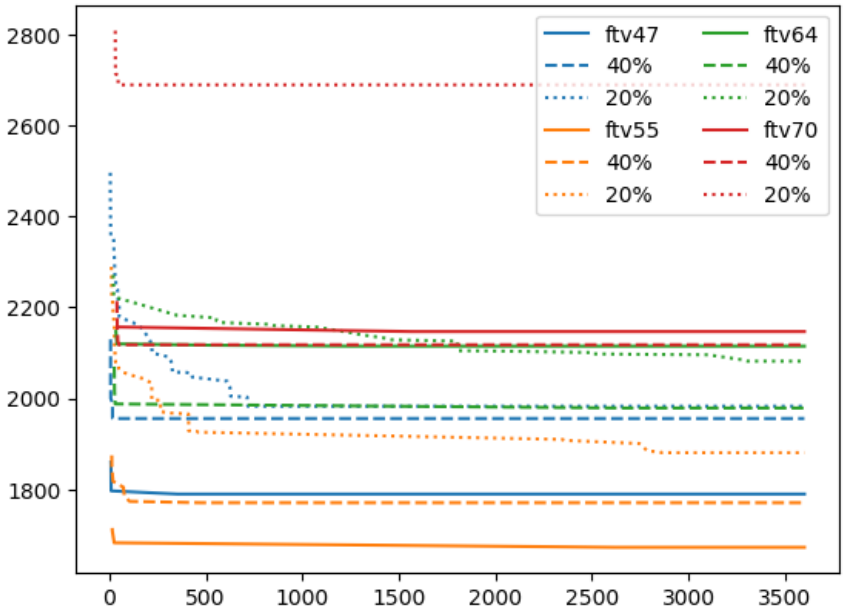
We use the search heuristic `costMaxSD` with sparsification rates of 40 and 20% to see the evolution of the best solution found over time in Figure 3.6. We see in the figure that the basic instance finds a good solution very fast, even when the sparsified version ends up with a better solution in the end. Nevertheless, the interpretation we can do of these results is limited by the fact that all instances time out, even the smaller ones. Given that information, it is difficult to know if the reduced instances will finish sooner and if the gaps will remain at the end. For this reason, we will consider another more complex CP model in the following section.

3.5.4 Talos

We use the model implemented by Isoart et al. in [38]. The model is implemented in Java with its own solver called Talos. Basic structures are implemented as well as all of the constraints defining the Traveling Salesman Problem. We use the model as is, so all of the experiments reported below are done on symmetric instances.



(a)



(b)

Figure 3.6 Evolution of best solution over time in sparsified instances solved with ILOG model. The horizontal axis represents the time in seconds. The bold line is the instance where no filtering is done while for the two other curves the percentage represents the number of arcs that are kept.

Guiding search

We report in Figure 3.7 the number of search nodes and the solving time for several values of ρ (see Section 3.3). The choice of ρ is a compromise between the precision of the solution

density and the amount of computation. Both subfigures of Figure 3.7 are normalized with the values for $\rho = 1$ that serves as reference (we can see that all of the points for $\rho = 1$ in both figures are merged in one point of coordinates $(1.0, 1.0)$)

The first figure represents the time needed for solving. We see that any value of ρ induces

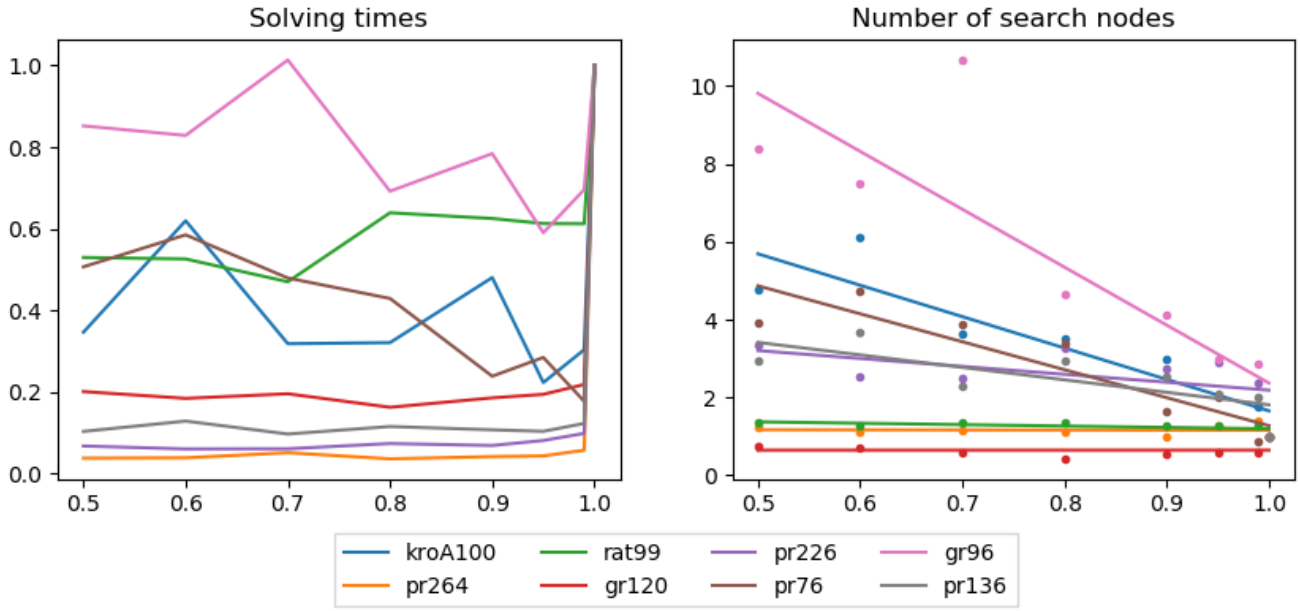


Figure 3.7 Times (left) and number of search nodes (right) as a function of ρ. The values are normalized by the value for $\rho = 1$ to be able to represent several instances together. In the right figure, we add the linear regression of the points for each instance to illustrate the general evolution.

a significant speedup for most instances, which is consistent with Figure 3.4: a lot of computation corresponded to a value of $r = 1$ so any parameter removing these values reduces drastically the number of SD computation. However, there is little information on the "best" value of ρ. Some of the instances show little dependence on ρ while others have an erratic behavior. A value of $\rho = 0.95$ seems to be among the minimal values for all the instances or so, but all of the values seem to be equivalent.

On the right plot, we show the number of search nodes for each value represented with a point, and the corresponding linear regression shown in a straight line. We see that all the lines are decreasing or (almost) flat. More precisely, all of the lines are descending except for the one corresponding to `gr120`, with a coefficient of 2.2×10^{-3}. This tendency means that in general a more precise solution density gives a better accuracy, with some instances gaining a $\times 3$ ratio in number of search nodes between $\rho = 0.5$ and $\rho = 1$. The small slope of some instances mean we lose little to no accuracy by giving up on some computation: these

instances show a good improvement in computation time when choosing a small value of ρ while bigger slopes means we increase the number of search nodes by a lot and hence the solving time. The left-figure curves corresponding to instances `gr96`, `kroA100` and `pr76` are heavily non-monotonous compared to the five other.

We will compare in Table 3.3 the search heuristic based on `costMaxSD` with $\rho = 0.95$ to the

Instance	minDeltaDeg		costMaxSD		$\rho = 1$
	Time (ms)	nb. Nodes	Time (ms)	nb. Nodes	nb. Nodes
pr76	80633	115805	67060	92415	46171
pr136	30467	18297	66634	28599	13547
pr226	3136	233	10830	839	287
pr264	6356	785	24021	661	523
gr48	255	29	822	83	159
gr96	2977	613	3732	923	309
gr120	2006	221	2584	189	317
rat99	833	105	1349	57	45
rat195	67351	29211	251187	62945	X
kroA100	3463	1471	7930	4959	2407
kroB150	248432	140243	506492	201487	X
kroB200	185363	81473	640135	146225	X
kroD100	1770	291	2642	405	317

Table 3.3 Comparison of time and number of search nodes between search heuristics `costMaxSD` and `minDeltaDeg`. Xs represent cases where the solving times out.

best-known search heuristic for Talos's TSP model, namely `minDeltaDeg`. The latter selects the arc with the minimal difference between the number of optional and mandatory arcs incident to its two extremities. In other words, if we denote O and M the sets of optional and mandatory arcs, we look for an arc (i, j) minimizing

$$\Delta_{ij} = \sum_{k \in n} (\mathbb{1}_O(i,k) + \mathbb{1}_O(j,k) - \mathbb{1}_M(i,k) - \mathbb{1}_M(j,k))$$

This heuristic selects arcs with a low number of neighbours, i.e., arcs in a sparse area. It is based on the principle of *first fail*, trying to backtrack as fast as possible to avoid searching in infeasible regions. We also report the number of nodes for $\rho = 1$, which serves as a guide for the expected number of nodes if we used the exact 1-tree solution density.

We conclude from the table that `costMaxSD` is not competitive with `minDeltaDeg` but compares on most instances with a ratio of 1.5 to 4 and even does better than `minDeltaDeg` search heuristic on one instance. A more complete implementation of our search heuristic including the 2-matching solution density may perform better.

With the choice of ρ, we were able to limit the amount of computation in general. Since we are looking at symmetric instances, we are also able to update for each step only the densities of the arcs we are looking at by inverting a single matrix, which is not possible for asymmetric instances. Other possibilities exist to speedup the process, that rely mainly on speeding up each computation:

- Inverting a definite positive matrix is faster using Cholesky's decomposition instead of the more general LU decomposition. However, the algorithm is not available in-place in the Colt matrix library that we use. According to [62], the Cholesky decomposition is about twice faster as the LU decomposition for definite positive matrices.
- As we go down the search tree, our graph includes paths from one node to another that can be contracted into a single node, since no other arc can be added to it: we simply connect the arcs incident to the two end nodes to a new "contracted" node. By doing so, we reduce the number of nodes in the graph and thus the size of the matrix to be inverted. However, this is only possible if we have a way to backtrack and so a specific reversible matrix. For instance, the removed nodes could be swapped to be at the end of the matrix with a parameter keeping in memory the submatrix active at each step. Again, a matrix structure including matrix inversions as well as reversibility for backtracking is not pre-implemented in Java or Talos.

In order to gain an additional speedup, there are a lot of structural modifications to be added that require to write specific matrix libraries. Since it takes a lot of time to implement, and this would still only be on the symmetric TSP, we stopped the experiments on this specific model here. Nevertheless, these additional improvements could probably help speeding up the process by a factor of 2, which would make our search heuristic more efficient. Additionally, we would need to implement the 2-matching heuristic in Java, since we currently use only 1-tree to guide search.

Influence of the upper bound on the initial filtering

The filtering of the Weighted Circuit Constraint (see Section 2.2.2) relies on the current value of the upper bound. With an optimal upper bound, the initial filtering can remove a great proportion of the arcs.
In [34], the authors represent a comparison of the remaining arcs for upper bounds of 700 and 675 on an instance, showing how dependent the initial filtering is on the upper bound. Initial upper bounds can be found by using a fast heuristic for the TSP, such as the Lin-Kernighan

Instance		OPT		1.05OPT		1.2OPT		1.5OPT	
		Time	Tree	Time	Tree	Time	Tree	Time	Tree
d198	cMSD	341	86797	2465	528667	$\geq$ 3h	X	$\geq$ 3h	X
	mDDeg	17.4	5095	342	129225	593	254915	749	319733
kroA100	cMSD	17.7	14115	193	155213	177	155911	323	278645
	mDDeg	2.11	1471	15.1	12865	96.6	104049	157	191553
kroB150	cMSD	730	322883	3059	1285625	9302	4420441	9395	4497889
	mDDeg	255	140243	1121	639717	1374	852495	1191	743415
kroB200	cMSD	542	148701	5101	1293193	$\geq$ 3h	X	$\geq$ 3h	X
	mDDeg	218	81473	3624	1309571	$\geq$ 3h	X	$\geq$ 3h	X
pr136	cMSD	100	42587	3378	1500863	2888	1380165	4293	2109815
	mDDeg	33.6	18297	723	416781	1141	760169	1907	1288453
rat195	cMSD	260	78009	3588	1059675	4076	1253379	3409	1044657
	mDDeg	84.5	29211	8875	3321431	$\geq$ 3h	X	$\geq$ 3h	X

Table 3.4 Results of symmetric instances in Talos with search heuristics `costMaxSD` and `minDeltaDeg` with different initial upper bounds

heuristic. However, instead of implementing an initial local search for the upper bound, Talos relies on a manual entry of an upper bound, that can be set to the best-known solution, thus becoming mainly a proof of optimality. We look in this section at how a suboptimal upper bound influences the result for `costMaxSD` and `minDeltaDeg`. Table 3.4 reports the times and sizes of the search trees for our two heuristics depending on the initial bound used for the filtering. We select the six "difficult" instances to compare. We see that the initial bound impacts heavily the efficiency of the filtering, with a factor higher than 10 for all instances between the optimal bound and a gap to the optimal bound of 20%. With inaccurate upper bounds, the model becomes impractical. In the following section, we will use both the optimal bound and a more realistic value of 1.05OPT.
However, no matter what value we choose for the initial upper bound, our search heuristic is less efficient than the state-of-the-art search strategy `minDeltaDeg` for these difficult instances.

Sparsification

We sparsify the graph after the initial filtering done by the WCC constraint. If we did our sparsification first, there is a big chance that there is little change with the original instance, since a lot of arcs are filtered out. We illustrate the rate of the initial WCC filtering with an optimal upper bound in Table 3.5.

We compare in Tables 3.6 and 3.7 the results for different rates of sparsification. We report

Instance	Percentage	Instance	Percentage
bays29	10.59	kroB150	14.19
d198	99.19	kroB200	12.47
eil101	5.267	kroD100	9.535
gr120	7.801	pr107	99.24
gr48	18.17	pr136	29.01
gr96	15.29	rat195	19.67
kroA100	13.43	rat99	4.927

Table 3.5 Percentage of remaining optional arcs after the initial WCC filtering.

Instance		20%		40%		60%		100%	
		Time	Tree	Time	Tree	Time	Tree	Time	Tree
d198	cMSD	147	41503	184	42227	342	86797	341	86797
	mDDeg	16.8	5261	14.33	3827	18.1	5095	17.4	5095
kroA100	cMSD	**5.23**	4051	**8.75**	6445	17.4	14115	17.7	14115
	mDDeg	**0.87**	539	**2.41**	1657	2.13	1471	2.11	1471
kroB150	cMSD	**362**	167091	746	331831	721	322883	730	322883
	mDDeg	**116**	62933	310	170687	262	140243	255	140243
kroB200	cMSD	518	150927	567	152517	540	148701	542	148701
	mDDeg	243	92823	264	95799	215	81473	218	81473
pr136	cMSD	35.8	14555	109	47313	98.0	42587	100	42587
	mDDeg	9.05	4559	38.5	21621	33.4	18297	33.6	18297
rat195	cMSD	220	75753	260	83979	259	78009	260	78009
	mDDeg	80.9	28239	46.2	16177	85.0	29211	84.5	29211

Table 3.6 Results of sparsified symmetric instances in Talos with optimal upper bounds. Numbers in bold are where no solution was found, i.e., we lost the optimal solution.

the results with and without initial upper bounds: in Table 3.6 the initial upper bounds are set to the best known solutions while in Table 3.7 the upper bounds are 5% higher than the optimal value. We compare the results for the two search heuristics `costMaxSD` based on the 1-tree search heuristic and `minDeltaDeg` defined previously.

In Table 3.6, the upper bound is set to the optimal value, which means that we can only find a solution if it is optimal. We write in **bold** the times where no solution was found. In Table 3.7, since they are the same instances, the optimal solution is found for the same instances than in Table 3.6, the rest find suboptimal solutions.

For these tables, we look at instances that take at least a few seconds to solve, since smaller instances give very unstable results. We observe in Table 3.6 that for most instances the solving times stay the same with a medium sparsification. The speedups for instances `kroA100` and `kroB150` are linked to a loss of the optimal solution. On the other hand, `d198` and `pr136`

Instance		20%		40%		60%		100%	
		Time	Tree	Time	Tree	Time	Tree	Time	Tree
d198	cMSD	1539	397515	1997	443395	2461	528667	2465	528667
	mDDeg	257	96421	336	133623	335	129225	342	129225
kroA100	cMSD	26.8	23629	73.3	61537	184	155213	193	155213
	mDDeg	3.23	2821	22.2	19367	15.2	12865	15.1	12865
kroB150	cMSD	1923	859051	3009	1322379	3089	1285625	3059	1285625
	mDDeg	2437	1436633	1223	691319	1119	639717	1121	639717
kroB200	cMSD	10162	2846877	3991	1037785	5122	1293193	5101	1293193
	mDDeg	5213	2023657	3929	1465191	3635	1309571	3624	1309571
pr136	cMSD	1331	679003	1577	747001	3332	1500863	3378	1500863
	mDDeg	298	177591	2434	1484319	715	416781	723	416781
rat195	cMSD	2322	809115	4027	1182285	3591	1059675	3588	1059675
	mDDeg	916	330653	2619	949879	8748	3321431	8875	3321431

Table 3.7 Results of sparsified symmetric instances in Talos with an upper bound of 1.05OPT (5% above optimal value). The times are reported in seconds. The gaps for the instances that lose optimality are reported in the text and are under 0.2%

gain a 2× speedup with 20% of arcs.

In Table 3.7, we relaxed the upper bound, so all instances are feasible, even if the optimal solution is not found. More precisely, the solution of `kroA100` with 20% of arcs has a gap of 0.12% and of 0.05% with 40% of arcs while `kroB150` has a gap of 0.18% with 20% of arcs. All of the other sparsified instances keep the optimal solution, as was already shown in Table 3.6. We see that the sparsification to 20% of arcs induces very small losses of optimality while reducing the solving times for almost all instances (except `kroB200` here). For less aggressive sparsification rates, the speedup is not as clear.

Graph of cutsets

As a first step to evaluate the interest of the idea developed in Section 3.1.5, we look at the cutsets found during the search. We are interested in the total number of cutsets found but also the structure of the cutsets. The idea is to have a sufficient number of cutsets to be able to gain information from it; the degree of the nodes is interesting as a graph with vertices of degree 0 or 1 only adds the value 0.5 for the possible values of the solution density. On the contrary, a graph with a very complicated structure may make it impossible to compute a vertex cover because of the dependence on the optimal value k of the complexity. We see that for instances bigger than the two reported here, there is no cutset found directly at the root node. This means that while we could use an adapted solution density for guiding search, we cannot use it for sparsification or any pre-computing, which means that any heuristic based

on the idea would need to be fast enough to be computed during search.

In Figure 3.8, we represent the transformation from the list of cutsets into a graph where

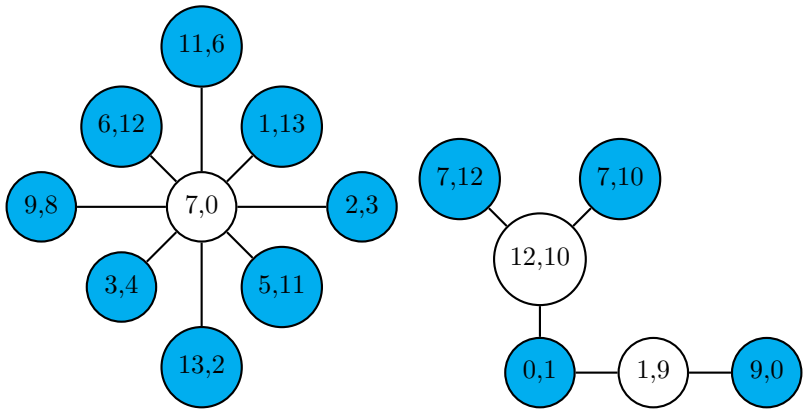


Figure 3.8 Representation of the graph of cutsets in instance `burma14`. Each node represents an arc of the instance and each arc a 3-cutset where both incident arcs are non-mandatory. Exactly one of the two arcs connected in the graph must be in every solution. We see that there are two connected components.

nodes represent arcs of the instance. In cyan are represented the arcs that are in the actual optimal solution. We see that contrary to what could be imagined, the vertex cover represented here is of size 12 whereas the complementary, of size 3, is not in the optimal solution. Of course this approach does not include the degree constraints nor the weights of the arcs. Even by taking into account the weights of the arcs, the blue set has a cost of almost 4× the rest. In Figure 3.9, we see another completely different structure for the cutsets graph.

Although all of the arcs are in one cutset at most, there is some information that can be inferred from the graph: for instance, two different cutsets have in common Vertex 1. Since for these two pairs of arcs one and only one must be in the solution, there are only 4 possibilities for Vertex 1's neighbours, and all of the other arcs incident to 1 can be discarded. For this instance, there is a significant number of cutsets that exist and are not taken into account by the binary filtering, which means there is an interest in finding a solution density to use this information. However, on these two examples, the Minimum Weighted Vertex Cover Problem seems to be of little interest in practice given the optimal solution in the first example and the structure of the second.

The cutsets that are represented in these figures are the information that is not used in the filtering of [38]. Although there are 14 arcs involved only at the root node, no arc is found in two different cutsets: for all these arcs, the value of a solution density would be of 0.5 with no possibility to discriminate between two arcs found in a same cutset.

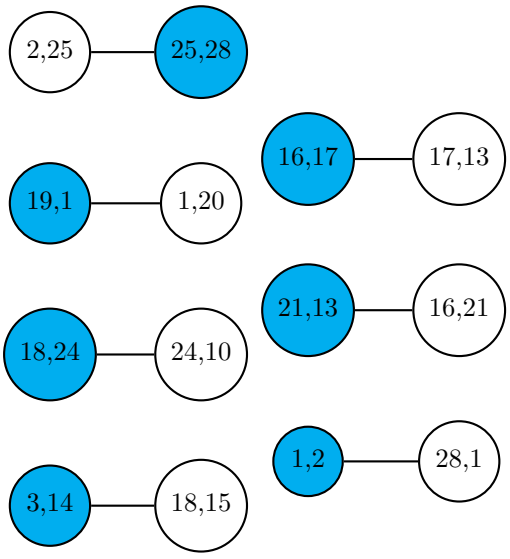


Figure 3.9 Representation of the graph of cutsets in instance `bays29`. In cyan are arcs found in an optimal solution, vertices represent arcs of the instance `bays29` and arcs represent a 3-cutset in which both arcs were simultaneously.

3.5.5 MIP model

We look at how sparsifying with 1-tree and 2-matching like described in Section 3.4 before solving with a MIP model affects performance. Besides the reports of solving times and gaps to optimality, we also trace the evolution of the lower and upper bounds during the search which illustrates the results we would have for each limitation of time.

Evolution of bounds during the solving

Graphically, we observe in Figure 3.10 the evolution of lower and upper bounds during the resolution by Gurobi of the MTZ model on 4 different instances. The times are in seconds. The orange lines represent the bounds for the original instances while the blue lines represent the bounds for an instance where only 20% of the arcs at each node were kept. The stars are the ending points when the resolution terminates.

We see that in our four experiments, the optimal solution is kept in the sparsified instance (the blue and orange stars are on the same horizontal line), and the sparsified version usually finds a first solution faster than the original version. The start of the upper bound line indicates when the first feasible solution is found.

The sparsified instance also terminates faster, but we see that most of the time both of the bounds are tighter: at any point, we would have a more precise surrounding for the value of the optimal and a better current solution. We see from this evolution that the sparsification

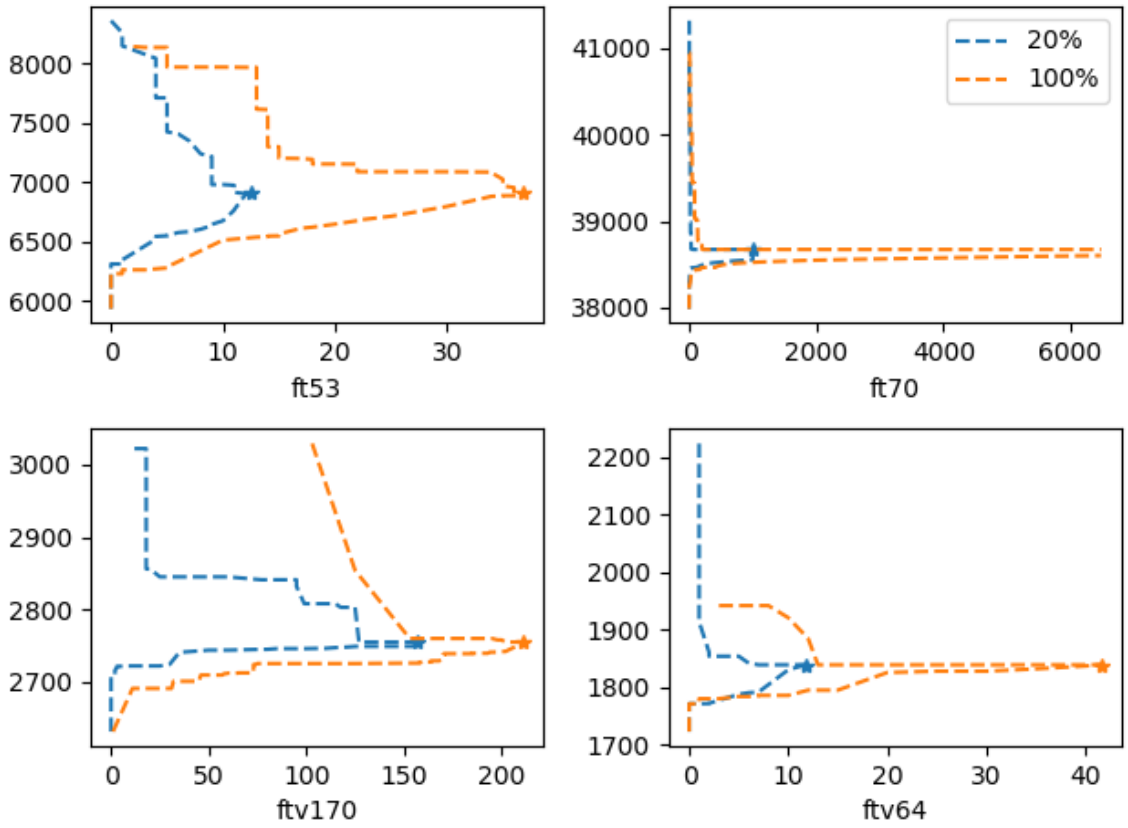


Figure 3.10 Evolution of the lower bound and the upper bound (best known solution) over time during the resolution of a few instances and their sparsified version

gives more impressive results than what was observed on both CP models and with Concorde with the same instances.
The following section will give more complete results including a higher number of instances and several sparsification rates.

Comprehensive results of solving time

In this last section, we report in Tables 3.8 and 3.9 the time needed to solve several instances in the model as well as the gap to optimality, i.e., the percentage we lose on our objective by using a sparsification method. A gap of optimality of 0 means that we found the optimal solution. Gaps are always non negative.

We see that the results are highly dependent on the instance. For a small instance like `br17`, we can have both very high gaps and very high time increases due to the low values of the

		br17	ft53	ft70	ftv33	ftv35	ftv38	ftv44	ftv47	ftv55	g.avg
5	Time(%)	0.00	1.98	$\leq$ 3.32	0.72	2.97	0.36	5.04	1.24	15.0	0.00
	Gap (%)	328.21	69.37	0.00	44.56	34.69	38.82	4.03	3.27	9.70	41.75
10	Time(%)	1.50	11.9	100	18.7	18.3	10.0	64.5	23.4	21.6	18.01
	Gap (%)	146.15	0.14	0.00	2.95	0.00	0.00	0.00	0.00	0.00	10.90
15	Time(%)	120	19.2	100	25.9	29.7	53.4	68.8	14.5	32.8	40.75
	Gap (%)	92.31	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	7.54
20	Time(%)	51.9	33.8	100	46.0	31.7	33.7	79.4	28.2	20.9	41.96
	Gap(%)	12.82	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.35
30	Time(%)	59.4	53.8	100	23.7	40.6	31.2	110	37.3	23.3	45.97
	Gap(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	Time(%)	467	47.6	100	89.2	126	145	173	24.5	80.5	102.4
	Gap (%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	Time(%)	602	53.4	100	20.9	150	82.4	71.2	35.1	43.9	76.63
	Gap (%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
60	Time(%)	468	78.0	100	41.7	122	376	63.1	54.7	34.3	97.91
	Gap(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100	Time (s)	1.33	36.79	>1000	1.39	2.02	2.79	4.17	17.74	11.97	
	Solution	39	6905	38673	1286	1473	1530	1613	1776	1608	

Table 3.8 Results of the MTZ model in Gurobi with sparsified instances. The rows correspond to the percentage of kept arcs for each vertex and each SD and the value are in percentage of the 100% configuration (without any sparsification). The columns represent 17 instances of the TSPLIB ATSP library

base case. For bigger instances like the `rbg*` reported here, the difference between the basic instance and the 60% sparsification is big and we can have significant speedups. We use a geometric mean to ensure that the mean is not too much impacted by one instance alone. Otherwise, the poor results we have on instance `br17` would affect the mean, even though it is an instance that can be solved in 1.33s without any filtering.
Besides instances `br17` and `rbg323`, all instances are solved to optimality with a 15% sparsification rate. We see that on average, we can have a 2.5× speedup for the smaller instances and a 10× speedup for bigger instances when taking 15% of the arcs. We recall that there is not exactly 15% of the arcs that are kept, but 15% of incoming and outgoing arcs for each vertex and each solution density plus a trivial cycle, see Section 3.4.1 for more detail.

3.5.6 Effectiveness of reduced costs compared to 2-matching

In this section, we compare the solution densities obtained by computing upper bounds with the reduced costs from the Hungarian Algorithm. We will see in Section 4.3.2 that the reduced costs are faster to compute than the original 2-matching heuristics, with a ratio of up to 1000 for large sizes. We want to see if this speedup means losing the quality of the

		ftv64	ftv70	ftv170	kro124p	rbg323	rbg358	rbg403	rbg443	g.avg
5	Time(%)	8.32	52.8	7.19	3.61	≤ 0.69	≤ 0.63	≤ 2.15	≤ 3.49	3.73
	Gap (%)	0.38	2.21	0.00	0.07	17.04	64.92	19.23	12.06	12.94
10	Time(%)	28.5	34.2	17.6	5.69	≤ 1.53	≤ 1.73	≤ 8.14	≤ 13.1	8.51
	Gap (%)	0.00	0.00	0.00	0.00	2.56	0.00	1.42	1.32	0.66
15	Time (%)	36.3	49.9	22.2	11.1	≤ 11.2	≤ 1.69	≤ 8.98	≤ 2.58	10.88
	Gap (%)	0.00	0.00	0.00	0.00	0.45	0.00	0.00	0.00	0.00
20	Time (%)	28.2	54.9	74.3	16.1	≤ 1.86	≤ 7.22	≤ 3.06	≤ 47.6	15.66
	Gap (%)	0.00	0.00	0.00	0.00	0.45	0.00	0.00	0.00	0.00
30	Time (%)	66.6	116	45.6	24.8	≤ 15.2	≤ 3.61	≤ 6.47	≤ 5.24	18.90
	Gap (%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	Time(%)	33.6	90.0	92.5	26.9	≤ 1.62	≤ 13.63	≤ 7.49	≤ 5.86	17.09
	Gap (%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	Time(%)	56.7	141	45.7	40.6	≤ 8.48	≤ 4.22	≤ 11.7	≤ 6.27	21.08
	Gap(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
60	Time(%)	45.3	143	97.9	52.2	≤ 11.4	≤ 3.54	≤ 4.73	≤ 6.91	21.38
	Gap (%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100	Time (s)	41.57	22.26	210.94	226.36	>1000	>1000	>1000	>1000	
	Solution	1839	1950	2755	36230	1326	1163	2465	2720	

Table 3.9 Results of the MTZ model in Gurobi with sparsified instances.

solution density compared to the results previously given with the original 2-matching SD.

Similarity of sparsified instances

We show in Figure 3.11 the arcs selected for 20%-sparsified instances with files `br17`, `ftv33` and `ftv35` from top to bottom. The nodes are represented in a circle in the order of an optimal solution. Figures (a) on the left are instances sparsified with 2-matching and Figures (b) on the right are instances sparsified with reduced costs only. We represent in blue dots the arcs that are kept in sparsified instances. We see that in general the structures are very similar, but for `br17` a lot fewer arcs are kept for the reduced costs even though a lot of those that are kept are also in the 2-matching-sparsified instance. The arcs that are lost from the optimal solution are those where the contour of the circle is not complete, for instance arcs $(8, 12)$ and $(14, 0)$ in the first figure. In Table 3.10, we look at more objective comparisons of the sets. We observe that usually the ratios and the Jaccard index are increasing as a function of the number of arcs, which is understandable as the sets tend to resemble the complete graph.

We recall the Jaccard index [63] of sets A, B is defined as $\frac{|A \cap B|}{|A \cup B|}$ and is symmetric. A Jaccard index close to 1 means the two sets are very close to one another. Since the number of arcs added with the two methods can differ, we also add the ratio (R) between the intersection

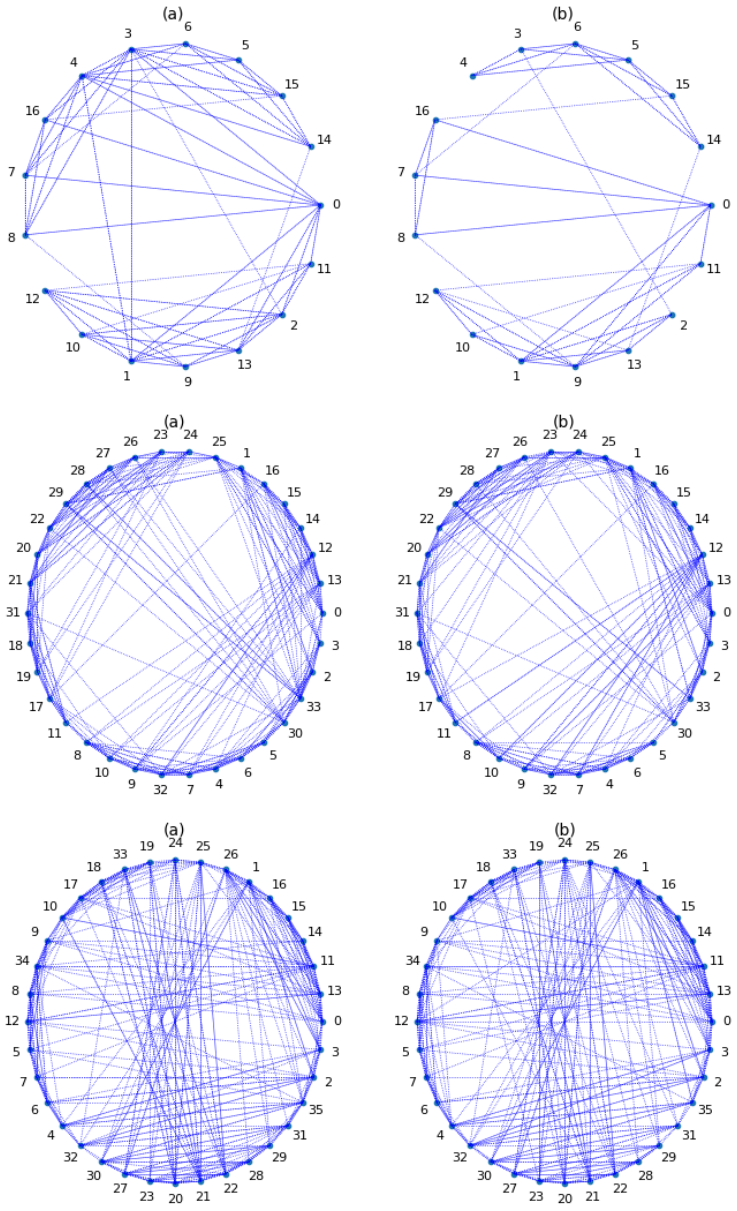


Figure 3.11 Sparsified instances with reduced costs (left) vs the real 2-matching solution density (right) for 20% of top arcs. From top to bottom: `br17`, `ftv33`, `ftv35`

and the smallest set $\frac{|A \cap B|}{\min(|A|,|B|)}$ that gives an upper bound on the similarity: if this value is close to 1, we tend to a situation where one set is included in the other. We see that the ratios are close to 0.9, which means 90% of one of the two sets are included in the other. The first three rows of the 20% column are the instances represented in Figure 3.11.

		br17	ftv33	ftv35	ftv38	ftv44	ftv47	ftv55	ftv64	ftv70	ftv170
5%	J	1.0	0.64	0.66	0.66	0.73	0.70	0.70	0.72	0.72	0.76
	R	1.0	0.79	0.85	0.85	0.89	0.87	0.84	0.88	0.88	0.91
10%	J	0.59	0.75	0.81	0.75	0.72	0.71	0.74	0.77	0.77	0.81
	R	1.0	0.90	0.97	0.93	0.90	0.89	0.91	0.88	0.91	0.94
15%	J	0.53	0.77	0.83	0.80	0.80	0.77	0.80	0.77	0.80	0.84
	R	1.0	0.89	0.94	0.93	0.90	0.89	0.91	0.88	0.91	0.94
20%	J	0.6	0.79	0.85	0.85	0.83	0.79	0.83	0.84	0.84	0.86
	R	0.95	0.90	0.93	0.94	0.92	0.89	0.92	0.92	0.93	0.95
30%	J	0.59	0.83	0.86	0.85	0.87	0.85	0.87	0.86	0.88	0.89
	R	0.93	0.93	0.93	0.93	0.94	0.92	0.93	0.92	0.94	0.95
40%	J	0.60	0.85	0.87	0.87	0.88	0.87	0.88	0.88	0.90	0.92
	R	0.90	0.93	0.94	0.94	0.94	0.93	0.94	0.93	0.95	0.96

Table 3.10 Similarity between instances sparsified with Reduced costs and 2-matching computed as the Jaccard index (J) of the arcs lists. We report the index for different instances and sparsification rates (columns).

Results of sparsified instances

We report in Table 3.11 and Table 3.12 the results of solving instances sparsified with reduced costs instead of 2-matching. This table is to be compared with Table 3.8 and Table 3.9: both are obtained with instances sparsified with the union of the 1-tree heuristic and a version of the 2-matching heuristic plus a trivial cycle included to have at least one solution. The percentages and instances are the same, we can compare the geometric averages of times and gaps to compare the efficiency of the two SDs. We observe in Table 3.11 that the gaps are a little bit higher and the average times a little bit faster, probably because in general there are a little fewer arcs in these instances. However, most instances are still solvable to optimality with 15% of the arcs, and it is especially the instance `br17` that induces higher gaps. We conclude that for further works, the reduced costs can effectively replace the 2-matching heuristic, at least for sparsification purposes.

		br17	ft53	ft70	ftv33	ftv35	ftv38	ftv44	ftv47	ftv55	g.avg
5	Time(%)	0.00	2.17	≤ 0.67	0.00	4.95	0.36	1.92	1.01	1.67	0.00
	Gap (%)	328.21	31.51	5.71	74.11	67.89	63.66	11.04	5.46	19.15	50.55
10	Time(%)	0.75	15.63	100.00	12.95	8.91	10.04	10.55	5.24	10.61	9.75
	Gap (%)	328.21	0.00	0.00	1.24	0.27	0.13	0.00	0.84	0.00	17.86
15	Time (%)	5.26	12.26	100.00	17.99	18.32	51.25	41.73	19.28	23.56	23.35
	Gap (%)	148.72	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	10.65
20	Time (%)	7.52	31.23	100.00	9.35	104.46	40.50	61.15	20.01	15.71	29.68
	Gap (%)	102.56	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	8.16
30	Time (%)	18.80	44.14	100.00	17.99	100.99	69.53	18.47	26.66	13.37	34.49
	Gap (%)	102.56	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	8.16
40	Time(%)	180.45	41.56	100.00	41.73	101.98	119.35	100.48	24.41	36.01	68.60
	Gap (%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	Time(%)	412.03	70.56	100.00	46.76	59.41	50.90	48.20	37.99	52.05	69.77
	Gap(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
60	Time(%)	415.04	125.74	100.00	37.41	169.31	43.73	60.91	46.84	34.75	80.58
	Gap (%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100	Time (s)	1.33	36.79	>1000	1.39	2.02	2.79	4.17	17.74	11.97	
	Solution	39	6905	38673	1286	1473	1530	1613	1776	1608	

Table 3.11 Results of the MTZ model in Gurobi with instances sparsified with reduced costs

		ftv64	ftv70	ftv170	kro124p	rbg323	rbg358	rbg403	rbg443	g.avg
5	Time(%)	9.48	9.61	8.59	5.06	≤ 0.13	≤ 0.57	≤ 0.64	≤ 1.90	2.08
	Gap (%)	0.65	0.36	0.00	0.31	76.32	248.50	58.01	60.59	41.20
10	Time(%)	12.08	28.57	9.71	6.38	≤ 0.39	≤ 1.37	≤ 2.95	≤ 1.05	3.70
	Gap (%)	0.00	0.00	0.00	0.00	2.34	49.36	17.00	12.34	9.11
15	Time (%)	16.79	35.76	15.22	7.78	≤ 1.38	≤ 7.51	≤ 3.66	≤ 2.52	7.15
	Gap (%)	0.00	0.00	0.00	0.00	0.00	1.20	3.08	1.91	0.77
20	Time (%)	48.83	73.58	20.82	6.63	≤ 2.15	≤ 4.15	≤ 4.00	≤ 2.95	9.22
	Gap (%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	Time (%)	17.25	90.57	41.02	14.08	≤ 5.57	≤ 3.31	≤ 3.76	≤ 7.06	12.04
	Gap (%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	Time(%)	46.36	84.41	114.12	19.99	≤ 4.61	≤ 3.00	≤ 6.35	≤ 6.39	16.31
	Gap (%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	Time(%)	64.95	95.42	34.32	31.04	≤ 7.10	≤ 4.97	≤ 7.91	≤ 7.27	18.44
	Gap(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
60	Time(%)	48.74	113.43	111.42	70.23	≤ 18.46	≤ 4.45	≤ 5.30	≤ 8.56	25.17
	Gap (%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100	Time (s)	41.57	22.26	210.94	226.36	>1000	>1000	>1000	>1000	
	Solution	1839	1950	2755	36230	1326	1163	2465	2720	

Table 3.12 Results of the MTZ model in Gurobi with instances sparsified with reduced costs

3.6 Conclusion

In this chapter we explored a search heuristic based on solution densities in two different CP models, as well as a sparsification method tested in four different models. We see that the CP search heuristic `costMaxSD` performs way better than the minDomain heuristic in our first model, but does not perform as well as the state of the art `minDeltaDeg` implemented in our second model. We were able to present several opportunities of improvement for speeding up the computation of the search heuristic in the Talos solver that may make it more competitive. We also explored the influence of the update parameter.
In terms of sparsification, we showed that the optimal solutions are kept with a low number of arcs. The sparsification induces high speedups in the MIP model Miller-Tucker-Zemlin, whereas the CP models are not or marginally sped up by this process.
Furthermore, this chapter presents the theoretical adaptation of the 1-tree solution density to asymmetric instances as well as an idea of new solution density based on cutsets, and an investigation of the best choice of parameter for 1-tree computation.

CHAPTER 4 REDUCING THE COMPUTATION OF THE TRAVELING SALESMAN PROBLEM WITH TIME WINDOWS BY A MIP MODEL

This chapter describes methods used to reduce the size of the instance described in the MIP model from Section 2.1.2 and solved with Gurobi. The model itself will stay the same, but some of the variables may be removed and some additional constraints will be added. We describe how we define and compute solution densities, the reduction methods that we use to simplify the problem and then report the experimental results we obtained with this model.

4.1 Computing densities

4.1.1 Overlap of time windows

In Section 2.3.5, we transformed the time windows into a partial order on the nodes. However, that construction only takes into account non-overlapping time windows to begin with. We could also pay attention to the importance of the overlap compared to the size of the windows: two time windows with a small interval of time possible for s_i with $i \prec j$, could be considered to be "almost ordered". This would mean not only looking at the condition $a_i + d(i,j) > b_j$ but also looking at the relative values of a_i, b_i, a_j, b_j and $d(i,j), d(j,i)$ when the condition is not respected.
We recall that for two random variables X_i and X_j following a uniform distribution on intervals $I_i = [a_i, b_i]$ and $I_j = [a_j, b_j]$ the probability of having $X_i < X_j$ can be written as

$$\mathbb{P}(X_i \leq X_j) = 1 - \frac{1}{2}\frac{\mu(I_i \cap I_j)^2}{\mu(I_i)\mu(I_j)}$$

where we suppose without loss of generality that $a_i \leq a_j$.
Instead of using exactly the algorithm from Section 2.3.5, we can replace the original order with this variant. Indeed, the cardinality of a set $S \subseteq E$ can be written as $|S| = \sum_{x \in E} \mathbb{1}_{x \in S}$. We can thus replace every set cardinality $|S|$ by $\sum_{x \in E} \mathbb{P}(x \in S)$, which is a relaxation of the ordering property where we have any value between 0 and 1 instead of $0-1$ values. We need to take the travel times $d(i,j)$ and $d(j,i)$ into account as they are of first importance in declaring a partial order. For instance, we can define X_i on $[a_i + d(i,j), b_i]$ and X_j on $[a_j + d(j,i), b_j]$. This relaxation can be used by itself or as a first brick for the algorithm from Section 2.3.5. It will then give an alternative solution density. The sections below refer to solution densities computed with or without taking into account the overlap of the time windows. The empirical comparison between the algorithm based only on non-overlapping

windows and this variant is reported in Section 4.3.8.

4.1.2 A solution density for the edges

There are two sets of variables in our model. The continuous variables s represent the start time for the visit of each vertex, and the binary variables x represent whether an edge is included or not in the model. Ordering constraints $i \prec j$ can be directly translated in terms of start times: $i \prec j \Leftrightarrow s_i \leq s_j$. This means that the solution density based on time windows defined in Section 2.3.5 is defined on start times variables. However, it is interesting for us to also have information on the x_{ij} variables, since it will be easier to compare with the solution densities based solely on the graph structure. We will denote $i \to j$ the fact that vertex j is visited directly after vertex i, which is equivalent to $x_{ij} = 1$.
The goal of this section is to underline the links between the probabilities of $i \prec j$ and $i \to j$: in general we have $i \to j \Rightarrow i \prec j$, but can we define a more specific equation for the probability of the edges? More precisely, we can write the probability for this event as

$$\mathbb{P}(i \to j) = \mathbb{P}(i \prec j \wedge \forall k, (i, j \prec k \vee k \prec i, j))$$

,
meaning that all the other vertices are either before or after i and j.
In general, there is no reason to suppose that events $i \prec j, i \prec k, \ldots$ are independent: on the contrary, the transitivity property means there are necessarily links among the precedence constraints. However, we can approximate a density on the variables x_{ij} by assuming that independence and writing

$$\mathbb{P}(i \to j) \simeq \mathbb{P}(i \prec j) \prod_{k \neq i,j} (\mathbb{P}(i \prec k)\mathbb{P}(j \prec k) + \mathbb{P}(k \prec i)\mathbb{P}(k \prec j)) \tag{4.1}$$

When the densities σ_{ij} for $i \prec j$ are computed with one of the two methods described before, the density for each edge (i, j) can be computed in linear time, which gives an additional pre-computing complexity of $\mathcal{O}(n^3)$. This expression will be used when we want to use the time windows to have a SD on the edges.

4.2 Reduction methods

4.2.1 A first exact reduction

The densities computed above will be used later on in approximated reductions of the problem. We can observe right away that extreme values for those densities have good properties

that will help reduce the instance without losing any solutions. We denote as d_{ij} the density attributed to edge (i, j) in Equation (4.1).

Theorem 4.2.1. *The values $d_{ij} = 0$ and $d_{ij} = 1$ are respectively equivalent to impossible and mandatory edges in any solution.*

Proof. We will note $\mathbb{P}(i \prec j) = p_{ij}$ and $\mathbb{P}(i \to j) = d_{ij}$ here.
Let us first suppose that for a tuple (i, j) we obtain $d_{ij} = 1$. Since all p_{ab} are in $[0, 1]$, necessarily, $p_{ij} = 1$ and for each $k \neq i, j$, $p_{ik}p_{jk} + p_{ki}p_{kj} = 1$. We always have $p_{ki} = 1 - p_{ik}$ and $p_{jk} = 1 - p_{kj}$: by studying the function $(x, y) \mapsto xy + (1 - x)(1 - y)$ for (x, y) in $[0, 1] \times [0, 1]$, we see that the value 1 is maximal and only obtained for $(x, y) = (1, 1)$ or $(0, 0)$. It means that we either have $k \prec i, k \prec j$ or $i \prec k, j \prec k$, which means that no vertex is between i and j. With $i \prec j$, it means that $i \to j$.
The reciprocal implication is trivial.
Suppose now that for a tuple (i, j) we obtain $d_{ij} = 0$. Since d_{ij} is a product, we have either $p_{ij} = 0$ or for some $k \neq i, j$, $p_{ik}p_{jk} + p_{ki}p_{kj} = 0$.
The first case directly implies that edge (i, j) cannot be in a solution.
The second case means that we have $p_{ik}p_{jk} = p_{ki}p_{kj} = 0$. If for instance $p_{ik} = 0$, we have $p_{ki} = 1$, which means that $p_{kj} = 0$ and that $p_{jk} = 1$. The symmetric case is also possible. In both cases, we necessarily have a vertex k such that $i \prec k \prec j$ or $j \prec k \prec i$, which means that edge (i, j) is impossible.
Again, the reciprocal implication is trivial by looking at a k such that $i \prec k \prec j$. □

From this simple result, we know that fixing to 1 the values of variables x_{ij} corresponding to edges of densities $d_{ij} = 1$ and removing the variables of edges with densities $d_{ij} = 0$ will not change the set of feasible solutions of an instance. The theory behind it shows that this edge elimination could be induced by simply mandatory precedences: having $i \prec j \prec k$ makes edges $(j, i), (k, j), (k, i)$ and (i, k) impossible in any solution. However, this simple deduction seems not to be caught by the basic preprocessing from Gurobi since we do in fact observe a speedup in many cases (see in results below). Gurobi has some parameters `PreSolve` and `PreSparsify` that make the presolve of the problem more agressive. Still, setting these parameters to their maximal values does not seem to catch these elimination rules.
This first reduction of the problem can now be the base case of all of the heuristic reductions that follow. Moreover, when we add the sparsification method from Section 4.2.2 afterwards, the corresponding edge densities are computed anyway, hence not losing any precomputation time with this elimination as it is a byproduct of the edge densities' computation.

4.2.2 Sparsifying with the edge densities

Once we have an adaptation of our solution density defined for the edges, we can use the same methods we used to sparsify with the basic TSP and remove unpromising edges from the model, hence limiting the number of variables in the model. With time windows, sparsifying the graph can lead to serious feasibility issues, as the 1-tree and 2-matching solution densities do not take the time window constraint into account: while most benchmarks have complete graphs and thus include a lot of tours, some instances have narrow time windows and specific edges need to be included in any tour respecting the time constraints.
Each edge can be described by a number of solution densities: here we have three different solution densities that are 1-tree, 2-matching and a solution density on time windows. This can be represented by a 3-dimensional vector. Selecting a subset of the edges based on their solution densities is the same as defining a subset of the 3-dimensional vector space of "promising edges". In our case, we will rather look at the relative rankings than the solution densities themselves: we can for instance select the union or the intersection of the top $k\%$ best ranking edges in each solution density, or have a variable percentage of edges selected in each ranking. The following paragraph investigates the structure obtained by each of these methods.

Distribution of optimal edges

Figure 4.1 shows the distribution in the 2-dimensional space of 1-tree and 2-matching rankings of the edges of the instance `ftv35`. This is an asymmetric instance. The coordinates of each dot in the graph represents the rankings of an edge for both densities. To visualize "good" edges, we plotted in blue the edges from one optimal solution: there may be other edges in other optimal solutions, we only look at one example. The orange points are the remaining edges of the instance.
The first observation we make is that there is a general correlation between the two densities: while the two densities are different, the general cloud of points is distributed along the diagonal. This means that some of the information from the two densities may be redundant, but is also a good indication that the solution densities could both be linked to the problem under study.
The only difference between the three other images is the black line separating the "good" edges from the "bad". In Figure 4.1b, we select the intersection of about 80% of both rankings, corresponding to the 0.8 coordinates of the black line's intersection with axes. In Figure 4.1c, we select about 40% of the rankings, which is the complementary of taking the intersection of 60% of bad edges. The last image, Figure 4.1d, represents a more complicated rule that

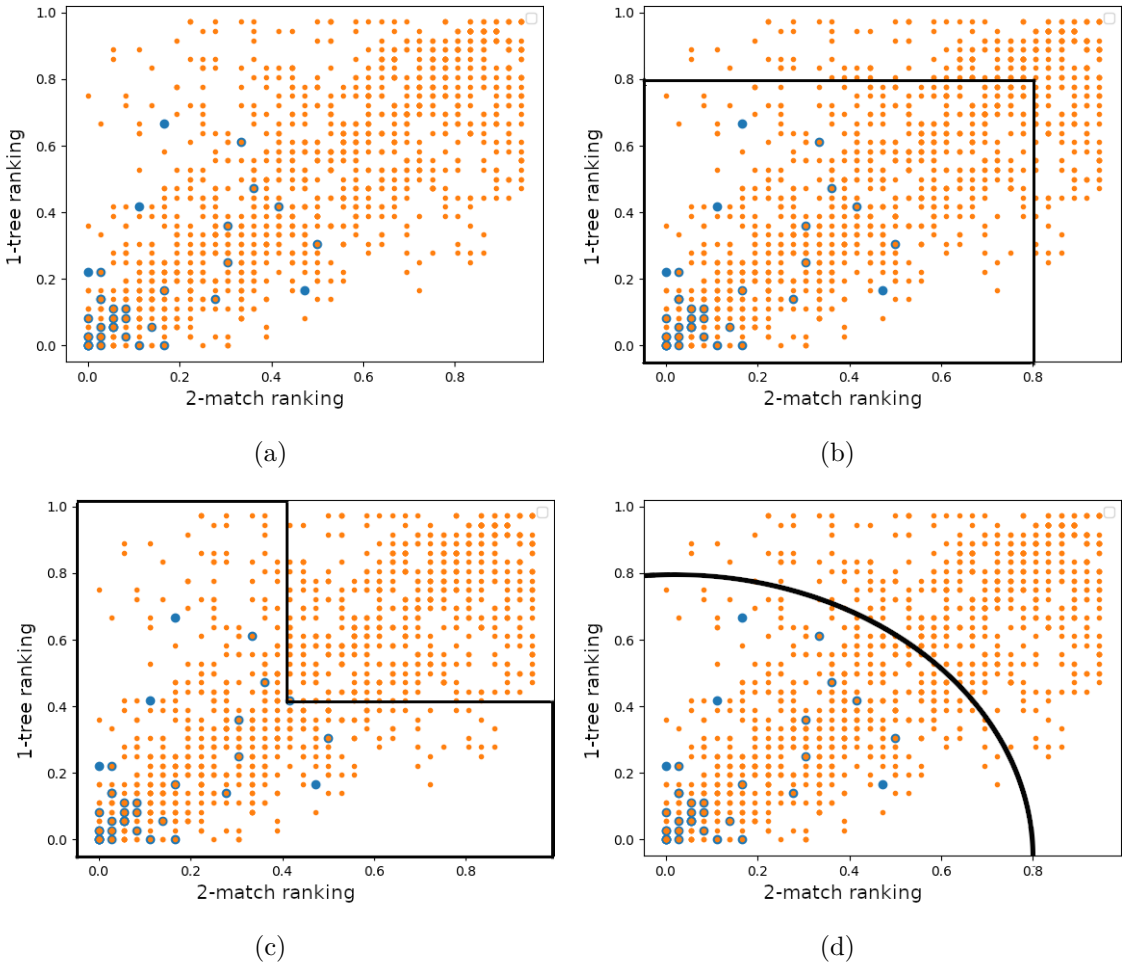


Figure 4.1 Distribution of edges in the SD-ranking space and three subsets of the space for file `ftv35`. 4.1a is the set of points in the space. On the three other subfigures, a black line shows the subset of points defined by a rule on the SDs: 4.1b intersection of rankings, 4.1c union of rankings, 4.1d maximum norm for the vector of ranks

could be defined by a circle equation: by calling r_{2m} and r_{1t} the normalized rankings of respectively 2-matching and 1-tree, the subset could be defined by $r_{2m}^2 + r_{1t}^2 \leq 0.8$ and is no longer simply defined by linear inequalities.

The objective to have an efficient filtering is to find a rule that minimizes the number of orange points while keeping all of the blue ones. Since a lot of orange points are found in the convex hull of the blue points, we have no hope of drastically sparsifying with simple rules, but we see that there are wide regions with no blue points in the upper-right corner, which is the region we try to remove from our models. In this specific case, we see without counting that Figure 4.1b contains a lot more points, but a whole section between 0.6 and 0.8 for the 2-matching ranking contains no blue point. An additional option is to uncorrelate

the percentage for each SD: we could have taken the top 60% of the 2-match ranking and the top 80% of the 1-tree ranking in this case. We also see from the correlation between solution densities that the union case Figure 4.1c contains few points on the upper-left and bottom-right corners, while the intersection from Figure 4.1b adds a lot of edges when the area grows in the upper-right direction. Since our goal is to minimize the number of orange points and not the global area selected, the union of ranks is a good candidate as we gain consistency without adding too many uninteresting edges.
Our last observation is that for each form of the subset, the critical edges, i.e., the blue points closer to the border, are different points, which means different filtering methods will yield different results. We should keep in mind that these figures are taken from one single instance that does not necessarily represent all of the instances, and that the blue edges are taken from one single optimal solution; the figure is given mostly for visualization and not to be fully representative.

Experimental parameters

After experiments on solving instances sparsified with union or intersection of rankings, we will use only the union in the following methods, but other rules can be investigated.
More specifically, in Section 4.3.4, we use the top $k\%$ of edges for each solution density among 1-tree, 2-matching and the solution density on time windows. We rank the incoming and outgoing edges of each vertex in decreasing order for each SD and select the top $\frac{kn}{100}$ in each of these rankings. Selecting edges for each vertex makes the degrees uniform in the graph, impeding a distant vertex to be disconnected when we look for a tour.

4.2.3 Adding ordering constraints

We describe another method for using these solution densities based on the time windows that does not use the solution densities from the TSP or the adaptation into edges: we simply look at the basic densities computed with the algorithm from Section 2.3.5.
Since these densities are related to the orderings $i \prec j$, we can simply rank the orderings by their solution densities and add additional linear constraints in the model of the form $s_i \leq s_j$. To make the constraints tighter, and since there are travel times, we can add constraints $s_i + d(i, j) \leq s_j$: additional constraints reduce the search space and will possibly reduce the number of search nodes.
We can see this method as the use of the time windows' SD alone for sparsification.

4.2.4 Fixing edges

We are looking at ways to combine our different solution densities to gain strength in our reduction and not simply use the different SDs in parallel. However, the solution density we defined on edges is based on an approximation without theoretical guarantee, and the previous subsection 4.2.3 describes a method that uses only the SD on time windows. In this section, we try to use the information of TSP and TW densities together without approximation. This last method uses only the implication that

$$i \to j \Rightarrow i \prec j \tag{4.2}$$

This implication is always true. A high probability that $i \prec j$ cannot directly be linked to a high probability that $i \to j$. On the contrary, one might even think that a high probability for $i \prec j$ means i will appear long before j and several other nodes will be visited in between. However, we can use the contrapositive from (4.2) to eliminate edges: a high probability that $i \prec j$ is linked to a very low probability for the edge $j \to i$.
Instead of our usual method of limiting the number of variables by removing unlikely edges, we will select some of the variables from the start. We rank the edges based on the best solution density among the 1-tree and 2-matching SDs, and thus get a ranking on all edges. We will then select some of these edges by removing the corresponding x_{ij} variable and writing the model with already an outgoing degree of 1 for node i and an incoming degree of 1 for node j.
We will need three parameters for this method: we will look at the top a edges for the 1-tree and 2-matching solution densities and select a maximum of b edges among them if and only if the corresponding precedence solution density is above a threshold of t: $\mathbb{P}(i \prec j) \geq t$
All of the experiments reported here use $t = 0.5$, which corresponds to the natural interpretation of "this edge is likely to at least be in the right direction". The double-setting of a and b allows to control both the number of additional constraints and the minimal likelihood of these constraints: if none of the top-ranking edges are likely to be in a solution that respects the time windows constraints, it is better not to add any of them than to add very bad edges.

4.3 Experimental results

4.3.1 Benchmarks

The benchmarks used in all of the following experiments are available at [64] as well as some information on best solutions. They consist of five different datasets.

- Dumas instances are parameterized by the number of nodes and the width of their time windows. The benchmark includes 135 instances from sizes 20 to 200 with time windows of width 20 to 100. For each size-width tuple in the benchmark, 5 different instances of the same parameters exist. For instance, the instance `n100w40.003` is the third of all five instances of size 100 with time-windows of size 40. Smaller width means the instances are more constrained in time and are usually easier to solve.

- AFG instances from [27] is the second biggest benchmark with 50 instances. The instances have varying sizes between 10 and 230. We use instances of sizes until 150 that can all be solved in less than a few hours for the longer ones.

- The 25 OhlmannThomas instances are of the same form as Dumas instances, but the sizes are 150 and 200 and the time windows sizes' vary from 120 to 160, which makes them much harder. Our results will show that more instances are not easily solved in several hours or even days.

- SolomonPesant instances are 27 instances of sizes between 20 and 45. The weights of the edges are not integral and the time windows are not of constant size, but some of them are usually very wide.

- SolomonPotvinBengio are 30 instances similar to SolomonPesant instances, with a few smaller instances (one of them only contains 4 nodes).

In all the following benchmarks, we report the results for instances that take at least $10s$ to solve in the model: first, because there is little to gain for these instances that do not really represent a challenge; second, because by doing so, we condense useful information in smaller tables. We use a timeout of $2,000$ seconds for benchmarks Dumas and AFG, $20,000$ seconds for Solomon* benchmarks, and a timeout of $200,000$ seconds for the benchmark OhlmannThomas, which is still not long enough to solve most of the instances. Because there is little relevant information when most instances reach the time limit, the OhlmannThomas results will be reported mainly in Appendix, as well as the Dumas results that consist of lots of instances and take a lot of room. We report in the following experimental results sections mainly results for the Solomon-Pesant, Solomon-Potvin-Bengio and AFG benchmarks, which makes a total of about 50 instances reported in full.

4.3.2 SDs computing times

In order to evaluate our reduction methods, we need to compare the times gained in the resolution with the losses of optimality and the times needed for preprocessing. This section

reports the times needed to compute the solution densities, so they can be compared with the times needed for resolution in the following sections.
The computing times reported in Tables 4.1 and 4.2 are averages of computing times on instances of the same size of the Dumas benchmark. This particular benchmark has the advantage of featuring instances of various "round" sizes, which gives a good idea of the dependence of the computation on the size of the instance. However, the size is not the only parameter impacting the computing times. The width of the time windows and number of comparable time windows (meaning there is a partial order defined between them) impacts heavily the computing times for the Time Windows densities.

Computing TSP densities

Size	1-tree (ms)	2-matching (ms)	Reduced costs (ms)
20	17	30	≤ 1
40	85	443	1
60	276	2 230	3
80	687	7 009	5
100	1 426	17 106	9
150	5 625	86 199	22
200	16 444	272 050	46

Table 4.1 Average time needed to compute three types of solution densities for different sizes of instances from the Dumas benchmark

For instances of size larger than 100, the computing time with the original 2-matching computation method is comparable to the resolution times and is a real problem to make our reductions based on TSP solution densities competitive. We see in Table 4.1 that replacing the 2-matching density with reduced costs as mentioned in Section 2.3.4 allows a more than 1000× speedup, making its computation negligible compared to that of 1-tree. The results obtained by replacing 2-matching by the reduced costs are shown in Section 3.5.6 to be very similar to the original SD.

Computing TW densities

The times needed for computing the time windows' densities are reported for both methods: heuristic and exact algorithm in Table 4.2. We add separately the transformation into edge densities.

Size	Heuristic (ms)	Exact (ms)	Edge densities (ms)
20	12	272	9
40	100	31 144	52
60	406	814 282	162
80	1 311	$\geq$ 1000s	369
100	3 048	$\geq$ 1000s	761
150	18 050	$\geq$ 1000s	2 751
200	68 110	$\geq$ 1000s	7 274

Table 4.2 Average time needed to compute the time windows solution density for different sizes of instances from the Dumas benchmark

The exact algorithm's complexity relies heavily on the width of the partial order, which is the cardinality of the maximal set of incomparable elements. This depends on the number of overlapping windows, i.e., it depends on the width of these windows: wider windows imply fewer comparable windows and means a higher complexity for the exact algorithm. The times reported here are thus only accurate for the Dumas benchmark, and others such as the AFG benchmark may take longer when fewer time windows are comparable.
Even for small instances, the exact algorithm is quickly dismissed as the order of magnitude of its computing time is much higher. We compare in Section 4.3.7 the results obtained with the exact algorithm vs. the approximation, but we can already suppose that the exact algorithm will be of little interest for bigger instances, since on top of the complexity, the implementation is more complicated and there is more risk of numerical instability. However, we can imagine using the exact algorithm during search in CP, when most of the problem is already fixed.

4.3.3 Exact reduction

Number of removed / fixed edges

Table 4.3 reports the number of edges selected or removed after the exact reduction. The number of mandatory edges is a percentage of the instance's size, since the total number of edges in a tour is n. The optional and deleted edges are in percentage of the total number of edges (which is $n(n-1)$).

We observe that as predicted the number of mandatory and deleted edges decreases as the width of the windows increases for each size, while the number of remaining optional edges (variables) increase. With windows of size 20 there always are mandatory edges, with a maximum of 42% of needed edges found with a size of 20. The number of remaining variables is divided by almost 20 for size 200 and windows of size 20.

Size	Width	Mand. (% of n)	Opt. (%)	Del. (%)
20	20	42.00	16.81	81.19
	40	14.00	27.19	72.14
	60	2.00	38.00	61.90
	80	0.00	46.90	53.10
	100	0.00	57.67	42.33
40	20	20.00	11.90	87.61
	40	2.50	19.38	76.90
	60	0.50	30.20	69.79
	80	0.00	35.22	64.78
	100	0.00	44.38	55.62
60	20	12.67	9.32	90.47
	40	6.33	16.16	83.73
	60	1.67	22.32	77.66
	80	1.33	31.03	68.95
	100	0.00	37.32	62.68
80	20	9.75	8.19	91.69
	40	3.00	14.23	85.73
	60	0.50	20.81	79.18
	80	0.25	25.97	74.03
100	20	6.60	5.97	93.92
	40	2.40	12.71	87.27
	60	1.40	19.31	80.68
150	20	4.93	5.72	94.25
	40	2.67	10.92	89.06
	60	1.20	14.81	85.18
200	20	3.70	5.09	94.89
	40	1.40	9.46	90.53

Table 4.3 Influence of the exact reduction on the number of edges in each instance of the Dumas benchmark. The Mand. column is the percentage of the tour found after the filtering. The Opt. and Del. columns are respectively the optional and deleted edges of the remaining. Optional edges will be the variables of the problem.

The number of edges that can be removed or fixed is heavily dependent on the number of already ordered nodes (and thus on the width of time windows) so we report averages for a constant time windows width.

Resolution times

We report the speedup gained by this reduction in Table 4.4. Here, the speedup observed for the Dumas benchmark really outperforms the other benchmarks, although the Ohlmann-

Thomas result is biased. Indeed, while the average ratio is high, this is due to timeouts from most of the instances after 200,000 seconds. Without any filtering, only one of the 25 instances finds a solution before timing out, while 16 find at least one solution in the filtered instances.

Benchmark	geom. avg filtered/base	arit. avg filtered/base
Dumas	13.49	28.30
AFG	44.72	85.17
Solomon-Pesant	81.41	103.0
Solomon-Potvin-Bengio	61.91	84.41
OhlmannThomas	44.35	75.08

Table 4.4 Average percentage of time needed by the exact reduction on different benchmarks. The rows are the sets of instances described before and the columns report the geometric and arithmetic averages of the fraction of times, i.e., the ratio in % between the filtered version and the original one. A value close to 100% means the exact reduction does not change the times needed for resolution.

4.3.4 Heuristic sparsification

This section reports the experimental results from the method described in Section 4.2.2. We sparsify the instances on top of the exact reduction introduced in Section 4.2.1. We select a percentage of n of the edges, which means a low rate of sparsification may converge rapidly towards the reduced instance. We could also take a percentage of the remaining edges, but in the following we decided to keep a percentage of n to make sure that the instance contains the instance sparsified without taking the time windows into account. Since we saw that for the Dumas benchmark the number of remaining edges (see Table 4.3) was below 50%, we will need to look at very low rates of sparsification to reduce the instance. The results for the Dumas benchmark are reported in Appendix. For other benchmarks, the speedup is smaller, so the reduced instances are closer to the original instance. We will look at percentages up to 80% of the edges.

Sparsified instances

We represent in Figure 4.2 the instances sparsified with only TSP densities (left) and the instances sparsified adding also the edge density from time windows (right). On the top, we have instance `rbg019c` and on the bottom, instance `rbg021.3`. We take the union of the best-ranking edges for the three SDs. Since we first do our exact reduction, the instance

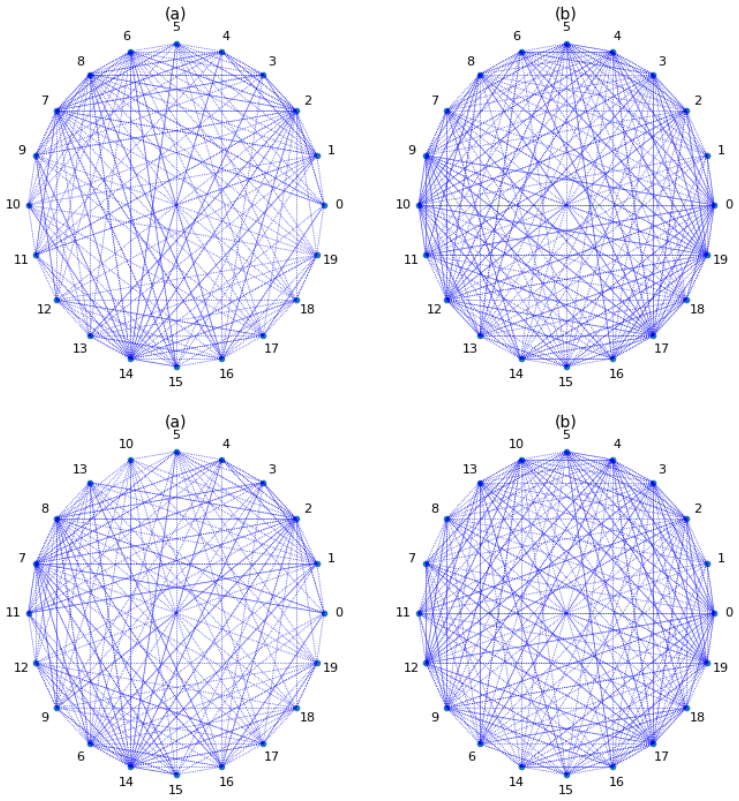


Figure 4.2 Difference between instances sparsified with two and three solution densities. The figures on the right also include the best-ranking edges for the SD on edges based on time windows. We see that a lot of additional edges are added, and that the degrees of the vertices are evenly distributed. The order of the circle is the order of the vertices in an optimal solution. Note: the orientation of the edges are omitted in the figures.

from the left may not be exactly contained in the instance from the right, but the edges from the optimal tour should be included in one another, since they are not removed by our exact filtering. The only difference that could appear there would be due to the rounding of the number of edges, since we always take an integral number of edges based on a percentage. In the first instance reported, the optimal tour was already included in the instance with only TSP densities. In the second, we see that we were able to recover edge $(13, 10)$. As opposed to the first similar visualization we did in Figure 3.11, having edges on the tour is not a sufficient condition to have the optimal tour as the orientation matters, but adding arrows would make the graph very difficult to read, so we keep it as a simple indication. We see that taking the top k edges for each vertex creates a very well-distributed instance, where each node has approximately the same degree.

SolomonPesant

We report in Table 4.5 the results for the benchmark SolomonPesant. The rows represent the different instances of the benchmark, with column "Base" corresponding to the exact reduction described in Section 4.2.1. The other columns represent configurations of different sparsification rates, with the percentage of time and of gap relative to the "Base" column. For instance, a 100.00 in the 80% column means that the solving times are the same as without sparsification. A 0.00 in the 80% gap column means that the optimal solution is kept during the sparsification process. A 5.00 in the 80% gap column means that the best solution that was found was of weight $1.05 \times z$ where z is the cost of the best solution found in the "Base" configuration. Xs represent cases where no solution was found at all. Positive results for our methods are when the time columns are under 100.00 and as small as possible, and the gaps are as close to 0 as possible.

We see from the results in Table 4.5 that an aggressive sparsification makes us lose both solutions and efficiency, with most of the instances not being solved as fast and high values for the gaps. Although we used the same general method as in the previous chapter for sparsification, we do not have the same results: our lower rates of sparsification give small gaps but not a real speedup and by pushing the cursor further, we actually make things worse.

Instance	Base		80%		50%		30%	
	Time (s)	Solution	Time (%)	Gap (%)	Time (%)	Gap (%)	Time (%)	Gap (%)
rc203.0	**20000**	727.453	100.00	0.00	100.00	0.00	100.00	5.81
rc203.1	**20000**	726.991	100.00	0.00	100.00	0.00	100.00	4.14
rc203.2	103.39	617.465	56.34	0.00	21.02	0.00	41.15	3.88
rc204.0	1249.66	541.448	215.66	0.00	302.00	0.00	843.25	10.73
rc204.1	244.56	485.367	180.65	0.00	47.34	2.45	238.80	24.28
rc204.2	**20000**	778.395	100.00	0.00	100.00	0.00	X	X
rc205.2	65.15	714.695	182.06	0.00	180.18	0.00	226.05	1.11
rc206.0	107.22	835.232	71.68	0.00	62.03	0.00	1988.60	3.85
rc206.1	15.27	664.731	102.49	0.00	209.04	0.00	595.09	3.41
rc207.0	85.85	806.689	68.60	0.00	87.68	0.00	12900.44	4.69
rc207.1	234.68	726.359	43.73	0.00	34.66	0.00	23.45	2.82
rc207.2	10.79	546.405	94.25	0.00	123.73	0.00	73.12	4.66
rc208.0	**20000**	inf	X	X	X	X	X	X
rc208.1	229.71	509.041	96.80	0.00	17.31	0.00	8706.64	24.88
rc208.2	54.17	503.923	20.07	0.00	94.61	0.00	1587.10	9.70
		geometric mean	87.78	0.00	79.90	0.17	400.29	7.75
		arithmetic mean	102.31	0.00	105.69	0.17	2109.51	7.99
		median	98.40	0.00	97.31	0.00	238.80	4.66

Table 4.5 Times and gaps to optimality for sparsification in the benchmark SolomonPesant

SolomonPotvinBengio

We report in Table 4.6 the results of the same method applied to the SolomonPotvinBengio benchmark. While the results are a little less catastrophic than in Table 4.5 and we even have positive speedups on average with higher number of edges, we see that the more aggressive configuration still induces high resolution times and high gaps. The median also stays around 100%, which means that as many instances are sped up than slowed down in the process.

Instance	Base		80%		50%		30%	
	Time (s)	Solution	Time (%)	Gap (%)	Time (%)	Gap (%)	Time (%)	Gap (%)
rc_202.1	**20000**	771.776	90.62	0.00	100.00	0.00	25.24	0.43
rc_202.4	97.77	793.030	57.37	0.00	49.91	0.00	111.24	2.85
rc_203.2	**20000**	784.158	100.00	0.00	100.00	0.00	44.67	4.98
rc_203.3	**20000**	817.526	100.00	0.00	100.00	0.00	61.65	4.71
rc_204.1	**20000**	878.640	100.00	0.00	100.00	0.00	X	X
rc_204.2	**20000**	662.159	100.00	1.39	100.00	2.53	100.00	11.38
rc_204.3	172.06	455.031	104.86	0.00	82.53	0.00	157.08	8.42
rc_205.3	17.60	825.058	141.76	0.00	154.72	0.00	6997.22	6.77
rc_205.4	29.70	760.470	142.36	0.00	93.97	0.00	47.07	0.00
rc_206.2	7104.06	828.059	45.86	0.00	31.95	0.00	281.53	3.31
rc_206.3	449.35	574.418	69.44	0.00	103.14	2.53	875.96	20.78
rc_206.4	**20000**	831.670	100.00	0.00	100.00	0.00	100.00	1.28
rc_207.1	12515.56	732.683	43.70	0.00	50.46	0.00	63.37	5.16
rc_207.2	**20000**	711.487	X	X	X	X	X	X
rc_207.3	**20000**	682.404	100.00	-0.00	100.00	0.29	100.00	0.64
rc_208.1	**20000**	inf	X	X	X	X	X	X
rc_208.2	81.88	533.780	138.97	0.00	119.20	0.00	X	X
rc_208.3	**20000**	640.399	100.00	0.00	100.00	1.01	X	X
		geometric mean	90.78	0.09	87.53	0.39	137.45	5.30
		arithmetic mean	95.93	0.09	92.87	0.40	689.62	5.44
		median	100.00	0.00	100.00	0.00	100.00	4.71

Table 4.6 Times and gaps to optimality for sparsification in the benchmark SolomonPotvinBengio

AFG

We report in Table 4.7 the results of the sparsified instances in the benchmark AFG. We see that compared to the two previous tables, we are able to go to higher rates of sparsification, which can be explained by the number of edges kept by our first reduction. Indeed, we see from Table 4.4 that the exact reduction induces a higher speedup in this benchmark than the Solomon*: this is linked to a lower number of edges kept after the reduction. When keeping 80% of the edges in the sparsification, we probably do not remove additional edges. However, this benchmark gives very promising results, with some instances being solved $10,000$ times

faster (0.01%). The average gaps are still quite high, especially with several instances going over a 5% gap.

Instance	Base		30%		20%		10%	
	Time (s)	Solution	Time (%)	Gap (%)	Time (%)	Gap (%)	Time (%)	Gap (%)
rbg019c	321.22	4536	1.11	0.24	0.03	0.24	0.01	0.24
rbg020a	69.18	4689	0.06	0.00	0.07	0.00	0.03	0.00
rbg021.3	127.28	4528	2.03	0.42	0.09	0.42	0.01	0.42
rbg021.4	108.5	4525	2.92	0.49	0.09	0.49	0.01	0.49
rbg021.5	25.84	4515	7.24	0.69	0.70	0.71	0.04	0.71
rbg021	330.12	4536	1.01	0.24	0.02	0.24	0.00	0.24
rbg027a	**2000**	inf	0.12	-100.00	0.00	-100.00	0.00	-100.00
rbg041a	95.7	2598	1.22	3.31	0.72	8.43	0.19	10.66
rbg042a	31.15	2772	97.40	4.51	0.22	4.94	0.19	7.22
rbg048a	**2000**	inf	X	X	X	X	X	X
rbg086a	65.94	8400	21.37	0.00	29.51	0.00	3.47	3.17
rbg092a	51.11	7158	170.89	0.00	70.12	0.39	7.65	5.57
rbg125a	20.84	7936	49.28	0.00	46.31	0.00	141.84	5.47
rbg132.2	729.94	8191	74.78	0.00	274.00	1.43	61.44	10.16
rbg132	31.36	8468	53.16	0.00	76.02	0.00	1220.03	2.17
rbg152.3	**2000.06**	9789	100.00	0.49	X	X	X	X
rbg152	30.57	10032	96.99	0.00	100.52	0.00	278.80	1.50
	geometric mean		8.01	0.68	0.00	1.21	0.00	3.37
	arithmetic mean		42.47	0.69	39.89	1.23	114.25	3.43
	median		14.31	0.21	0.70	0.32	0.19	1.84

Table 4.7 Times and gaps to optimality for sparsification in the benchmark AFG

Dumas

Since there are a lot of instances in the Dumas benchmark, the full results are available in Table B.1 in Appendix. Table 4.8 reports only the average results, where we see that although the gaps induced by sparsifying are small, the sparsification does not help the solving times.

	30%		20%		10%	
	Time (%)	Gap(%)	Time(%)	Gap(%)	Time(%)	Gap(%)
geometric mean	136.94	0.54	124.06	0.34	104.92	1.43
arithmetic mean	212.82	0.56	173.86	0.35	126.65	1.47
median	111.82	0.00	107.07	0.00	100.00	0.41

Table 4.8 Average times and gaps for sparsifying in the Dumas benchmark

Global analysis

We see that the method gives uneven results over the different benchmarks in terms of time, but induces high gaps to optimality in a number of instances. While for the ATSP alone, we saw in the previous chapter that the sparsification based on SD gave very promising results, using the times windows as a density on edges does not yield the same improvements. This can be because our transformation into edge densities is based on a rough approximation, and also because we do not control the feasibility as well. Indeed, while most instances are complete and thus contain a lot of feasible tours of different weights, the time windows restrict the possible orderings of the visits.

4.3.5 Ordering constraints

In this section, we report the results of the "ordering constraints" method described in Section 4.2.3. The percentages giving names to the columns are percentages of the instances' dimensions. For instance, the 50% column for an instance of size 20 corresponds to an addition of 10 ordering constraints of the form $s_i + d_{ij} \leq s_j$. Since there are an order of magnitude of n^2 pairs of nodes, 100% is not an upper bound here. However, since a solution can be described by n ordering constraints $s_0 \leq s_1 \leq \ldots \leq s_n$, we choose a parameter linear in n.

SolomonPesant

We report the results for the method of adding ordering constraints in Table 4.9 with 50, 100 and 150% of n constraints added to the model.

We see in this table that adding ordering constraints has on average a negative effect, increasing by a lot the time needed for solving. The difference between geometric and arithmetic means is linked to the small number of instances that really have big values for the time ratio: on the other hand, some instances are sped up by the process. This observation will be confirmed in the following sections, with some instances influencing negatively the average while the majority is usually sped up, as we can see from the medians that are at 100% or under. This particular benchmark performs once again worse than the ones in SolomonPotvinBengio and AFG reported next.

SolomonPotvinBengio

In Table 4.10, we see that the geometric means of the solving times indicate a general speedup as we add ordering constraints, while some instances become infeasible with 100% and 150% of ordering constraints. Nevertheless, three instances cause high values which disturb the

Instance	Base		50%		100%		150%	
	Time (s)	Solution	Time (%)	Gap (%)	Time (%)	Gap (%)	Time (%)	Gap (%)
rc203.0	**20000**	727.453	100.00	0.00	100.00	0.10	100.00	1.95
rc203.1	**20000**	726.991	100.00	0.00	100.00	0.00	100.00	0.64
rc203.2	103.39	617.465	51.76	0.00	13.78	3.42	8.35	3.42
rc204.0	1249.66	541.448	794.87	2.25	116.51	2.25	20.55	3.45
rc204.1	244.56	485.367	23.77	0.00	241.18	3.13	421.31	8.03
rc204.2	**20000**	778.395	X	X	X	X	X	X
rc205.2	65.15	714.695	50.97	0.00	22.49	0.00	10.82	0.00
rc206.0	107.22	835.232	107.39	2.65	18653.25	8.57	18653.25	10.23
rc206.1	15.27	664.731	63.59	0.00	50.16	0.00	24.17	0.00
rc207.0	85.85	806.689	37.76	0.00	122.19	0.00	61.00	0.00
rc207.1	234.68	726.359	25.78	1.14	19.72	1.63	13.17	1.73
rc207.2	10.79	546.405	70.25	0.00	89.53	1.90	120.11	2.96
rc208.0	**20000**	inf	X	X	X	X	X	X
rc208.1	229.71	509.041	8706.63	15.56	8706.63	18.91	8706.64	28.11
rc208.2	54.17	503.923	36920.86	12.16	36920.86	12.72	X	X
		geometric mean	166.40	2.49	232.00	3.90	109.50	4.79
		arithmetic mean	3620.00	2.60	5012.00	4.05	2353.00	5.04
		median	70.25	0.00	100.00	1.90	80.50	2.46

Table 4.9 Times and gaps to optimality for adding ordering constraints in the benchmark SolomonPesant

arithmetic mean to be very high once again, since we take an average on the relative times and not weighted by the solving times themselves. We also see that most of the instances that time out at the start are still not solved within the time limits by adding ordering constraints. We still observe small gaps on average, with less than 2% on average in all configurations.

AFG

In Table 4.11, we report the same results for the set of instances AFG. Here, the speedups are much more impressive, with one instance being solved 5000 times faster and a lot of them more than 100 times. On average, we reach a geometric average of 2.7% with 150% of ordering constraints, still having a gap under 1% to the base best solution and no instance losing feasibility compared to the start. While not all benchmark give results that positive, we can at least see that for these instances, we have a drastic improvement of the solving times.

On those two last benchmarks, we have a lot less loss of feasibility and smaller gaps than the first method of sparsification.

Instance	Base		50%		100%		150%	
	Time (s)	Solution	Time (%)	Gap (%)	Time (%)	Gap (%)	Time (%)	Gap (%)
rc_202.1	**20000**	771.776	100.00	0.00	89.79	0.07	36.25	0.64
rc_202.4	97.77	793.03	183.95	2.28	3856.07	5.62	X	X
rc_203.2	**20000**	784.158	100.00	0.00	100.00	0.00	100.00	0.88
rc_203.3	**20000.02**	817.526	100.00	4.77	X	X	X	X
rc_204.1	**20000**	878.64	100.00	0.00	100.00	0.75	100.00	1.55
rc_204.2	**20000**	662.159	100.00	-0.00	100.00	-0.00	100.00	0.17
rc_204.3	172.06	455.031	19.10	0.00	11623.86	5.47	X	X
rc_205.3	17.6	825.058	105.74	0.00	70.74	0.00	65.40	1.03
rc_205.4	29.7	760.47	29.39	0.00	7.95	0.00	3.64	0.26
rc_206.2	7104.06	828.059	5.47	0.00	1.70	0.14	2.01	0.35
rc_206.3	449.35	574.418	9.55	1.62	7.74	2.15	7.11	2.15
rc_206.4	**20000**	831.67	100.00	0.00	67.51	0.25	34.96	0.25
rc_207.1	12515.56	732.683	6.90	0.00	4.22	0.00	3.06	0.00
rc_207.2	**20000**	711.487	100.00	-1.44	100.00	-1.29	100.00	-1.29
rc_207.3	**20000**	682.404	100.00	-0.00	100.00	-0.00	100.00	1.85
rc_208.1	**20000**	inf	X	X	X	X	X	X
rc_208.2	81.88	533.78	24426.04	19.72	X	X	X	X
rc_208.3	**20000**	640.399	100.00	0.53	100.00	9.64	X	X
	geometric mean		76.09	1.52	72.76	1.48	26.70	0.65
	arithmetic mean		1511.00	1.62	1089.00	1.52	54.40	0.65
	median		100.00	0.00	100.00	0.07	50.83	0.50

Table 4.10 Times and gaps to optimality for adding ordering constraints in the benchmark SolomonPotvinBengio

Dumas

Like in the previous Section 4.3.4, we only report the average results in this chapter, the full results are available in Table C.1, as well as comparison with the exact SD and the variant based on partial overlaps (see Section 4.3.7 and 4.3.8). Table 4.12 reports only the averages and median for this benchmark. Compared to the previous sparsification method, we see a real speedup and slightly higher gaps.

Global analysis

Once again, the results are highly dependent on the set of instances, We can report positive results from the AFG benchmark, that have the advantage over the previous method of Section 4.3.4 of reducing the average gaps to optimality. On all these benchmarks, a few instances drastically impact the means of the solving times: we observe a big difference between the geometric mean or the median and the arithmetic mean. However, for two benchmarks out of three, the medians of the solving times stay close to 100%, which means we do not improve the solving of a majority of instances.

Instance	Base		50%		100%		150%	
	Time (s)	Solution	Time (%)	Gap (%)	Time (%)	Gap (%)	Time (%)	Gap (%)
rbg019c	321.22	4536.0	0.02	0.00	0.09	0.26	0.09	0.26
rbg020a	69.18	4689.0	0.23	0.17	0.19	0.17	0.07	0.17
rbg021.3	127.28	4528.0	0.29	0.00	0.27	0.00	0.24	0.11
rbg021.4	108.5	4525.0	0.08	0.00	0.26	0.09	0.05	0.11
rbg021.5	25.84	4515.0	0.31	0.00	0.62	0.20	0.46	0.20
rbg021	330.12	4536.0	0.01	0.00	0.08	0.26	0.09	0.26
rbg027a	**2000**	inf	0.27	-100.00	8.64	-100.00	X	X
rbg041a	95.7	2598.0	41.42	0.12	4.73	0.69	0.93	2.00
rbg042a	31.15	2772.0	67.16	0.00	74.99	0.36	159.52	0.90
rbg048a	**2000**	inf	X	X	X	X	X	X
rbg086a	65.94	8400.0	3.09	0.05	2.53	0.43	2.84	1.49
rbg092a	51.11	7158.0	430.29	0.03	31.19	0.07	19.06	0.32
rbg125a	20.84	7936.0	16.55	0.58	43.86	1.30	33.97	1.89
rbg132.2	729.94	8191.0	274.00	0.57	108.84	0.89	62.53	1.89
rbg132	31.36	8468.0	27.14	0.70	29.59	1.56	18.40	2.22
rbg152.3	**2000**	9789.0	X	X	100.00	0.19	100.00	0.10
rbg152	30.57	10032.0	65.72	0.39	56.43	0.71	30.06	1.08
	geometric mean		2.68	0.19	4.25	0.48	2.70	0.86
	arithmetic mean		61.77	0.19	28.89	0.48	28.44	0.87
	median		3.09	0.03	6.69	0.26	2.84	0.32

Table 4.11 Times and gap to optimality for adding ordering constraints in the benchmark AFG

	50%		100%		150%	
	Time(%)	Gap(%)	Time(%)	Gap(%)	Time(%)	Gap(%)
geometric mean	73.39	0.30	46.53	0.84	41.05	1.93
arithmetic mean	101.25	0.30	59.07	0.85	62.77	1.95
median	71.45	0.00	52.2	0.73	49.71	1.22

Table 4.12 Average times and gaps for adding ordering constraints in the Dumas benchmark

4.3.6 Fixing edges

This section reports the results from the method described in Section 4.2.4. We study two different configurations, named $5-10$ and $15-30$: the first number is the maximum number of edges that can be added, and the second the maximum number of edges that are considered as candidate. For a configuration $a-b$, we add at most a edges among the top b ranked with their solution densities, if and only if the corresponding ordering has a SD of at least 0.5. In practice, a very low number of edges are added in general.

In a following section we will compare with adding randomly selected edges with the same process to see if the solution densities add useful information.

Number of fixed edges

Tables 4.13 and 4.14 report the number of edges that are fixed in the real instances: when we use the configuration 5-10, we suggest 5 candidates for selection but we may add only 1, for instance. We look at the distribution of instances where each number of edges is fixed in reality.
We see that the number of edges that are in fact fixed is in general much lower than the maximum we allowed (5 or 15). The number of edges depends on the instance, we see mainly that the distribution is very different for Dumas instances and for the AFG instances, where almost no edge is added into the model.

Benchmark	0	1	2	3	4+
Dumas	21	49	51	14	0
AFG	24	4	1	0	0
SolomonPesant	4	6	4	1	0
SolomonPotvinBengio	3	4	9	2	0

Table 4.13 Number of instances with n added edges for each benchmark in the configuration $5-10$

Benchmark	0-2	3	4	5	6-8	9+
Dumas	24	31	31	35	13	0
AFG	18	9	1	1	0	0
SolomonPesant	6	3	5	1	0	0
SolomonPotvinBengio	5	2	5	5	1	0

Table 4.14 Number of instances with n added edges for each benchmark in the configuration $15-30$

SolomonPesant

We report in Table 4.15 the number of edges added, the time and gap ratios for the configurations $5-10$ and $15-30$. We see that on average, a low number of edges are actually selected: for our two configurations $a-b$, it is almost never the case that the parameter a influences the number of selected edges. We can increase the number of edges selected by changing the value of b and accepting to select edges that have a higher rank for our solution densities, or to reduce the minimum value of the density for $(i < j)$, accepting edges that are less likely to respect the partial order on time windows. While this particular set of instances proved

Instance	Base		5-10			15-30		
	Time (s)	Solution	Time (%)	Gap (%)	Edges	Time (%)	Gap (%)	Edges
rc203.0	**20000**	727.453	100.00	1.05	1	100.00	2.19	4
rc203.1	**20000**	726.991	100.00	0.78	2	100.00	1.31	3
rc203.2	103.39	617.465	42.13	0.00	1	7.09	5.28	4
rc204.0	1249.66	541.448	198.89	0.00	1	129.14	0.66	2
rc204.1	244.56	485.367	149.54	0.75	1	149.27	0.75	1
rc204.2	**20000**	778.395	100.00	0.00	0	100.00	inf	1
rc205.2	65.15	714.695	124.33	0.00	1	12.92	1.20	4
rc206.0	107.22	835.232	20.86	0.03	2	9.74	0.33	4
rc206.1	15.27	664.731	5.63	0.00	2	54.29	1.40	3
rc207.0	85.85	806.689	43.98	0.00	1	38.32	1.61	4
rc207.1	234.68	726.359	10.65	1.37	3	6.08	2.51	5
rc207.2	10.79	546.405	67.75	0.83	2	7.04	3.35	3
rc208.0	**20000**	inf	X	X	0	X	X	0
rc208.1	229.71	509.041	100.00	0.00	0	100.00	0.00	0
rc208.2	54.17	503.923	100.00	0.00	0	100.00	0.00	0
		geometric mean	58.73	0.34		38.43	1.57	
		arithmetic mean	83.13	0.34		65.28	1.58	
		median	100.00	0.00		100.00	1.31	

Table 4.15 Times and gaps to optimality for pre-selecting edges in the benchmark Solomon-Pesant

resistant to our two previous attempts of reducing the resolution time, we see here a clear improvement of the solving times with average gaps under 2% and a maximum of 5.28% for a single instance. We see that the instances that were impacted the most by adding specific ordering constraints in advance correspond here to instances where none of the candidate edges is selected into the model, meaning that the 30 best edges for the resolution of the TSP without time windows do not coincide with good orderings for our SD.

SolomonPotvinBengio

We report in Table 4.16 the results obtained with the benchmark SolomonPotvinBengio when we fix some edges before solving. We see that in the configuration $5-10$ two instances lose feasibility, and 5 in the configuration $15-30$. In the first configuration, we see a positive speedup, while fixing more edges after that reduces the number of solutions without speeding up the resolution further. Compared with the previous benchmark, the second configuration is less interesting. We see from Tables 4.13 and 4.14 that more edges are selected in this benchmark on average, which means the top ranking edges are more frequently coherent with our partial orders. The instances that lose feasibility are among those where the highest number of edges are selected.

Once again, the instances where no edge is added correspond to instances that lost feasibility or had high gaps and high solving times in the previous method (`rc_204.3`, `rc_208.2`,

rc_208.3).

Instance	Base		5-10			15-30		
	Time (s)	Solution	Time (%)	Gap (%)	Edges	Time (%)	Gap (%)	Edges
rc_202.1	**20000**	771.776	10.42	0.11	1	100.52	1.01	4
rc_202.4	97.77	793.030	13.25	0.00	2	6.53	0.78	5
rc_203.2	**20000**	784.158	$\leq$ 98.12	0.17	2	100.00	3.88	4
rc_203.3	**20000**	817.526	100.00	0.00	1	100.00	4.21	4
rc_204.1	**20000**	878.640	100.00	1.41	2	100.00	1.41	2
rc_204.2	**20000**	662.159	100.00	3.08	1	100.00	3.08	2
rc_204.3	172.06	455.031	100.00	0.00	0	100.00	0.00	0
rc_205.3	17.60	825.058	66.14	0.03	1	X	X	4
rc_205.4	29.70	760.470	X	X	3	X	X	5
rc_206.2	7104.06	828.059	X	X	2	X	X	5
rc_206.3	449.35	574.418	16.12	0.64	2	X	X	6
rc_206.4	**20000**	831.670	68.5	0.24	2	41.83	0.44	3
rc_207.1	12515.56	732.683	2.62	0.00	2	1.74	0.00	4
rc_207.2	**20000**	711.487	100.00	-1.13	3	X	X	5
rc_207.3	**20000**	682.404	100.00	0.97	2	100.00	1.90	5
rc_208.1	**20000**	inf	X	X	2	X	X	3
rc_208.2	81.88	533.780	100.00	0.00	0	100.00	0.00	0
rc_208.3	**20000**	640.399	100.00	0.00	0	100	0.00	0
		geometric mean	49.46	0.36		52.88	1.38	
		arithmetic mean	71.68	0.37		79.22	1.39	
		median	100.00	0.03		100.00	0.90	

Table 4.16 Times and gaps to optimality for pre-selecting edges in the benchmark Solomon-PotvinBengio

AFG

We see from Table 4.13 that a majority of the instances are not changed with the $5-10$ configuration. It means that the information from solving the TSP is not very compatible with the partial orderings. Even when taking more edges into account, we see from Table 4.14 that there is generally three edges or fewer added among the top 30. We will not report in detail the results of the instances with the $5-10$ configuration as only five instances select at least one edge. Since reporting the same tables as for the previous benchmarks with most instances not adding a single edge, we will focus on the instances where at least one edge is added and report only the results of configuration $15-30$ in Figure 4.3. The other instances will always have solving times close to 100.00 and gaps of 0.00 since nothing is changed in the instance. We see that even zooming in, the resolution of a majority of instances is longer than it was in the base case. We note that we report only the results for about 20 instances. We will report the same histograms for the Dumas instances in the following section.

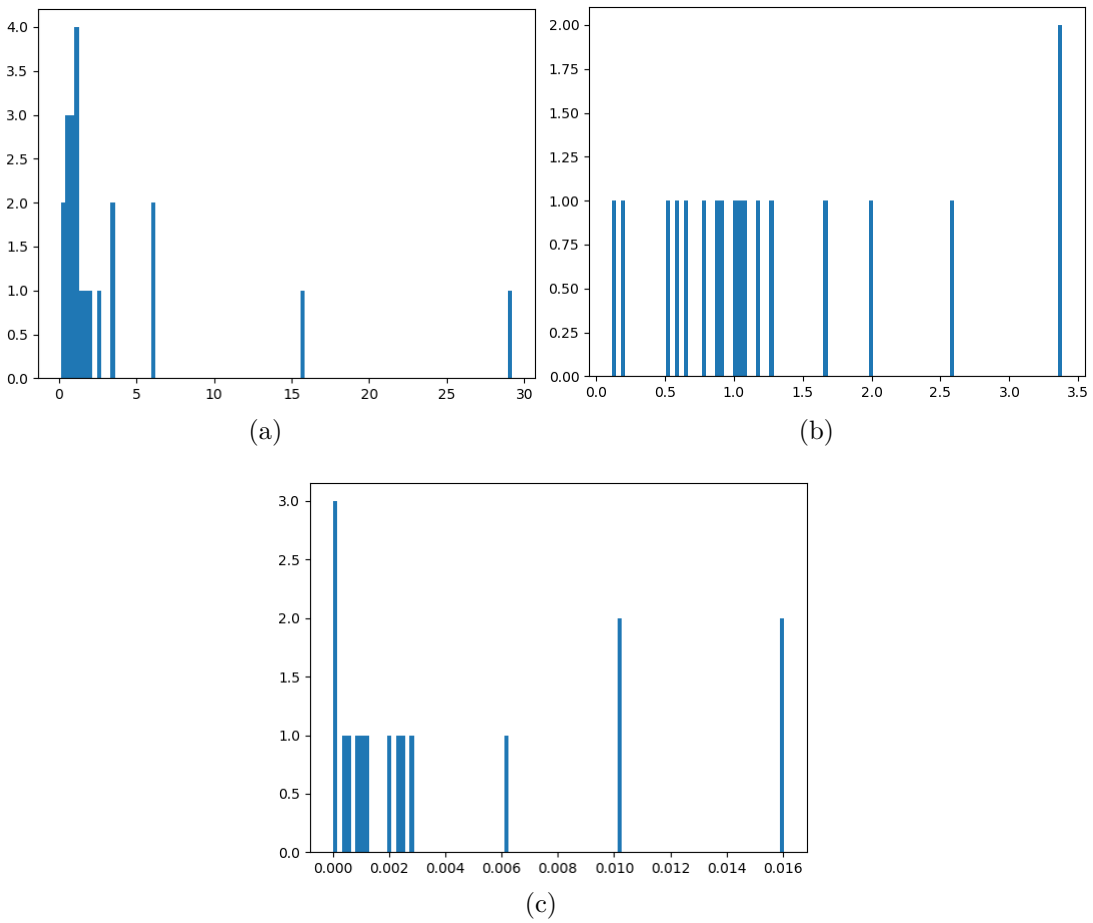


Figure 4.3 Histograms of the times and gaps of instances where at least one edge is added. We see a zoom of Figure 4.3a in Figure 4.3b and the last Figure 4.3c represents the gaps to the base instance

Dumas

In this section, we report the results of the configuration $5 - 10$ in Figure 4.4 and of the configuration $15 - 30$ in Figure 4.5. For both configurations, we zoom in on the instances where the ratio of time is under 5, which allows to see more detail. Indeed, we see in Figure 4.4a that most results are compressed in one column since one instance gives a time ratio close to 3000. The third figure, namely Figures 4.4c and 4.5c report the gaps to the base instance of the versions with selected edges. We observe that for the first configuration, zoomed in in Figure 4.4b, a majority of bars are after 1: the histograms show the ratio between the sparsified instance and the simply filtered one. This means that in a majority of cases, the time needed for solving is longer than before. On the contrary, a majority of the results are below 1 in the second configuration. We see there that selecting edges induces a

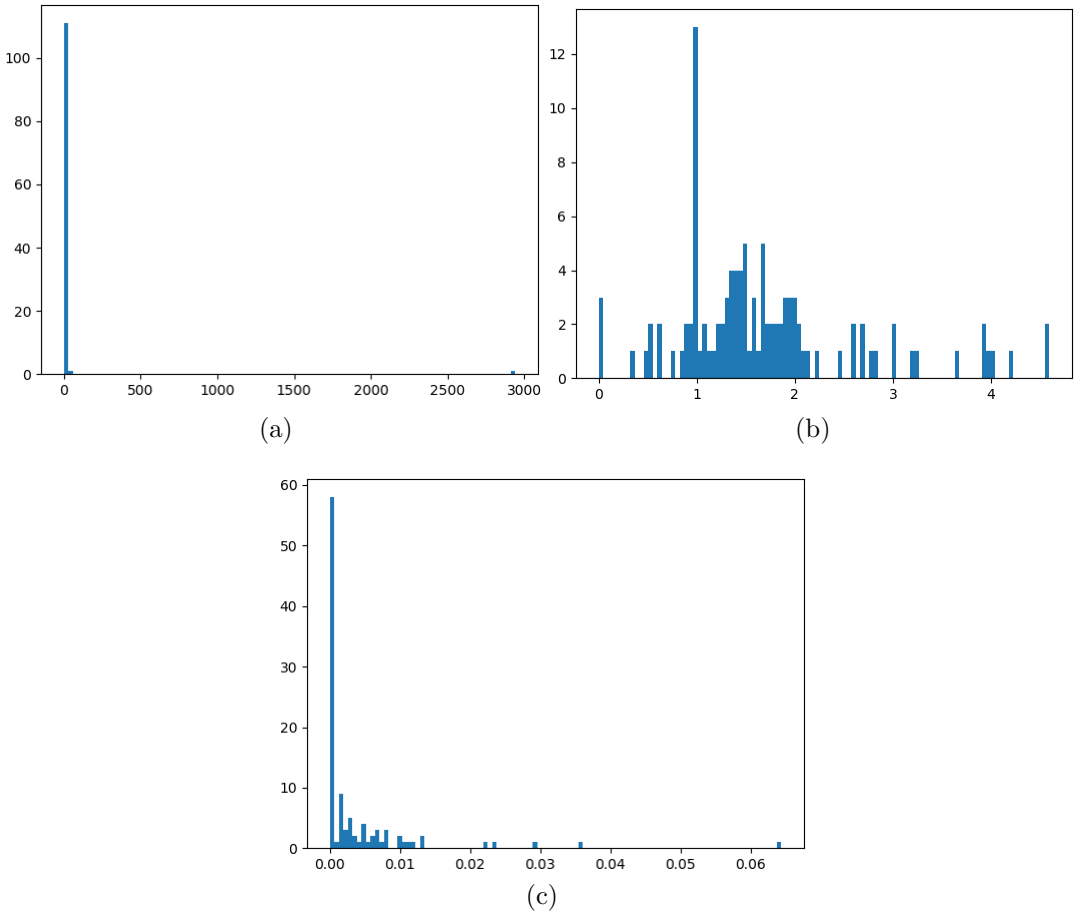


Figure 4.4 Histograms of the times and gaps of instances where at least one edge is added in the $5-10$ configuration. We see a zoom of Figure 4.4a in Figure 4.4b and the last Figure 4.4c represents the gaps to the base instance

speedup while the gaps are still all lower than 10%.

For this benchmark, since there are a lot of instances, we do not include the full results for each instance, that can be found in Appendix.

Comparison with randomly fixed edges

After having shown in Figures 4.4 and 4.5 the distribution of the times and gaps when we use our method for pre-selecting edges, we will show in Figures 4.6 and 4.7 the same results when we use the same process but we select the candidates for selection at random. This means that instead of ranking the best edges for our SDs and then checking if they respect our SD for the order of visiting times, we check if random edges respect these orders before adding them. The first difference we observe with our previous experiments is that we have

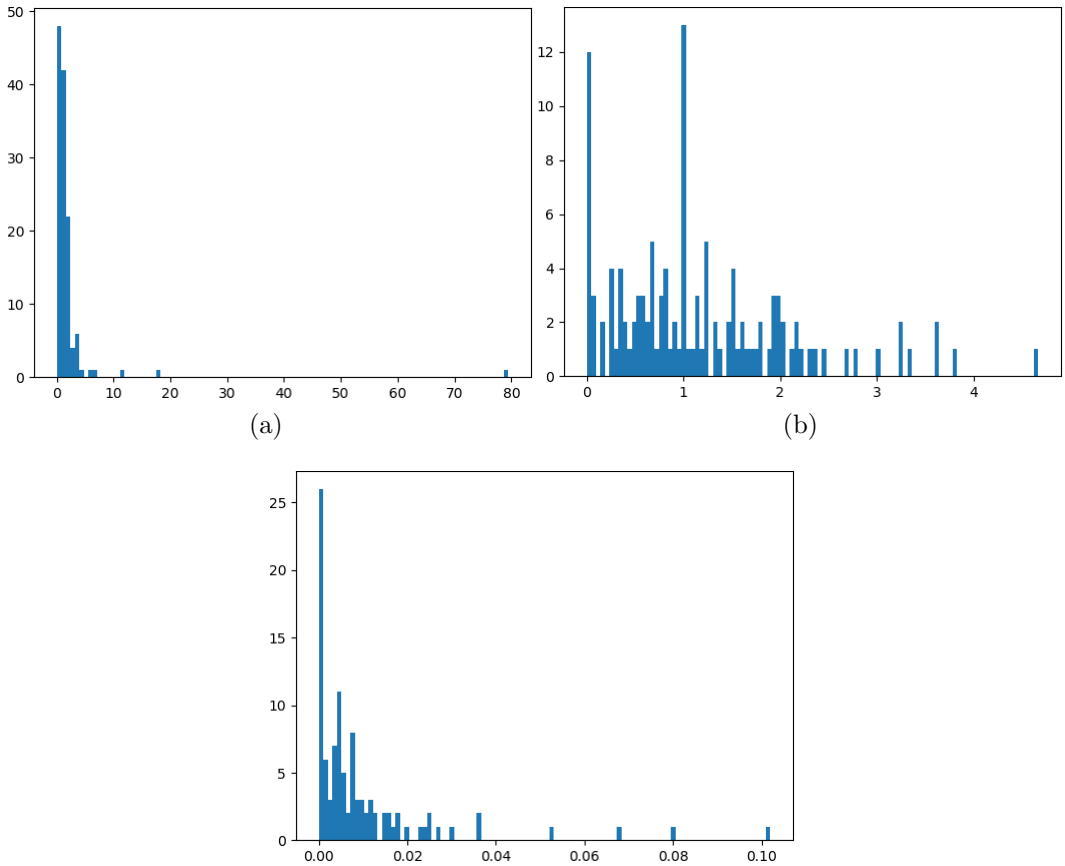


Figure 4.5 Histograms of the times and gaps of instances where at least one edge is added in the $15-30$ configuration. We see a zoom of Figure 4.5a in Figure 4.5b and the last Figure 4.5c represents the gaps to the base instance

fewer points in our second sets, and even half as much in the first sets corresponding to the $5-10$ configuration. Since we only plot instances with at least one selected edge, the reason for that is that there are fewer edges that go positively through the controlling phase. We then see that on average, more instances than before have a positive speedup, i.e., there are more points below 1 in the first two graphs. The speedup is generally linked to the number of added edges: in order to have a more accurate comparison, we would need to compare separately the instances where the same number of edges are added, or even restrain the comparison to instances where the same number of edges is selected in both methods. Of course, having precise data while separating our instance into a lot of categories would need a higher number of instances. Since we only changed the pre-selection method for the edges and not the way we actually add them, changing the pre-selection should not really affect the time needed for the solving. However, the edges selected should be less "good" edges for

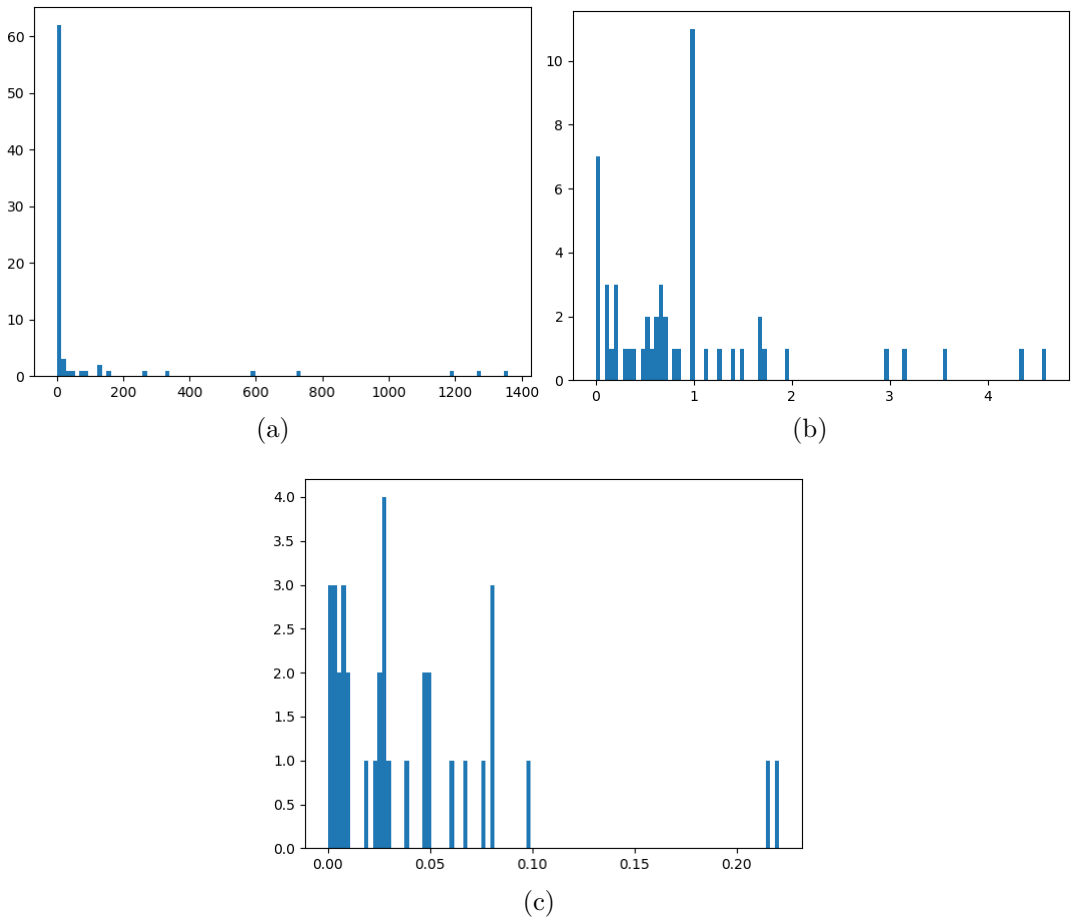


Figure 4.6 Comparison of the results of fixing edges based on SD with pre-selecting $5-10$ random edges

the objective function, and we observe that the gaps obtained are indeed higher than when we choose our candidates more carefully.

Global analysis

In this section, we saw that the two benchmarks that performed badly in our previous experiments report improvements of their solving times and low gaps compared to the previous attempts. Moreover, we see that selecting edges based on our SD and not only on ordering constraints performs better. We confirm the interest of the heuristic SD for Time Windows in regard to the exact algorithm presented in [53]. The main drawback of this method is that for a number of instances, there is no interesting edge that can be selected: the majority of the AFG benchmark is left unchanged. However, different parameters could help find interesting edges, and this affects a benchmark whose resolution is greatly improved by our

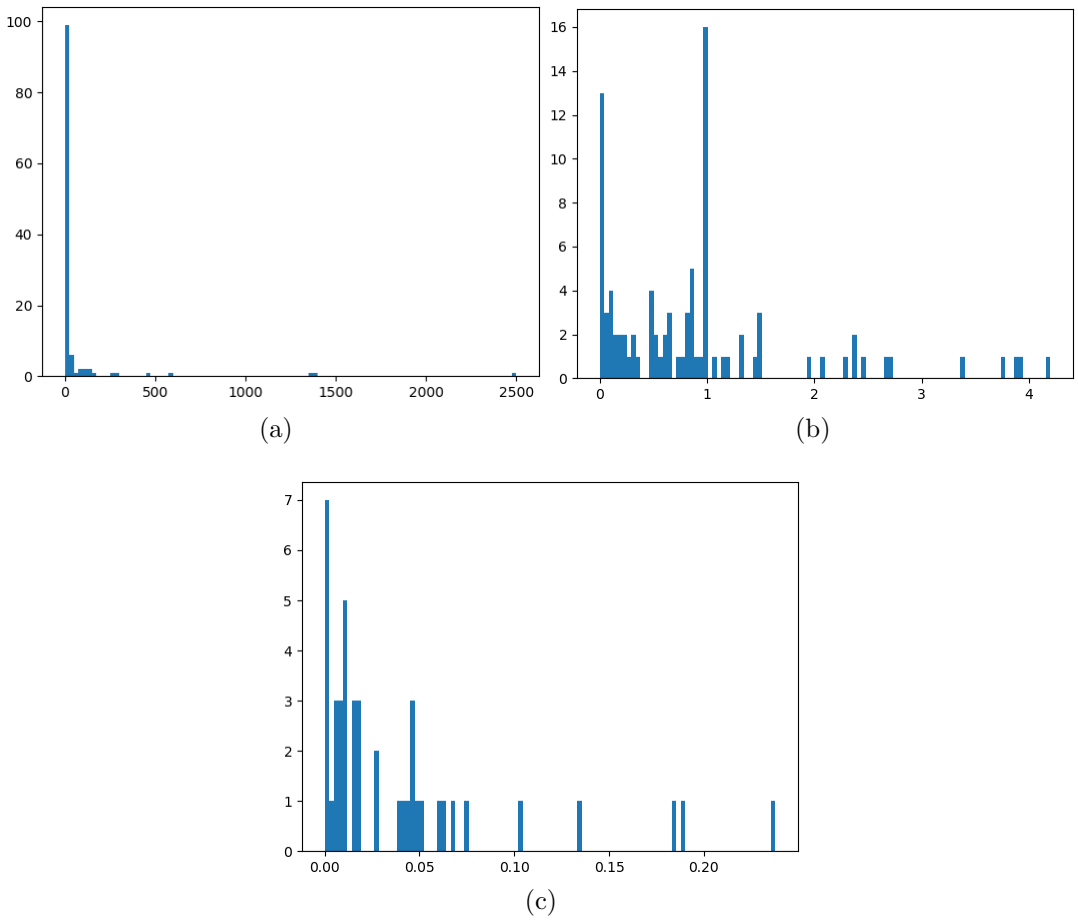


Figure 4.7 Comparison of the results of fixing edges based on SD with pre-selecting $15 - 30$ random edges

other methods: we see that a combination of the reduction methods based on the structure of the instance could be helpful.

4.3.7 Effectiveness of the TW-SD heuristic compared to the exact algorithm

Table 4.17 is to be compared with Table 4.11 seen before in this chapter. For instances of sizes bigger than 20, the time needed to compute the densities alone can take hours (timeout after 1h). We report results for sizes up to 20, which limits the quantity of data. In any case, the time needed for computation makes the method impractical, but we want to compare on a few instances to compare the quality of the density obtained.
Since only a few instances of the benchmark are small enough to compute the exact densities, we add the means of Table 4.11 for this subset of instances to help the comparison. The results for all the small instances of the Dumas benchmark can be found in Appendix: in the

Instance	Base		50%		100%		150%	
	Time (s)	Solution	Time (%)	Gap (%)	Time (%)	Gap (%)	Time (%)	Gap (%)
rbg019c	321.22	4536.0	25.45	0.13	31.27	0.26	194.58	0.93
rbg020a	69.18	4689.0	180.72	0.17	32.05	0.17	19.72	0.17
rbg021.3	127.28	4528.0	44.44	0.00	19.75	0.13	2.58	0.44
rbg021.4	108.5	4525.0	62.20	0.00	6.88	0.20	0.62	0.29
rbg021.5	25.84	4515.0	171.94	0.29	3.25	0.29	1.70	0.38
rbg021	330.12	4536.0	23.44	0.13	29.27	0.26	181.23	0.93
rbg027a	**2000**	inf	X	X	X	X	X	X
		geometric mean	60.94	0.12	15.33	0.22	11.12	0.52
		arithmetic mean	84.70	0.12	20.41	0.22	66.74	0.52
		median	53.32	0.13	24.51	0.23	11.15	0.41
geom.mean for approximation			0.10	0.03	0.34	0.16	0.12	0.18
arit.mean for approximation			0.17	0.03	1.45	0.16	0.17	0.18
median for approximation			0.23	0.00	0.26	0.19	0.09	0.19

Table 4.17 Results of "ordering constraints" with exact TW solution density on the benchmark AFG

3 comparable instances `n40w80.002`, `n40w100.002` and `n40w100.005`, the exact SD performs better than the approximation. However, it seems from the few AFG instances that the exact densities do not speed up or improve the gaps to optimality, which is surprising. Even though we only look at a few instances, we would expect for the exact algorithm to perform better than the approximation. We see that the small AFG instances are those that give the best results among all benchmarks for adding ordering constraints, so the results for the exact algorithm looks more like the average of the other instances: one explanation could be that we only compare on instances where the approximation performs really well. It could also be because of numerical instability linked to the size of the numbers involved in the computation: the exact algorithm relies on counting linear extensions. Even in the smallest instance `rbg019c`, the number of linear extensions for the partial order is close to 10^{13}. It could also be simply a random deviation since we look at a small number of instances.

4.3.8 Using partial overlaps as a solution density

This section compares the variant of the TW SD introduced in Section 4.1.1 based on the intersection of time windows rather than only non-overlapping time windows. Once again, Table 4.18 is to be compared with Table 4.11. We use the sum of overlaps instead of the number of precedences in computing the densities for the orderings: we see that this proposition of extension of the solution density gives on these instances smaller gaps with comparable speedups (higher medians but means of the same magnitude). Experiments on a higher number of examples could give a more precise idea of the potential of the heuristic, but it seems from this first experiment that the variant is promising. We report in Appendix in Table E.1

Instance	Base		50%		100%		150%	
	Time (s)	Solution	Time (%)	Gap (%)	Time (%)	Gap (%)	Time (%)	Gap (%)
rbg019c	321.22	4536.0	0.10	0.00	0.41	0.13	0.03	0.24
rbg020a	69.18	4689.0	0.13	0.00	0.38	0.15	0.32	0.19
rbg021.3	127.28	4528.0	0.13	0.00	0.09	0.00	0.16	0.49
rbg021.4	108.5	4525.0	0.15	0.00	0.17	0.22	0.09	0.22
rbg021.5	25.84	4515.0	0.46	0.20	0.19	0.31	0.19	0.40
rbg021	330.12	4536.0	0.08	0.00	0.40	0.13	0.03	0.24
rbg027a	**2000**	inf	0.27	-100.00	0.34	-100.00	100.00	-100.00
rbg041a	95.7	2598.0	108.07	0.00	55.25	0.00	27.27	0.42
rbg042a	31.15	2772.0	68.60	0.00	67.64	0.14	28.64	0.14
rbg048a	**2000**	inf	X	X	X	X	X	X
rbg086a	65.94	8400.0	13.18	0.00	14.18	0.12	13.91	0.58
rbg092a	51.11	7158.0	132.69	0.00	41.73	0.04	21.89	0.21
rbg125a	20.84	7936.0	54.08	0.01	56.05	0.01	11.32	0.28
rbg132.2	729.94	8191.0	62.37	0.05	83.17	0.23	39.70	0.42
rbg132	31.36	8468.0	74.27	0.00	58.13	0.15	49.84	0.84
rbg152.3	**2000**	9789.0	100.00	-0.01	100.00	-0.01	100.00	0.06
rbg152	30.57	10032.0	78.28	0.00	82.04	0.05	60.35	0.18
	geometric mean		4.74	0.02	5.21	0.11	3.87	0.33
	arithmetic mean		43.30	0.02	35.01	0.11	28.36	0.33
	median		33.63	0.00	27.96	0.13	17.90	0.24

Table 4.18 Results of "ordering constraints" with partial overlaps on the benchmark AFG

the same results for the Dumas benchmark where we also see that, even if the average time needed for computation is longer than with the basic solution density, more instances are feasible with 150% of ordering constraints, and the average gaps are also smaller.

4.4 Conclusion

In this chapter, we presented the computation of solution densities with variants and four reduction methods with the corresponding experimental results. We see that the exact reduction presented, though simple, outperforms the precomputation done by the solver itself and improves the solving times of all benchmarks at different levels. For the three heuristics presented, we see that the results vary from one benchmark to the other. The sparsification method tends to make instances infeasible, even by adding a SD based on the time windows. Adding some ordering constraints based on the extension of partial orders induces very long solving times for a few instances, which ruins the average, but speeds up most instances and performs really well on the AFG benchmark. Finally, the main drawback of the method based on pre-selecting some edges is that in most cases we cannot find an edge that is good for the TSP and the ordering in some benchmarks, which gives little possibilities of improvement. However, this third method gives good results on average in all our benchmarks.
Finally, this chapter compared the results obtained by variants of our SDs: we suggested a

variant for the extension of the partial orderings based on the intersection of the time windows, and compared the exact and approximated algorithms on a few instances.
The variability of results between the benchmarks for each method means the performance relies a lot on the structure of the instance. A further development of this project could be to create an algorithm analyzing the instances and selecting a good reduction method before solving.

CHAPTER 5 CONCLUSION

5.1 Summary of Works

This thesis presented a compilation of several uses of solution densities for the ATSP and TSPTW: we described them, presented computation methods and evaluated them empirically.
We presented adaptations of solution densities for the TSP into an asymmetric version as well as approximations. We also introduced a solution density for the TSPTW based on non-overlapping Time Windows, its approximation and suggested a possible variant.
We also developed several ways of using the information given by these solution densities, mainly reduction methods, that use the SDs to reduce the number of variables or add additional constraints that restrict the search space. We investigated a search heuristic based on those SDs and compared it to search heuristics in two different models. We also considered several MIP models, both for the ATSP and the TSPTW and looked at how our reduction methods can reduce the computing times in two of these models.
In addition to the solving times, we reported the losses of optimality or feasibility induced by our heuristic methods. We effectively reduced the solving time for the ATSP by MIP models, and have positive results on several methods for solving the TSPTW with a basic MIP model, depending on the benchmarks.

5.2 Limitations

Due to the number of different models and techniques tried, a lot of improvements that demand more work were dismissed to be able to focus on more promising leads and cover all of the research directions. For instance the experiments about using a SD-based search heuristic in the CP solver Talos were inconclusive because of too high computational times although several leads are mentioned to increase the speed. All of them demand coding reversible matrix structures and inversion algorithms that are time-consuming without being central to this Master's project. A model adapted to include asymmetric instances and time windows could be of high interest for our experiments, and is currently the focus of the team who published [38], but does not yet exist. In general, we did not have enough time to try all of the methods and all of their parameters on all solvers.
This leads to our methods not helping the state-of-the-art methods in each problem, and having positive results on non-optimal methods only. Furthermore the methods presented here

are heuristic, which means that there is no proof of optimality for any choice of parameters. While we present some choices of parameters for our different methods and show average values indicating what to expect with each configuration, there is a significant variance on the instances in terms of time as well as optimality gap.

5.3 Future Research

The main (quick) improvement that can be done to the project would be continuing to implement variants of the solution densities, parameters and methods to go to the end of our exploration: for instance accelerating the computation of the 1-tree density, using reduced costs to also guide search if possible, combining several reductions together to get an additional speedup, changing the parameter of edge-fixing ...

Since we get different results depending on the benchmark, a more profound analysis of what methods work well with the different characteristics of the instances could give way to a tool selecting the best reduction method before solving.

On a larger scale, the objective would be to adapt the method to other routing problems with different relaxations, such as capacitated problems or with multiple vehicles. On an even larger scale, this project is part of the exploration of counting-based search and preprocessing, that can be adapted to other difficult problems having efficient relaxations.

REFERENCES

[1] H. de Balzac, *L'Illustre Gaudissart*, 1833.

[2] A. Miller, *Death of a Salesman*, 1949.

[3] A. Bartolomei, C. Lemercier, and S. Marzagalli, "Les commis voyageurs, acteurs et témoins de la grande transformation," *Entreprises et histoire*, vol. 66, pp. 7–21, 2012. [Online]. Available: https://www.cairn.info/revue-entreprises-et-histoire-2012-1-page-7.htm

[4] S. A. Marin, "Le commis voyageur allemand : une image mythifiée dans la France de 1900," *Entreprises et histoire*, vol. 66, pp. 177–193, 2012. [Online]. Available: https://www.cairn.info/revue-entreprises-et-histoire-2012-1-page-177.htm

[5] D. Andreozzi, "Vers la professionnalisation. Les agents de commerce et les commis voyageurs à Trieste au XIXe siècle," *Entreprises et histoire*, vol. 66, pp. 146–163, 2012. [Online]. Available: https://www.cairn.info/revue-entreprises-et-histoire-2012-1-page-146.htm

[6] W. J. Cook, D. L. Applegate, R. E. Bixby, and V. Chvátal, *The Traveling Salesman Problem: A computational study.*, 2006.

[7] G. Pesant, "Counting-based search for constraint optimization problems," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, p. 3441–3447.

[8] P. Coste, "Accelerating tsp solving by using cost-based solution densities of relaxations," Master's thesis, École Polytechnique de Montréal, 2019.

[9] J. K. Lenstra and A. H. G. R. Kan, "Some simple applications of the travelling salesman problem," *Journal of the Operational Research Society*, vol. 26, no. 4, pp. 717–733, 1975.

[10] D. Gusfield, R. Karp, L. Wang, and P. Stelling, "Graph traversals, genes and matroids: An efficient case of the travelling salesman problem," *Discrete Applied Mathematics*, vol. Volume 88, Issues 1–3, pp. 167–180, 1998.

[11] D. A. Spielman and S.-H. Teng, "Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time," *Journal of the ACM*, May 2004.

[12] L. Khachiyan, "A polynomial algorithm for linear programming," *Doklady Akademii Nauk SSSR*, vol. 224 (5), p. 1093–1096., 1979.

[13] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Proceedings of the sixteenth annual ACM symposium on Theory of computing - STOC '84*, p. 302, 1984.

[14] E. Tsang, *Foundations of Constraint Satisfaction*, 1996.

[15] R. Dechter, "Constraint optimization," *Constraint Processing*, pp. 363–397, 2003.

[16] ——, "Consistency-enforcing and constraint propagation," *Constraint Processing*, pp. 51–83, 2003.

[17] R. Karp, "Reducibility among combinatorial problems." *Miller R.E., Thatcher J.W., Bohlinger J.D. (eds) Complexity of Computer Computations. The IBM Research Symposia Series. Springer, Boston, MA.*, 1972.

[18] W. I. Gasarch, "Guest column: The third p =? np poll," 2019. [Online]. Available: https://www.cs.umd.edu/users/gasarch/BLOGPAPERS/pollpaper3.pdf

[19] P. Toth and D. Vigo, *Vehicle Routing*, D. Vigo and P. Toth, Eds. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2014. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611973594

[20] G.B.Danzig and D. Fulkerson, "Minimizing the number of tankers to meet a fixed schedule." *Naval Research Logistics Quarterly*, vol. 1(3), pp. 217—222, 1954.

[21] R. Jonker and T. Volgenant, "Transforming asymmetric into symmetric traveling salesman problems." *Oper Res Lett*, vol. 2(4), pp. 161—163, 1983.

[22] M. Bellmore and G. L. Nemhauser, "The traveling salesman problem: A survey," *Mathematical Models in Marketing. Lecture Notes in Economics and Mathematical Systems (Operations Research)*, vol. 132, 1976.

[23] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the Operations Research Society of America*, vol. vol. 2, no. 4, pp. 393—410, 1954.

[24] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer programming formulation of traveling salesman problems," *Journal of the ACM*, vol. vol. 7, no. 4, pp. 326—329, 1960. [Online]. Available: https://dl.acm.org/citation.cfm?id=321046

[25] M. Velednitsky, "Short combinatorial proof that the dfj polytope is contained in the mtz polytope for the asymmetric traveling salesman problem," *Operations Research Letters*, vol. 45, pp. 323–324, 2017.

[26] Sanjeeb Dash, Oktay Günlük, Andrea Lodi, and Andrea Tramontani, "A time bucket formulation for the traveling salesman problem with time windows," *INFORMS Journal on Computing*, vol. 24(1), pp. 132–147, 2012.

[27] N. Ascheuer, M. Fischetti, and M. Grötschel, "Solving the asymmetric travelling salesman problem with time windows by branch-and-cut." *Mathematical Programming*, vol. 90(3), p. 475–506, 2001.

[28] J. Albiach, J. M. Sanchis, and D. Soler, "An asymmetric tsp with time windows and with time-dependent travel times and costs: An exact solution through a graph transformation," *European Journal of Operational Research*, vol. 189(3), pp. 789–802, 2008.

[29] C. van Eijl, "A polyhedral approach to the delivery man problem," *Memorandum COSOR*, vol. vol. 9519, 1995.

[30] F. Maffioli and A. Sciomachen, "A mixed-integer model for solving ordering problems with side constraints," *Annals of Operations Research*, vol. 69, p. 277 – 297, 1997.

[31] "Concorde tsp solver," "http://www.math.uwaterloo.ca/tsp/concorde/index.html".

[32] S. L. B. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, vol. 21 (2), p. 498–516, 1973.

[33] N. Isoart and J.-C. Régin, "Adaptive cp-based lagrangian relaxation for tsp solving," *CPAIOR Conference*, 2020.

[34] P. Benchimol, H. Hoeve, and J.-C. R. et al., "Improved filtering for weighted circuit constraints." *Constraints*, vol. 17, pp. 205—233, 2012.

[35] M. Held and R. Karp, "The traveling-salesman problem and minimum spanning trees." *Operations Research*, p. 1138–1162, 1970.

[36] M. L. Fisher, *Management Science*, vol. Vol. 27, No. 1, 1981.

[37] J.-G. Fages, X. Lorca, and L.-M. Rousseau, "The salesman and the tree: the importance of search in cp." *Constraints*, vol. 21(2), pp. 145—-162, 2014.

[38] N. Isoart and J.-C. Régin., "Integration of Structural Constraints into TSP Models." *Principles and Practice of Constraint Programming*, pp. 284–299, 2019.

[39] Y. Tsin, "Yet another optimal algorithm for 3-edge-connectivity." *Journal of Discrete Algorithms*, vol. 7(1), pp. 130–146, 2009.

[40] L. Yeh, B. Wang, and H. Su, "Efficient algorithms for the problems of enumerating cuts by non-decreasing weights." *Algorithmica*, vol. 56(3), pp. 297—-312, 2010.

[41] M. Gendreau and C. D. Tarantilis, "Solving large-scale vehicle routing problems with time windows: The state-of-the-art," *Technical Report 04*, 2010. [Online]. Available: https://www.cirrelt.ca/documentstravail/cirrelt-2010-04.pdf

[42] R. Baldacci, A. Mingozzi, and R. Roberti, "New state-space relaxations for solving the traveling salesman problem with time windows," *INFORMS Journal on Computing*, vol. 24(3(, p. 356–371, 2012.

[43] J. Desrosiers, F. Soumis, and M. Desrochers, "Routing with time windows by column generation," *Networks*, vol. 14(4), pp. 545–565, 1984.

[44] J. Clausen, "Branch and bound algorithms -principles and examples," 1999.

[45] F. Focacci, A. Lodi, and M. Milano, "Cost-based domain filtering," *Principles and Practice of Constraint Programming*, pp. 189–203, 1999.

[46] N. Christofides, A. Mingozzi, and P. Toth, "State space relaxation procedures for the computation of bounds to routing problems," *Networks*, vol. 11, pp. 145–164, 1981.

[47] G. Pesant, "Counting solutions of csps: A structural approach." *IJCAI*, pp. 260–265, 2005.

[48] S.Chaiken and D.J.Kleitman, "Matrix tree theorems," *Journal of Combinatorial Theory, Series A*, vol. Volume 24, Issue 3, pp. 377–381, 1978.

[49] J. Sherman and W. J. Morrison, "Adjustment of an inverse matrix corresponding to changes in the elements of a given column or a given row of the original matrix," *Annals of Mathematical Statistics*, vol. 20, p. 621, 1949.

[50] L. Valiant, "The complexity of computing the permanent," *Theoretical Computer Science*, vol. Volume 8, Issue 2, pp. 189–201, 1979.

[51] G. Pesant, "Counting-based search for constraint optimization problems," *AAAI, D. Schuurmans and M. P. Wellman, Eds. AAAI Press*, pp. 3441—3448, 2016.

[52] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83—97, 1955.

[53] R. Cherkaoui El Azzouzi, “Algorithmes de dénombrement d’extensions linéaires d’un ordre partiel et application aux problèmes d’ordonnancement disjonctif,” Master’s thesis, École Polytechnique de Montréal, 2015.

[54] G. Karakostas, “A better approximation ratio for the vertex cover problem,” *International Colloquium on Automata, Languages, and Programming - Automata, Languages and Programming*, pp. 1043–1050, 2004.

[55] F. N. Abu-Khzam, R. L. Collins, M. R. Fellows, M. A. Langston, W. H. Suters, and C. T. Symons, “Kernelization algorithms for the vertex cover problem: Theory and experiments.” *ALENEX/ANALC, 69.*, 2004.

[56] S. Hougardy and R. T. Schroeder, “Edge elimination in tsp instances,” *Graph-Theoretic Concepts in Computer Science, WG.2014, Lecture Notes in Computer Science, vol 8747*, pp. 275–286, 2014.

[57] [Online]. Available: https://www.gurobi.com/documentation/9.1/quickstart_mac/cs_grbpy_the_gurobi_python.html

[58] S.Sahni and T. F. Gonzalez, “P-complete approximation problems,” *Journal of the ACM*, vol. Vol. 23, No. 3, 1976.

[59] S. Gagnon, “Improvement and integration of counting-based search heuristics in constraint programming,” Master’s thesis, École Polytechnique de Montréal, 2018.

[60] G. Reinelt, “Tsplib—a traveling salesman problem library.” *ORSA Journal on Computing*, vol. 3(4), p. 376–384, 1991.

[61] [Online]. Available: https://github.com/pdrozdowski/TSPLib.Net/tree/master/TSPLIB95

[62] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing (second ed)*, C. U. E. EPress., Ed., 1992.

[63] P. Jaccard, “Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines,” *Bulletin de la Société vaudoise des sciences naturelles*, vol. 37, pp. 241–272, 1901.

[64] [Online]. Available: http://lopez-ibanez.eu/tsptw-instances

APPENDIX A BEST SOLUTIONS FOUND IN BASIC CP MODEL

Instance	iloMinSizeInt	costMaxSD	Instance	iloMinSizeInt	costMaxSD
br17	39	39	ftv55	2834	1673
ft53	12181	7223	ftv64	3522	2115
ft70	53317	39692	ftv70	3382	2147
ftv170	6025	3089	kro124p	175195	39531
ftv33	1323	1364	p43	5995	5697
ftv35	1509	1475	rbg323	6287	6287
ftv38	1644	1532	rbg358	6999	6999
ftv44	1863	1706	rbg403	7566	7566
ftv47	2889	1790	rbg443	8392	8392

Table A.1 Comparison between `costMaxSD` and `iloMinSizeInt` in ILOG CP Model. We report results with a 3600s timeout, and we observe that almost all experiments reach the time limit.

Instance	100%	40%	20%	Instance	100%	40%	20%
br17	39	39	X	ftv55	1673	1771	1881
ft53	7223	8600	8349	ftv64	2115	1979	2082
ft70	39692	42481	44197	ftv70	2147	2118	2689
ftv170	3089	3359	3969	kro124p	39531	41503	45015
ftv33	1364	1364	1286	p43	5697	5698	5660
ftv35	1475	1533	1499	rbg323	6287	X	X
ftv38	1532	1511	1555	rbg358	6999	X	X
ftv44	1706	1701	1772	rbg403	7566	X	X
ftv47	1790	1956	1983	rbg443	8392	X	X

Table A.2 Results of sparsified instances with costMaxSD

APPENDIX B SPARSIFICATION IN THE DUMAS BENCHMARK

Instance	Base		30%		20%		10%	
	Time (s)	Solution	Time (%)	Gap (%)	Time (%)	Gap (%)	Time (%)	Gap (%)
n40w80.002	66.16	429.0	67.70	2.10	X	X	X	X
n40w100.002	75.96	358.0	86.12	0.56	X	X	X	X
n40w100.005	24.59	377.0	2741.76	11.94	X	X	X	X
n60w60.003	13.18	485.0	75.80	0.41	120.18	2.47	X	X
n60w80.001	24.97	457.0	111.65	0.00	66.32	0.22	X	X
n60w80.002	54.87	498.0	210.92	0.00	47.13	0.20	X	X
n60w80.003	21.92	550.0	89.87	0.00	64.87	0.91	X	X
n60w80.004	11.76	566.0	164.88	0.18	175.85	2.83	X	X
n60w80.005	152.47	468.0	151.79	1.92	X	X	X	X
n60w100.001	75.51	515.0	111.99	1.55	X	X	X	X
n60w100.002	1755.23	534.0	46.66	1.12	X	X	X	X
n60w100.003	162.19	557.0	507.37	2.15	X	X	X	X
n60w100.004	28.10	510.0	115.87	0.39	144.06	1.18	X	X
n80w40.002	35.48	613.0	108.57	0.00	134.19	0.00	54.90	0.65
n80w60.001	14.80	554.0	177.30	0.00	198.31	0.18	X	X
n80w60.004	200.19	612.0	201.57	0.00	43.24	0.00	X	X
n80w60.005	**2000**	575.0	100.00	0.00	100.00	0.00	20.13	6.26
n80w80.001	15.43	624.0	400.32	0.00	349.25	0.00	X	X
n80w80.002	189.64	591.0	277.08	0.00	123.97	0.00	50.72	10.49
n80w80.003	91.27	589.0	512.74	0.17	108.48	0.17	X	X
n80w80.004	**2000**	594.0	100.00	0.34	100.00	0.34	100.00	4.04
n80w80.005	12.89	570.0	220.17	0.00	110.40	0.00	X	X
n100w40.004	14.94	651.0	105.56	0.00	123.63	0.00	170.82	0.61
n100w60.001	48.19	655.0	108.18	0.00	599.09	2.29	X	X
n100w60.002	44.91	656.0	146.92	0.00	176.26	0.00	X	X
n100w60.003	**2000**	743.0	54.29	0.00	100.00	0.00	X	X
n100w60.004	19.31	764.0	239.15	0.00	1501.19	1.31	X	X
n150w40.001	23.68	912.0	255.15	0.00	141.98	0.00	350.84	1.10
n150w40.002	25.17	941.0	129.28	0.00	107.07	0.00	157.69	0.21
n150w40.003	17.74	727.0	177.40	0.00	157.05	0.00	119.95	0.41
n150w40.005	**2000**	823.0	100.00	0.00	100.00	0.00	100.00	0.00
n150w60.001	**2000**	851.0	X	X	X	X	100.00	1.53
n150w60.002	60.71	781.0	71.83	0.00	87.46	0.00	86.66	0.90
n150w60.003	**2000**	789.0	100.00	0.13	100.00	0.00	X	X
n150w60.004	**2000**	817.0	99.16	0.00	100.00	0.00	X	X
n150w60.005	**2000**	840.0	89.71	0.00	59.94	0.00	100.00	0.83
n200w20.002	24.65	970.0	119.39	0.00	135.78	0.00	127.71	0.00
n200w20.004	11.80	984.0	88.39	0.00	82.71	0.00	117.80	0.00
n200w40.001	**2000**	1021.0	100.00	0.00	100.00	0.00	100.00	0.00
n200w40.002	**2000**	945.0	100.00	0.00	100.00	0.00	100.00	0.21
n200w40.003	1028.89	929.0	76.74	0.00	94.27	0.00	59.21	0.22
n200w40.004	739.19	979.0	178.87	0.00	77.68	0.00	148.88	0.31
n200w40.005	319.09	1030.0	486.35	0.00	254.74	0.00	341.09	0.00
	geometric mean		136.94	0.54	124.06	0.34	104.92	1.43
	arithmetic mean		212.82	0.56	173.86	0.35	126.65	1.47
	median		111.82	0.00	107.07	0.00	100.00	0.41

Table B.1 Results of time and gaps to optimality for sparsification in the Dumas benchmark

APPENDIX C ORDERING CONSTRAINTS IN THE DUMAS BENCHMARK

Instance	Base		50%		100%		150%	
	Time (s)	Solution	Time (%)	Gap (%)	Time (%)	Gap (%)	Time (%)	Gap (%)
n40w80.002	66.16	429.0	7.33	0.00	2.78	0.00	1.60	6.53
n40w100.002	75.96	358.0	251.54	1.12	46.71	1.12	12.69	1.12
n40w100.005	24.59	377.0	88.41	0.00	45.99	0.00	160.55	0.00
n60w60.003	13.18	485.0	18.06	0.00	15.17	0.82	12.52	0.82
n60w80.001	24.97	457.0	91.63	0.00	57.07	0.88	85.82	4.16
n60w80.002	54.87	498.0	61.80	0.00	40.55	0.00	53.76	1.20
n60w80.003	21.92	550.0	68.43	0.00	36.86	0.55	40.47	0.55
n60w80.004	11.76	566.0	138.44	0.00	181.29	1.77	271.17	1.77
n60w80.005	152.47	468.0	151.56	0.00	34.77	0.43	33.98	2.56
n60w100.001	75.51	515.0	26.78	0.00	55.08	0.58	21.72	1.75
n60w100.002	1755.23	534.0	40.67	2.06	18.82	3.56	23.67	6.74
n60w100.003	162.19	557.0	64.23	0.00	75.71	0.36	125.36	0.36
n60w100.004	28.10	510.0	31.81	0.00	41.28	0.00	59.96	1.76
n80w40.002	35.48	613.0	55.10	0.98	45.55	1.96	X	X
n80w60.001	14.80	554.0	74.46	0.00	56.76	0.00	77.64	0.54
n80w60.004	200.19	612.0	333.25	0.00	X	X	X	X
n80w60.005	**2000**	575.0	30.60	0.35	11.89	3.13	42.59	3.65
n80w80.001	15.43	624.0	28.58	0.00	120.41	1.60	49.71	2.24
n80w80.002	189.64	591.0	161.25	0.17	52.20	0.17	12.02	2.03
n80w80.003	91.27	589.0	389.89	0.00	67.95	0.00	68.14	0.51
n80w80.004	**2000**	594.0	100.00	0.00	100.00	0.00	27.48	0.67
n80w80.005	12.89	570.0	28.63	0.18	73.78	0.35	104.65	4.04
n100w40.004	14.94	651.0	55.22	0.31	13.86	0.77	11.45	2.46
n100w60.001	48.19	655.0	68.23	0.00	107.82	1.22	88.38	1.22
n100w60.002	44.91	656.0	229.08	1.52	X	X	X	X
n100w60.003	**2000**	743.0	34.91	0.00	46.34	0.81	29.46	0.81
n100w60.004	19.31	764.0	84.88	0.00	71.21	0.00	61.21	0.26
n150w40.001	23.68	912.0	82.69	0.22	X	X	X	X
n150w40.002	25.17	941.0	40.80	0.32	X	X	X	X
n150w40.003	17.74	727.0	94.08	0.00	X	X	X	X
n150w40.005	**2000**	823.0	35.40	0.00	39.09	0.73	X	X
n150w60.001	**2000**	851.0	X	X	X	X	X	X
n150w60.002	60.71	781.0	65.23	0.26	72.16	1.54	X	X
n150w60.003	**2000**	789.0	100.00	0.00	100.00	0.00	100.00	0.76
n150w60.004	**2000**	817.0	100.00	0.86	38.39	0.86	X	X
n150w60.005	**2000**	840.0	X	X	X	X	X	X
n200w20.002	24.65	970.0	57.48	1.24	X	X	X	X
n200w20.004	11.80	984.0	56.95	0.20	61.61	2.13	18.90	3.05
n200w40.001	**2000**	1021.0	100.00	0.00	100.00	0.88	100.00	0.98
n200w40.002	**2000**	945.0	100.00	0.63	X	X	X	X
n200w40.003	1028.89	929.0	194.39	1.29	X	X	X	X
n200w40.004	739.19	979.0	X	X	X	X	X	X
n200w40.005	319.09	1030.0	308.42	0.19	X	X	X	X
		geometric mean	73.39	0.30	46.53	0.84	41.05	1.93
		arithmetic mean	101.25	0.30	59.07	0.85	62.77	1.95
		median	71.45	0.00	52.2	0.73	49.71	1.22

Table C.1 Times and gaps to optimality for adding ordering constraints in the Dumas benchmark

APPENDIX D ORDERING CONSTRAINTS WITH EXACT ALGORITHM IN THE DUMAS BENCHMARK

Instance	Base		50%		100%		150%	
	Time (s)	Solution	Time (%)	Gap (%)	Time (%)	Gap (%)	Time (%)	Gap (%)
n20w20.001	0.01	378.0	X	X	X	X	X	X
n20w20.003	0.01	394.0	0.00	3.81	X	X	X	X
n20w40.001	0.05	254.0	40.00	4.72	X	X	X	X
n20w40.002	0.01	333.0	X	X	X	X	X	X
n20w40.003	0.02	317.0	50.00	5.05	X	X	X	X
n20w40.004	0.02	388.0	X	X	X	X	X	X
n20w40.005	0.04	288.0	25.00	0.00	0.00	11.81	X	X
n20w60.001	0.04	335.0	100.00	0.00	225.00	3.58	25.00	12.24
n20w60.002	0.02	244.0	100.00	0.00	50.00	3.69	X	X
n20w60.003	0.03	352.0	33.33	1.99	X	X	X	X
n20w60.004	0.16	280.0	81.25	0.00	50.00	0.00	12.50	0.00
n20w60.005	0.04	338.0	75.00	2.66	X	X	X	X
n20w80.001	0.06	329.0	100.00	0.30	66.67	0.30	16.67	6.08
n20w80.002	0.03	338.0	100.00	0.00	100.00	7.10	100.00	17.46
n20w80.003	0.04	320.0	75.00	0.94	50.00	8.12	50.00	8.12
n20w80.004	0.05	304.0	320.00	7.57	80.00	7.57	20.00	9.21
n20w80.005	0.39	264.0	53.85	0.00	117.95	3.03	66.67	6.06
n20w100.001	0.45	237.0	102.22	0.00	13.33	0.00	57.78	16.88
n20w100.002	0.25	222.0	288.00	11.71	204.00	13.51	184.00	19.37
n20w100.003	0.49	310.0	79.59	0.00	10.20	1.29	6.12	2.58
n20w100.004	0.05	349.0	80.00	0.00	80.00	5.73	80.00	8.31
n20w100.005	0.09	258.0	66.67	0.00	44.44	0.00	44.44	11.63
n40w20.001	0.02	497.0	0.00	1.61	X	X	X	X
n40w20.002	0.02	552.0	50.00	3.08	X	X	X	X
n40w20.003	0.01	478.0	100.00	0.84	X	X	X	X
n40w20.004	0.01	404.0	100.00	2.23	X	X	X	X
n40w20.005	0.01	499.0	X	X	X	X	X	X
n40w40.001	0.03	465.0	100.00	0.43	X	X	X	X
n40w40.002	0.25	461.0	108.00	1.95	24.00	5.21	X	X
n40w40.003	0.06	470.0	33.33	0.00	33.33	5.74	X	X
n40w40.004	0.11	452.0	63.64	1.99	27.27	3.54	18.18	4.87
n40w40.005	0.21	453.0	61.90	0.22	19.05	1.10	14.29	1.77
n40w60.001	0.68	494.0	123.53	0.00	75.00	0.00	88.24	7.69
n40w60.002	2.34	470.0	5.98	0.00	3.85	2.77	4.70	2.77
n40w60.003	0.74	408.0	102.70	0.74	82.43	0.98	35.14	2.94
n40w60.004	0.66	382.0	71.21	0.00	81.82	1.83	22.73	2.88
n40w60.005	0.20	328.0	45.00	0.00	125.00	4.27	60.00	10.37
n40w80.001	0.79	395.0	92.41	0.00	96.20	1.01	32.91	4.05
n40w80.002	66.16	429.0	11.28	0.00	6.67	0.00	1.50	0.00
n40w80.003	3.37	410.0	118.69	1.95	70.92	2.68	12.17	3.66
n40w80.004	0.64	417.0	164.06	0.00	79.69	1.68	40.62	2.64
n40w80.005	4.71	344.0	74.31	0.00	20.38	0.00	23.14	1.74
n40w100.001	7.65	429.0	129.28	0.00	153.07	0.23	70.59	0.23
n40w100.002	75.96	358.0	30.29	1.12	12.01	1.12	12.60	1.40
n40w100.003	4.76	359.0	121.01	0.00	63.87	2.79	75.21	4.18
n40w100.004	3.39	357.0	70.50	0.00	48.38	0.00	37.17	0.00
n40w100.005	24.59	377.0	63.16	0.00	42.66	0.00	36.76	0.00
	geometric mean		0.00	1.25	0.00	3.00	28.82	5.47
	arithmetic mean		83.38	1.28	65.40	3.05	43.07	5.60
	median		75.00	0.00	50.00	1.83	35.14	4.05

Table D.1 Times and gaps to optimality for Dumas instances where ordering constraints have been added on the basis of the exact algorithm. In this case only, we add all the instances with no filtering on the solving times, to have more data available

APPENDIX E ORDERING CONSTRAINTS WITH PARTIAL OVERLAPS IN THE DUMAS BENCHMARK

Instance	Base		50%		100%		150%	
	Time (s)	Solution	Time (%)	Gap (%)	Time (%)	Gap (%)	Time (%)	Gap (%)
n40w80.002	66.16	429.0	22.61	0.00	15.80	0.00	15.70	6.53
n40w100.002	75.96	358.0	111.69	0.00	142.19	0.00	X	X
n40w100.005	24.59	377.0	64.54	0.00	89.10	0.00	228.75	0.00
n60w60.003	13.18	485.0	73.98	0.00	13.96	0.00	56.15	0.00
n60w80.001	24.97	457.0	70.85	0.00	34.40	0.00	X	X
n60w80.002	54.87	498.0	75.25	0.00	78.48	0.00	46.49	0.00
n60w80.003	21.92	550.0	65.33	0.00	94.94	0.55	101.37	0.55
n60w80.004	11.76	566.0	86.99	0.00	145.49	0.00	86.14	0.00
n60w80.005	152.47	468.0	61.53	0.21	56.37	0.21	27.59	0.43
n60w100.001	75.51	515.0	133.24	0.00	216.16	0.00	179.53	0.00
n60w100.002	1755.23	534.0	99.02	0.00	22.98	0.00	14.49	1.87
n60w100.003	162.19	557.0	49.02	0.00	56.33	0.00	56.57	0.00
n60w100.004	28.10	510.0	127.72	0.00	14.56	0.98	X	X
n80w40.002	35.48	613.0	101.63	1.14	60.15	1.14	18.21	1.47
n80w60.001	14.80	554.0	203.31	0.00	66.55	0.00	X	X
n80w60.004	200.19	612.0	109.04	0.00	134.33	0.00	132.38	0.00
n80w60.005	**2000**	575.0	100.00	0.35	100.00	1.39	X	X
n80w80.001	15.43	624.0	127.28	0.00	128.32	0.00	83.67	0.00
n80w80.002	189.64	591.0	196.11	0.00	133.38	0.00	90.17	0.17
n80w80.003	91.27	589.0	71.41	0.00	92.09	0.00	116.11	0.00
n80w80.004	**2000**	594.0	100.00	0.00	100.00	0.17	100.00	0.17
n80w80.005	12.89	570.0	152.52	0.00	139.88	0.00	151.51	0.00
n100w40.004	14.94	651.0	102.95	0.00	162.25	0.46	63.99	0.46
n100w60.001	48.19	655.0	140.28	0.00	151.59	0.00	128.26	0.31
n100w60.002	44.91	656.0	149.14	0.00	335.52	0.00	157.07	0.15
n100w60.003	**2000**	743.0	100.00	0.00	100.00	0.27	93.07	0.27
n100w60.004	19.31	764.0	51.68	0.00	196.69	0.00	60.95	0.00
n150w40.001	23.68	912.0	129.56	0.00	52.32	0.00	158.02	0.22
n150w40.002	25.17	941.0	82.52	0.00	85.86	0.32	X	X
n150w40.003	17.74	727.0	116.01	0.00	118.94	0.28	22.32	0.28
n150w40.005	**2000**	823.0	100.00	0.00	96.88	0.00	21.60	0.12
n150w60.001	**2000**	851.0	X	X	X	X	X	X
n150w60.002	60.71	781.0	85.50	0.00	69.74	0.00	94.07	0.00
n150w60.003	**2000**	789.0	100.00	0.00	100.00	0.00	100.00	2.15
n150w60.004	**2000**	817.0	100.00	0.00	100.00	0.00	100.00	0.00
n150w60.005	**2000**	840.0	80.89	0.00	89.51	0.00	52.25	0.00
n200w20.002	24.65	970.0	47.51	0.21	X	X	X	X
n200w20.004	11.80	984.0	76.19	0.00	46.78	0.20	X	X
n200w40.001	**2000**	1021.0	100.00	0.00	X	X	X	X
n200w40.002	**2000**	945.0	100.00	0.00	100.00	0.42	X	X
n200w40.003	1028.89	929.0	86.84	0.00	187.11	0.11	194.39	0.75
n200w40.004	739.19	979.0	270.57	0.00	159.24	0.00	54.28	0.41
n200w40.005	319.09	1030.0	79.28	0.00	166.89	0.00	93.67	0.00
		geometric mean	93.97	0.05	88.68	0.16	72.12	0.50
		arithmetic mean	102.43	0.05	109.39	0.16	90.59	0.51
		median	100.00	0.00	100.00	0.00	91.62	0.14

Table E.1 Times and gaps to optimality for adding ordering consraints based on the variant with partial overlaps in the Dumas benchmark

APPENDIX F ORDERING CONSTRAINTS IN THE OHLMANNTHOMAS BENCHMARK

Instance	Base		50%		100%		150%	
	Time (s)	Solution	Time (%)	Gap (%)	Time (%)	Gap (%)	Time (%)	Gap (%)
n150w120.001	**200000**	734.0	65.45	0.00	100.00	0.00	100.00	0.00
n150w120.002	**200000**	674.0	100.00	0.89	100.00	1.93	100.00	1.63
n150w120.003	4650.7	746.0	58.70	0.00	43.11	0.00	56.80	0.00
n150w120.004	**200000**	inf	68.28	-100.00	90.20	-100.00	85.45	-100.00
n150w120.005	17581.58	685.0	755.99	1.02	573.64	1.02	166.75	1.02
n150w140.001	**200000**	756.0	100.00	0.00	100.00	0.53	100.00	0.00
n150w140.002	**200000**	753.0	100.00	0.00	100.00	0.00	86.35	0.00
n150w140.003	**200000**	617.0	100.00	-1.13	100.00	0.00	100.00	0.00
n150w140.004	1715.99	672.0	38.19	0.00	28.50	0.00	48.73	0.00
n150w140.005	**200000**	664.0	100.00	0.00	100.00	-0.15	100.00	1.05
n150w160.001	81163.34	705.0	109.86	0.00	111.35	0.00	60.50	0.00
n150w160.002	**200000**	708.0	100.00	0.00	100.00	0.00	100.00	0.42
n150w160.003	**200000**	605.0	X	X	100.00	0.66	X	X
n150w160.004	**200000**	670.0	47.28	1.64	100.00	0.75	100.00	0.00
n150w160.005	**200000**	661.0	29.33	-0.45	100.00	0.45	100.00	0.61
n200w120.001	93022.06	795.0	95.68	0.00	93.40	0.00	215.00	0.75
n200w120.002	**200000**	721.0	100.00	0.14	100.00	0.00	100.00	0.00
n200w120.003	**200000**	inf	X	X	X	X	X	X
n200w120.004	5217.51	772.0	209.88	0.00	314.91	0.00	68.70	0.00
n200w120.005	**200000**	841.0	100.00	0.00	100.00	0.12	100.00	0.48
n200w140.001	**200000**	inf	99.97	-100.00	X	X	99.97	-100.00
n200w140.002	**200000**	inf	100.00	-100.00	100.00	-100.00	100.00	-100.00
n200w140.003	**200000**	741.0	100.00	-0.27	100.00	0.40	100.00	0.00
n200w140.004	**200000**	809.0	100.00	0.37	100.00	2.35	100.00	2.22
n200w140.005	**200000**	inf	X	X	100.00	-100.00	X	X
		geometric mean	93.57	0.11	103.24	0.42	94.73	0.43
		arithmetic mean	121.75	0.12	124.14	0.42	99.47	0.43
		median	100.00	0.00	100.00	0.00	100.00	0.00

Table F.1 Times and gaps to optimality for adding ordering constraints in the OhlmannThomas benchmark

APPENDIX G SPARSIFICATION IN THE OHLMANNTHOMAS BENCHMARK

Instance	Base		50%		30%		20%	
	Time (s)	Solution	Time (%)	Gap (%)	Time (%)	Gap (%)	Time (%)	Gap (%)
n150w120.001	**200000**	734.0	100.00	0.00	100.00	1.09	100.00	3.13
n150w120.002	**200000**	674.0	X	X	100.00	0.89	100.00	1.19
n150w120.003	4650.7	746.0	125.13	0.00	147.32	0.13	80.63	0.94
n150w120.004	**200000**	inf	100.00	-100.00	100.00	-100.00	13.08	-100.00
n150w120.005	17581.58	685.0	288.04	0.00	318.75	0.73	749.61	2.48
n150w140.001	**200000**	756.0	100.00	0.00	100.00	2.65	100.00	6.75
n150w140.002	**200000**	753.0	100.00	0.00	100.00	1.20	X	X
n150w140.003	**200000**	617.0	100.00	-1.30	100.00	1.78	X	X
n150w140.004	1715.99	672.0	90.19	0.00	106.97	1.04	X	X
n150w140.005	**200000**	664.0	100.00	0.00	100.00	0.00	100.00	0.45
n150w160.001	81163.34	705.0	136.36	0.00	165.01	2.27	10.64	8.23
n150w160.002	**200000**	708.0	100.00	0.99	100.00	1.98	X	X
n150w160.003	**200000**	605.0	100.00	-0.66	100.00	0.17	X	X
n150w160.004	**200000**	670.0	100.00	0.75	X	X	X	X
n150w160.005	**200000**	661.0	X	X	X	X	X	X
n200w120.001	93022.06	795.0	59.50	0.00	81.55	0.00	215.00	1.51
n200w120.002	**200000**	721.0	100.00	0.14	100.00	0.28	X	X
n200w120.003	**200000**	inf	X	X	X	X	X	X
n200w120.004	5217.51	772.0	115.72	0.00	70.18	0.00	3833.25	3.89
n200w120.005	**200000**	841.0	100.00	0.12	100.00	0.36	100.00	1.07
n200w140.001	**200000**	inf	X	X	X	X	X	X
n200w140.002	**200000**	inf	100.00	-100.00	100.00	-100.00	100.00	-100.00
n200w140.003	**200000**	741.0	100.00	0.40	100.00	0.40	X	X
n200w140.004	**200000**	809.0	100.00	0.25	X	X	100.00	1.85
n200w140.005	**200000**	inf	X	X	100.00	-100.00	X	X
		geometric mean	105.74	0.04	108.10	0.88	116.06	2.84
		arithmetic mean	110.75	0.04	114.49	0.88	430.94	2.86
		median	100.00	0.00	100.00	0.73	100.00	1.85

Table G.1 Times and gaps to optimality for adding sparsification in the OhlmannThomas benchmark