



UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

**GRAPHICAL TRAVELING SALESMAN PROBLEM WITH RELEASE
DATES IN PATH-LIKE GRAPHS: ALGORITHMS AND COMPLEXITY**

Thailsson Clementino de Andrade

February 2026

Manaus - AM



UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

**GRAPHICAL TRAVELING SALESMAN PROBLEM WITH RELEASE DATES IN
PATH-LIKE GRAPHS: ALGORITHMS AND COMPLEXITY**

Thailsson Clementino de Andrade

Master's dissertation submitted to the
Postgraduate Program in Computer
Science/Informatics (PPGI) at the Institute of
Computing, Universidade Federal do Amazonas,
in partial fulfillment of the requirements for the
degree of Master of Science.

Advisor: Rosiane de Freitas Rodrigues, Ph.D.

February 2026

Manaus - AM

Ficha Catalográfica

Elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

A553g Andrade, Thailsson Clementino de
 Graphical Traveling Salesman Problem With Release Dates in Path-like
 Graphs: Algorithms and Complexity / Thailsson Clementino de Andrade. -
 2026.
 74 f. : il., color. ; 31 cm.

 Orientador(a): Rosiane de Freitas Rodrigues.
 Dissertação (mestrado) - Universidade Federal do Amazonas, Programa
 de Pós-Graduação em Informática, Manaus, 2026.

 1. Polynomial-time algorithms. 2. Combinatorial Optimization. 3.
 Complexity analysis. 4. Dynamic Programming. 5. Graph Theory. I.
 Rodrigues, Rosiane de Freitas. II. Universidade Federal do Amazonas.
 Programa de Pós-Graduação em Informática. III. Título



Ministério da Educação
Universidade Federal do Amazonas
Coordenação do Programa de Pós-Graduação em Informática

FOLHA DE APROVAÇÃO

"GRAPHICAL TRAVELING SALESMAN PROBLEM WITH RELEASE DATES IN PATH-LIKE GRAPHS: ALGORITHMS AND COMPLEXITY"

THAILSSON CLEMENTINO DE ANDRADE

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof.^a Dr.^a Rosiane de Freitas Rodrigues - **Presidente**

Prof. Dr. Edleno Silva de Moura - **Membro Interno**

Prof. Dr. Diego Delle-Donne - **Membro Externo**

 Firmado digitalmente
por Diego Delle Donne
Fecha: 2026.03.03
18:30:33 +01'00'

Documento assinado digitalmente
gov.br SANTIAGO VALDES RAVELO
Data: 03/03/2026 11:56:28-0300
Verifique em <https://validar.iti.gov.br>

Prof. Dr. Santiago Valdés Ravelo - **Membro Externo**

Manaus, 20 de Fevereiro de 2026.



Documento assinado eletronicamente por **Rosiane de Freitas Rodrigues, Professor do Magistério Superior**, em 23/02/2026, às 15:04, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



Documento assinado eletronicamente por **Edleno Silva de Moura, Professor do Magistério Superior**, em 23/02/2026, às 15:59, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



A autenticidade deste documento pode ser conferida no site

[https://sei.ufam.edu.br/sei/controlador_externo.php?](https://sei.ufam.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0)

[acao=documento_conferir&id_orgao_acesso_externo=0](https://sei.ufam.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0), informando o código verificador **3020349** e o código CRC **B58BDDDB3**.

Avenida General Rodrigo Octávio, 6200 - Bairro Coroados I Campus Universitário Senador Arthur Virgílio
Filho, Setor Norte - Telefone: (92) 3305-1181 / Ramal 1193
CEP 69080-900, Manaus/AM, coordenadorppgi@icomp.ufam.edu.br

Referência: Processo nº 23105.005952/2026-94

SEI nº 3020349

*Para Maria, Jozenira
e todas as pessoas que amo.*

Acknowledgments

Primeiramente, gostaria de agradecer minha orientadora, Rosiane de Freitas, pelo suporte, sugestões, discussões e encorajamento ao longo dos mais de dois anos de desenvolvimento deste trabalho.

Agradeço também aos colegas e amigos da UFAM que estiveram presentes ao longo desse período, em especial Guilherme, Castro e Ricardo, bem como ao pessoal do grupo de programação competitiva, que tive a oportunidade de coordenar durante o mestrado, muito obrigado pelo aprendizado que tive. Aos colegas de mestrado Lia e Jessé, com quem construí uma amizade que espero levar para a vida, e a tantos outros não mencionados aqui, com quem pude compartilhar conhecimento e experiências.

Registro ainda meu agradecimento ao Jonas Costa, pelas conversas sobre o problema e pela revisão de algumas provas durante sua estadia. De forma mais ampla, sou grato a todos que contribuíram, direta ou indiretamente, para a realização deste trabalho, seja pela companhia no laboratório, por um café compartilhado, por palavras de incentivo, momentos de descontração ou pequenos gestos de apoio ao longo do caminho.

Por fim, gostaria de expressar minha gratidão à minha família, que sempre me apoiou, aos meus amigos em geral, ao meu gatinho Francisco e à minha futura esposa, Maria Eloá do Nascimento Arantes.

A todos aqueles que contribuíram para esta dissertação, de maneiras grandes e pequenas, ofereço meus sinceros agradecimentos.

Errare humanum est

— *Jorge Ben*

Abstract of Dissertation presented to IComp/UFAM as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

GRAPHICAL TRAVELING SALESMAN PROBLEM WITH RELEASE DATES IN PATH-LIKE GRAPHS: ALGORITHMS AND COMPLEXITY

Thailsson Clementino de Andrade

Advisor: Rosiane de Freitas Rodrigues

Program: Computer Science (PPGI)

In this dissertation, we investigate the Graphical Traveling Salesman Problem with Release Dates (GTSP-rd), a generalization of the classical NP-hard TSP in which each vertex has an associated release date, the earliest time a vertex becomes available to be served. Unlike the traditional approach of transforming the input into a complete graph, the GTSP-rd is solved directly on the original graph, preserving structural properties that may admit more efficient algorithms for special cases. Motivated by real-world applications such as logistics in grid-based warehouses, rural road networks, and same-day express deliveries in general, this work aims to determine for which graph classes the GTSP-rd can be solved in polynomial time. Based on previous polynomial-time results for path graphs and a few other structures, this research focuses on Path-like graphs, including paths, cycles, stars, caterpillars, lobsters, and subdivided stars. Thus, this work provides: (i) a precise and explicit formulation of the GTSP-rd, filling gaps in the existing literature; (ii) improved linear and near-linear algorithms for paths with the depot at an extremity; (iii) new polynomial-time dynamic programming algorithms for general paths, stars, and cycles; and (iv) structural insights and partial results for caterpillar, lobster and subdivided stars. Overall, this work advances the understanding of how graph structure influences the computational complexity of the GTSP-rd, expands the set of graph classes known to admit polynomial-time solutions, and establishes new theoretical insights that may support further progress toward a broader characterization of other tractable cases.

Keywords: polynomial-time algorithms; combinatorial optimization; complexity analysis; dynamic programming; graph theory.

Contents

List of Figures	x
List of Tables	xii
1 Introduction	1
1.1 Research Question and Goals	4
1.2 Methodology	5
1.3 Main Contributions and Text Organization	7
2 Theoretical Basis	9
2.1 Graph Theory Related Concepts	9
2.1.1 Path-like Graphs	10
2.1.2 Other Concepts and Graph Classes	11
2.2 Routing Problems on Graph Classes	13
2.3 Graphical Traveling Salesman Problem - GTSP	15
3 Graphical Traveling Salesman Problem with release dates (GTSP-rd)	19
3.1 Problem Definition	19
3.2 Related Works	22
3.2.1 GTSP-rd on Paths	22
3.2.2 GTSP-rd on Stars	26
4 Main Results: Algorithms and Proofs	27
4.1 GTSP-rd on Paths with Depot at Extremity	28
4.1.1 Proposed Algorithm for GTSP-rd (time) in $O(n)$	28
4.1.2 Proposed Algorithm for GTSP-rd (distance) in $O(n \log n)$	32
4.2 GTSP-rd on Paths with Depot located anywhere	34

4.2.1	Algorithm $O(n^2)$ for GTSP-rd (time)	35
4.2.2	Algorithm $O(n^2 \log n)$ for GTSP-rd (distance)	39
4.3	GTSP-rd on Stars	40
4.3.1	Algorithm $O(n^2)$ for GTSP-rd(time)	41
4.3.2	Algorithm $O(n^3)$ for GTSP-rd(distance)	44
4.4	GTSP-rd on Cycle Graphs	48
4.4.1	Algorithm $O(n^3)$ for GTSP-rd(time)	48
4.4.2	Algorithm $O(n^3)$ for GTSP-rd(distance)	55
4.5	Discussion of Results	61
4.5.1	GTSP-rd on Subdivided Stars	61
4.5.2	GTSP-rd on Catterpillars	64
5	Concluding Remarks	67
	References	70

List of Figures

1.1	Delivery problem with release dates associated with each delivery point. The objective is to minimize the completion time. Two tours are executed: the first to serve the furthest points and the second to serve the nearby points.	2
1.2	Examples of special graph classes arising in real-world problems.	4
1.3	Master dissertation big-picture.	6
2.1	Examples of Path-like graphs.	11
2.2	Hierarchy of path-like graph classes.	12
2.3	Graphs M_1 , M_2 , and M_3 defined by Fonlupt and Naddef (1992). The TSP- perfect graphs are graphs that exclude these graph as minors.	17
3.1	A GTSP-rd solution, containing the three routes $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3$	21
3.2	Example of a tree instance in which it is impossible to construct a route without repeating vertices. In the route $\mathcal{R} = \langle v_0, v_1, v_3, v_4, v_3, v_5, v_3, v_1, v_0 \rangle$, vertex v_3 is traversed three times.	21
3.3	Route of customer v_i along with other vertices $v_{j+1}, \dots, v_{i-1}$	25
3.4	Star with depot located in a leaf.	26
4.1	A general Path instance with the depot in an arbitrary vertex.	34
4.2	Equation 4.5 represents the choice of creating a new non-interlacing route for each side, containing only the customers on that side.	36

4.3	Illustration of the use of multiple <code>minqueue</code> data structures in the dynamic programming computation of $L(i, j)$. Each column j maintains its own minqueue QL_j , which stores the values required to efficiently compute the minimum over the interval $[k_j^l + 1, i - 1]$ in the corresponding recurrence. Separate minqueues are necessary because the relevant values depend on the column being processed.	39
4.4	Star graph with the depot at a leaf. The center is v_n , and the remaining vertices $v_1, \dots, v_{n-1}$ are sorted so that $r_1 \leq r_2 \leq \dots \leq r_{n-1}$	41
4.5	Example illustrating the arrangement of vertices in a GTSP-rd instance on a cycle.	48
4.6	Path P_e , given the edge $e = \{v_s, v_t\} \in E$, remove it, and obtain a path whose extremities correspond to the two vertices incident to e	52
4.7	GTSP-rd has polynomial solutions to star and path, but not to subdivided star.	62
4.8	The spine with vertices $\dots, v_{i-1}, v_i, v_{i+1}, \dots$, each connected to multiple leaf vertices (legs). At node v_i , the tour must choose a subset of legs to visit before move to v_{i-1} or v_{i+1}	64

List of Tables

2.1	Best Known results on time complexity to the CVRP (Archetti et al., 2011).	14
2.2	Best Known results on time complexity to the SDVRP (Archetti et al., 2011).	14
2.3	Best Known results on time complexity to the UVRP-rd. Reyes et al. (2018) address the problem with a fixed deadline while Archetti et al. (2015) address without a deadline.	15
2.4	Known polynomial-time complexity results for GTSP on graph classes. . .	18
3.1	Previous time and distance complexity results for the GTSP-rd on path graphs. All listed algorithms are based on dynamic programming.	22
4.1	Updated complexity results for GTSP-rd on paths.	61
4.2	Summary of complexity results obtained in this work for the GTSP-rd on path-like graph classes.	62

Chapter 1

Introduction

The Traveling Salesman Problem (TSP) is a well-known combinatorial optimization problem that seeks to determine the shortest possible tour to visit a given set of cities exactly once and return to the origin city (Cook, 2015; Jünger et al., 1995). A relative new variation of this problem is the Traveling Salesman Problem with release dates (TSP-rd) (Archetti et al., 2015). In this problem we name an origin city as depot. Each city has an associated release date, which is the earliest time that the tour containing this city can begin. From this constraint, it is possible that the solution could be composed by several tours beginning and ending at depot.

The goal is to find tours with minimum cost that satisfy the release date constraints of each city, ensuring that the salesman does not depart to any city before its release date, and pass through all cities. The TSP-rd adds a layer of complexity to the classical TSP by incorporating temporal constraints, making it significantly more challenging to solve.

For instance, consider a package delivery scenario where each package has a ready time (a release date) and a single vehicle must minimize the total completion time. In such a case, it is often better to split the deliveries into two tours. The first tour can serve distant locations that become available early, while the second tour serves nearby locations that have later release dates. This strategy can be more efficient than waiting for all packages to become available. Therefore, the challenge is not only to find the shortest tour but also to schedule visits to deal with the release dates. Figure 1.1 illustrates this example.

Both the TSP and the TSP-rd are NP-hard (Garey and Johnson, 1979; Archetti et al., 2015). In particular, assuming a metric graph as input, when all release dates are

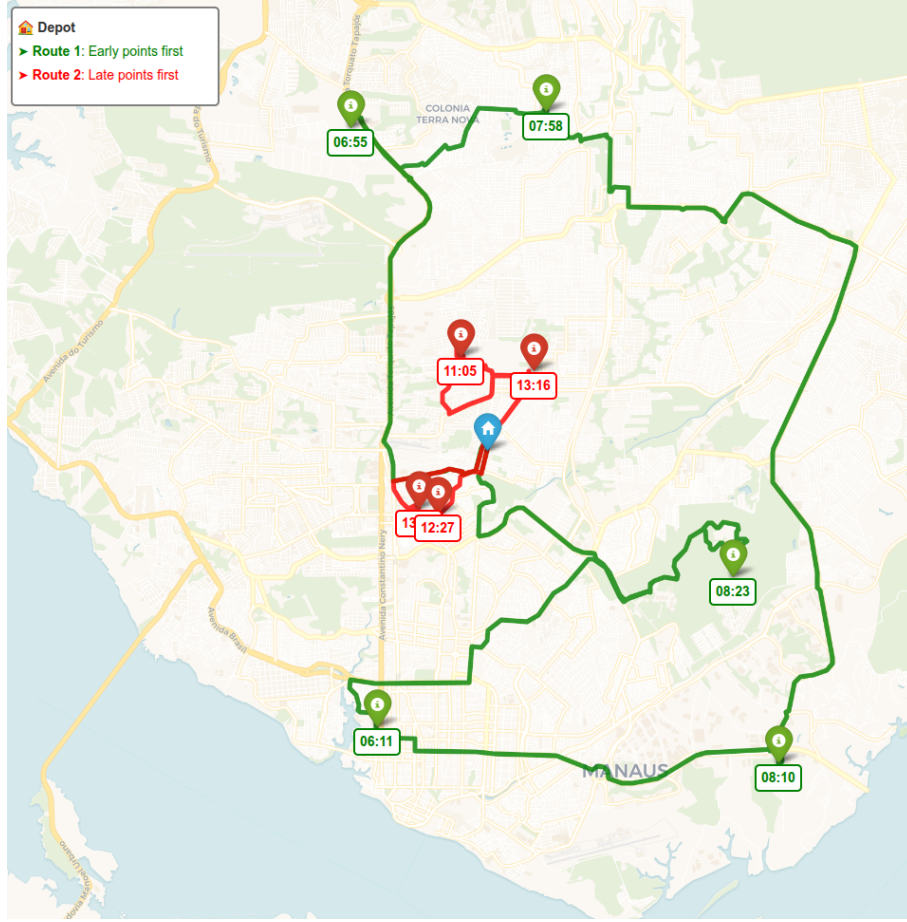


Figure 1.1: Delivery problem with release dates associated with each delivery point. The objective is to minimize the completion time. Two tours are executed: the first to serve the furthest points and the second to serve the nearby points.

equal, the TSP-rd reduces to the classical TSP, since minimizing the cost in this case is equivalent to determining the minimum-cost Hamiltonian tour that visits each city exactly once.

Usually, the TSP is modeled as a graph $G = (V, E)$, where each city is represented by a vertex $v \in V$ and each edge $\{u, v\}$ represents a road between cities u and v . Each edge has a length d_{uv} , which is the distance between cities. Then, the graph problem is to determine the Hamiltonian Cycle with minimum cost.

Although two cities may not have a direct road connecting them, if the input graph $G = (V, E)$ is not a complete one, the standard approach in the literature is transforming it in a complete graph $K = (V, E')$ on the same vertex set, where each edge $\{u, v\} \in E'$ has the cost the shortest path from u to v in the original graph G . K is called the metric closure of G (Hargrave and Nemhauser, 1962; Vazirani, 2001; Goyal and Mömke, 2020).

Consequently, it is possible to transform any graph into a complete graph and

then solve the TSP using metaheuristics or exact methods, as several have been proposed in the literature. However such transformation can discard potentially polynomial-time solutions inherent to the original graph structure. This loss is particularly significant when the original graph has characteristics that allow for easier or faster solutions.

Studying the TSP and its variants on specific, graph classes yields significant practical implications. A canonical example is found in warehouse logistics, where the physical layout of aisles and storage locations can be precisely modeled as a grid graph¹. In this model, vertices represent picking locations and intersections, while edges correspond to navigable aisles, as exemplified in Figure 1.2a (Ratliff and Rosenthal, 1983; Lüke et al., 2024; Bock et al., 2025).

Specific graph topologies emerge naturally in urban and regional routing. City plans with perpendicular street networks, such as Manhattan-style layouts, inherently resemble grid graphs. Conversely, road networks characterized by a long, central path with connecting side streets can be modeled as caterpillar graphs². This topology provides a natural abstraction for scenarios ranging from rural highways (Yang et al., 2020) to isolated infrastructure such as the Stuart Highway in central Australia or roads traversing the Amazon rain-forest (see Figure 1.2b).

Furthermore, the relevance of specialized graph classes extends to scenarios of network disruption. For instance, the analysis of post-disaster delivery networks, such as those impacted by an earthquake, necessitates the study of graph classes that capture the resulting fragmented or altered connectivity (Khare et al., 2021).

The approach of solving the TSP directly on the original graph, without converting it into a complete one, is referred to as the Graphical Traveling Salesman Problem (GTSP) in order to differentiate from classical TSP. Some works have already used this approach, is the case of Cornuéjols et al. (1983) and Ratliff and Rosenthal (1983).

In this dissertation, we address the Graphical Traveling Salesman Problem with release dates (GTSP-rd), which is a variation of TSP-rd where the problem is solved directly on the given graph, just like GTSP. Following the approach taken by previous works on GTSP, a natural question arises for GTSP-rd: for which classes of graphs does

¹A grid graph is a graph whose vertices correspond to points in an $m \times n$ grid, with edges connecting each vertex to its horizontally and vertically adjacent neighbors.

²A caterpillar graph is a tree in which all vertices lie on a central path or are connected to it by a single edge.

³See: <https://www.swisslog.com/>



(a) Micro fulfillment based on Autostore (Source: Swisslog³).



(b) Caterpillar (or even a Path) graph formed on road in Amazon rain-forest.

Figure 1.2: Examples of special graph classes arising in real-world problems.

the GTSP-rd admit a polynomial-time solution?

Although they did not explicitly refer to the problem as the GTSP-rd, Archetti et al. (2015) and Reyes et al. (2018) proposed polynomial-time algorithms for it on specific graph classes, paths and a particular case of stars. These two studies form the foundation of our research. Since the case of paths has already been solved in polynomial time, we focus on graph classes that can be constructed using paths as their main structural components. In this context, we refer to these graphs as **Path-like graphs**. Examples of such graphs include paths, cycles (closed paths), trees like caterpillars and lobsters (where the central structure of the graph is a path), and subdivided stars (trees with a central vertex from which multiple paths emanate).

1.1 Research Question and Goals

The central **research question** is: **Can better asymptotic upper bounds be achieved for the GTSP-rd in Path-like graphs?**

The **principal goal** of this research is to **provide polynomial cases with better upper bounds for the Graphical Traveling Salesman Problem with Release Dates (GTSP-rd) on Path-like graphs**. This includes both cases where polynomial-time algorithms are known and cases where the problem remains computationally intractable. The **secondary goals** are:

- Investigate and characterize the structural properties of Path-like graphs that enable the development of polynomial-time algorithms for GTSP-rd.

- Provide formal proofs of optimality for the proposed algorithms, ensuring their correctness and exactness in solving GTSP-rd.
- Provide time complexity analysis for the proposed algorithms, giving its respective asymptotic upper bound in big-O notation.

1.2 Methodology

This work is a theoretical study in Computer Science, focusing on algorithm design, computational complexity, and graph theory, with applications to routing and scheduling problems. It is a **pure research**, as it aims to advance fundamental knowledge by exploring the underlying principles of computational complexity and algorithm design for the Graphical Traveling Salesman Problem with release dates (GTSP-rd).

In terms of its objective, the research is **explanatory**, as it seeks to explain why certain structural properties of Path-like graphs influence the computational complexity of GTSP-rd. By providing insights into the relationships between graph structures and algorithmic complexity.

In terms of its approach, the research adopts a **qualitative approach**, involving the development of mathematical formalization, modeling, and algorithms that are evaluated using the asymptotic time complexity, and classifying the algorithms in categories. Figure 1.3 illustrate the methodology applied into this work.

The tools employed in this research are primarily theoretical, including graph theory, recursion, recurrence relations, and dynamic programming for developing the algorithms. For the analysis, big-O notation, complexity theory, and formal proof systems were also utilized. Additionally, LaTeX was used for documenting the findings.

The methodological procedures of this work were defined to achieve the main and secondary objectives. The methodological steps adopted are described below:

- **Literature Review:** realize a comprehensive review of published research on the GTSP-rd, GTSP, and related problems. The objective is to identify existing algorithms, analyze their complexities, and explore the approaches employed in previous works.
- **Analysis of GTSP-rd Properties:** study of properties as scheduling, density,

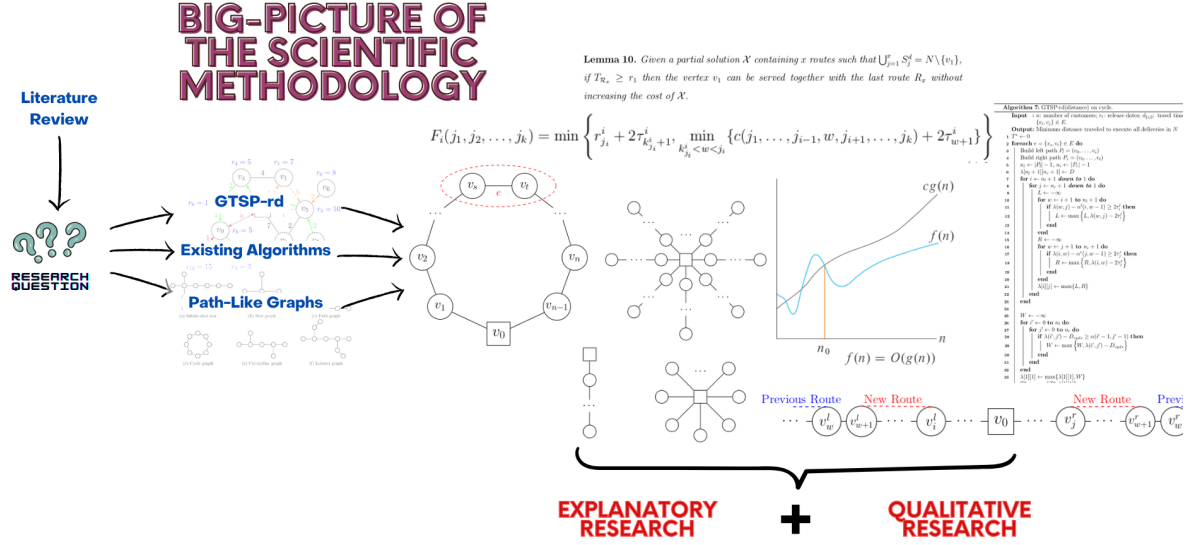


Figure 1.3: Master dissertation big-picture.

connectivity, and explore how they can impact on the hardness of GTSP-rd. It involves analyze particular cases and graph classes.

- **Analysis and Improvement of Existing Algorithms:** review existing algorithms to GTSP-rd, analyze its asymptotic complexities, approaches and limitations. Propose improvements to algorithms, providing better lower bounds to graph classes already addressed in literature.
- **Selection of Graph Classes:** explore different graph classes like path, cycle, star, subdivided star, tree, and verify if its structural properties such that connectivity, cycles and subgraphs can be used to solve the GTSP-rd. The objective is identifying graph classes which the GTSP-rd could be solved in polynomial time.
- **Proofs for the Proposed Algorithms:** provide proofs of complexity and correctness for the algorithms proposed to solve the GTSP-rd in any investigated graph class.
- **Elaboration of Articles and Participation in Scientific Events:** prepare and submit research articles about the research results. Participate in relevant scientific events, workshops, and conferences to present findings, exchange ideas with other researchers, and gather feedback to refine the work.

- **Documentation and Writing:** document all the research process, the algorithms, proofs, and analysis. Write the dissertation text, and articles to disseminate the results.

1.3 Main Contributions and Text Organization

As part of the **contributions** of this master's dissertation, we:

- Provide a correct and explicit definition of the Graphical Traveling Salesman Problem with Release Dates (GTSP-rd). This contribution includes a review of the existing literature on routing problems and works focused on specific graph classes. This review allowed us to validate the proposed definition, addressing the lack of a precise formulation in previously well-published works.
- **Improve** existing dynamic programming algorithms for GTSP-rd on **path graphs with the depot located at an extremity**:
 - an $O(n)$ algorithm for minimizing the total completion time;
 - an $O(n \log n)$ algorithm for minimizing the total distance traveled.
- Propose new algorithms for additional classes of graphs:
 - Dynamic programming algorithms for GTSP-rd on general **paths**, allowing the depot to be located at any vertex:
 - * an $O(n^2)$ algorithm for minimizing the total completion time;
 - * an $O(n^2 \log n)$ algorithm for minimizing the total distance traveled.
 - Dynamic programming algorithms for GTSP-rd on **star graphs**, particularly when the depot is located at a leaf. These results show that GTSP-rd on stars is polynomial-time solvable:
 - * an $O(n^2)$ algorithm for minimizing the total completion time;
 - * an $O(n^3)$ algorithm for minimizing the total distance traveled.
 - Dynamic programming algorithms for GTSP-rd on **cycle graphs**, showing that GTSP-rd on cycles is also polynomial-time solvable:
 - * an $O(n^3)$ algorithm for minimizing the total completion time;

* an $O(n^3)$ algorithm for minimizing the total distance traveled.

- Provide analysis and insights about **caterpillar (lobster) graphs** and **subdivided stars**, two additional path-like graphs explored in this dissertation.
- Additionally, during the development of this dissertation, we contributed both implementation and theoretical results to a colleague's master dissertation Rosas (2025), and collaborated on an experimental research paper developed within our research group Castro et al. (2025).

The remainder of this dissertation is organized as follows. Chapter 2 introduces the fundamental concepts used throughout the text, including basic definitions in algorithms and complexity analysis, fundamental notions of graph theory and graph classes, and a literature review of works addressing routing problems on specific graph classes, as well as existing results on the Graphical Traveling Salesman Problem (GTSP). Chapter 3 provides a formal definition of the Graphical Traveling Salesman Problem with Release Dates (GTSP-rd) and reviews previous work on this problem, establishing the definitions and structural properties used in Chapter 4. Chapter 4 presents the results obtained for path, star, and cycle graphs, including discussions on the implications of these results and the challenges of extending them to other path-like graph classes. Finally, Chapter 5 presents concluding remarks.

Chapter 2

Theoretical Basis

This chapter introduces key concepts used throughout the text. Section 2.1 presents elements of graph theory, including basic properties and examples of graph classes. Section 2.2 offers a literature review addressing routing problems defined on particular graph classes. Finally, Section 2.3 focuses on related works which address the GTSP.

2.1 Graph Theory Related Concepts

The definitions and notations used in this section are based on books West (2001), Bondy and Murty (2008) and Szwarcfiter (2018).

A **graph** $G = (V, E)$ consists of a finite non-empty set V and a set E of unordered pairs of distinct elements from V . The elements of V are called **vertices**, and the elements of E are called **edges** of G . Each edge $e \in E$ can be represented by the pair of vertices $e = \{v, w\}$ (or simply $e = vw$) where v and w are the vertices connected by e . In this case, the vertices v and w are the **endpoints** of the edge e and are referred to as **adjacent** (or neighbors). The edge e is said to be **incident** to both v and w . The set of neighbors of a vertex v in a graph G is denoted by $N_G(v)$. The **degree** of a vertex $v \in V$, denoted as $d_G(v)$, is defined as the number of vertices adjacent to v .

A **path** is a graph $P = (V, E)$ of the form:

$$V = \{x_0, x_1, \dots, x_k\} \quad E = \{x_0x_1, x_1x_2, \dots, x_{k-1}x_k\},$$

where the elements of V are all distinct. The vertices x_0 and x_k are connected by P and called **extremities**; the vertices $x_1, \dots, x_{k-1}$ are the **internal** vertices of P . The

number of edges of a path is its **length** and a path with length k is denoted by P^k . We often refer to a path by the natural sequence of its vertices, writing, $P = x_0x_1 \dots x_k$.

A **subgraph** of a graph G is a graph H such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$ and the assignment of endpoints to edges in H is the same as in G . We then write $H \subseteq G$ and say that G contains H .

A graph $G = (V, E)$ is called **connected** if there exists a path between every pair of vertices in G . Otherwise, G is **disconnected**. If G has a path between u and v , then the **distance** from u to v , written $d(u, v)$, is the least length of a path between u and v . The **longest path** in G is the path with maximum length in G .

2.1.1 Path-like Graphs

The main graph classes studied in this dissertation are those whose structure is primarily based on paths. Here, we refer to them as **Path-like** graphs. In the remainder of this subsection, we provide formal definitions of these graph classes.

If $P = x_0 \dots x_{k-1}$ is a path and $k \geq 3$, then the graph $C := P + x_{k-1}x_0$ is a **cycle**. The cycle *length* is the number of edges. The cycle of length k is called k -*cycle* and is denoted by C^k . A *Hamiltonian Cycle* of G is a cycle that contains all the vertices of G . Similarly, a path in G containing every vertex of G is a *Hamiltonian Path*.

A graph that has no cycles is called *acyclic*. A graph $T = (V, E)$ that is acyclic and connected is called a *tree*. If a vertex v of a tree T has degree 1, then v is a *leaf*. Otherwise, v is an *internal vertex*. The leaves of a tree T is denoted by $L(T)$.

Let P_T be a longest path in a tree T . We call T a k -*distant tree* if every vertex of T is at distance at most k from P_T . A **caterpillar** is a 1-distant tree, and the path P_T is called the *spine* of the caterpillar. A **lobster** is a 2-distant tree, the path P_T is also called the *spine* of the lobster (Bermond, 1979).

Let $r \geq 2$ be an integer. A graph $G = (V, E)$ is called r -*partite* if V can be partitioned into r classes such that each edge connects vertices from different classes: vertices within the same partition class must not be adjacent.

Given that $G = (V, E)$ is a 2-partite (*bipartite*) graph, its vertex set V can be partitioned into two subsets V_1 and V_2 such that every edge in G connects a vertex in V_1 to a vertex in V_2 . It is useful to denote the bipartite graph G as $(V_1 \cup V_2, E)$. A *complete bipartite graph* has an edge between every pair of vertices v_1, v_2 , where $v_1 \in V_1$

and $v_2 \in V_2$. By denoting $n_1 = |V_1|$ and $n_2 = |V_2|$, a complete bipartite graph is denoted by K_{n_1, n_2} . Graphs of the form $K_{1, n}$ are called **stars**. The vertex that stands alone in one of the partition classes of this $K_{1, n}$ is the *center* of the star.

A **generalized (subdivided) star** graph is a tree T that has exactly one vertex of degree greater than 2, while all other vertices have degree at most 2. The *central vertex* of T is the unique vertex v with degree $k > 2$. Its neighbors $u_1, \dots, u_k$ lie at the ends of k disjoint branches $T_1, \dots, T_k$, and each branch T_i forms a path whose endpoint is u_i (Johnson et al., 2003).

Figure 2.1 illustrates the graph classes defined in this session.

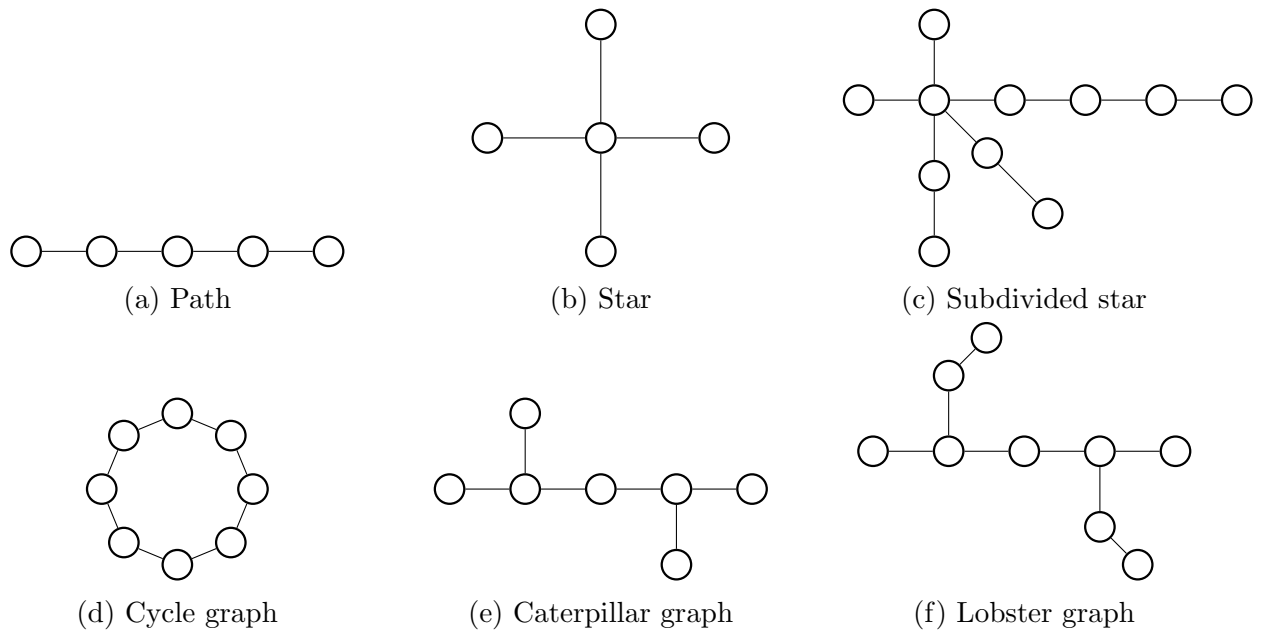


Figure 2.1: Examples of Path-like graphs.

Figure 2.2 illustrates a hierarchy of Path-like graphs, demonstrating that most of these structures can be categorized as trees.

2.1.2 Other Concepts and Graph Classes

We write $G - e$ or $G - M$ for the subgraph of G obtained by **deleting an edge** e or the set of edges M . We write $G - v$ or $G - S$ for the subgraph obtained by **deleting a vertex** v or set of vertices S and its incident edges. An **induced subgraph** is a subgraph obtained by deleting a set of vertices. We write $G[T]$ for $G - \bar{T}$, where $\bar{T} = V \setminus T$, this is the subgraph induced by T .

In graph G , **contraction** of edge $\{u, v\}$ is the replacement of u and v with a single

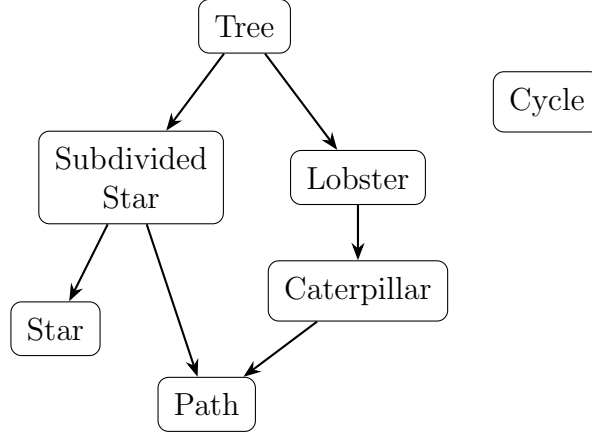


Figure 2.2: Hierarchy of path-like graph classes.

vertex whose incident edges are the edges other than $\{u, v\}$ that were incident to u or v . The resulting graph $G \cdot \{u, v\}$ has one less edge than G . A graph H is a **minor** of a graph G if a copy of H can be obtained from G by deleting and/or contracting edges of G .

The **connectivity** of G , is the minimum size of a vertex set S such that $G - S$ is disconnected or has only one vertex. A graph is **k -connected** if its connectivity is at least k . Given $S, T \subseteq V$, we write $[S, T]$ for the set of edges having one endpoint in S and the other in T . An **edge cut** is an edge set of the form $[S, \bar{S}]$, where S is a nonempty proper subset of V and $\bar{S}$ denotes $V \setminus S$.

A graph is *planar graph* if it has a drawing without crossings. Such a drawing is a *planar embedding* of G . A *plane graph* is a particular planar embedding of a planar graph. A *plane tree* is a tree equipped with such a fixed planar embedding. In a plane tree, the embedding determines a cyclic order of the leaves around the outer face. A cycle C connects the leaves of T in cyclic order if $V(C) = L(T)$ and the edges of C join consecutive leaves according to this order.

A **Halin graph** is a graph $H = T \cup C$, where T is a plane tree with at least four vertices and no vertex of degree two, and C is a cycle that connects the leaves of T in the cyclic order determined by the embedding of T .

The graph obtained by taking the union of graphs G and H with disjoint vertex sets is the **disjoint union** (or sum), written $G + H$. The **join** of graphs G and H , written $G \vee H$, is the graph obtained from the disjoint union $G + H$ by adding the edges $\{\{x, y\} : x \in V(G), y \in V(H)\}$. The **wheel** graph is $K_1 \vee C^{n-1}$.

For the following definition, we assume the possibility of the creation of multi-graphs with the Parallel extension. A **series-parallel graph** is any graph obtained

from a single edge connecting two distinct vertices by repeatedly applying the following operations:

- **Parallel extension:** replace an edge $\{u, v\}$ by two parallel edges between u and v .
- **Series extension:** replace an edge $\{u, v\}$ by a two edges $\{u, w\}, \{w, v\}$, where w is a new vertex.

2.2 Routing Problems on Graph Classes

In the literature, few works propose algorithms to solve routing problems in polynomial-time for specific graph classes. Following are the problems found and the respective works that address them.

Several works have studied the **Capacitated Vehicle Routing Problem (CVRP)** when the input graph is a tree. Chandran and Raghavan (2008) present two new integer programming formulations that explicitly exploit the tree structure, while Mathieu and Zhou (2022) provide an approximation algorithm for a variant of CVRP on trees. More recently, Mathieu and Zhou (2023) introduced a polynomial-time approximation scheme (PTAS) for the unit-demand CVRP on trees, and Becker (2018) developed a $4/3$ -approximation algorithm for the CVRP on Trees. Despite these algorithmic advances, the problem remains computationally difficult: Labbé et al. (1991) proved that CVRP is NP-hard even when restricted to trees, and Busch (1991), as cited by Archetti et al. (2011), showed that the problem is NP-hard even for simple graph classes such as stars and paths.

In Archetti et al. (2011) was considered a version of CVRP with an unlimited number of vehicles to be used, in this case, CVRP is solvable in $O(n)$ when the input is a star. Table 2.1 show the summary of results to **CVRP Unlimited and Limited Fleet** (UF and LF).

Table 2.1: Best Known results on time complexity to the CVRP (Archetti et al., 2011).

Graph Class	CVRP - LF	CVRP - UF
Path	NP-hard	NP-hard
Cycle	NP-hard	NP-hard
Star	NP-hard	$O(n)$
Tree	NP-hard	NP-hard

Another problem addressed by Archetti et al. (2011) is the **Split Delivery Vehicle Routing Problem (SDVRP)**. In this problem it is allowed to make the delivery in one vertex using more than one route splitting the demand between the vehicles that passed through this vertex. The work also consider two versions, limited and unlimited fleet. It proposes polynomial-time algorithms to path, cycle and star, and also prove NP-hardness for other classes, Table 2.2 summarize the results.

Table 2.2: Best Known results on time complexity to the SDVRP (Archetti et al., 2011).

Graph Class	SDVRP - LF	SDVRP - UF
Path	$O(n)$	$O(n)$
Cycle	$O(n^2)$	$O(n^2)$
Star	NP-hard	$O(n)$
Tree	NP-hard	NP-hard

In most of the NP-hardness proofs in the cited works, the difficulty of the problem stems from the capacity constraints imposed on the routes, which are the main source of computational hardness. The studies by Archetti et al. (2015) and Reyes et al. (2018) explore routing problems that replace capacity constraints with time restrictions, specifically addressing the **Traveling Salesman Problem with release dates (TSP-rd)** and the **Uncapacitated Vehicle Routing Problem with release dates (UVRP-rd)**. In these problems, each vertex v has an associated release date, which defines the earliest time the route serving v can depart from the depot.

For the UVRP-rd, Archetti et al. (2015) propose quadratic and linear-time algorithms for path and star graphs, respectively. In addition, Reyes et al. (2018) study a variant in which each vertex is also associated with a deadline. Given a release date r_v for a vertex v , the deadline is defined as $r_v + c$, where c is a fixed integer constant. Table 2.3

summarizes the known results for these settings and graph classes, while entries marked with “-” indicate cases for which no results are currently available in the literature.

Table 2.3: Best Known results on time complexity to the UVRP-rd. Reyes et al. (2018) address the problem with a fixed deadline while Archetti et al. (2015) address without a deadline.

Graph Class	distance	distance w/deadline
Path	$O(n^2)$	$O(n^3)$
Cycle	-	-
Star	$O(n)$	-
Tree	-	-

The next section discusses in further detail the works that address another problem without capacity and time constraints, a variation of TSP very close to GTSP-rd, the Graphical Traveling Salesman Problem (GTSP).

2.3 Graphical Traveling Salesman Problem - GTSP

Given a simple connected graph $G = (V, E)$. A **tour** on G is a sequence of nodes $T = [v_1, \dots, v_i, v_{i+1}, \dots, v_p, v_{p+1}]$ such that $\{v_i, v_{i+1}\} \in E$, $v_1 = v_{p+1}$, and each node of V appears at least once in T . Each edge $\{v_i, v_{i+1}\} \in E$ is associated with a travel time (distance), denoted by $d_{v_i, v_{i+1}}$. The total distance traveled on a tour is determined by $d_T = \sum_{i=0}^p d_{v_i, v_{i+1}}$. The **Graphical Traveling Salesman Problem (GTSP)**, consists of finding a tour on G of minimum cost.

The GTSP differs from the TSP because it does not necessarily seek for a Hamiltonian Cycle with minimum cost in G . Then, even graphs that do not contain a Hamiltonian cycle can still have a valid solution to the GTSP. Also, in sparse graphs with Hamiltonian cycles, a route passing by a vertex more than once can be an optimal solution to GTSP. For the TSP, when dealing with sparse graphs where the existence of a Hamiltonian cycle is uncertain, a common approach is to transform the graph into a complete graph $K = (V, E')$ over the same vertex set. In this transformation, each edge $\{u, v\} \in E'$ is assigned a cost equal to the shortest path between u and v in the original graph G (Hargrave and Nemhauser, 1962; Cornuéjols et al., 1985a; Sebő and Vygen, 2014; Cook, 2015).

The GTSP is NP-hard, as noted by Cornuéjols et al. (1985a). Consider a graph G , where each edge $e \in E$ has a cost $d_e = 1$. In this case, the solution to the GTSP will be equivalent to determining whether G is Hamiltonian or not, which is a known NP-complete problem. If G is Hamiltonian, the cost of solution will be $|V|$.

The Miliotis et al. (1981) were the first to consider solve the problem in the original graph. It compares two approaches, first solve the problem creating a complete graph as described above and second, propose an integer linear programming directly on input graph. The second method achieve better results in computational tests.

In 1983, the first studies began to identify non-trivial cases where the GTSP could be solved in polynomial time. Ratliff and Rosenthal (1983) address the GTSP in the context of order picking in a rectangular warehouse, where the vertices of graph represent the products and the edges correspond to the warehouse aisles. They proposed a dynamic programming algorithm that runs linearly with respect to the number of aisles. The graph representing the rectangular warehouse is a series-parallel graph. Later, Cornuéjols et al. (1985a) propose a new algorithm that extended these results, showing that the algorithm could also be applied to all series-parallel graphs with linear time complexity. The work of Cornuéjols et al. (1985a) was also the first one to call the problem as Graphical Traveling Salesman Problem.

Cornuéjols et al. (1983) demonstrates that the GTSP can be solved in polynomial time for Halin graphs. The paper presents a recursive algorithm based on shrinking a subset of vertices until the resulting graph becomes the simplest Halin graph, a wheel graph. In this graph, denoted by $W = C \cup \{v\}$, the GTSP is solved by removing the edge $k \in E(C)$ that minimizes the sum of the costs of the two edges connecting the nodes incident with k to the central vertex v (the center of the wheel). The two edges are then added to the solution, while all other edges of C are retained. The overall complexity of this procedure is $O(n)$.

A follow-up to this work is Cornuéjols et al. (1985b), which extends the results beyond Halin graphs. The paper shows that the GTSP can be solved in polynomial time for all fully reducible graph classes. Two examples of fully reducible classes are Halin graphs and prismatic graphs. A class $\mathfrak{C}$ of 3-connected graphs is considered **fully reducible** if it satisfies the following conditions:

1. If $G \in \mathfrak{C}$ has a 3-edge cut with shores S and $\overline{S}$, then both $G[S]$ and $G[\overline{S}]$ are in $\mathfrak{C}$;

2. GTSP can be solved in polynomial time for the graphs in $\mathfrak{C}$ that do not have a 3-edge cut.

Fonlupt and Naddef (1992) introduced the concept of TSP-perfect graphs, which are graphs that exclude certain specific graphs as minors (Figure 2.3). They demonstrate that the Graphical Traveling Salesman Problem (GTSP) can be solved in polynomial time on TSP-perfect graphs by utilizing the ellipsoid method (Grötschel et al., 1981).

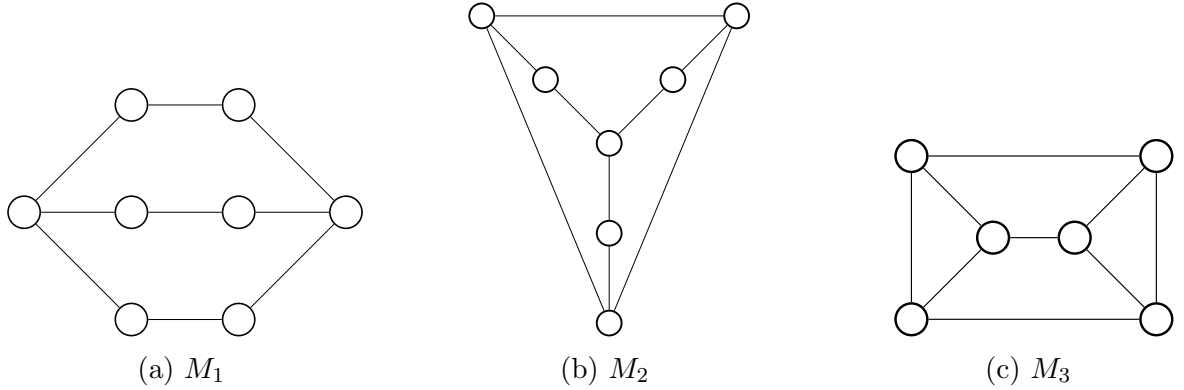


Figure 2.3: Graphs M_1 , M_2 , and M_3 defined by Fonlupt and Naddef (1992). The TSP-perfect graphs are graphs that exclude these graph as minors.

Let $G_1 = (V_1 \cup S, E_1 \cup F_1)$ and $G_2 = (V_2 \cup S, E_2 \cup F_2)$ be two graphs intersecting on a vertex set S with $|S| = r$, such that (S, F_1) and (S, F_2) are cliques. Let $G = (V, E)$ be the graph with $V = V_1 \cup V_2 \cup S$ and $E = E_1 \cup E_2$. We say that G is the r -sum of G_1 and G_2 .

Fonlupt and Nachev (1993) propose a general framework for studying polynomial-time algorithms for the GTSP on different graph classes. Their approach relies on the r -sum composition of graphs, which enables the systematic construction of larger graphs from simpler components while preserving tractability. This framework provides new insights into solving GTSP on more complex structures, demonstrating its applicability to basic graphs, series-parallel graphs, wheels, and other composite graph classes.

As outlined in this section, several solutions have been proposed to solve certain graph classes in polynomial time. The most commonly used techniques in these works involve decomposing graphs using operations such as r -sum, deletion, contraction, and edge-cutting. Along with graph decomposition, the algorithms frequently employ recursion and dynamic programming strategies. Table 2.4 summarizes the graph classes addressed in each study.

Table 2.4: Known polynomial-time complexity results for GTSP on graph classes.

Graph Class	Reference Paper	Algorithm Approach
Rectangular Warehouse Graph	Ratliff and Rosenthal (1983)	Dynamic Programming
Halin Graph	Cornuéjols et al. (1983) Cornuéjols et al. (1985b)	Recursion
Series-Parallel	Cornuéjols et al. (1985a)	Dynamic Programming
Prismatic Graph	Cornuéjols et al. (1985b)	Recursion
TSP-perfect graphs	Fonlupt and Naddef (1992)	Ellipsoid Method
r-sum composed graphs	Fonlupt and Nachev (1993)	Dynamic Programming

Chapter 3

Graphical Traveling Salesman

Problem with release dates

(GTSP-rd)

Although previous and recently published works propose solutions for specific graph classes in the TSP-rd, they define (model) the problem as a complete graph. This results in a mismatch between the problem definition and the proposed solutions. To encompass potential solutions for specific graph classes, we define the problem considering not only complete graph as input. A similar approach was taken in the creation of the Graphical TSP (Fonlupt and Nachev, 1993; Cornuéjols et al., 1985a; Carr et al., 2023).

In this chapter we define the GTSP with release dates (Section 3.1) and detail previous results from literature addressing graph classes (Section 3.2). Some notation used along the work is also defined.

3.1 Problem Definition

The Graphical Traveling Salesman Problem with release dates (GTSP-rd) can be defined as follows: Given a simple connected graph $G = (V, E)$, with $V = \{v_0, v_1, \dots, v_n\}$. The vertex v_0 denotes the depot, while the set of vertices $N = \{v_1, \dots, v_n\}$ represents the set of customers to be visited. Each edge $\{v_i, v_j\} \in E$ is associated with a travel time (based on distance), denoted by $d_{ij} \geq 0$. Additionally, a release date $r_i \geq 0$ is associated with each vertex $v_i \in N$, which represents the earliest point in time that a delivery to v_i can

depart from the depot. Therefore, a deadline $D \geq 0$ may be imposed to complete all routes.

A *route* $\mathcal{R}$ is a closed walk in G that starts and ends at the depot. Formally, $\mathcal{R}$ is the ordered sequence $\langle u_0, u_1, \dots, u_p, u_{p+1} \rangle$, for some integer $p \geq 1$, where $u_{p+1} = u_0 = v_0$, $S_{\mathcal{R}} = \{u_1, \dots, u_p\} \subseteq N$ and $\{u_0, u_1\}, \{u_k, u_{k+1}\}, \{u_p, u_0\} \in E$ for all $k \in \{1, 2, \dots, p-1\}$.

The vertices in $S_{\mathcal{R}}$ are partitioned into two subsets: $S_{\mathcal{R}}^d$, which contains the vertices where deliveries are made, referred to as *delivery vertices*, and $S_{\mathcal{R}}^t = S_{\mathcal{R}} \setminus S_{\mathcal{R}}^d$, referred to as *traversal vertices*. The total distance traveled on a route is determined by $d_{\mathcal{R}} = \sum_0^p d_{(v_k, v_{k+1})}$. The dispatch time of a route, $T_{\mathcal{R}}$, is defined as the moment the salesman departs from the depot to serve the set $S_{\mathcal{R}}^d$. The route $\mathcal{R}$ must begin only after the latest release date in S^d , ensuring $T_{\mathcal{R}} \geq \max_{v \in S^d} \{r_v\} = \alpha(\mathcal{R})$.

A *solution* $\mathcal{X}$ to GTSP-rd consists of a sequence of x routes $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_x$ containing the vertices set $S_{\mathcal{R}_1}, S_{\mathcal{R}_2}, \dots, S_{\mathcal{R}_x}$, these routes must be traversed consecutively by the salesman in order of dispatch time $T_{\mathcal{R}_1} \leq T_{\mathcal{R}_2} \leq \dots \leq T_{\mathcal{R}_x}$. A route $\mathcal{R}_j$ can leave the depot only if the previous route has already been traversed, that is, $T_{\mathcal{R}_{j-1}} + d_{\mathcal{R}_{j-1}} \leq T_{\mathcal{R}_j}$ for $j \in \{2, \dots, x\}$. A solution to GTSP-rd is feasible if all the sets $S_{\mathcal{R}_j}^d \subseteq S_{\mathcal{R}_j}$ form a partition of N and the total time completion time respects the deadline constraint, $T_{\mathcal{R}_x} + d_{\mathcal{R}_x} \leq D$. Note that the feasibility of a solution depends on the value of D . If D is too small, no feasible solution may exist.

Figure 3.1 provides an example of solution containing three routes. $\mathcal{R}_1 = \langle v_0, v_4, v_8, v_9, v_3, v_2, v_7, v_5, v_0 \rangle$ (green), $\mathcal{R}_2 = \langle v_0, v_1, v_5, v_6, v_5, v_0 \rangle$ (orange) and $\mathcal{R}_3 = \langle v_0, v_9, v_{10}, v_9, v_0 \rangle$ (red) with the delivery vertices $S_{\mathcal{R}_1}^d = \{v_4, v_8, v_9, v_3, v_2, v_7\}$, $S_{\mathcal{R}_2}^d = \{v_1, v_5, v_6\}$ and $S_{\mathcal{R}_3}^d = \{v_{10}\}$. The dispatch times could be $T_{\mathcal{R}_1} = 5$, $T_{\mathcal{R}_2} = 22$ and $T_{\mathcal{R}_3} = 43$. Route $\mathcal{R}_1$ leaves the vertex 0 in time 5 and complete at time 22 when route $\mathcal{R}_2$ can start.

Given the set of solutions, we focus on those that optimize two objective functions previously investigated in the literature Archetti et al. (2011, 2015); Reyes et al. (2018); Montero et al. (2023). The first objective function is to minimize the total distance traveled, given by $\sum_{i=1}^x d_{\mathcal{R}_i}$ (**GTSP-rd(distance)**) under the deadline constraint D . The second objective function, is to minimize the total completion time of all routes, expressed as $T_{\mathcal{R}_x} + d_{\mathcal{R}_x}$, known as the (**GTSP-rd(time)**).

When all release dates are equal, that is $r_1 = r_2 = \dots = r_n = 0$, the GTSP-rd(time) and GTSP-rd(distance) are equivalent. Furthermore, GTSP and GTSP-rd are

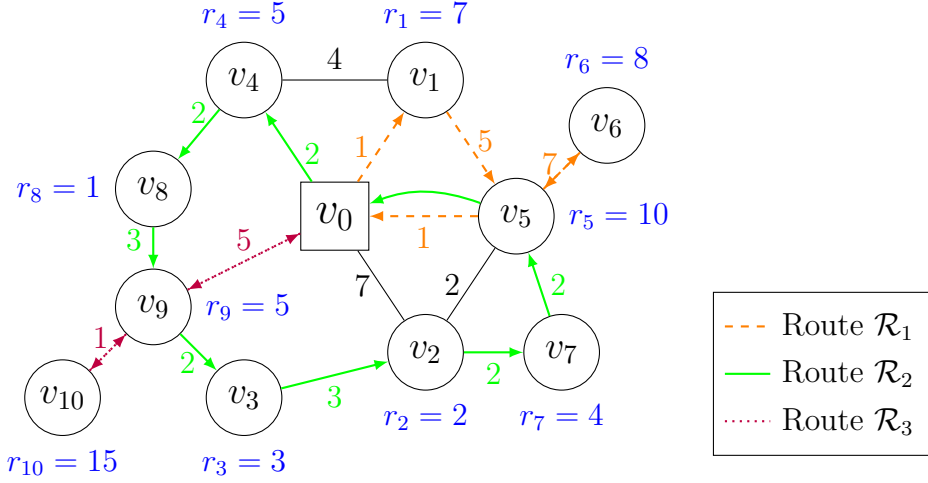


Figure 3.1: A GTSP-rd solution, containing the three routes $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3$.

also equivalent in this scenario, making GTSP a special case of GTSP-rd. Hence, the GTSP-rd problem is NP-Hard for both objective functions.

In GTSP-rd, the solutions will consist of one or more routes, where, by definition, at least the vertex v_0 is repeated in each route. Moreover, in some graph configurations a vertex must be revisited several times. As illustrated in Figure 3.2, to serve vertices v_4 and v_5 as depicted in, it is inevitable that we must traverse vertices v_1 and v_3 more than once.

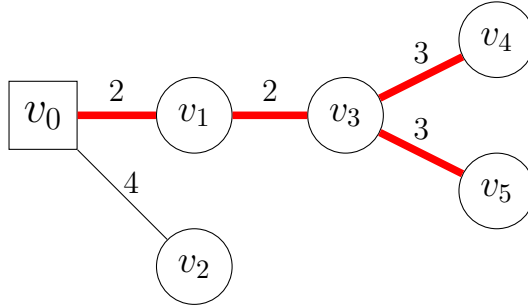


Figure 3.2: Example of a tree instance in which it is impossible to construct a route without repeating vertices. In the route $\mathcal{R} = \langle v_0, v_1, v_3, v_4, v_3, v_5, v_3, v_1, v_0 \rangle$, vertex v_3 is traversed three times.

Although we can propose some variation of a GTSP integer programming formulation, or use some metaheuristic to solve the general case of GTSP-rd, we are interested in exploring special graph classes discerning the levels of complexity and identifying potential gaps for efficient solutions exploiting the inherent graph structure. This examination serves to delineate the boundaries of tractability and the challenges posed by various

graph structures.

3.2 Related Works

The main results in the literature for GTSP-rd in graph classes are Archetti et al. (2015) and Reyes et al. (2018). These works address GTSP-rd to Stars and Paths. In the rest of the chapter we explore the problem in these graph classes and shows the results.

3.2.1 GTSP-rd on Paths

We address the GTSP-rd on paths focusing on two scenarios: (1) the special case where the depot is located at one of the extremities of the path, and (2) the general case where the depot is located anywhere along the path. For the special case, we consider a path $P = (\{v_0\} \cup N, E = \{(v_i, v_{i+1}) : 0 \leq i \leq n-1\})$. Without loss of generality, we assume that vertex v_0 is the left extremity and the remaining vertices are named $v_1, v_2, \dots, v_n$. Furthermore, this ordering can be used to construct a solution to the general case where the depot is located anywhere on the path.

Previous works Archetti et al. (2015); Reyes et al. (2018) propose algorithms to solve the GTSP-rd in paths (Table 3.1). They are related works, since the first one solves the general path case in $O(n^3)$ and the second one solves the special case where the depot is at a path extremity in $O(n^2)$. Both of them using dynamic programming as technique. In the remainder of this section we explore some properties and the two dynamic programming solutions proposed by Reyes et al. (2018) to solve the GTSP-rd for the two objective functions. A solution to path in general is provided in the Chapter 4 above the properties and solutions described here.

Table 3.1: Previous time and distance complexity results for the GTSP-rd on path graphs. All listed algorithms are based on dynamic programming.

Reference paper	Path			
	Depot at extremity		General case	
	time	distance	time	distance
Archetti et al. (2015)	-	-	$O(n^3)$	$O(n^3)$
Reyes et al. (2018)	$O(n^2)$	$O(n^2)$		

The study discussed in Reyes et al. (2018) focuses on the GTSP-rd and proposes polynomial time algorithms for scenarios where customers are arranged in a path where

the depot is located at an extremity.

For simplicity, in the path with the depot at an extremity, the distance from vertex u to vertex v_0 (the depot) is denoted as τ_u . The distance can be computed using a graph traversal algorithm (e.g, Depth First Search), where for a subpath $P' \subseteq P$ with u at the other, $\tau_u = \sum_{e \in E(P')} d_e$.

The following properties and definitions are essential for solving the problem. We adapt the notation used in Reyes et al. (2018) and Archetti et al. (2015), here.

Without loss of generality, we can assume that the vertices in N are indexed such that $r_1 \leq r_2 \leq \dots \leq r_n$. This assumption corresponds to a renumbering of the vertices based on their release dates and does not affect the generality of the result. With this assumption, Lemma 1 can also be applied:

Lemma 1. (Archetti et al., 2015) *Given two vertices v_i and v_j such that $i < j$, if $\tau_i < \tau_j$ then there exists an optimal solution such that v_i and v_j are delivered in the same route.*

Thus, from this point forward, we only need to consider instances with pairs of vertices v_i, v_j where $i < j$ and $\tau_i \geq \tau_j$ as input. When these conditions do not hold, we can simply disregard v_i and retain only v_j on the path, preserving its distance to v_0 . This transformation leads to an equivalent solution. Next, we introduce two key definitions that help characterize the structure of feasible solutions.

Definition 1. (Reyes et al., 2018) Two routes $\mathcal{R}_1$ and $\mathcal{R}_2$ with $\min\{r_i \mid v_i \in S_{\mathcal{R}_1}^d\} < \min\{r_j \mid v_j \in S_{\mathcal{R}_2}^d\}$ are said to be *non-interlacing* if and only if $\max\{r_i \mid v_i \in S_{\mathcal{R}_1}^d\} < \min\{r_j \mid v_j \in S_{\mathcal{R}_2}^d\}$.

Definition 2. (Reyes et al., 2018) A solution $\mathcal{X}^{\text{NI}}$ containing only non-interlacing routes can be characterized by the set of customers with the highest index $i_{\mathcal{R}}^{\uparrow}$ in each route, i.e., $\mathcal{X}^{\text{NI}} = \{v_{i_{\mathcal{R}_1}^{\uparrow}}, v_{i_{\mathcal{R}_2}^{\uparrow}}, \dots, v_{i_{\mathcal{R}_k}^{\uparrow}}, v_n\}$ with $1 \leq i_{\mathcal{R}_1}^{\uparrow} \leq i_{\mathcal{R}_2}^{\uparrow} \leq \dots \leq i_{\mathcal{R}_k}^{\uparrow} \leq n$, indicating that customers $S_{\mathcal{R}_1}^d = \{v_1, \dots, v_{i_{\mathcal{R}_1}^{\uparrow}}\}$ are served on the first route, orders $S_{\mathcal{R}_2}^d = \{v_{i_{\mathcal{R}_1}^{\uparrow}+1}, \dots, v_{i_{\mathcal{R}_2}^{\uparrow}}\}$ are served on the second route, and so on.

These definitions lead us to the following Lemma 2, which simplifies the problem by restricting our attention to non-interlacing routes.

Lemma 2. (Reyes et al., 2018) *Any feasible solution for a GTSP-rd on a path with*

the depot at an extremity can be transformed into a feasible solution with non-interlacing routes, without increase in the total travel time.

Lemma 2 allows us to construct solutions using only non-interlacing routes, which are formed by contiguous sequences of vertices. Lemma 3 establishes the optimal substructure property of the problem, which is essential for dynamic programming approaches.

Lemma 3. (Reyes et al., 2018) *If a solution minimizing the completion time exists, then there exists an optimal non-interlacing solution $\mathcal{X}^{NI} = \{v_{i_{\mathcal{R}_1}^\uparrow}, v_{i_{\mathcal{R}_2}^\uparrow}, \dots, v_{i_{\mathcal{R}_k}^\uparrow}, v_n\}$ with the property that each partial solution $\mathcal{X}_{[1,p]}^{NI} = \{v_{i_{\mathcal{R}_1}^\uparrow}, v_{i_{\mathcal{R}_2}^\uparrow}, \dots, v_{i_{\mathcal{R}_p}^\uparrow}\}$, delivering orders: $\{v_1, v_2, \dots, v_{i_{\mathcal{R}_p}^\uparrow}\}$, for $p = 1, \dots, k$, completes the delivery of these order subsets as early as possible.*

In the following sections, we present the dynamic programming approach proposed by Reyes et al. (2018).

A - GTSP-rd (time)

The following recurrence relation is proposed (Equation 3.1), $c(i)$ calculates the minimum completion time to serve the customers $\{v_1, v_2, \dots, v_i\}$:

$$c(i) = \begin{cases} 0, & \text{if } i = 0 \\ \min_{0 \leq j < i} \{\max\{c(j), r_i\} + 2\tau_{j+1}\}, & \text{otherwise.} \end{cases} \quad (3.1)$$

Consider the recursive step to compute $c(i)$. The delivery to customer v_i can be included in two types of routes: the first one, is along with other delivery customers $\{v_{j+1}, \dots, v_{i-1}\}$, and it is added to the partial solution serving the customers $\{v_1, \dots, v_j\}$ when $j \leq i - 2$. The second type of route involves creating a new route containing only v_i when $j = i - 1$. In both cases, the minimum completion time for the new route including v_i is the earliest possible dispatch time for this route, $\max\{c(j), r_i\}$, added to the travel time of route that would be $2\tau_{j+1}$. Figure 3.3 illustrates the idea of inclusion of customer v_i in a route.

It is not difficult to observe that this recurrence relation can be calculated in $O(n^2)$ using dynamic programming, since for each $i \in [1, n]$ we need to compute the terms $\max\{c(j), r_i\} + 2\tau_{j+1}$ for minimization where $0 \leq j < i$.

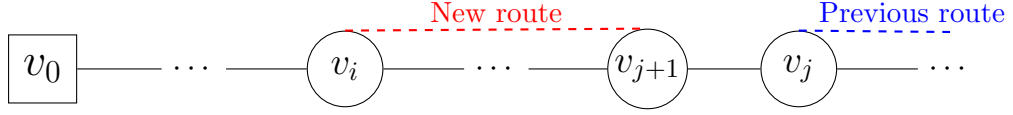


Figure 3.3: Route of customer v_i along with other vertices $v_{j+1}, \dots, v_{i-1}$.

B - GTSP-rd (distance)

When the objective is to minimize the total distance traveled by the Traveling Salesman. If we consider a version of this problem where there is no final deadline D (or if D is sufficiently large), the optimal strategy is to wait until all vertices are available before initiating a route, thus completing a single route covering all customers, with a final cost be $2\tau_1$.

When a final deadline D exists, we wait as long as possible to initiate deliveries, including all available customers in the first route. Subsequent routes begin with the first customer u not included in the previous route at the latest possible dispatch time of u , repeating until all customers are served.

Let $\lambda(i)$ denote the latest possible time at which customer v_i can be dispatched so that the customers $\{v_i, v_{i+1}, \dots, v_n\}$ is served using non-interlacing routes. By definition, $\lambda(1)$ is the latest dispatch time for v_1 such that all subsequent routes can still depart without missing the deadline. Therefore, no idle time is allowed between two consecutive routes. Otherwise, we could postpone the dispatch of v_1 even further, contradicting the definition of $\lambda(1)$. As a result, the minimum total distance traveled by the routes is $D - \lambda(1)$. This insight led Reyes et al. (2018) to formulate the following recurrence relation:

$$\lambda(i) = \begin{cases} D, & \text{if } i = n + 1 \\ \max_{j > i} \{\lambda(j) - 2\tau_i \mid \lambda(j) - 2\tau_i \geq r_{j-1}\}, & \text{otherwise} \end{cases} \quad (3.2)$$

The base case $\lambda(n + 1) = D$, represents the latest time by which all routes must be completed. As we are working with non-interlacing routes to determine the latest time to dispatch customer v_i , we attempt to incorporate it into all previously established routes, such as $\{v_i, \dots, v_{j-1}\}$. To postpone dispatch as much as possible, the latest time to dispatch this route will be $\lambda(j)$ (the latest time for the next route dispatch) minus the cost of executing this route, which is $2\tau_i$.

The feasibility condition $\lambda(j) - 2\tau_i \geq r_{j-1}$ ensures that the time of dispatch from this route is at least the greatest release date of the customers into this route, r_{j-1} .

Therefore, it is not difficult to observe that this recurrence relation can be calculated in $O(n^2)$ time complexity using dynamic programming, since for each $i \in [1, n]$, we need to compute the terms $\lambda(j) - 2\tau_i$ and choose the maximum among them, for each $j \in [i + 1, n + 1]$.

3.2.2 GTSP-rd on Stars

When the graph is a star, the works only consider that the depot is located in the center of star. It is easy to verify that the solution is composed by n routes leaving from depot, one route for each vertex, independently of its release dates. Then the distance traveled by Salesman is constant and equal to $\sum_{i \in N} d_{v_0, v_i}$. When the objective function is to minimize the distance traveled by Salesman (GTSP-rd(distance)), this is already the solution, if a feasible solution exist.

When the objective is to minimize the total time to complete all routes, we must focus on reducing the waiting time between routes. The optimal solution in this case is to visit the vertices in non-decreasing order of their release dates. Then, if the input is a star, the GTSP-rd can be solved in polynomial-time in both objective functions.

The algorithms above are proposed in Archetti et al. (2015). An interesting question to be answered is what happen if the depot are located in a leaf. This is the simplest instance (Figure 3.4) where it consider a bifurcation in the path between the depot and any other vertex. In Section 4.3, we describe algorithms to solve this specific case, showing that in any star the GTSP-rd can be solved in polynomial time.

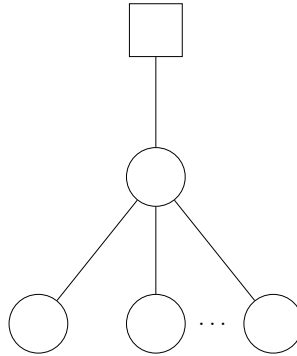


Figure 3.4: Star with depot located in a leaf.

Chapter 4

Main Results: Algorithms and Proofs

This chapter presents the main contributions of this dissertation. We begin addressing the GTSP-rd on paths, first when the depot is located at one of the extremities, in Section 4.1. For this case, we show how the recurrence relations introduced by Reyes et al. (2018) can be reformulated to obtain tighter bounds. Specifically, we present a linear-time algorithm for the GTSP-rd(time) and an $O(n \log n)$ algorithm for the GTSP-rd(distance). These results improve the best-known bounds in the literature.

Next, we extend these ideas to general paths in which the depot may be arbitrarily located along the path (Section 4.2). In this case, the presence of two directions from the depot introduces additional complexity. We demonstrate how to decompose the instance and combine left- and right-side optimal substructures to obtain an $O(n^2)$ algorithm for the GTSP-rd(time) and an $O(n^2 \log n)$ algorithm for the GTSP-rd(distance).

In Section 4.3, we turn our attention to GTSP-rd on Stars. While the special case with the depot at the center has already been solved in polynomial time, the scenario in which the depot is located at a leaf had not been addressed prior to this work. We show that both objective functions remain polynomial-time solvable in this more general configuration, presenting explicit algorithms with time complexities $O(n^2)$ and $O(n^3)$, respectively.

Finally, in Section 4.4, we examine GTSP-rd on cycles. Although cycles are structurally simple, the temporal constraints introduced by release dates add some challenge. By reducing the cycle to a collection of path instances, each one obtained by removing

a single edge, we provide polynomial-time algorithms for both objective functions with $O(n^4)$ time complexity.

Section 4.5 concludes the chapter with a discussion of these results, highlighting the structural properties that enable polynomial-time solvability and identifying graph classes, such as caterpillar graphs and subdivided stars, for which the complexity of GTSP-rd remains an open question.

4.1 GTSP-rd on Paths with Depot at Extremity

In this section, we revisit recurrence relations from Section 3.2.1 and show that they can be reformulated to exploit the linear structure of the path more effectively. By carefully analyzing the order in which vertices become available and the monotonicity properties induced by the depot being at an extremity, we obtain tighter bounds on the running time.

We begin by presenting an optimal linear-time algorithm for the GTSP-rd with the objective of minimizing the total completion time and later we address the distance-minimization variant.

4.1.1 Proposed Algorithm for GTSP-rd (time) in $O(n)$

Here we demonstrate how to modify Equation 3.1 to compute $c(n)$ in $O(n)$. For this, we need to show that the function $c(i)$ is non-decreasing.

Lemma 4. *The function $c(i)$ is non-decreasing.*

Proof. By contradiction, let's suppose that exist a $k \in N$ that $c(k) > c(k+1)$. By definition,

$$c(k) = \min_{0 \leq j < k} \{\max\{c(j), r_k\} + 2\tau_{j+1}\}, \quad c(k+1) = \min_{0 \leq j \leq k} \{\max\{c(j), r_{k+1}\} + 2\tau_{j+1}\}.$$

Since $r_{k+1} \geq r_k$, each term when $j < n$ for $c(k+1)$ satisfies $\max\{c(j), r_{k+1}\} + 2\tau_{j+1} \geq \max\{c(j), r_k\} + 2\tau_{j+1} \geq c(k)$, and when $j = k$, the term satisfies $\max\{c(k), r_{k+1}\} + 2\tau_{k+1} \geq$

$c(k)$. Hence, each candidate in the minimization for $c(k+1)$ is at least $c(k)$, so

$$c(k+1) = \min \left\{ \underbrace{\min_{0 \leq j < k} \{\max\{c(j), r_{k+1}\} + 2\tau_{j+1}\}}_{\geq c(k)}, \underbrace{\max\{c(k), r_{k+1}\} + 2\tau_{k+1}}_{\geq c(k)} \right\} \geq c(k),$$

contradicting $c(k+1) < c(k)$. Since such a k does not exist, it follows that the function $c(i)$ is non-decreasing.

□

As $c(i)$ is a non-decreasing function (Lemma 4), we can divide the general case of Equation 3.1 into two parts. For a fixed i , the first term of the sum $\max\{c(j), r_i\} + 2\tau_{j+1}$ will be r_i as long as the inequality $c(j) \leq r_i$ holds true. Otherwise, $c(j)$ will be the first term of the sum. Let's define k as the last value of j such that $c(j) \leq r_i$. Formally, $k = \max_{0 \leq j \leq n} \{j \mid c(j) \leq r_i\}$. So, the base case remain the same $c(0) = 0$, but the general case of Equation 3.1 can be written in the following way:

$$c(i) = \min_{0 \leq j < i} \begin{cases} r_i + 2\tau_{j+1}, & \text{if } j \leq k \\ c(j) + 2\tau_{j+1}, & \text{otherwise} \end{cases} \quad (4.1)$$

It means that we have two sets of routes we can add customer v_i . If $j \leq k$, then the previously constructed route has already finished, and the earliest possible dispatch time is r_i . Otherwise, the previous route will finish after the v_i release, and the earliest possible dispatch time is $c(j)$.

Equation 4.1 can be rewritten as follows:

$$c(i) = \min \left\{ \min_{0 \leq j \leq k} \{r_i + 2\tau_{j+1}\}, \min_{k < j < i} \{c(j) + 2\tau_{j+1}\} \right\} \quad (4.2)$$

Based on Equation 4.2, we aim to prove that $c(n)$ can be computed in $O(n)$ time complexity. To achieve this, we will show the following:

- The value of k (Lemma 5) and the term $\min_{k < j < i} \{c(j) + 2\tau_{j+1}\}$ (Lemma 7) can be computed in amortized $O(1)$ running time for each $i \in [1, n]$;
- The term $\min_{0 \leq j \leq k} \{r_i + 2\tau_{j+1}\}$ (Lemma 6) can be computed in worst-case $O(1)$ running time for each $i \in [1, n]$.

These results imply that all values $c(i)$, for $1 \leq i \leq n$, can be computed in overall linear time, which establishes that $c(n)$ can be obtained in $O(n)$.

Lemma 5. *The value $k = \max_{0 \leq j \leq n} \{j \mid c(j) \leq r_i\}$ can be computed in amortized $O(1)$ time, for each $i \in [1, n]$.*

Proof. We compute the value of k for each $i \in [1, n]$ incrementally. Let $k = 0$ initially. For each $i \in [1, n]$, we increase k while $c(k+1) \leq r_i$. Since $c(j)$ and r_i are non-decreasing sequences (Lemma 4), we do not need to re-check values of $j < k$, as $c(j) \leq c(k) \leq r_{i-1} \leq r_i$. Therefore, the value of k only increases throughout the process.

Over all n iterations, k can be incremented at most n times. Thus, the total time to compute all values of k is $O(n)$, which implies an amortized time of $O(1)$ per iteration. $\square$

Once we have calculated the k , Lemma 6 shows that $\min_{0 \leq j \leq k} \{r_i + 2\tau_{j+1}\}$ can be transformed into a constant sum.

Lemma 6. *The value $\min_{0 \leq j \leq k} \{r_i + 2\tau_{j+1}\}$ can be computed in $O(1)$ time, for each $i \in [1, n]$.*

Proof. As shown in Lemma 1, we work only with instances where for some pair i, j , $i < j \Rightarrow \tau_i \geq \tau_j$. It ensures that the distance array τ is ordered in non-increasing way. Hence, $\min_{0 \leq j \leq k} \{r_i + 2\tau_{j+1}\} = r_i + 2\tau_{k+1}$ and it can be computed in $O(1)$ time complexity. $\square$

This occurs because all customers v_j where $0 \leq j \leq k$ have completion times $c(j)$ smaller than r_i . Then it is optimal to include v_i into the route with customers $v_{k+1}, \dots, v_{i-1}$. We can rewrite recurrence of Equation 4.2 as:

$$c(i) = \min \left\{ r_i + 2\tau_{k+1}, \min_{k < j < i} \{c(j) + 2\tau_{j+1}\} \right\} \quad (4.3)$$

Equation 4.3 is now expressed as a minimization of a sum, combined with a larger minimization over the interval $[k+1, i-1]$. Lemma 7 shows how to compute this efficiently.

Lemma 7. *The value $\min_{k < j < i} \{c(j) + 2\tau_{j+1}\}$ can be computed in amortized $O(1)$ time, for each $i \in [1, n]$.*

Proof. We show that the expression $\min_{k < j < i} \{c(j) + 2\tau_{j+1}\}$ can be computed for all $i \in [1, n]$ in overall $O(n)$ time, and hence in amortized constant time per iteration.

Let $a_j = c(j) + 2\tau_{j+1}$, and observe that the objective is to find the minimum value of a_j such that $k < j < i$. To accomplish this efficiently, we use a data structure known as a

minqueue — a queue that supports the standard operations **enqueue(x)** and **dequeue()**, as well as a **find_min()** operation, which returns the minimum element currently in the queue. This structure can be implemented with amortized $O(1)$ time per operation Brass (2008) or worst-case $O(1)$ time Sundar (1989).

As established in Lemma 5, the value of k can be computed in $O(1)$ time for each i . Thus, for each new index i , we:

- Insert a_{i-1} into the **minqueue** using **enqueue(a_{i-1})**;
- Remove from the front all values a_j for $j \in [k' + 1, k]$, where k' is the value of k from the previous iteration.

Since each a_j is inserted and removed at most once, the total number of insertions and deletions is $O(n)$. Therefore, the overall complexity for computing the term $\min_{k < j < i} \{c(j) + 2\tau_{j+1}\}$ across all values of $i \in [1, n]$ is $O(n)$, which implies amortized $O(1)$ running time per i . $\square$

Algorithm 1 computes $c(n)$ by implementing the procedures described in Lemmas 5, 6, and 7.

Algorithm 1: GTSP-rd(time) with depot at the extremity

Input : n : number of vertices, r_i : release dates, τ_i : travel time from v_i to depot, $i \in [1, n]$.

Output: $c[i]$: completion time for each vertex v_i , $i \in [1, n]$.

```

1  $Q \leftarrow \text{minqueue}()$ 
2  $k \leftarrow 0$ 
3  $c[0] \leftarrow 0$ 
4  $c[i] \leftarrow \infty \quad \forall i \in [1, n]$ 
5
6 for  $i \leftarrow 1$  to  $n$  do
7   while  $k < n$  and  $c(k + 1) \leq r_i$  do
8      $k \leftarrow k + 1$ 
9      $Q.\text{dequeue}(a_k)$ 
10  end
11
12   $\min_{ki} \leftarrow Q.\text{find\_min}()$ 
13   $c[i] \leftarrow \min(r_i + 2\tau_{k+1}, \min_{ki})$ 
14   $a_i \leftarrow c[i] + 2\tau_{i+1}$ 
15   $Q.\text{enqueue}(a_{i-1})$ 
16 end
```

Theorem 1. *The time complexity to compute $c(n)$ using Algorithm 1 is $O(n)$.*

Proof. Lemma 5 shows that k can be computed in amortized $O(1)$ time. Lemma 6 shows that the first term can be computed in worst-case $O(1)$ time for each i , while Lemma 7 shows that the second term can also be computed in amortized $O(1)$ time for each i .

Therefore, each $c(i)$ can be computed in amortized $O(1)$ time, and computing all values $c(1), c(2), \dots, c(n)$ takes overall $O(n)$ time. $\square$

4.1.2 Proposed Algorithm for GTSP-rd (distance) in $O(n \log n)$

In this section we describe how to modify Equation 3.2 to calculate $\lambda(1)$ in $O(n \log n)$ running time. First, we can rewrite the feasibility inequality $\lambda(j) - 2\tau_i \geq r_{j-1}$ as $\lambda(j) - r_{j-1} \geq 2\tau_i$. Hence, the general case of Equation 3.2 can be rewritten as:

$$\lambda(i) = \max_{j>i} \{\lambda(j) \mid \lambda(j) - r_{j-1} \geq 2\tau_i\} - 2\tau_i \quad (4.4)$$

The base case remains $\lambda(n+1) = D$. The value of Equation 4.4 can be computed efficiently by maintaining appropriate auxiliary heap structures (Thorup, 2000; Williams, 1964; Costa et al., 2025). Algorithm 2 provides an implementation that realizes Equation 4.4.

Theorem 2 shows how to calculate $\lambda(1)$ using Equation 4.4 in $O(n \log n)$.

Theorem 2. *The time complexity to compute $\lambda(1)$ using Algorithm 2 is $O(n \log n)$.*

Proof. We process the values of $\lambda(i)$ in decreasing order of i , starting from $i = n$ down to $i = 1$. To efficiently compute each $\lambda(i)$, we maintain two heaps:

- A max-heap H_1 , which stores values of $\lambda(j)$ such that the feasibility constraint $\lambda(j) - r_{j-1} \geq 2\tau_i$ holds.
- A min-heap H_2 , which stores auxiliary values $a_j = \lambda(j) - r_{j-1}$, used to detect and remove infeasible entries from H_1 .

For each index i , we perform the following steps:

1. While the minimum element a_k in H_2 violates the constraint, remove a_k from H_2 and remove the corresponding $\lambda(k)$ from H_1 .

Algorithm 2: GTSP-rd(distance) on path with depot at the extremity

Input : n : number of vertices, r_i : release dates, τ_i : travel time from v_i to depot, $i \in [1, n]$.
Output: $\lambda[i]$: latest time to dispatch customer v_i in order to serve customers $\{v_i, \dots, v_n\}$ with non-interlacing routes.

```

1  $H_1 \leftarrow \text{maxHeap}()$  ,  $H_2 \leftarrow \text{minHeap}()$ 
2  $\lambda[n+1] \leftarrow D$ 
3  $H_1.\text{insert}(\lambda[n+1])$  ,  $H_2.\text{insert}(\lambda[n+1] - r_n)$ 
4
5 for  $i \leftarrow n$  down to 1 do
6   while  $H_2.\text{find\_min}() < 2\tau_i$  do
7      $k \leftarrow H_2.\text{minIndex}()$ 
8      $H_1.\text{remove}(\lambda[k])$ 
9      $H_2.\text{remove}(H_2.\text{find\_min}())$ 
10  end
11
12   $\lambda[i] \leftarrow H_1.\text{find\_max}() - 2\tau_i$ 
13   $H_1.\text{insert}(\lambda[i])$ 
14   $H_2.\text{insert}(\lambda[i] - r_{i-1})$ 
15 end
```

2. If H_1 is non-empty, let $\lambda_{\max}$ be its maximum element in H_1 . Then, set $\lambda(i) = \lambda_{\max} - 2\tau_i$.
3. Insert $\lambda(i)$ into H_1 , and insert $a_i = \lambda(i) - r_{i-1}$ into H_2 .

The correctness of this procedure follows from the invariant that H_1 always contains exactly the feasible $\lambda(j)$ values satisfying the inequality in Equation 4.4. This is ensured by the removal step using H_2 . Moreover, the monotonicity of τ_i (i.e., $\tau_i \geq \tau_{i+1}$) guarantees that once a value becomes infeasible, it will remain so for all future iterations and can be safely removed.

For each $i \in [1, n]$, there is exactly one insertion into both H_1 and H_2 . Additionally, each element is removed at most once. Using binary heaps Williams (1964), each insertion or removal takes $O(\log n)$ time, leading to a total time complexity of $O(n \log n)$. $\square$

Note that the `minqueue` employed in Lemma 7 is not suitable here, as insertions and deletions in the present setting do not occur in the same order, and thus the structure cannot rely on a first-in, first-out policy. The time complexity depends on the choice of the auxiliary heap used Brodal (2013). If the values involved are integers within a bounded range, we can achieve a better upper bound using the heap proposed by Thorup (2000),

the time for `insert` and `remove` operations drops to $O(\log \log n)$, and `find_max` becomes $O(1)$. Consequently, the total time complexity to compute $\lambda(1)$ reduces to $O(n \log \log n)$. A drawback of using the Thorup Heap is the space cost, which is $O(n2^{\epsilon\omega})$, where ϵ is any positive constant and ω is the number of bits used to represent the highest number in the heap. Also, there is a randomized implementation giving $O(\log \log n)$ expected time and $O(n)$ space.

4.2 GTSP-rd on Paths with Depot located anywhere

In the general case where the depot can be located anywhere along the path, not just at the extremities, if we remove the depot from the path it divides the original path into two disconnected paths. Without loss of generality, we denote the set of vertices for these two paths as the *left vertices* N_l and the *right vertices* N_r , such that $N = N_l \cup N_r$. Additionally, let $n_l = |N_l|$, $n_r = |N_r|$, and $n_r + n_l = n$.

Observe that any route on a path P that serves customers from both N_r and N_l can be decomposed into two disjoint routes, each serving customers from only one side, without increasing the total cost. Hence, there always exists an optimal solution in which every route contains customers exclusively from a single side. The same reasoning applies when multiple depot-rooted paths are present. Since each side is treated independently in the construction of the routes, Lemma 1 and Lemmas 2–3 continue to hold for each side separately.

We use the following notation in this section. To the vertices in N_l , release dates and distance are denoted by r_i^l and τ_i^l , respectively. The same is valid to vertices in N_r , which are denoted as r_i^r and τ_i^r . Without loss of generality we relabel the indices of vertices of N_l and N_r to $\{v_1^l, v_2^l, \dots, v_{n_l}^l\}$ and $\{v_1^r, v_2^r, \dots, v_{n_r}^r\}$ in such way that $r_i^l \leq r_{i+1}^l$ for each $i \in [1, n_l]$ and $r_i^r \leq r_{i+1}^r$ for each $i \in [1, n_r]$. Based on Lemma 1, to the left (right) side, $i < j \Rightarrow \tau_i^l \geq \tau_j^l$ ($\tau_i^r \geq \tau_j^r$). Then we have an instance like Figure 4.1.

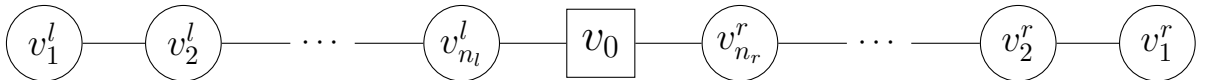


Figure 4.1: A general Path instance with the depot in an arbitrary vertex.

4.2.1 Algorithm $O(n^2)$ for GTSP-rd (time)

Based on Lemma 2, we can proceed with our solution construction, employing only non-interlacing routes within the sets N_l and N_r independently. Extending the recurrence of Equation 4.3, we define $c(i, j)$ as the minimum completion time to serve the customers $\{v_1^l, \dots, v_i^l\} \subseteq N_l$ and $\{v_1^r, \dots, v_j^r\} \subseteq N_r$. To compute $c(i, j)$, we select the optimal choice between incorporating customer v_j^r into a non-interlacing route to the right of depot $R(i, j)$ or including customer v_i^l into a non-interlacing route to the left of depot $L(i, j)$.

$$c(i, j) = \begin{cases} 0, & \text{if } i = j = 0 \\ \min\{L(i, j), R(i, j)\}, & \text{otherwise} \end{cases} \quad (4.5)$$

As the recurrence relations $L(i, j)$ and $R(i, j)$ for each i, j with $i \in [0, n_l]$ and $j \in [0, n_r]$ calculate the best choice for each side only containing customers of this side, they are very similar to recurrence relation of Equation 4.3. Function $L(i, j)$ includes a variable analogous to k , in Equation 4.3. The goal is to mark the set of customers that can or can not depart at r_i^l . But here, for each i we have n_r possible values for j then we define $k_j^l = \max_{0 \leq w \leq n_l} \{w \mid c(w, j) \leq r_i^l\}$ for each $j \in [0, n_r]$. Similarly, for $R(i, j)$, we define $k_i^r = \max_{0 \leq w \leq n_r} \{w \mid c(i, w) \leq r_j^r\}$ for each $i \in [0, n_l]$. Functions L and R are defined below:

$$L(i, j) = \min \left\{ r_i^l + 2\tau_{k_j^l+1}^l, \min_{k_j^l < w < i} \{c(w, j) + 2\tau_{w+1}^l\} \right\} \quad (4.6)$$

$$R(i, j) = \min \left\{ r_j^r + 2\tau_{k_i^r+1}^r, \min_{k_i^r < w < j} \{c(i, w) + 2\tau_{w+1}^r\} \right\} \quad (4.7)$$

On the left, customer v_i^l can be integrated into a route along with customers $v_{k_j^l+1}^l, \dots, v_{i-1}^l$ where k_j^l represents the closest customer on the left of depot such that the minimum completion time is less than r_i^l . The cost in this case is $r_i^l + 2\tau_{k_j^l+1}^l$. Customer v_i^l can also be incorporated into a route alongside customers $v_{w+1}^l, \dots, v_{i-1}^l$ when $w \leq i-2$, or it can form a new route comprising only itself when $w = i-1$. In both cases, the cost is determined by the shortest possible dispatch time for this route, denoted as $c(w, j)$, added to the travel cost of $2\tau_{w+1}^l$. From all these possible ways to add v_i in a route, we chose the one that return the minimum cost. Similarly, the same principle applies to customers on the right, including customer v_j^r . Figure 4.2 illustrates the idea of Equation 4.5.

To solve the recurrence relation using dynamic programming, we evaluate the

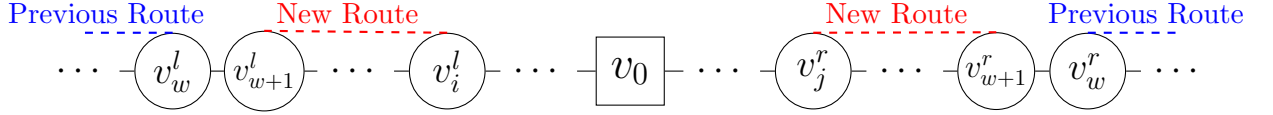


Figure 4.2: Equation 4.5 represents the choice of creating a new non-interlacing route for each side, containing only the customers on that side.

function $c(i, j)$ for all $n_l \cdot n_r$ possible states. As shown in Lemmas 8 and 9, each evaluation requires amortized constant time. Therefore, the value of $c(n_l, n_r)$ can be computed in $O(n^2)$ total time complexity.

Lemma 8. *The number of subproblems required to compute $c(n_l, n_r)$ in Equation 4.5 is $O(n^2)$.*

Proof. The recurrence in Equation 4.5 defines the function $c(i, j)$ over pairs $(i, j) \in [0, n_l] \times [0, n_r]$. Each such pair corresponds to a distinct subproblem.

Although the recurrence includes nested minimization over ranges in $L(i, j)$ and $R(i, j)$ (Equations 4.6 and 4.7), these refer only to values of $c(w, j)$ and $c(i, w)$ with strictly smaller indices. Thus, they do not introduce new subproblems beyond the rectangular domain. Therefore, the total number of subproblems that need to be computed to evaluate $c(n_l, n_r)$ is at most $(n_l + 1)(n_r + 1) = O(n^2)$. $\square$

Lemma 9. *Given Equation 4.5, the value of $c(i, j)$ can be computed in amortized $O(1)$ time for each $(i, j) \in [0, n_l] \times [0, n_r]$.*

Proof. The computation of $c(i, j)$ involves evaluating the functions $L(i, j)$ and $R(i, j)$, each defined as the minimum between a sum of two terms and a minimum over an interval. Compute the sum of two known variables is $O(1)$, we show that others components can be evaluated in amortized constant time.

- Computing k_j^l and k_i^r in amortized $O(1)$ time:

Consider a fixed $j \in N_r$. We describe how to compute k_j^l for each $i \in [1, n_l]$. Let $k_j^{l'}$ denote the value of k_j^l from the previous iteration (i.e., computed for $i - 1$). For the initial case $i = 0$, we set $k_j^l = 0$. Then, for each $i \in [1, n_l]$, we incrementally update k_j^l by iterating over w starting from $k_j^{l'}$, increasing w while the condition $c(w, j) \leq r_i^l$ holds. The updated value of k_j^l is the last w for which the inequality remains true.

Since each w is used at most once, computing all k_j^l takes $O(n_l)$ time, implying $O(1)$ amortized per value. A symmetric argument applies to the computation of each k_i^r

for a fixed $i \in N_l$.

- The term $\min_{k_j^l < w < i} \{c(w, j) + 2\tau_{w+1}^l\}$ can be computed in amortized $O(1)$ time:

We employ the **minqueue** data structure of Sundar (1989), as previously discussed in Lemma 7. Since the minimization is performed over w only, we maintain a separate **minqueue** for each $j \in N_l$ to efficiently compute the minimum, given that $c(w, j)$ varies with j .

Define $a_{ij}^l = c(i, j) + 2\tau_{i+1}^l$. For a fixed j , our goal is to compute $\min\{a_{wj}^l \mid w \in [k_j^l + 1, i - 1]\}$. For each $j \in N_r$, we maintain a **minqueue** of cost $O(n_l)$, analogous to the approach in Lemma 7. Similarly, to compute the R function, we maintain one queue per $i \in N_l$ with cost $O(n_r)$.

This results in n_r queues of cost $O(n_l)$ and n_l queues of cost $O(n_r)$, totaling $O(n_l \cdot n_r)$ overall. Hence, the amortized cost per computation is $O(1)$. $\square$

Algorithm 3 presents the dynamic programming procedure for solving Equation 4.5, employing the operations described in Lemma 9. For each $j \in [1, n_r]$, we maintain a minqueue QL_j , which is used to compute the values of the function L . Figure 4.3 illustrates the necessity of maintaining multiple distinct minqueues. Suppose the algorithm is currently computing $L(i, j)$. The recurrence requires taking a minimum over the interval of indices $[k_j^l + 1, i - 1]$ in column j . To support this minimization efficiently, all relevant values are maintained in the minqueue QL_j . When the algorithm proceeds to compute $L(i, j')$ for some $j' \neq j$, the relevant values come from a different column and therefore require a separate minqueue $QL_{j'}$. This approach is efficient because, when computing $L(i + 1, j)$, the corresponding data are already stored in QL_j . An analogous construction is applied to maintain the minqueues QR_i for $i \in [1, n_l]$, used to compute the values of R in the corresponding recurrence.

Theorem 3. *The time complexity to compute $c(n_l, n_r)$ using Algorithm 3 is $O(n^2)$.*

Proof. Following Lemma 8, the total number of subproblems required to compute $c(n_l, n_r)$ is $O(n^2)$. Moreover, Lemma 9 establishes that each subproblem can be solved in amortized constant time, $O(1)$. Since both statements hold, we conclude that Equation 4.5 can be computed via dynamic programming Algorithm 3 in $O(n^2)$. $\square$

Algorithm 3: GTSP-rd(time) on path using MinQueue

Input : n_l : number of left vertices, n_r : number of right vertices, r : release dates, τ : travel time from depot.

Output: $c[i, j]$: the minimum completion time to serve the customers $\{v_1^l, \dots, v_i^l\} \subseteq N_l$ and $\{v_1^r, \dots, v_j^r\} \subseteq N_r$

```
1  $c[i, j] \leftarrow \infty \quad \forall i \in [0, n_l], \forall j \in [0, n_r]$ 
2  $k_j^l \leftarrow 0 \quad \forall j \in [0, n_r]$ 
3  $k_i^r \leftarrow 0 \quad \forall i \in [0, n_l]$ 
4
5  $QL_j \leftarrow \text{minqueue}() \quad \forall j \in [0, n_r]$ 
6  $QR_i \leftarrow \text{minqueue}() \quad \forall i \in [0, n_l]$ 
7
8 for  $i \leftarrow 0$  to  $n_l$  do
9   for  $j \leftarrow 0$  to  $n_r$  do
10    if  $i = 0$  and  $j = 0$  then
11       $c[i, j] \leftarrow 0$ 
12    end
13
14    while  $c[k_j^l, j] \leq r_i^l$  do
15       $QL_j.\text{remove}(a_{k_j^l j}^l)$ 
16       $k_j^l \leftarrow k_j^l + 1$ 
17    end
18
19    while  $c[i, k_i^r] \leq r_j^r$  do
20       $QR_i.\text{remove}(a_{ik_i^r}^r)$ 
21       $k_i^r \leftarrow k_i^r + 1$ 
22    end
23
24     $L \leftarrow \min(r_i^l + 2\tau_{k_j^l}^l, QL_j.\text{find\_min}())$ 
25     $R \leftarrow \min(r_j^r + 2\tau_{k_i^r}^r, QR_i.\text{find\_min}())$ 
26     $c[i, j] \leftarrow \min(L, R)$ 
27
28     $a_{ij}^l \leftarrow c[i, j] + 2\tau_{i+1}^l$ 
29     $QL_j.\text{insert}(a_{ij}^l)$ 
30     $a_{ij}^r \leftarrow c[i, j] + 2\tau_{j+1}^r$ 
31     $QR_i.\text{insert}(a_{ij}^r)$ 
32  end
33 end
```

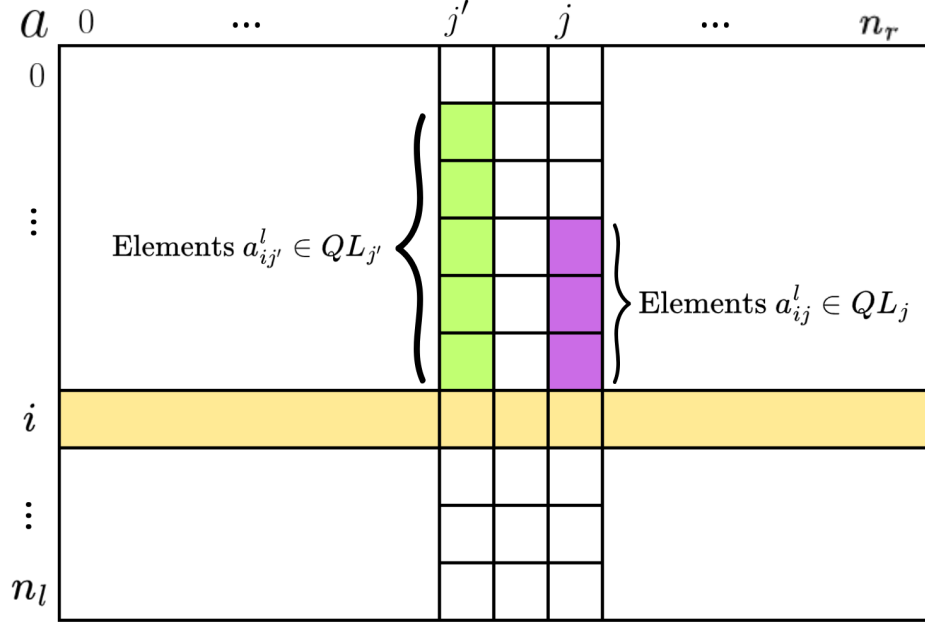


Figure 4.3: Illustration of the use of multiple `minqueue` data structures in the dynamic programming computation of $L(i, j)$. Each column j maintains its own minqueue QL_j , which stores the values required to efficiently compute the minimum over the interval $[k_j^l + 1, i - 1]$ in the corresponding recurrence. Separate minqueues are necessary because the relevant values depend on the column being processed.

4.2.2 Algorithm $O(n^2 \log n)$ for GTSP-rd (distance)

As in GTSP-rd (time), where the optimal solution consists of routes containing customers from only one side of the depot, we define a recurrence relation similar to Equation 4.5.

Given that $\lambda(i, j)$ represents the latest time to dispatch a route that serve $i \in N_l$ or $j \in N_r$, and is possible to serve the customers $\{v_i^l, \dots, v_{n_l}^l\} \in N_l$ and $\{v_j^r, \dots, v_{n_r}^r\} \in N_r$. To compute $\lambda(i, j)$, we select the optimal choice between incorporating customer v_j^r into a non-interlacing route to the right of depot $R(i, j)$ or including customer v_i^l in a non-interlacing route to the left of depot $L(i, j)$. This is expressed by the following recurrence:

$$\lambda(i, j) = \begin{cases} D, & \text{if } i = n_l + 1, j = n_r + 1 \\ \max\{L(i, j), R(i, j)\}, & \text{otherwise} \end{cases} \quad (4.8)$$

Functions L and R closely resemble Equation 4.4. In L , we examine all feasible non-interlacing routes that could involve customer v_i^l as the farthest customer from the depot, similarly to R .

$$L(i, j) = \max_{w > i} \{\lambda(w, j) \mid \lambda(w, j) - r_{w-1}^l \geq 2\tau_i^l\} - 2\tau_i^l \quad (4.9)$$

$$R(i, j) = \max_{w > j} \{\lambda(i, w) \mid \lambda(i, w) - r_{w-1}^r \geq 2\tau_j^r\} - 2\tau_j^r \quad (4.10)$$

Just like in Algorithm 2 presented to compute Equation 4.4 in $O(n \log n)$, we utilize two heaps to calculate the functions L and R in $O(\log n)$ time at each iteration.

To compute the function L , we employ n_r min-heaps storing $\lambda(w, j) - r_{w-1}^l$ and n_r max-heaps storing $\lambda(w, j)$, one for each $j \in [0, n_r]$. These heaps perform the same operations outlined in the proof of Theorem 2, independently for each j . A similar process is used to compute R .

The same argument presented in Section 3.2.1, can be used to show that the total distance traveled by the Salesman is $D - \lambda(1, 1)$.

Theorem 4. *The time complexity to compute $\lambda(1, 1)$, given by Equation 4.8, is $O(n^2 \log n)$.*

Proof. Each state of L and R can be computed using binary heaps Williams (1964) with time complexity of $O(\log n)$. Since there are $n_r \cdot n_l$ states, the overall time complexity is $O(n^2 \log n)$. A similar argument to the one used in the proof of Theorem 3 applies here, but is omitted for brevity. $\square$

As discussed in Section 4.1.2, the overall complexity depends on the choice of auxiliary heap. If we use Thorup's heap Thorup (2000), which supports integer keys on the RAM model, the total complexity improves to $O(n^2 \log \log n)$.

4.3 GTSP-rd on Stars

Let $K_{s,t} = (V, S \cup T)$ be a star with $n + 1$ vertices, where $|S| = s = 1$ and $|T| = t = n$. If $S = \{v_0\}$, then the depot is located at the center of the star, and all edges are of the form $\{v_0, u\}$ for $u \in N$. In this case, the GTSP-rd can be solved in $O(n \log n)$ time (or in $O(n)$ time if the customers are already given in non-decreasing order of release dates). This solution, originally presented by Archetti et al. (2015), consists of serving the customers in non-decreasing order of their release dates, producing a route for each customer. For GTSP-rd(time), this strategy yields the minimum completion time C^* .

For GTSP-rd(distance) with a deadline D , if a feasible solution exists, its optimal cost is $2 \sum_{e \in E} d_e$. Feasibility can be checked by verifying whether $C^* \leq D$.

We now consider the case in which the depot is located on a leaf of the star (Figure 3.4). In this setting, it is not known whether a polynomial-time algorithm exists. The goal of this section is to present a polynomial-time algorithm for both objective functions when the depot lies on a leaf, thereby solving the problem for the entire class of star graphs.

Let G be such a star graph with $n + 1$ vertices. Without loss of generality, assume that the center is vertex v_n . Furthermore, we index the remaining vertices $v_1, \dots, v_{n-1}$ in non-decreasing order of release dates, so that $r_i \leq r_{i+1}$ for all $i \in \{1, \dots, n-2\}$. The ordering is illustrated in Figure 4.4.

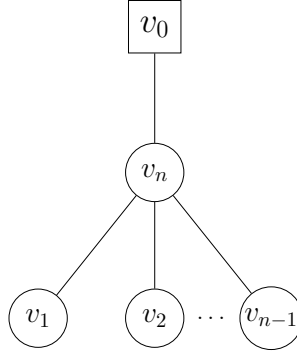


Figure 4.4: Star graph with the depot at a leaf. The center is v_n , and the remaining vertices $v_1, \dots, v_{n-1}$ are sorted so that $r_1 \leq r_2 \leq \dots \leq r_{n-1}$.

Any route $\mathcal{R} = \langle u_0, u_1, \dots, u_s, u_{s+1} \rangle$ in G must traverse the edge $\{v_0, v_n\}$ exactly twice, once on departure and once on return, since this edge is the unique connection between the depot and the remainder of the graph. Moreover, visiting each customer in S^d (except possibly v_n) requires moving back and forth between a leaf and the center (v_n). Hence, every route has the structure $\langle v_0, v_n, w_1, v_n, w_2, \dots, v_n, w_{|S^d|}, v_n, v_0 \rangle$, where $w_i \in S^d$. Consequently, the total length of route $\mathcal{R}$ is

$$d_{\mathcal{R}} = 2d_{0n} + \sum_{u \in S^d} 2d_{un}. \quad (4.11)$$

4.3.1 Algorithm $O(n^2)$ for GTSP-rd(time)

The completion time $c(\mathcal{R})$ of a route $\mathcal{R}$ is the dispatch time $\mathcal{T}_{\mathcal{R}}$ added by the time to traverse the route, $d_{\mathcal{R}}$:

$$c(\mathcal{R}) = \mathcal{T}_{\mathcal{R}} + d_{\mathcal{R}} \quad (4.12)$$

Given a solution $\mathcal{X}$ containing x routes, the cost of $\mathcal{X}$ is defined by the completion time of the last route, $c(\mathcal{R}_x)$.

Lemma 10 shows the existence of an optimal solution in which the central vertex is served in the last route. This observation allows us to restrict our attention to the behavior of the leaf customers.

Lemma 10. *There exists an optimal solution $\mathcal{X}$ with x routes that v_n is served in the last route $\mathcal{R}_x$.*

Proof. Let $\mathcal{X}^*$ be an optimal solution in which v_n is served in a route $\mathcal{R}_j$ with $j < x$. We construct another solution by moving v_n from $\mathcal{R}_j$ to the last route $\mathcal{R}_x$. We have $T_{\mathcal{R}_x} \geq T_{\mathcal{R}_j} \geq r_n$, which ensures that inserting v_n into $\mathcal{R}_x$ preserves feasibility with respect to release dates. Let $\mathcal{R}'_j$ and $\mathcal{R}'_x$ be the routes obtained after removing v_n from $\mathcal{R}_j$ and inserting it into $\mathcal{R}_x$, respectively. Removing v_n from $\mathcal{R}_j$ can only decrease its completion time, hence $c(\mathcal{R}'_j) \leq c(\mathcal{R}_j)$. The routes $\mathcal{R}_i$ with $j < i < x$ remain unchanged, and therefore their completion times do not increase. Finally, inserting v_n into $\mathcal{R}_x$ does not increase its completion time, since v_n can be served without delaying the service of last route, as $T_{\mathcal{R}_x} \geq r_n$ and v_n is in the path to any other vertex in $\mathcal{R}_x$. Thus, the total cost of the solution does not increase. Since $\mathcal{X}^*$ is optimal, the modified solution is also optimal and serves v_n in the last route. $\square$

Lemma 11 characterizes an strategy for serving the leaf customers by exploiting the notion of non-interlacing routes introduced in Section 3.2.1. In particular, it shows that any solution can be transformed into one that satisfy this structural property without increase the cost.

Lemma 11. *Any solution $\mathcal{X}$ with v_n into last route can be transformed in a solution $\mathcal{X}^{NI}$ containing only non-interlacing routes without increase the solution cost.*

Proof. Let $\mathcal{R}_a$ and $\mathcal{R}_b$ be interlacing routes in $\mathcal{X}$ such that $S_{\mathcal{R}_a}^d \supseteq \{v_i, v_k\}$, $S_{\mathcal{R}_b}^d \supseteq \{v_j\}$ and $i < j < k$. Assume further that $i = \min \{y \mid v_y \in S_{\mathcal{R}_a}^d\}$, $j = \min \{y \mid v_y \in S_{\mathcal{R}_b}^d\}$ and that $k = \min \{y \mid v_y \in S_{\mathcal{R}_a}^d, y > j\}$.

Case 1: Suppose $\mathcal{R}_a$ is dispatched before $\mathcal{R}_b$, $c(\mathcal{R}_a) \leq c(\mathcal{R}_b)$. In this case, k could not be equal to n , because n is in the last route as assumed in the statement. The

completion times are the following, $c(\mathcal{R}_a) = T_{\mathcal{R}_a} + d_{\mathcal{R}_a}$ and $c(\mathcal{R}_b) = \max\{c(\mathcal{R}_a), \alpha(\mathcal{R}_b)\} + d_{\mathcal{R}_b}$. Now consider the modified routes $\mathcal{R}'_a = \mathcal{R}_a \cup \{v_j\}$ and $\mathcal{R}'_b = \mathcal{R}_b \setminus \{v_j\}$. Since $r_k \geq r_j$, the addition of v_j does not increase the dispatch time of $\mathcal{R}_a$, $T'_{\mathcal{R}_a} = T_{\mathcal{R}_a}$, then $c(\mathcal{R}'_a) = c(\mathcal{R}_a) + 2d_{jn}$. The distance traveled in $\mathcal{R}'_b$ is $d_{\mathcal{R}'_b} = d_{\mathcal{R}_b} - 2d_{jn}$, and $\alpha(\mathcal{R}'_b) \leq \alpha(\mathcal{R}_b)$. Then, $c(\mathcal{R}'_b) = \max\{c(\mathcal{R}_a) + 2d_{jn}, \alpha(\mathcal{R}'_b)\} + d_{\mathcal{R}_b} - 2d_{jn}$.

- if $c(\mathcal{R}_a) + 2d_{jn} \geq \alpha(\mathcal{R}'_b)$, then $c(\mathcal{R}'_b) \leq c(\mathcal{R}_b)$ since $c(\mathcal{R}_b) = \max\{c(\mathcal{R}_a), \alpha(\mathcal{R}_b)\} + d_{\mathcal{R}_b}$.
- otherwise, $c(\mathcal{R}'_b) = \alpha(\mathcal{R}'_b) + d_{\mathcal{R}_b} - 2d_{jn} \leq \alpha(\mathcal{R}_b) + d_{\mathcal{R}_b} = c(\mathcal{R}_b)$

Case 2: Suppose $\mathcal{R}_b$ is dispatched before $\mathcal{R}_a$, $c(\mathcal{R}_b) \leq c(\mathcal{R}_a)$. The completion times are the following, $c(\mathcal{R}_b) = T_{\mathcal{R}_b} + d_{\mathcal{R}_b}$ and $c(\mathcal{R}_a) = \max\{c(\mathcal{R}_b), \alpha(\mathcal{R}_a)\} + d_{\mathcal{R}_a}$. Now consider the modified routes $\mathcal{R}'_a = \mathcal{R}_a \setminus \{v_i\}$ and $\mathcal{R}'_b = \mathcal{R}_b \cup \{v_i\}$. Since $r_j \geq r_i$, the addition of v_i does not increase the dispatch time of $\mathcal{R}_b$, $T'_{\mathcal{R}_b} = T_{\mathcal{R}_b}$, then $c(\mathcal{R}'_b) = c(\mathcal{R}_b) + 2d_{in}$. The distance traveled in $\mathcal{R}'_a$ is $d_{\mathcal{R}'_a} = d_{\mathcal{R}_a} - 2d_{in}$, and $\alpha(\mathcal{R}'_a) = \alpha(\mathcal{R}_a) \geq r_k$. Then, $c(\mathcal{R}'_a) = \max\{c(\mathcal{R}_b) + 2d_{in}, \alpha(\mathcal{R}'_a)\} + d_{\mathcal{R}_a} - 2d_{in}$.

- if $c(\mathcal{R}_b) + 2d_{in} \geq \alpha(\mathcal{R}'_a)$, then $c(\mathcal{R}'_a) \leq c(\mathcal{R}_a)$ since $c(\mathcal{R}_a) = \max\{c(\mathcal{R}_b), \alpha(\mathcal{R}_a)\} + d_{\mathcal{R}_a}$.
- otherwise, $c(\mathcal{R}'_a) = \alpha(\mathcal{R}'_a) + d_{\mathcal{R}_a} - 2d_{in} \leq \alpha(\mathcal{R}_a) + d_{\mathcal{R}_a} = c(\mathcal{R}_a)$

Since each transformation strictly reduces the number of interlacing pairs of routes without increasing the solution cost, and since the number of such pairs is finite, the process terminates. Therefore, we obtain a solution with non-interlacing routes and cost no greater than the original one. $\square$

Corollary 1. *If there exists an optimal solution $\mathcal{X}^*$, then there exists an optimal solution $\mathcal{X}^{\text{NI}^*}$ consisting only of non-interlacing routes.*

From Corollary 1 and Lemma 11, we can construct an optimal solution for the GTSP-rd (time), minimizing the completion time, using only non-interlacing routes for all the vertices. To compute the minimum completion time we could use a dynamic programming algorithm to test all possible non-interlacing routes.

Let $f(i)$ be the minimum completion time of an optimal partial solution consisting only of non-interlacing routes serving the vertices $\{v_1, \dots, v_i\}$. The base case is $f(0) = 0$, this means the empty set of vertices.

$$f(i) = \begin{cases} 0, & \text{if } i = 0, \\ \min_{0 \leq j < n} \left\{ \max\{f(j), r_{n-1}, r_n\} + 2d_{0n} + 2 \sum_{v=j+1}^{n-1} d_{vn} \right\}, & \text{if } i = n, \\ \min_{0 \leq j < i} \left\{ \max\{f(j), r_i\} + 2d_{0n} + 2 \sum_{v=j+1}^i d_{vn} \right\}, & \text{otherwise.} \end{cases} \quad (4.13)$$

For each vertex $i \in [1, n]$, the value $f(i)$ represents the minimum completion time of a partial solution that forms a non-interlacing route ending at v_i , constructed using some prefix of vertices $\{v_1, \dots, v_i\}$. In particular, for each possible predecessor index $j \in [0, i-1]$, we consider the route segment formed by the vertices $\{v_{j+1}, \dots, v_i\}$ and compute the cost of appending this segment to the optimal partial solution ending at v_j . The recurrence in Equation 4.13 then selects the predecessor j that minimizes the resulting completion time. When $i = n$, we just adjust the cost to reflect the properties of the central vertex.

The term $\sum_{v=j+1}^i d_{vn}$ can be precomputed using prefix sums of the distances from v_n to the remaining vertices. Thus, the recurrence can be evaluated in $O(1)$ complexity time per choice of j , yielding to an $O(n^2)$ algorithm to compute $f(n)$.

The full procedure is showed in Algorithm 4, which computes the dynamic programming table used to compute the minimum completion time.

4.3.2 Algorithm $O(n^3)$ for GTSP-rd(distance)

In this section, we address the variation of the problem in which the objective is to minimize the distance traveled by the salesman, subject to a deadline D for completing all routes, that is, $T_{\mathcal{R}_x} + d_{\mathcal{R}_x} \leq D$.

If the smallest completion time (previous section) $C^* \leq D$, then a solution for the GTSP-rd(distance) exists. In this case, any solution such that $C^* \leq T_{\mathcal{R}_x} + d_{\mathcal{R}_x} \leq D$ is feasible. Following, we show how to compute, among all feasible solutions, the one that minimizes the total distance traveled by the salesman.

Given a solution $\mathcal{X}$ containing x routes, the total distance traveled by the salesman is equal to the sum of the distances traveled on each route $d_{\mathcal{R}}$ (Equation 4.11), that is,

Algorithm 4: GTSP-rd(time) on star.

Input : n : number of customers, r_i : release dates, d_{ij} : travel time at $\{v_i, v_j\} \in E$.

Output: Minimum completion time to execute delivery in N

```

1  $f[0] \leftarrow 0$ 
2 for  $i \leftarrow 1$  to  $n - 1$  do
3    $f[i] \leftarrow \infty$ 
4   for  $j \leftarrow 0$  to  $i - 1$  do
5      $f[i] \leftarrow \min\{f[i], 2d_{0n} + \max\{f[j], r_i\} + 2 \sum_{v=j+1}^i d_{vn}\}$ 
6   end
7 end
8
9  $f[n] \leftarrow \infty$ 
10 for  $j \leftarrow 0$  to  $n - 1$  do
11    $f[n] \leftarrow \min\{f[n], 2d_{0n} + \max\{f[j], r_n, r_{n-1}\} + 2 \sum_{v=j+1}^{n-1} d_{vn}\}$ 
12 end
13
14 return  $f[n]$ 

```

$\sum_{i=1}^x d_{\mathcal{R}_i}$. Lemma 12 shows that, to minimize the total distance traveled, it is sufficient to minimize the number of routes the salesman performs.

Lemma 12. *Minimize the distance traveled, $\sum_{i=1}^x d_{\mathcal{R}_i}$, is equivalent to minimizing the number of routes x .*

Proof. By Equation 4.11:

$$\begin{aligned}
\sum_{i=1}^x d_{\mathcal{R}_i} &= \sum_{i=1}^x \left(2d_{0n} + \sum_{u \in S_i^d} 2d_{un} \right) \\
&= \sum_{i=1}^x 2d_{0n} + \sum_{e \in E \setminus \{\{v_0, v_n\}\}} 2d_e \\
&= 2d_{0n}x + \sum_{e \in E \setminus \{\{v_0, v_n\}\}} 2d_e
\end{aligned}$$

Second step follows from the fact that each customer is visited exactly once in a single route, $\bigcup_{i=1}^x S_i^d = N$, thus each edge, except for the edge $\{v_0, v_n\}$ is used twice in all routes.

As in the final equation, the only variable is x and the remaining terms are constants, it suffices to minimize x to obtain the optimal solution, as desired. $\square$

The following Lemma shows that, from a solution serving the leaf customers is possible to build a solution containing only non-interlacing routes with the same set of vertices and without increasing the number of routes.

Lemma 13. *Any solution $\mathcal{X}$ containing x routes can be transformed into a partial solution $\mathcal{X}^{\text{NI}*}$ consisting only non-interlacing routes without increasing the number of routes.*

Proof. The transformation shown in the proof of Lemma 11 does not create additional routes and can be applied here. $\square$

Thus, to calculate the minimum number of routes needed to serve the leaf customers we can extend Equation 4.13 in the following way. Let $f(i, k)$ be the minimum completion time to serve the leaf customers $\{v_1, \dots, v_i\}$ with k routes. $f(i, k)$ can be calculated in the following way:

$$f(i, k) = \begin{cases} 0, & \text{if } i = 0, \\ \infty, & \text{if } k = 0 \\ \min_{0 \leq j < n} \left\{ \max\{f(j, k-1), r_{n-1}, r_n\} + 2d_{0n} + 2 \sum_{v=j+1}^{n-1} d_{vn} \right\}, & \text{if } i = n, \\ \min_{0 \leq j < i} \left\{ \max\{f(j, k-1), r_i\} + 2d_{0n} + 2 \sum_{v=j+1}^i d_{vn} \right\}, & \text{otherwise.} \end{cases} \quad (4.14)$$

If $f(n, k) \leq D$, then the customers N could be served in k routes. The minimum number of routes x^* that could be executed to serve all customers in N , can be calculated in the following way, $x^* = \min\{k \mid f(n, k) \leq D\}$. Moreover, the minimum distance traveled is (Lemma 12):

$$2d_{0n}x^* + \sum_{e \in E \setminus \{\{v_0, v_n\}\}} 2d_e \quad (4.15)$$

Algorithm 5 computes the minimum total travel distance required to serve all customers in N , as defined in Equation 4.15. In the worst case, the optimal solution may contain up to n routes (one per customer). The dynamic programming algorithm has an overall running time of $O(n^3)$.

Algorithm 5: GTSP-rd(distance) on star.

Input : n : number of customers, r_i : release dates, $d_{\{ij\}}$: travel time at $\{v_i, v_j\} \in E$.

Output: Minimum distance traveled to execute delivery in N

```
1  $f[1][i] \leftarrow 0 \quad \forall i \in [1, n], f[i][0] \leftarrow \infty \quad \forall i \in [2, n]$ 
2
3 for  $k \leftarrow 1$  to  $n$  do
4   for  $i \leftarrow 1$  to  $n - 1$  do
5      $f[i][k] \leftarrow \infty$ 
6     for  $j \leftarrow 0$  to  $i - 1$  do
7        $f[i][k] \leftarrow \min\{f[i][k], 2d_{0n} + \max\{f[j][k - 1], r_i\} + 2 \sum_{v=j+1}^i d_{vn}\}$ 
8     end
9   end
10
11    $f[n][k] \leftarrow \infty$ 
12   for  $j \leftarrow 0$  to  $n - 1$  do
13      $f[n][k] \leftarrow \min\{f[n][k], 2d_{0n} + \max\{f[j][k - 1], r_n, r_{n-1}\} + 2 \sum_{v=j+1}^{n-1} d_{vn}\}$ 
14   end
15 end
16
17  $x^* = \infty$ 
18 for  $k \leftarrow 1$  to  $n$  do
19   if  $f[n][k] \leq D$  AND  $x^* == \infty$  then
20      $x^* = k$ 
21   end
22 end
23
24 return  $2d_{0n}x^* + \sum_{e \in E \setminus \{\{v_0, v_n\}\}} 2d_e$ 
```

4.4 GTSP-rd on Cycle Graphs

Let C^{n+1} be an cycle with $n + 1$ vertices. The GTSP-rd on cycles, presents a different challenge compared to star and path, that is, all the vertices are connected not necessarily passing by the depot. Figure 4.5 illustrates an example of instance.

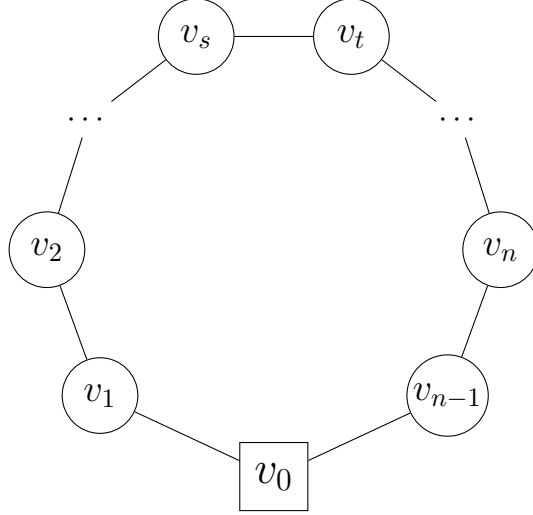


Figure 4.5: Example illustrating the arrangement of vertices in a GTSP-rd instance on a cycle.

Definition 3. A route $\mathcal{R}$ on C^{n+1} is said to be *circular* if $d_{\mathcal{R}} = D_{cycle} = \sum_{e \in E} d_e$. A circular route serves a set of customers by traversing the entire cycle, starting and ending at the depot v_0 .

The circular routes take an important role in the solution of GTSP-rd on cycles.

4.4.1 Algorithm $O(n^3)$ for GTSP-rd(time)

In this section we study the GTSP-rd under the objective of minimizing the *completion time* (GTSP-rd(time)) on cycle graphs.

Lemma 14. Let $\mathcal{R}_i$ and $\mathcal{R}_j$ be two circular routes in a cycle graph C^{n+1} , with $c(\mathcal{R}_i) \leq c(\mathcal{R}_j)$. Then there exists a circular route $\mathcal{R}'$ such that:

1. $S_{\mathcal{R}'}^d = S_{\mathcal{R}_i}^d \cup S_{\mathcal{R}_j}^d$;
2. $c(\mathcal{R}') \leq c(\mathcal{R}_j)$.

Proof. For any circular route $\mathcal{R}$, $c(\mathcal{R}) = \max\{c_{\text{prev}}(\mathcal{R}), \alpha(\mathcal{R})\} + D_{\text{cycle}}$, where $c_{\text{prev}}(\mathcal{R})$ denotes the completion time of the predecessor route (if any). Since $\mathcal{R}_i$ is performed before $\mathcal{R}_j$, $c(\mathcal{R}_i) \leq \max\{c_{\text{prev}}(\mathcal{R}_j), \alpha(\mathcal{R}_j)\}$

Using $c(\mathcal{R}_i) = \max\{c_{\text{prev}}(\mathcal{R}_i), \alpha(\mathcal{R}_i)\} + D_{\text{cycle}}$ we obtain $\max\{c_{\text{prev}}(\mathcal{R}_i), \alpha(\mathcal{R}_i)\} + D_{\text{cycle}} \leq \max\{c_{\text{prev}}(\mathcal{R}_j), \alpha(\mathcal{R}_j)\}$. This implies, $\alpha(\mathcal{R}_i) \leq \max\{c_{\text{prev}}(\mathcal{R}_j), \alpha(\mathcal{R}_j)\}$.

Let $\mathcal{R}'$ be a circular route that serves $S_{\mathcal{R}_i}^d \cup S_{\mathcal{R}_j}^d$. Its release-date term is $\alpha(\mathcal{R}') = \max\{\alpha(\mathcal{R}_i), \alpha(\mathcal{R}_j)\}$. From the inequalities $\alpha(\mathcal{R}_i) \leq \max\{c_{\text{prev}}(\mathcal{R}_j), \alpha(\mathcal{R}_j)\}$ and $\alpha(\mathcal{R}_j) \leq \max\{c_{\text{prev}}(\mathcal{R}_j), \alpha(\mathcal{R}_j)\}$, we get $\alpha(\mathcal{R}') \leq \max\{c_{\text{prev}}(\mathcal{R}_j), \alpha(\mathcal{R}_j)\}$.

Removing $\mathcal{R}_i$ from solution cannot increase the completion time of the predecessor of $\mathcal{R}_j$, then $c_{\text{prev}}(\mathcal{R}') \leq c_{\text{prev}}(\mathcal{R}_j)$. Consequently the dispatch time of $\mathcal{R}'$ satisfies

$$\max\{c_{\text{prev}}(\mathcal{R}'), \alpha(\mathcal{R}')\} \leq \max\{c_{\text{prev}}(\mathcal{R}_j), \alpha(\mathcal{R}_j)\}$$

Finally, since both $\mathcal{R}'$ and $\mathcal{R}_j$ are circular routes, we obtain $c(\mathcal{R}') \leq c(\mathcal{R}_j)$, which proves the claim. $\square$

Corollary 2. *In a cycle graph C^{n+1} , there exists an optimal solution $\mathcal{X}^*$ to the GTSP-rd(time) that contains at most one circular route.*

Proof. Let $\mathcal{X}$ be any optimal solution. If $\mathcal{X}$ contains two circular routes $\mathcal{R}_i$ and $\mathcal{R}_j$, Lemma 14 guarantees the existence of a single circular route $\mathcal{R}'$ serving all customers of $\mathcal{R}_i \cup \mathcal{R}_j$ with $c(\mathcal{R}') \leq c(\mathcal{R}_j)$. Replacing $\mathcal{R}_i$ and $\mathcal{R}_j$ by $\mathcal{R}'$ yields another feasible solution whose completion time is no larger than that of $\mathcal{X}$, and therefore also optimal. Repeating this replacement until no two circular routes remain produces an optimal solution with at most one circular route. $\square$

Lemma 15. *There exists an optimal solution $\mathcal{X}^*$ to GTSP-rd(time) on C^{n+1} such that, if it contains a circular route $\mathcal{R}_W$, then $\mathcal{R}_W$ must be the first route in $\mathcal{X}^*$ to serve any customer set S_W^d and any route $\mathcal{R}'$ that precedes $\mathcal{R}_W$ can be merged into $\mathcal{R}_W$ without increasing the completion time.*

Proof. Let $\mathcal{X}$ be a solution where a non-circular route $\mathcal{R}_p$ is executed immediately before a circular route $\mathcal{R}_W$. Let $\mathcal{R}_p$ serve S_p^d and $\mathcal{R}_W$ serve S_W^d . The completion time of $\mathcal{R}_W$ is $c(\mathcal{R}_W) = \max\{c(\mathcal{R}_p), \alpha(\mathcal{R}_W)\} + D_{\text{cycle}}$.

Let $\mathcal{X}'$ be a solution where $\mathcal{R}_W$ and $\mathcal{R}_p$ are merged and replaced by a circular route

$\mathcal{R}'_W$, where $S_W^{d'} = S_W^d \cup S_p^d$. The completion time of $\mathcal{R}'_W$ is $c(\mathcal{R}'_W) = \max\{c_{prev}, \alpha(\mathcal{R}'_W)\} + D_{cycle}$, where $\alpha(\mathcal{R}'_W) = \max\{\alpha(\mathcal{R}_p), \alpha(\mathcal{R}_W)\}$.

Since $c(\mathcal{R}_p) = \max\{c_{prev}, \alpha(\mathcal{R}_p)\} + d_{\mathcal{R}_p}$. It follows that:

$$c(\mathcal{R}'_W) = \max\{c_{prev}, \alpha(\mathcal{R}'_W)\} + D_{cycle} \leq \max\{c(\mathcal{R}_p), \alpha(\mathcal{R}_W)\} + D_{cycle} = c(\mathcal{R}_W)$$

Replacing $c(\mathcal{R}_p)$:

$$\max\{c_{prev}, \alpha(\mathcal{R}'_W)\} + D_{cycle} \leq \max\{\max\{c_{prev}, \alpha(\mathcal{R}_p)\} + d_{\mathcal{R}_p}, \alpha(\mathcal{R}_W)\} + D_{cycle}$$

If $\alpha(\mathcal{R}'_W) = \alpha(\mathcal{R}_p)$ or $\alpha(\mathcal{R}'_W) = \alpha(\mathcal{R}_W)$ the inequality holds and then $c(\mathcal{R}'_W) \leq c(\mathcal{R}_W)$. Replacing $\mathcal{R}_p$ and $\mathcal{R}_W$ by $\mathcal{R}'_W$ therefore yields a feasible solution with no larger completion time; repeating this merging for any routes preceding a circular route produces an (equally optimal) solution in which the circular route is the first route serving its customers. This concludes the proof. $\square$

Definition 4. A *non-circular route* $\mathcal{R}$ is said to be *clockwise*, denoted by $\mathcal{R}^{cw}$, if it departs from the depot, visits vertices in clockwise order up to its furthest vertex, and then returns to the depot following the counterclockwise direction. Analogously, a *non-circular route* $\mathcal{R}$ is said to be *counterclockwise*, denoted by $\mathcal{R}^{ccw}$, if it departs from the depot, visits vertices in counterclockwise order up to its furthest vertex, and returns to the depot following the clockwise direction.

Let τ_v^{cw} (τ_v^{ccw}) denote the distance from the depot to a vertex v along the cycle in the clockwise (counterclockwise) direction. The total length of a clockwise (counterclockwise) non-circular route is twice the maximum distance from the depot to any vertex in the route, $d_{\mathcal{R}^{cw}} = 2 \max_{v \in S_{\mathcal{R}^{cw}}} \{\tau_v^{cw}\}$, $d_{\mathcal{R}^{ccw}} = 2 \max_{v \in S_{\mathcal{R}^{ccw}}} \{\tau_v^{ccw}\}$.

Lemma 16. *There exists an optimal solution $\mathcal{X}^*$ to GTSP-rd(time) on C^{n+1} such that:*

- *for every clockwise non-circular route $\mathcal{R}^{cw} \subseteq \mathcal{X}^*$, $d_{\mathcal{R}^{cw}} < D_{cycle}$;*
- *for every counterclockwise non-circular route $\mathcal{R}^{ccw} \subseteq \mathcal{X}^*$, $d_{\mathcal{R}^{ccw}} \leq D_{cycle}$.*

Proof. Consider an optimal solution $\mathcal{X}^*$. Suppose, by contradiction, that there exists a clockwise non-circular route $\mathcal{R}^{cw} \in \mathcal{X}^*$ such that $d_{\mathcal{R}^{cw}} \geq D_{cycle}$, serving a set of vertices S^d . Then, $\mathcal{R}^{cw}$ can be replaced by a circular route $\mathcal{R}$ that visits all vertices in S^d , with total distance $d_{\mathcal{R}} = D_{cycle} \leq d_{\mathcal{R}^{cw}}$. Hence, $c(\mathcal{R}) \leq c(\mathcal{R}^{cw})$.

Thus, replacing $\mathcal{R}^{cw}$ by $\mathcal{R}$ yields a solution no worse than $\mathcal{X}^*$, contradicting the assumption that $\mathcal{R}^{cw}$ appears in an optimal solution with $d_{\mathcal{R}^{cw}} \geq D_{cycle}$. Therefore, we must have $d_{\mathcal{R}^{cw}} < D_{cycle}$. The argument for counterclockwise non-circular routes is analogous.

□

Lemma 15 that an optimal solution may consist of a sequence of non-circular routes, optionally preceded by a circular route, or a single circular route. Let v_s be the clockwise furthest customer such that clockwise distance from depot is less than $\frac{D_{cycle}}{2}$, $v_s = \max\{v \mid v \in N, d_v^{cw} < \frac{D_{cycle}}{2}\}$, and v_t be the same for counterclockwise route, $v_t = \max\{v \mid v \in N, d_v^{ccw} \leq \frac{D_{cycle}}{2}\}$.

Based on Lemma 16 the following approach can be used to solve the GTSP-rd(time) in a cycle graph C^{n+1} . In order to consider the non-circular routes, let $e = \{v_s, s_t\} \in E$, we conceptually remove it, and obtain a path P_e whose extremities correspond to the two vertices incident to e . The depot remains an internal vertex of this path. We index customers to the *clockwise side* of the depot as $N_l = \{v_1^l, \dots, v_{n_l}^l\}$ and the customers to the *counter-clockwise side* as $N_r = \{v_1^r, \dots, v_{n_r}^r\}$, in increasing order of their distance from the depot along the corresponding. Let τ_i^l be the clockwise distance from depot to v_i^l and τ_j^r be the counter-clockwise distance to v_j^r . Thus, $v_1^l = v_s$ and $v_1^r = v_t$ are the vertices farthest from v_0 in the respective sets. Figure 4.6 illustrate P_e . We derive P_e from C^{n+1} to separate clockwise non-circular routes from counterclockwise non-circular routes. The remainder of this section show how to optimal combine a possible first circular route with remaining non-circular routes. Lemma 17 shows that is optimal build the circular route serving vertices $\{v_1^l, \dots, v_i^l\} \cup \{v_1^r, \dots, v_j^r\}$ for some i, j , with $i \leq n_l$ and $j \leq n_r$, that is, the possible circular route serve the vertices in the extremity of P_e .

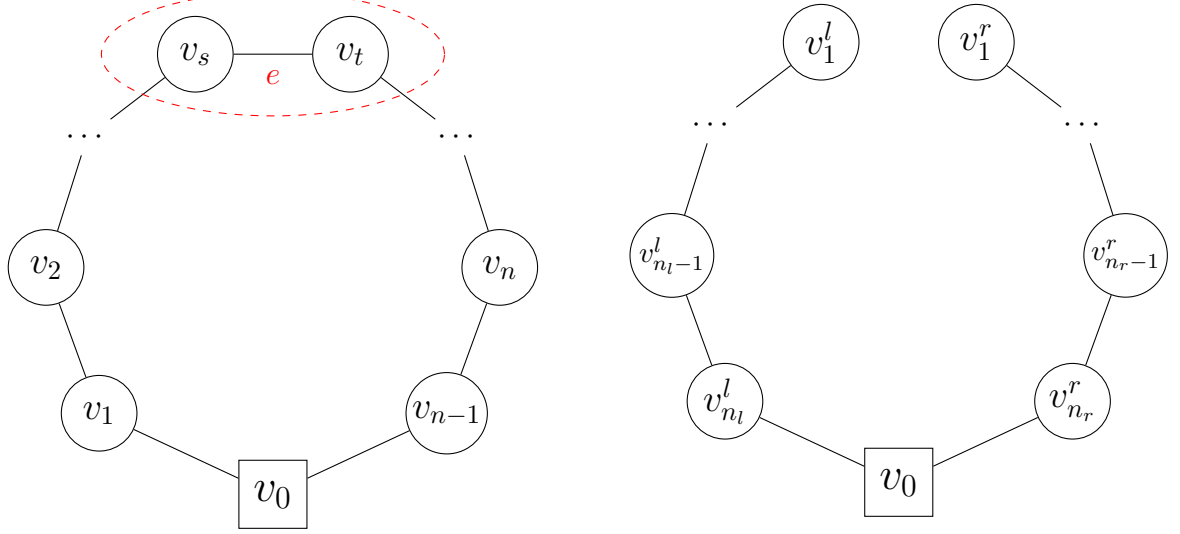


Figure 4.6: Path P_e , given the edge $e = \{v_s, v_t\} \in E$, remove it, and obtain a path whose extremities correspond to the two vertices incident to e

Lemma 17. *There exist an optimal solution to $GTSP\text{-}rd(\text{time})$, such that, if it contain a circular route $\mathcal{R}$, then the customers served by $\mathcal{R}$ on each side of the depot form a prefix. That is, if $\mathcal{R}$ serves v_p^r , then it also serves every v_q^r with $q < p$; and similarly for the clockwise side.*

Proof. Suppose, towards a contradiction, that $\mathcal{R}$ does not serve a prefix: let q be the smallest index such that $v_q^r \notin S_{\mathcal{R}}^d$ but there exists $p > q$ with $v_p^r \in S_{\mathcal{R}}^d$.

Let $A = \{v_t^r \in S_{\mathcal{R}}^d : t > q\}$ be the set of right-side customers of $\mathcal{R}$ strictly farther than v_q^r , and let $\mathcal{S}$ be the route that serves v_q^r , by Lemma 15, $\mathcal{S}$ occurs after $\mathcal{R}$. We consider two cases:

Case 1: $\alpha(v_q^r) \leq \alpha(\mathcal{R})$.

Then v_q^r is already available at the start of $\mathcal{R}$, so including v_q^r into $\mathcal{R}$ does not increase the release-date term $\alpha(\mathcal{R})$. Since $\mathcal{R}$ is circular its travel time equals D_{cycle} , so adding one more customer on the cycle does not change the travel time. Hence we can modify $\mathcal{R}$ to also serve v_q^r without increasing its completion time, contradicting the choice of q .

Case 2: $\alpha(v_q^r) > \alpha(\mathcal{R})$.

Because every vertex in A is served by $\mathcal{R}$, we have $\alpha(a) \leq \alpha(\mathcal{R})$ for every $a \in A$. Combining this with the assumption of this case gives

$$\alpha(a) \leq \alpha(\mathcal{R}) < \alpha(v_q^r) \leq \alpha(\mathcal{S}) \quad \text{for every } a \in A,$$

where the last inequality holds because $\alpha(\mathcal{S})$ is the maximum release date of the customers served by $\mathcal{S}$, and $v_q^r \in S_{\mathcal{S}}^d$.

Now construct a new feasible solution as follows:

1. Remove the customers in A from $\mathcal{R}$.
2. Add the customers in A to $\mathcal{S}$.

This modification does not increase the completion time of either $\mathcal{R}$ or $\mathcal{S}$.

- For $\mathcal{R}$: removing A can only decrease its release-date term $\alpha(\mathcal{R})$, and the travel time of $\mathcal{R}$ remains D_{cycle} . Hence $c(\mathcal{R})$ does not increase.
- For $\mathcal{S}$: before the change $\alpha(\mathcal{S}) \geq \alpha(v_q^r)$. As we showed $\max_{a \in A} \alpha(a) \leq \alpha(\mathcal{S})$, then after adding A we have $\alpha(\mathcal{S}') = \alpha(\mathcal{S})$. Thus the release-date term of $\mathcal{S}$ does not change.

Concerning travel time: each $a \in A$ lies on the same (right) side of the depot as v_q^r , and also the distance from depot $\tau_a^l \leq \tau_q^r$ for all $a \in A$. Then the travel time does not increase also.

Therefore $\mathcal{S}$'s completion time after the modification is not larger than the larger of the original completion times of $\mathcal{R}$ and $\mathcal{S}$. Since $\mathcal{S}$ originally started after $\mathcal{R}$, the overall makespan cannot increase.

Because neither affected routes completion time increases, the global makespan does not increase. The modification strictly reduces the number of non-prefix customers served by $\mathcal{R}$. Repeating this operation until no such q exists yields an optimal solution in which the customers of $\mathcal{R}$ on the right side form a prefix, as claimed. $\square$

Finally, just like in GSTP-rd on paths, we consider build the non-circular routes like non-interlacing routes in path P_e . The idea of the algorithm is try to build non-circular routes with vertices containing vertex from only one side N_l or N_r , and an optional first circular route containing vertex from both sets $N_l \cup N_r$. Let $c(i, j)$ the minimum completion time to serve the vertices $\{v_1^l, \dots, v_i^l\}$ and $\{v_1^r, \dots, v_j^r\}$ where the non-circular routes serves only each set N_l and N_r separately.

$$c(i, j) = \begin{cases} 0, & \text{if } i = j = 0 \\ \min \{L(i, j), R(i, j), W(i, j)\}, & \text{otherwise} \end{cases} \quad (4.16)$$

$W(i, j)$ represents the cost to execute a circular route serving vertices $\{v_1^l, \dots, v_i^l\} \cup \{v_1^r, \dots, v_j^r\}$ and ending the execution. Let $\alpha(i, j) = \max \{r_a \mid v_a \in \{v_1^l, \dots, v_i^l\} \cup \{v_1^r, \dots, v_j^r\}\}$. $\alpha^l(w, i) = \max \{r_a \mid v_a \in \{v_w^l, \dots, v_i^l\}\}$, $\alpha^r(w, j) = \max \{r_a \mid v_a \in \{v_w^r, \dots, v_j^r\}\}$.

$$L(i, j) = \min_{0 \leq w < i} \{ \max \{ \alpha^l(w+1, i), c(w, j) \} + 2\tau_{w+1}^l \} \quad (4.17)$$

$$R(i, j) = \min_{0 \leq w < j} \{ \max \{ \alpha^r(w+1, j), c(i, w) \} + 2\tau_{w+1}^r \} \quad (4.18)$$

$$W(i, j) = \alpha(i, j) + D_{cycle} \quad (4.19)$$

Theorem 5. *The recurrence relation c (4.16) correctly computes the minimum completion time $c(n_l, n_r)$ for the GTSP-rd on cycle graphs.*

Proof. We prove correctness by induction on $i + j$.

Base. For $i = j = 0$ the claim is trivial: no customers remain and $c(0, 0) = 0$.

Inductive hypothesis. Assume $c(i', j')$ is correct for all pairs with $i' + j' < i + j$. Let $c(i, j)$ as defined by (4.16) equals the minimum possible completion time to serve the vertices $\{v_1^l, \dots, v_i^l\} \cup \{v_1^r, \dots, v_j^r\}$.

Let $\mathcal{S}$ be an optimal feasible solution that serves exactly those vertices and takes the minimum completion time. Consider the last route executed in $\mathcal{S}$. By the structural lemmas proved earlier, three exhaustive and mutually exclusive possibilities arise:

Case W (last route is circular). If the last route is a circular route that serves exactly the vertices $\{v_1^l, \dots, v_i^l\} \cup \{v_1^r, \dots, v_j^r\}$, then its completion time equals

$$c(\mathcal{R}) = \alpha(i, j) + D_{cycle},$$

since a circular route always takes travel time D_{cycle} and its start is determined by the maximum release date among its customers. Thus the optimal cost in this case is $W(i, j)$.

Case L (last route is a non-circular route serving a block on the left). If the last route is non-circular and serves only left-side customers, then because non-circular routes are restricted to a single side and the left-side customers its optimal structure resemble 4.6 and form intervals, the last route must serve a contiguous block $v_{w+1}^l, \dots, v_i^l$ for some $0 \leq w < i$. Let the completion time before this last route be $c(w, j)$. The

last left-only route must wait until all its customers are released, so it effectively starts at $\max\{\alpha^l(w, i), c(w, j)\}$. Its travel distance is the usual go-and-return distance to the farthest served left vertex, which equals $2\tau_{w+1}^l$ under the path model, hence its completion time equals

$$\max\{\alpha^l(w, i), c(w, j)\} + 2\tau_{w+1}^l.$$

Minimizing over the possible split index w yields $L(i, j)$; by the inductive hypothesis $c(w, j)$ is correct, so the DP correctly evaluates the best schedule of this type.

Case R (last route is a non-circular route serving a block on the right). Analogous to Case L, the last route serves $v_{w+1}^r, \dots, v_j^r$ for some $0 \leq w < j$, and the completion time is

$$\max\{\alpha^r(w, j), c(i, w)\} + 2\tau_{w+1}^r,$$

and minimizing over w yields $R(i, j)$.

Since these three cases are exhaustive for the structure of an optimal solution and each case is captured by one of the terms $W(i, j)$, $L(i, j)$ and $R(i, j)$, the optimal completion time $c(i, j)$ is the minimum of those three terms. The inductive step is thus established, and the recurrence (4.16) is correct. $\square$

Algorithm 6 computes $c(n_l, n_r)$ using dynamic programming and the recurrence relation 4.16. Since the input graph is a cycle, we have $|E| = n + 1$. Assuming that the functions α and τ are precomputed. DP table contains $O(n^2)$ states, and computing each state requires two independent minimizations over $O(n)$ candidates. Thus, the running time is $O(n^3)$. Therefore, Algorithm 6 runs in $O(n^3)$ time to compute $c(n_l, n_r)$. Consequently, the GTSP-rd (time) on cycle graphs is polynomial-time solvable.

4.4.2 Algorithm $O(n^3)$ for GTSP-rd(distance)

In this subsection we study the GTSP-rd under the objective of minimizing the total distance traveled on cycle graphs. As in the GTSP-rd(time) case, we will show that there exist a optimal solution on C^{n+1} that admit a structured form: they consist of an optional circular route, followed by a sequence of non-circular routes serving a contiguous block of customers. We first show that an optimal solution can be constructed using at most one circular route (Lemma 18).

Algorithm 6: GTSP-rd(time) on cycle.

Input : n : number of customers; r_i : release dates; $d_{\{ij\}}$: travel time on edge $\{v_i, v_j\} \in E$.

Output: Minimum completion time to execute all deliveries in N

```

1 Build left path  $P_l = (v_0, \dots, v_s)$ 
2 Build right path  $P_r = (v_0, \dots, v_t)$ 
3  $n_l \leftarrow |P_l| - 1$ ,  $n_r \leftarrow |P_r| - 1$ 
4
5 for  $i \leftarrow 0$  to  $n_l$  do
6   for  $j \leftarrow 0$  to  $n_r$  do
7     if  $i = 0$  and  $j = 0$  then
8        $c[0][0] \leftarrow 0$ 
9       continue
10    end
11     $L \leftarrow \infty$ 
12    for  $w \leftarrow 0$  to  $i - 1$  do
13       $L \leftarrow \min \left\{ L, \max \{ c[w][j], \alpha^l(w, i) \} + 2\tau_{w+1}^l \right\}$ 
14    end
15     $R \leftarrow \infty$ 
16    for  $w \leftarrow 0$  to  $j - 1$  do
17       $R \leftarrow \min \left\{ R, \max \{ c[i][w], \alpha^r(w, j) \} + 2\tau_{w+1}^r \right\}$ 
18    end
19     $W \leftarrow \alpha(i, j) + D_{cycle}$ 
20     $c[i][j] \leftarrow \min \{ L, R, W \}$ 
21  end
22 end
23
24 return  $c[n_l][n_r]$ 

```

Lemma 18. *In a cycle graph C^{n+1} , there exists an optimal solution $\mathcal{X}^*$ to the GTSP-rd(distance) that contains at most one circular route.*

Proof. Let $\mathcal{X}$ be any optimal solution. If $\mathcal{X}$ contains two circular routes $\mathcal{R}_i$ and $\mathcal{R}_j$, with $\mathcal{R}_i$ executed before $\mathcal{R}_j$. We can replace the two routes by $\mathcal{R}'$ serving all customers of $\mathcal{R}_i \cup \mathcal{R}_j$ with dispatch time $T_{\mathcal{R}'} = T_{\mathcal{R}_j}$ and with distance traveled as D_{cycle} instead $2D_{cycle}$ distance traveled of circular routes $\mathcal{R}_i$ and $\mathcal{R}_j$. Repeating this replacement until no two circular routes remain produces an optimal solution with at most one circular route. $\square$

Lemmas 19 shows that if the optimal solution contains an circular route this route is the first route.

Lemma 19. *There exists an optimal solution $\mathcal{X}^*$ to GTSP-rd(distance) on C^{n+1} such*

that, if it contains a circular route $\mathcal{R}_W$, then $\mathcal{R}_W$ is the first route serving any customer in S_W^d .

Proof. Suppose, by contradiction, that in an optimal solution $\mathcal{X}$ the circular route $\mathcal{R}_W$ is not the first route to serve its customer set S_W^d . Let $\mathcal{R}_p$ be the route executed immediately before $\mathcal{R}_W$ and serving some customers in S_p^d .

By Lemma 15, merging $\mathcal{R}_p$ into $\mathcal{R}_W$ produces a single circular route $\mathcal{R}'$ whose completion time is no larger than that of $\mathcal{R}_W$. Hence the merged solution remains feasible. And $\mathcal{R}_W$ alone contributes D_{cycle} to the distance, and the non-circular route $\mathcal{R}_p$ contributes a non-negative amount. After merging, the route $\mathcal{R}'$ is a single circular route of distance D_{cycle} . Therefore, the total distance strictly decreases.

This contradicts the optimality of $\mathcal{X}$. Hence, in every optimal solution, if a circular route is present, it must be the first route serving its customers. $\square$

We adopt the same strategy used in the GTSP-rd(time) case. We conceptually break the cycle C^{n+1} into a path P_e by removing $e = \{v_s, v_t\}$, and we attempt to construct all feasible non-circular routes on P_e as non-interlacing routes that serve customers from only one side of the depot, exactly as in the general path case (Section 4.2).

As in the GTSP-rd(time) analysis, we index the customers on the clockwise side of the depot as $N_l = \{v_1^l, \dots, v_{n_l}^l\}$ and those on the counter-clockwise side as $N_r = \{v_1^r, \dots, v_{n_r}^r\}$, according to their distance from the depot along their respective directions.

Lemma 20 guarantees that, in any optimal solution containing a circular route, the customers served by that route must form a prefix on each side. Thus the circular route serves precisely $\{v_1^l, \dots, v_i^l\} \cup \{v_1^r, \dots, v_j^r\}$ for some $i \leq n_l$ and $j \leq n_r$.

Lemma 20. *There exist an optimal solution to GTSP-rd(distance), such that, if it contain a circular route $\mathcal{R}$, then the customers served by $\mathcal{R}$ on each side of the depot form a prefix. That is, if $\mathcal{R}$ serves v_p^r , then it also serves every v_q^r with $q < p$; and similarly for the clockwise side.*

Proof. Suppose, by contradiction, that $\mathcal{R}$ violates the prefix property on the right side. Then there exist indices $q < p$ such that $v_p^r \in S_{\mathcal{R}}^d$ while $v_q^r \notin S_{\mathcal{R}}^d$. Let $\mathcal{S}$ be the route serving v_q^r ; by previous structural lemmas, $\mathcal{S}$ occurs after $\mathcal{R}$.

Since $\mathcal{R}$ violates the prefix property, we may apply the transformation of Lemma 17 to the pair $(\mathcal{R}, \mathcal{S})$. This transformation preserves feasibility and does not increase total

distance, while eliminating at least one violation of the prefix property.

Repeating the argument removes all such violations on the right side. The left side is symmetric. Since every step preserves feasibility and does not increase distance, the original solution cannot have been optimal. $\square$

Analogously to Section 4.1.2, we compute the latest feasible dispatch time T^* for a solution of the form described above. Once the first route is dispatched at time T^* , no idle time may occur between consecutive routes; otherwise, the dispatch of the first route could be delayed further, contradicting the definition of T^* . Consequently, the minimum total distance traveled by the routes in an optimal solution is $D - T^*$.

Let $\lambda(i, j)$ as the latest feasible time to start a route that serves v_i^l or v_j^r , assuming all customers $\{v_i^l, \dots, v_{n_l}^l\}$ and $\{v_j^r, \dots, v_{n_r}^r\}$ will be served.

$$\lambda(i, j) = \begin{cases} D, & \text{if } i = n_l + 1, j = n_r + 1 \\ \max \{L(i, j), R(i, j), W(i, j)\}, & \text{if } i = 1, j = 1 \\ \max \{L(i, j), R(i, j)\}, & \text{otherwise} \end{cases} \quad (4.20)$$

The transitions are defined as follows:

- **Base case** ($i = n_l + 1, j = n_r + 1$): All customers have been served and the latest dispatch time is D .
- **Non-circular route in N_l** : A route beginning at v_i^l will serve $\{v_i^l, \dots, v_{w-1}^l\}$ and $\{v_w^l, \dots, v_{n_l}^l\}$ must be served next. If $\lambda(w, j)$ is the latest time at which the next route may start, then this route must start at $\lambda(w, j) - 2\tau_i^l$. Feasibility requires the release condition $\lambda(w, j) - \alpha^l(i, w - 1) \geq 2\tau_i^l$. Maximizing over admissible w yields $L(i, j)$.

$$L(i, j) = \max_{w > i} \{\lambda(w, j) \mid \lambda(w, j) - \alpha^l(i, w - 1) \geq 2\tau_i^l\} - 2\tau_i^l \quad (4.21)$$

- **Non-circular route in N_r** : Symmetric to the left side.

$$R(i, j) = \max_{w > j} \{\lambda(i, w) \mid \lambda(i, w) - \alpha^r(j, w - 1) \geq 2\tau_j^r\} - 2\tau_j^r \quad (4.22)$$

- **A first Circular route (when $i = 1, j = 1$)**: From this state we may choose a circular route serving prefixes on both sides. If the remaining state is (i', j') , feasibility requires $\lambda(i', j') - D_{cycle} \geq \alpha(i' - 1, j' - 1)$. This yields the value $W(1, 1)$.

$$W(1, 1) = \max_{i', j' : i' > 1, j' > 1} \{ \lambda(i', j') - D_{cycle} \mid \lambda(i', j') - D_{cycle} \geq \alpha(i' - 1, j' - 1) \} \quad (4.23)$$

Algorithm 7 implements Recurrence Relation 4.20.

Algorithm 7: GTSP-rd(distance) on cycle.

Input : n : number of customers; r_i : release dates; $d_{\{ij\}}$: travel time on edge $\{v_i, v_j\} \in E$.

Output: Minimum distance traveled to execute all deliveries in N

```

1 Build left path  $P_l = (v_0, \dots, v_s)$ 
2 Build right path  $P_r = (v_0, \dots, v_t)$ 
3  $n_l \leftarrow |P_l| - 1$ ,  $n_r \leftarrow |P_r| - 1$ 
4
5  $\lambda[n_l + 1][n_r + 1] \leftarrow D$ 
6 for  $i \leftarrow n_l + 1$  down to 1 do
7   for  $j \leftarrow n_r + 1$  down to 1 do
8      $L \leftarrow -\infty$ 
9     for  $w \leftarrow i + 1$  to  $n_l + 1$  do
10      if  $\lambda(w, j) - \alpha^l(i, w - 1) \geq 2\tau_i^l$  then
11         $L \leftarrow \max \{ L, \lambda(w, j) - 2\tau_i^l \}$ 
12      end
13    end
14     $R \leftarrow -\infty$ 
15    for  $w \leftarrow j + 1$  to  $n_r + 1$  do
16      if  $\lambda(i, w) - \alpha^r(j, w - 1) \geq 2\tau_j^r$  then
17         $R \leftarrow \max \{ R, \lambda(i, w) - 2\tau_j^r \}$ 
18      end
19    end
20     $\lambda[i][j] \leftarrow \max \{ L, R \}$ 
21  end
22 end
23
24  $W \leftarrow -\infty$ 
25 for  $i' \leftarrow 0$  to  $n_l$  do
26   for  $j' \leftarrow 0$  to  $n_r$  do
27     if  $\lambda(i', j') - D_{cycle} \geq \alpha(i' - 1, j' - 1)$  then
28        $W \leftarrow \max \{ W, \lambda(i', j') - D_{cycle} \}$ 
29     end
30   end
31 end
32
33 return  $D - \lambda[1][1]$ 

```

We conclude this section by formally establishing the correctness and computational complexity of Algorithm 7.

Lemma 21. *Algorithm 7 returns the minimum total distance required to serve all customers on a cycle graph C^{n+1} satisfying release dates.*

Proof. The correctness relies on the decomposition of the cycle and the duality between maximizing start time and minimizing distance. First, Lemmas 18, 19, and 20 establish that the optimal solution space is restricted to a specific structure: optionally one circular route serving a prefix of customers on both sides of the depot, followed by a sequence of non-circular routes.

The recurrence relation $\lambda(i, j)$ computes the latest feasible time a route can start such that all customers $\{v_i^l \dots v_{n_l}^l\}$ and $\{v_j^r \dots v_{n_r}^r\}$ are served by the deadline D . Specifically, for a transition involving distance cost C (where C is either 2τ for non-circular or D_{cycle} for circular) and release constraints α , the recurrence ensures: $T_{current} + C \leq T_{next}$ and $T_{current} \geq \alpha$. Because the algorithm pushes all routes as close to the deadline D as possible (maximizing the start time), no idle time exists between the dispatch of the first route and the deadline D . Consequently, the interval $[T^*, D]$ consists exclusively of travel time. Thus, maximizing the start time $T^* = \lambda(1, 1)$ is mathematically equivalent to minimizing the total distance traveled, given by $D - T^*$. $\square$

Theorem 6. *The time complexity of Algorithm 7 is $O(n^3)$.*

Proof. We construct a Dynamic Programming table of size $(n_l + 2) \times (n_r + 2)$. Since $n_l + n_r = n$, the table size is $O(n^2)$. Computing the value of each cell $\lambda(i, j)$ requires iterating over possible next states w . In the worst case, this inner iteration runs $O(n_l)$ or $O(n_r)$ times, which is $O(n)$. Therefore, filling the table takes $O(n^2 \times n) = O(n^3)$. The calculation of the circular route option W_{opt} also involves a nested loop over i' and j' , taking $O(n^2)$. Combining these, the total complexity is $O(n^3 + n^2) = O(n^3)$. $\square$

Together with the structural lemmas and the correctness theorem, this establishes that Algorithm 7 is both exact and efficient for GTSP-rd(distance), running in $O(n^3)$ time complexity. Consequently, the GTSP-rd (distance) on cycle graphs is polynomial-time solvable.

4.5 Discussion of Results

Building upon the context provided in Section 3.2, we have established that the GTSP-rd is solvable in polynomial time on paths for both objective functions. Our results make significant contributions to the state-of-the-art regarding the GTSP-rd on paths. Specifically, for paths with depot at the extremities, we presented an $O(n)$ algorithm for the GTSP-rd (time) and an $O(n \log n)$ algorithm for the GTSP-rd (distance). Furthermore, for the general case where depots can be located at any vertex, we introduced an $O(n^2)$ solution for the GTSP-rd (time) and an $O(n^2 \log n)$ solution for the GTSP-rd (distance).

Table 4.1 updates the literature comparison to reflect these new bounds.

Table 4.1: Updated complexity results for GTSP-rd on paths.

Reference Paper	Path			
	Depot at extremity		General case	
	time	distance	time	distance
Archetti et al. (2015)	-	-	$O(n^3)$	$O(n^3)$
Reyes et al. (2018)	$O(n^2)$	$O(n^2)$	-	-
This work	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(n^2 \log n)$

Regarding other path-like graphs, we demonstrated that the GTSP-rd on Stars and Cycles is also solvable in polynomial time, providing dynamic programming algorithms for both cases. For the GTSP-rd on Stars, a polynomial-time strategy was already known for the specific case where the depot is located at the center. We extended this by addressing the case where the depot is a leaf vertex, proposing an $O(n^2)$ algorithm for the GTSP-rd(time) and an $O(n^3)$ algorithm for the GTSP-rd(distance). For the GTSP-rd on Cycles, we provided $O(n^3)$ algorithms for both the time and distance objectives.

Table 4.2 summarizes the results obtained in this work for path-like graphs and outlines the immediate frontiers for future research. The time complexities shown in the columns *time* and *distance* refer to the worst case for each graph class, where the depot is located in an arbitrary vertex. For instance, in paths, the worst-case is when the depot is located at an internal vertex. For stars, the case where the depot is located in a leave.

4.5.1 GTSP-rd on Subdivided Stars

Moreover, we can extend the results achieved to GTSP-rd on paths to provide an algorithm to solve the GTSP-rd on subdivided stars when the depot is the center. We know that

Table 4.2: Summary of complexity results obtained in this work for the GTSP-rd on path-like graph classes.

Graph Class	GTSP-rd	
	time	distance
Path	$O(n^2)$	$O(n^2 \log n)$
Star	$O(n^2)$	$O(n^3)$
Subdivided Star (depot at center)	$O(n^k)$	$O(n^k \log n)$
Cycle	$O(n^3)$	$O(n^3)$
Caterpillar	-	-
Lobster	-	-

the polynomial-time solvable classes for the GTSP-rd, paths and stars, are subclasses of subdivided stars. A path can be viewed as a special case where only two paths emerge from the central vertex, while a star is the case where $n - 1$ paths emanate from the central vertex (Figure 4.7).

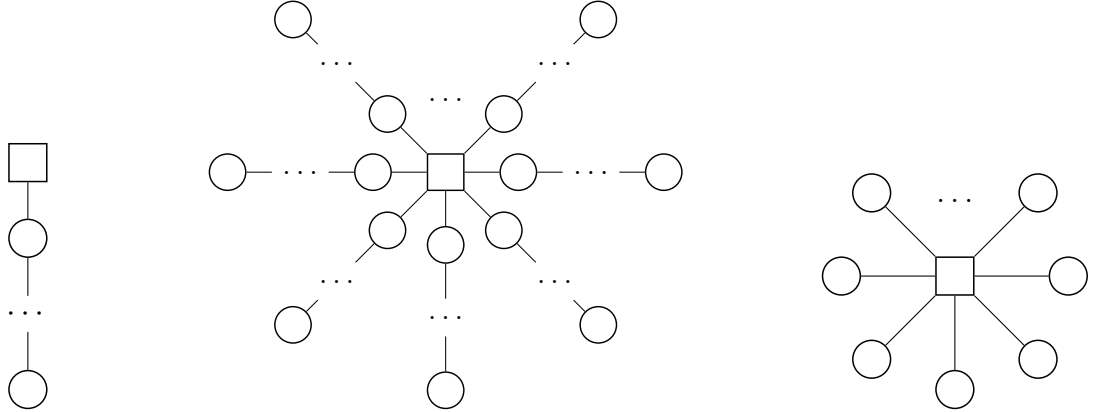


Figure 4.7: GTSP-rd has polynomial solutions to star and path, but not to subdivided star.

Despite the fact that the paths and the star with center as depot can be solved in polynomial time, the general subdivided star is an open question, with an exponential-time solution. By extending the recurrence relation used for paths, we can generalize the solution to the subdivided star. The idea is to consider routes that serve the vertices separately in each path emanating from central vertex. Hence a similar dynamic programming can be applied in this case with the complexity depending on the number k of paths emanating from the central vertex. For the GTSP-rd (time), this results in a complexity of $O(n^k)$, and for the GTSP-rd (distance), the complexity is $O(n^k \log n)$. For a fixed k the problem turns polynomial, which is the case of paths ($k \leq 2$).

Equation 4.24 show this generalization to GTSP-rd(time). Let $c(j_1, j_2, \dots, j_k)$ be

the minimum completion time to a solution containing only non-interlacing routes serving vertices $\{v_1^i, \dots, v_{j_i}^i\} \in N_i$ for all $i \in [1, k]$. In this case we consider the the set N_i as the vertices in the i -th path emanating from the central vertex. The base case is the following, $c(0^k) = 0$.

$$c(j_1, j_2, \dots, j_k) = \min\{F_1(j_1, j_2, \dots, j_k), \dots, F_k(j_1, j_2, \dots, j_k)\} \quad (4.24)$$

$F_i(j_1, j_2, \dots, j_k)$ (Equation 4.25) computes the minimum completion time if an non-interlacing route include the the vertex $v_{j_i}^i$ is executed in the i -th path emanating from the central vertex.

$$F_i(j_1, j_2, \dots, j_k) = \min \left\{ r_{j_i}^i + 2\tau_{k_{j_i}^i+1}^i, \min_{k_{j_i}^i < w < j_i} \{c(j_1, \dots, j_{i-1}, w, j_{i+1}, \dots, j_k) + 2\tau_{w+1}^i\} \right\} \quad (4.25)$$

This recurrence relation is a direct extension of Equation 4.5. An analogous extension can be applied to Equation 4.8. Let $\lambda(j_1, j_2, \dots, j_k)$ be the latest dispatch time to serve vertices $\{v_{j_i}^i, \dots, v_{n_i}^i\} \in N_i$ for all $i \in [1, k]$, only with non-interlacing routes. The base case is the following $\lambda(n_1 + 1, n_2 + 1, \dots, n_k + 1) = D$, just like in Equation 4.8. Equation 4.26 define the recurrence relation.

$$\lambda(j_1, j_2, \dots, j_k) = \max\{F_1(j_1, j_2, \dots, j_k), \dots, F_k(j_1, j_2, \dots, j_k)\} \quad (4.26)$$

$F_i(j_1, j_2, \dots, j_k)$ (Equation 4.27) computes the latest dispatch time if an non-interlacing route include the the vertex $v_{j_i}^i$ is executed in the i -th path emanating from the central vertex.

$$F_i(j_1, j_2, \dots, j_k) = \max_{w > j_i} \left\{ \lambda(j_1, \dots, j_{i-1}, w, j_{i+1}, \dots, j_k) \mid \lambda(j_1, \dots, j_{i-1}, w, j_{i+1}, \dots, j_k) - r_{w-1}^i \geq 2\tau_{j_i}^i \right\} - 2\tau_{j_i}^i \quad (4.27)$$

The same argument presented in Section 3.2.1 shows that the total distance traveled by the Salesman is $D - \lambda(1, 1, \dots, 1)$.

4.5.2 GTSP-rd on Catterpillars

While the results in Table 4.2 successfully resolve the **GTSP-rd** for Paths and Stars, it leaves an open question about Caterpillar and Lobster graphs. A Caterpillar graph can be seen as a Path with the radial geometry of multiple Stars. While this might suggest tractability, it might complicate to solve efficiently the problem with a Dynamic Programming (DP) algorithm.

The primary obstacle is the exponential combinatorial selection of the leaf vertices (legs) adjacent to different vertices along the spine. The Path DP proposed (Section 3.2.1) lies on the non-interlacing route property, which ensures that the unvisited customer vertices always form a simple contiguous segment. However, in a Caterpillar graph, this fundamental assumption breaks down. An optimal tour at a spine node v_i must make a complex decision: visit a subset of v_i 's legs immediately, skip some to be served later, and then decide whether to move to v_{i+1} or v_{i-1} (Figure 4.8).

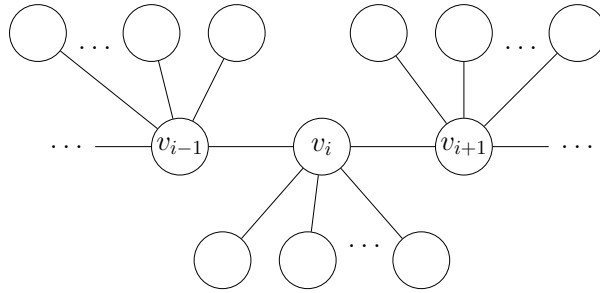


Figure 4.8: The spine with vertices $\dots, v_{i-1}, v_i, v_{i+1}, \dots$, each connected to multiple leaf vertices (legs). At node v_i , the tour must choose a subset of legs to visit before move to v_{i-1} or v_{i+1} .

The presence of unvisited legs attached to a visited segment of the spine results in a difficult way to keep the unvisited customers, the boundary between visited and unvisited customers is no longer defined by two simple endpoints. The DP state must now account not just for the spine segment endpoints, but also for the specific visitation status of the leaves at the boundary nodes.

An approach analogous to Held and Karp (1962) is employed to construct an exponential DP. Algorithm 8 solves the GTSP-rd(time) on caterpillar graphs. It utilizes the recursive function $\text{GTSPrdCatterpillar}(S, t)$, which calculates the minimum completion time required to serve the customers in the unserved set S , starting at time t . The overall minimum completion time is returned by the initial call $\text{GTSPrdCatterpillar}(N, 0)$. A memoization table, **Memo**, is maintained to cache the results for computed states (S, t) and

avoid unnecessary recomputations.

The time complexity of this algorithm depends critically on the magnitude of the time parameter t . Let T be the maximum possible completion time (and assumed to be discrete). The number of unique states in this dynamic programming is $O(2^n T)$, it came from the combination of 2^n subsets of customers and T discrete time points. Since each recursive call passes through all subsets $S' \subseteq S$ (leading to a total transition factor of $O(3^n)$) and executes a route calculation $\text{DFS_Route_Cost}(S')$ which takes $O(n)$ time on the caterpillar graph, the final time complexity is $O(3^n n T)$. This time complexity remains classified as exponential-time. An analogous solution, adjusting only the objective function, can be applied to the GTSP-rd(distance) on caterpillar graphs.

Algorithm 8: DP Algorithm for GTSP-rd(time) with Memoization

Input : n : number of customers; r_i : release dates; $d_{\{ij\}}$: travel time edges.
Output: Minimum completion time to execute all deliveries in N

- 1 **Global Memo** $\leftarrow$ initialized to $\perp$
- 2 **Function** GTSPrdCatterpillar(S, t):

- /* Base Case: No customers left */
 - 3 **if** $S = \emptyset$ **then**
 - 4 | **return** t
 - 5 **end**
 - /* Memoization Check */
 - 6 **if** $\text{Memo}[S, t] \neq \perp$ **then**
 - 7 | **return** $\text{Memo}[S, t]$
 - 8 **end**
 - 9
 - 10 $\text{min_completion} \leftarrow \infty$
 - /* Try all non-empty subsets */
 - 11 **foreach** $S' \subseteq S, S' \neq \emptyset$ **do**
 - 12 | $r_{\max} \leftarrow \max_{i \in S'} \{r_i\}$
 - 13 | $t_{\text{start}} \leftarrow \max(t, r_{\max})$
 - 14 |
 - /* Add routing cost (DFS) */
 - 15 | $\text{cost} \leftarrow \text{DFS_Route_Cost}(S')$
 - 16 | $t_{\text{arrival}} \leftarrow t_{\text{start}} + \text{cost}$
 - 17 |
 - 18 | $\text{val} \leftarrow \text{GTSPrdCatterpillar}(S \setminus S', t_{\text{arrival}})$
 - 19 | $\text{min_completion} \leftarrow \min(\text{min_completion}, \text{val})$
 - 20 **end**
 - 21
 - 22 $\text{Memo}[S, t] \leftarrow \text{min_completion}$
 - 23 **return** min_completion

Thus, an important question for future research is whether polynomial-time solv-

ability for Caterpillar or Lobster graphs can be achieved, or whether these structures fundamentally break the tractability observed in simpler path-like graphs.

Chapter 5

Concluding Remarks

In this dissertation, we addressed the Graphical Traveling Salesman Problem with release dates (GTSP-rd), which introduces temporal constraints to the Graphical Traveling Salesman Problem (GTSP). While recent works on the GTSP-rd does not mention GTSP, and solve the problem without the this definition, we revisited and extended the GTSP definition adapting it to handle with release dates.

We explore previous works about routing problems on graph classes, such as VRP, GTSP and GTSP-rd, and we analyzed the graph classes and algorithmic approaches used to address these problems. Based on algorithms proposed by Archetti et al. (2015) and Reyes et al. (2018) for solving the GTSP-rd on paths, we introduced improvements for both variants of the problem: GTSP-rd (time) and GTSP-rd (distance). Our results demonstrate that GTSP-rd (time) can now be solved in $O(n^2)$, and GTSP-rd (distance) in $O(n^2 \log n)$ for any path.

We also provide new polynomial-time algorithms for GTSP-rd on stars and cycles, showing that these graph classes are solvable in polynomial time for both variations of the problem, GTSP-rd(distance) and GTSP-rd(time). The algorithms were build using dynamic programming technique and recurrence relations. To star graphs the time complexities achieved were $O(n^2)$ and $O(n^3)$ for GTSP-rd(distance) and GTSP-rd(time), respectively. And on cycle graphs the algorithms achieves $O(n^3)$ for both variations of the problem.

The GTSP-rd showed to be more difficult than the GTSP; the inclusion of release dates appears to make the problem nontrivial even on a tree. As future works, several direction can be identified. One natural question is whether the GTSP-rd on others path-

like graphs, starting with caterpillars and lobsters graphs, can be solved in polynomial time or whether these cases are NP-hard. Clarifying this issue may help to further characterize the computational complexity of the problem. Other avenue is to try to prove the proposed algorithms optimality or improve the upper bound complexity to solve the GTSP-rd in path-like graph classes.

The GTSP-rd showed to be more difficult than the GTSP, the introduction of release dates constraints turn the problem nontrivial even on tree graphs. The temporal constraints fundamentally modify the structural properties that usually make the GTSP easy on restricted graph classes. Several directions for future work can be identified. A natural question is whether the GTSP-rd on caterpillar and lobster graphs admits polynomial-time algorithms or whether these cases become NP-hard due to the interaction between graph topology and release date constraints. Solve this question would contribute to a better characterization of the computational complexity boundary for the problem. Another promising avenue is the refinement of the algorithms proposed in this work, as the current solutions may not be tight bounds for the corresponding graph classes and could potentially be improved through tighter dynamic programming formulations or alternative algorithmic techniques.

Some of the results discussed in Sections 4.1 and 4.2 were previously presented at the *IX Encontro de Teoria da Computação* (Clementino and Freitas, 2024), and in the 11th Latin American Workshop on Cliques in Graphs with a special issue published at Clementino and de Freitas (2025a). Additionally, a preprint of an article on GTSP-rd on paths, submitted to the *RAIRO-Operations Research*, is available at Clementino and de Freitas (2025b).

In conclusion, this Master dissertation contributes to the ongoing effort to understand of the complexity landscape of the GTSP-rd. By presenting efficient algorithms for certain graph classes, it provides a basis for subsequent studies and indicates directions for next steps with more general and challenging graph classes or generalizations of the GTSP-rd.

Acknowledgments. I would like to thank Loggi for introducing me to routing problems through the *Programa de Bolsas de Pesquisa (PBP)* da Loggi, and for the interaction with the company made possible by Professor José Carlos Maldonado. I would also like to thank Professor Eduardo Uchoa, with whom I had the opportunity to attend lectures

on column generation. I am especially grateful to the LAWCG 2022 conference, held in celebration of Jayme Szwarcfiter's 80th birthday, where the motivation for this work first emerged.

This research was partially supported by the Coordination for the Improvement of Higher Education Personnel - Brazil (CAPES-PROEX) - Funding Code 001, the National Council for Scientific and Technological Development (CNPq), and the Amazonas State Research Support Foundation - FAPEAM - through the POSGRAD 2024-2025 project. Also, under Brazilian Federal Law No. 8,387/1991, Motorola Mobility partially sponsored this research through the SWPERFI Research, Development, and Technological Innovation Project on intelligent software performance and through agreement No. 004/2021, signed with UFAM. The authors are part of the Algorithms, Optimization, and Computational Complexity (ALGOX) CNPq research group from the Postgraduate Program in Computer Science (PPGI), IComp/UFAM.

References

- Archetti, C., Feillet, D., Gendreau, M., and Speranza, M. G. (2011). Complexity of the VRP and SDVRP. *Transportation Research Part C: Emerging Technologies*, 19(5):741–750.
- Archetti, C., Feillet, D., and Speranza, M. G. (2015). Complexity of routing problems with release dates. *European Journal of Operational Research*, 247(3):797–803.
- Becker, A. a. (2018). A tight $4/3$ approximation for capacitated vehicle routing in trees. *arXiv preprint arXiv:1804.08791*.
- Bermond, J.-C. (1979). Graceful graphs, radio antennae and french windmills. In *Proceedings One day Combinatorics Conference, Research Notes in Mathematics, Pitman*, volume 34, pages 18–37. Pitman.
- Bock, S., Bomsdorf, S., Boysen, N., and Schneider, M. (2025). A survey on the traveling salesman problem and its variants in a warehousing context. *European Journal of Operational Research*, 322(1):1–14.
- Bondy, J. A. and Murty, U. S. R. (2008). *Graph theory*. Springer Publishing Company, Incorporated.
- Brass, P. (2008). *Advanced data structures*, volume 193. Cambridge university press Cambridge.
- Brodal, G. S. (2013). A survey on priority queues. In *Space-Efficient Data Structures, Streams, and Algorithms: Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 150–163. Springer.
- Busch, I. K. (1991). *Vehicle routing on acyclic networks*. The Johns Hopkins University.

- Carr, R., Ravi, R., and Simonetti, N. (2023). A new integer programming formulation of the graphical traveling salesman problem. *Mathematical Programming*, 197(2):877–902.
- Castro, L., Clementino, T., and de Freitas, R. (2025). Implementation and brief experimental analysis of the duan et al.(2025) algorithm for single-source shortest paths. *arXiv preprint arXiv:2511.03007*.
- Chandran, B. and Raghavan, S. (2008). Modeling and solving the capacitated vehicle routing problem on trees. *The vehicle routing problem: latest advances and new challenges*, pages 239–261.
- Clementino, T. and de Freitas, R. (2025a). Complexity on paths for the graphical TSP with release dates. *Matemática Contemporânea*, pages 1–8.
- Clementino, T. and de Freitas, R. (2025b). Improving polynomial bounds for the graphical traveling salesman problem with release dates on paths. *arXiv preprint arXiv:2502.02680*.
- Clementino, T. and Freitas, R. (2024). Reduzindo para complexidade linear a solução do TSP com datas de liberação em caminhos com depósito na extremidade. In *Anais do IX Encontro de Teoria da Computação*, pages 81–85. SBC.
- Cook, W. J. (2015). *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton University Press.
- Cornuéjols, G., Fonlupt, J., and Naddef, D. (1985a). The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical programming*, 33(1):1–27.
- Cornuéjols, G., Naddef, D., and Pulleyblank, W. (1985b). The traveling salesman problem in graphs with 3-edge cutsets. *Journal of the ACM (JACM)*, 32(2):383–410.
- Cornuéjols, G., Naddef, D., and Pulleyblank, W. R. (1983). Halin graphs and the traveling salesman problem. *Mathematical programming*, 26(3):287–294.
- Costa, J., Castro, L., and de Freitas, R. (2025). Exploring monotone priority queues for dijkstra optimization. *RAIRO-Operations Research*, 59(5):2419–2436.
- Fonlupt, J. and Nachev, A. (1993). Dynamic programming and the graphical traveling salesman problem. *Journal of the ACM (JACM)*, 40(5):1165–1187.

- Fonlupt, J. and Naddef, D. (1992). The traveling salesman problem in graphs with some excluded minors. *Mathematical Programming*, 53:147–172.
- Garey, M. R. and Johnson, D. S. (1979). *Computers and intractability*, volume 174. freeman San Francisco.
- Goyal, K. and Mömke, T. (2020). Robust reoptimization of steiner trees. *Algorithmica*, 82(7):1966–1988.
- Grötschel, M., Lovász, L., and Schrijver, A. (1981). The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197.
- Hargrave, W. and Nemhauser, G. (1962). On the relation between the travelling salesman problem and the longest path problem. *Operations Research*, 10:647–657.
- Held, M. and Karp, R. M. (1962). A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied mathematics*, 10(1):196–210.
- Johnson, C. R., Duarte, A. L., and Saiago, C. M. (2003). Inverse eigenvalue problems and lists of multiplicities of eigenvalues for matrices whose graph is a tree: the case of generalized stars and double generalized stars. *Linear Algebra and its Applications*, 373:311–330.
- Jünger, M., Reinelt, G., and Rinaldi, G. (1995). The traveling salesman problem. *Handbooks in operations research and management science*, 7:225–330.
- Khare, A., Batta, R., and Kang, J. E. (2021). On the analysis of last-mile relief delivery on a tree network: Application to the 2015 nepal earthquake. *Journal of the Operational Research Society*, 72(4):727–743.
- Labbé, M., Laporte, G., and Mercure, H. (1991). Capacitated vehicle routing on trees. *Operations Research*, 39(4):616–622.
- Lüke, L., Heßler, K., and Irnich, S. (2024). The single picker routing problem with scattered storage: modeling and evaluation of routing and storage policies. *OR Spectrum*, 46(3):909–951.

- Mathieu, C. and Zhou, H. (2022). A tight $(1.5 + \epsilon)$ -approximation for unsplittable capacitated vehicle routing on trees. *arXiv preprint arXiv:2202.05691*.
- Mathieu, C. and Zhou, H. (2023). A PTAS for capacitated vehicle routing on trees. *ACM Transactions on Algorithms*, 19(2):1–28.
- Miliotis, P., Laporte, G., and Nobert, Y. (1981). Computational comparison of two methods for finding the shortest complete cycle or circuit in a graph. *RAIRO-Operations Research*, 15(3):233–239.
- Montero, A., Méndez-Díaz, I., and Miranda-Bront, J. J. (2023). Solving the traveling salesman problem with release dates via branch and cut. *EURO Journal on Transportation and Logistics*, 12:100121.
- Ratliff, H. D. and Rosenthal, A. S. (1983). Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. *Operations research*, 31(3):507–521.
- Reyes, D., Erera, A. L., and Savelsbergh, M. W. (2018). Complexity of routing problems with release dates and deadlines. *European Journal of Operational Research*, 266(1):29–34.
- Rosas, J. S. (2025). Explorando o VRPTW na logística de entrega urbana expressa de Última milha usando grandes instâncias reais de cidades brasileiras. Dissertação de mestrado, Universidade Federal do Amazonas, Manaus, AM.
- Sebő, A. and Vygen, J. (2014). Shorter tours by nicer ears: $7/5$ -approximation for the graph-tsp, $3/2$ for the path version, and $4/3$ for two-edge-connected subgraphs. *Combinatorica*, pages 1–34.
- Sundar, R. (1989). Worst-case data structures for the priority queue with attrition. *Information processing letters*, 31(2):69–75.
- Szwarcfiter, J. L. (2018). *Teoria computacional de grafos: Os Algoritmos*. Elsevier Brasil, Rio de Janeiro, 1 edition.
- Thorup, M. (2000). On RAM priority queues. *SIAM Journal on Computing*, 30(1):86–109.
- Vazirani, V. V. (2001). *Approximation algorithms*, volume 1. Springer.

- West, D. B. (2001). *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River.
- Williams, J. W. J. (1964). Algorithm 232: Heapsort. *Communications of the ACM*, 7(6):347–348.
- Yang, F., Dai, Y., and Ma, Z.-J. (2020). A cooperative rich vehicle routing problem in the last-mile logistics industry in rural areas. *Transportation Research Part E: Logistics and Transportation Review*, 141:102024.