Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

The Generalized Travelling Salesman Problem in the Polygonal Domain

Štěpán Holejšovský

Supervisor: RNDr. Miroslav Kulich, Ph.D
Field of study: Open Informatics
Subfield: Artificial Intelligence and Computer Science
May 2025

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Holejšovský Štěpán** Personal ID number: **516205**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

The Generalized Travelling Salesman Problem in the Polygonal Domain

Bachelor’s thesis title in Czech:

Zobecněný problém obchodního cestujícího v polygonální doméně

Guidelines:

In the Generalized Traveling Salesman Problem (GTSP), we are given a set of cities grouped into possibly intersecting clusters. The objective is to find a closed path of minimum cost that visits at least one city in each cluster. The aim of the thesis is twofold. First, the state of the art metaheuristic GLNS will be extended with an operator inspired by the Gutin neighborhood. Second, the student will study recent shortest path finding approaches and their usage in GLNS for problems in the polygonal domain. The student's tasks are the following:

1. Get acquainted with modern shortest path finding approaches [1,2,3,4].
2. Modify the method [4] so it computes all pairs shortest paths effectively.
3. Compare the modified method with the shortest path finder in the Třivis library [7].
4. Get acquainted with the Generalized Travelling Salesman Problem (GTSP) and state-of-the-art metaheuristics for routing problems.
5. Extend the GLNS solver [6] with the Gutin neighborhood. This consists of two steps. First, design and realize various destroy operators taking into account specifics of the Gutin neighborhood. Second, implement Gutin operator as a new repair operator.
6. Extend the GLNS solver to work in the polygonal domain by utilization of a chosen shortest path finder.
7. Evaluate experimentally properties of the extended GLNS. Describe and discuss the obtained results.

Bibliography / sources:

[1] Ye Li, Leong Hou U, Man Lung Yiu, and Ngai Meng Kou. 2017. An experimental study on hub labeling based shortest path algorithms. Proc. VLDB Endow. 11, 4 445–457. https://doi.org/10.1145/3186728.3164141
[2] Michael L. Cui, Daniel D. Harabor, and Alban Grastien. 2017. Compromise-free pathfinding on a navigation mesh. In Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17). AAAI Press, 496–502.https://www.ijcai.org/proceedings/2017/70
[3] Bojie Shen, Muhammad Aamir Cheema, Daniel D. Harabor, and Peter J. Stuckey. 2021. Euclidean pathfinding with compressed path databases. In Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI'20). Article 584, 4229–4235. https://www.ijcai.org/proceedings/2020/584
[4] Jinchun Du, Bojie Shen, and Muhammad Aamir Cheema. 2023. Ultrafast Euclidean shortest path computation using hub labeling. In Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence (AAAI'23/IAAI'23/EAAI'23), Vol. 37. AAAI Press, Article 1394, 12417–12426. https://ojs.aaai.org/index.php/AAAI/article/view/26463.
[5] Jeanette Schmidt, Stefan Irnich, New neighborhoods and an iterated local search algorithm for the generalized traveling salesman problem, EURO Journal on Computational Optimization, Volume 10, 2022, 100029, ISSN 2192-4406, https://doi.org/10.1016/j.ejco.2022.100029.
[6] Stephen L. Smith, Frank Imeson, GLNS: An effective large neighborhood search heuristic for the Generalized Traveling Salesman Problem, Computers & Operations Research, Volume 87, 2017, Pages 1-19, ISSN 0305-0548, https://doi.org/10.1016/j.cor.2017.05.010.
[7] J. Mikula, M. Kulich and L. Přeučil, "TřiVis: Versatile, Reliable, and High-Performance Tool for Computing Visibility in Polygonal Environments," 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Abu Dhabi, United Arab Emirates, 2024, pp. 10503-10510, https://doi.org/ 10.1109/IROS58592.2024.10801476.

Name and workplace of bachelor's thesis supervisor:

RNDr. Miroslav Kulich, Ph.D. Intelligent and Mobile Robotics CIIRC

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **29.01.2025** Deadline for bachelor thesis submission: __________

Assignment valid until: **20.09.2026**

prof. Dr. Ing. Jan Kybic
Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Vice-dean´s signature on behalf of the Dean

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student's signature

FAKULTA ELEKTROTECHNICKÁ
FACULTY OF ELECTRICAL ENGINEERING
Technická 2
166 27 Praha 6

DECLARATION

I, the undersigned

Student's surname, given name(s): Holejšovský Štěpán
Personal number: 516205
Programme name: Open Informatics

declare that I have elaborated the bachelor's thesis entitled

The Generalized Travelling Salesman Problem in the Polygonal Domain

independently, and have cited all information sources used in accordance with the Methodological Instruction on the Observance of Ethical Principles in the Preparation of University Theses and with the Framework Rules for the Use of Artificial Intelligence at CTU for Academic and Pedagogical Purposes in Bachelor's and Continuing Master's Programmes.

I declare that I used artificial intelligence tools during the preparation and writing of this thesis. I verified the generated content. I hereby confirm that I am aware of the fact that I am fully responsible for the contents of the thesis.

In Prague on 21.05.2025

Štěpán Holejšovský
....
student's signature

Acknowledgements

I would like to express my gratitude to my supervisor, RNDr. Miroslav Kulich, Ph.D., for his valuable guidance, regular consultations, and patience throughout the course of my thesis. I would also like to thank Ing. Jan Mikula for generously providing me with datasets and code on multiple occasions, as well as for his time.

Abstract

This thesis focuses on solving the Generalized Traveling Salesman Problem (GTSP) in polygonal environments with obstacles. The main contributions are twofold: (1) the integration of new heuristic operators into the GLNS metaheuristic GTSP solver, inspired by the Gutin Neighborhood, and (2) the adaptation of the Euclidean Hub Labeling (EHL) algorithm for efficient all-pairs shortest path computations within polygonal domains.

Keywords: Generalized Traveling Salesman Problem, GTSP, polygonal environment, shortest path, visibility graph, Euclidean Hub Labeling, TřiVis, GLNS solver, Gutin neighborhood, metaheuristic, Watchman Route Problem, path planning, combinatorial optimization, cluster routing

Supervisor: RNDr. Miroslav Kulich, Ph.D
Intelligent and Mobile Robotics, CIIRC,
Jugoslávských partyzánů 1580/3,
160 00 Praha 6

Abstrakt

Tato práce se zaměřuje na řešení zobecněného problému obchodního cestujícího (GTSP) v polygonálním prostředí s překážkami. Hlavní přínosy jsou dva: (1) integrace nových heuristických operátorů do metaheuristického řešitele GTSP GLNS, inspirovaného Gutinovým okolím, a (2) adaptace algoritmu Euclidean Hub Labeling (EHL) pro efektivní výpočty nejkratších cest všech párů v polygonálních oblastech.

Klíčová slova: Zobecněný Problém Obchodního Cestujícího, GTSP, polygonální prostředí, nejkratší cesta, graf viditelnosti, Eucldian Hub Labeling, TřiVis, GLNS solver, Gutinovo okolí, metaheuristika, Problém Hledání Cesty Hlídače, plánování cest, kombinatorická optimalizace, shlukové směrování

Překlad názvu: Zobecněný problém obchodního cestujícího v polygonální doméně

Contents

Figures

Tables

Chapter 1

Introduction

The Generalized Traveling Salesman Problem (GTSP) is a natural extension of the well-known Traveling Salesman Problem (TSP). In the classical TSP, an agent seeks the shortest possible tour that visits every city in a given set exactly once and returns to the starting point. In contrast, GTSP generalizes this by partitioning the cities into *clusters*, with the objective of finding the shortest tour that visits exactly one city from each cluster.

Both TSP and GTSP are well-established NP-hard combinatorial optimization problems. However, GTSP presents an additional layer of complexity due to the cluster-based selection constraint. As a result, computing exact solutions for large GTSP instances is often computationally infeasible. This motivates the development of efficient, near-optimal heuristics and metaheuristics.

These problems have been extensively studied since the 1970s, resulting in the development of a wide array of both exact and heuristic solution approaches. Their versatility and practical relevance have led to widespread application in numerous domains.

For example, in logistics and supply chain management, they are critical for optimizing transportation routes and warehouse operations. In robotics, they help with efficient route planning and task scheduling. Similarly, in delivery and routing systems, these approaches help minimize costs and improve service efficiency. Beyond these areas, GTSP has also found applications in biomedicine and electronic circuit design [1].

1.1 Motivation for Shortest Paths in GTSP Context

In order to apply GTSP solving methods within polygonal environments, such as warehouses or indoor robotic navigation, an essential prerequisite is the construction of a cost matrix that reflects the true distances between candidate vertices. While traditional GTSP instances often use Euclidean distances or precomputed costs, these metrics become insufficient when obstacles are present.

In polygonal domains, the cost between any two vertices must account for obstacle-avoiding navigation and is therefore defined by the length of the shortest path within the free space. Consequently, the cost matrix must be derived from all pairs shortest path computations that honor the visibility and connectivity constraints of the environment [2].

This introduces a significant computational challenge. For large-scale instances, constructing the all-pairs shortest paths cost matrix may become more time-consuming than solving the GTSP itself. Efficient pathfinding is thus not merely a preprocessing step, but a core computational bottleneck. Addressing this challenge is critical to enabling scalable GTSP solutions in real-world geometric environments and is therefore a central component of this thesis.

1.2 Goals and Thesis Structure

The primary objective of this thesis is twofold. First, it aims to extend the Metaheuristic GTSP solver developed by Smith and Imeson called GLNS [3], by incorporating new operators inspired by the Gutin Neighborhood. These operators are designed to improve the destroy and repair search mechanism of the solver, enabling it to explore more diverse regions of the solution.

Second, the thesis focuses on the efficient computation of shortest paths in polygonal domains, a critical preprocessing step for constructing the cost matrix required by GTSP solvers in environments with obstacles. To address this, a recent algorithm by Du et al. [4], based on Euclidean Hub Labeling, is adapted to support distance queries of all pairs among a set of points. The performance of this method is compared against the TřiVis library [5], a well-established tool for computations in polygonal environments.

The thesis is organized into four main parts. The first part introduces the Generalized Traveling Salesman Problem and describes the GLNS solver, including the modifications made to its search operators. The second part focuses on the problem of finding shortest paths in polygonal environments. A specific algorithm, based on Euclidean Hub Labeling, is adapted to support repeated distance queries between multiple pairs of points.

The third and fourth parts present the experiments that evaluate our approaches. The third part presents a comparison between the adapted algorithm and the TřiVis library, with a focus on performance in terms of computation time. The fourth part evaluates the extended GLNS solver using benchmark instances. It also includes results from experiments combining the solver with the geometric preprocessing pipeline, demonstrating how GTSP can be applied to problems involving obstacles and visibility constraints.

Chapter 2

Problem Formulation

2.1 Shortest Path in a Polygonal Domain Formulation

Let us formulate the problem of finding the shortest paths in a polygonal domain, we adopt terminology and concepts based on the work of Shen et al. [6].

We consider the problem of computing shortest paths within a polygonal domain containing polygonal obstacles. Two points are considered *visible* if the straight line segment connecting them lies entirely in the free (obstacle-free) space of the domain.

A valid path is a sequence of points

$$P = \langle p_1, p_2, \ldots, p_k \rangle,$$

where each consecutive pair (p_i, p_{i+1}) is mutually visible.

The total cost of the path is given by the sum of pairwise Euclidean distances between adjacent points, denoted by a distance function $d(p_i, p_{i+1})$:

$$\mathrm{cost}(P) = \sum_{i=1}^{k-1} d(p_i, p_{i+1}).$$

To find the shortest path, we seek a path P that minimizes this total cost.

2.2 GTSP Formulation

Let us also formulate the Generalized Traveling Salesman Problem (GTSP), which we solve via GLNS metaheuristics incorporating new operators. This formulation is based on the work of Schmidt and Irnich [1].

The Generalized Traveling Salesman Problem (GTSP) is a generalization of the classical Traveling Salesman Problem (TSP). In the standard TSP, a salesman seeks to find the shortest possible tour that visits each city (vertex) exactly once and returns to the starting point. The problem is well-known for its combinatorial complexity and has been extensively studied in operations research and computer science.

In contrast, the GTSP modifies the visitation requirement: instead of visiting all vertices in the graph, the goal is to visit exactly one vertex from each predefined group of vertices, referred to as clusters. The resulting sequence of selected vertices forms a *tour*, which is a closed path that starts and ends at the same vertex and visits exactly one vertex from each cluster in some order. This generalization retains the NP-hard nature of the TSP.

Formally, consider a complete undirected graph $G = (V, E)$, where V is the set of vertices and E the set of edges, each associated with a non-negative weight d_{vw} for every pair of vertices $v, w \in V$. The vertex set V is partitioned into N non-empty, mutually disjoint subsets called *clusters*, denoted as $C_1, C_2, \ldots, C_N$. These clusters satisfy:

$$V = \bigcup_{i=1}^{N} C_i, \quad \text{with} \quad C_i \cap C_j = \emptyset \quad \text{for } i \neq j.$$

The objective in GTSP is to determine a cycle, often referred to as a *G-tour*, that includes exactly one vertex from each cluster, such that the total length of the tour is minimized. The cost of such a solution x is given by:

$$\text{cost}(x) = \sum_{(v,w) \in x} d_{vw},$$

where x represents the selected cycle visiting one vertex from each cluster.

It should be noted that GTSP clusters are not required to reflect geographical or spatial proximity. Instead, they can represent any logical grouping of vertices, based on problem-specific constraints or structural properties.

Chapter 3

Generalized Travelling Salesman Problem

This chapter provides an overview of methods for solving the GTSP, ranging from classical exact algorithms to modern metaheuristic approaches. We begin by summarizing existing solution strategies and highlighting the limitations of exact methods for large-scale problem instances. We then focus on heuristic and metaheuristic techniques, with special attention to the Generalized Large Neighborhood Search (GLNS) algorithm, which is currently one of the most effective heuristic solvers for GTSP.

Subsequently, we describe the inner workings of the GLNS algorithm, detailing its adaptive heuristic selection, removal and insertion operators, and mechanisms for intensification and diversification. We then introduce our proposed extension to the GLNS framework, which integrates a new family of removal heuristics based on the Gutin neighborhood structure, as well as a novel repair operator that leverages the Hungarian algorithm to perform optimal reinsertion.

3.1 Overview of Existing Methods

Since its introduction in the late 1960s, the GTSP has been extensively studied. Several exact approaches have been developed to solve GTSP, including the branch-and-cut algorithm by Fischetti et al. [7] and the Lagrangian-based method for asymmetric GTSP instances proposed by Noon and Bean [8]. While these exact methods have proven effective for instances with up to 100 vertices, finding optimal solutions for larger instances remains a significant challenge even today [1].

Due to the computational complexity of exactly solving large-scale GTSP instances, a variety of heuristic approaches have been proposed in the literature. These heuristics aim to produce high-quality, near-optimal solutions within reasonable computational times, often by exploiting problem-specific characteristics [3].

More advanced of these heuristics combine tour construction with iterative local improvement strategies, typically employing neighborhood-search techniques such as 2-opt or swap moves adapted for the GTSP context. These local search heuristics systematically explore the neighborhood of the solution by making incremental changes and maintaining improvements. Although capable of rapidly improving initial solutions, they are prone to getting trapped in local optima [1].

To mitigate this, many heuristics-based GTSP solvers incorporate metaheuristic frameworks, such as Genetic Algorithms, which guide the heuristic search. In this thesis, we extended one of such metaheuristic frameworks called the Generalized Large Neighborhood Search [3].

The Generalized Large Neighborhood Search (GLNS) solver represents one of the most effective and versatile metaheuristic approaches for GTSP to date. Building on the principles of Adaptive Large Neighborhood Search (ALNS), GLNS iteratively destroys a portion of the current tour and repairs it in order to find a cheaper solution.

At each iteration, a subset of vertices is selectively removed from the current tour using various removal heuristics, followed by the reinsertion of vertices using insertion heuristics designed to preserve feasibility by ensuring that each cluster is visited exactly once. The selection of these heuristics is managed adaptively through a weighting system, where heuristics that contribute to better solutions are favored more frequently in future iterations. This adaptive component allows GLNS to dynamically balance intensification and diversification dynamically, avoiding premature convergence to local optima while steadily improving the quality of the solution.

Furthermore, GLNS integrates classic insertion heuristics from the TSP literature, generalized for the clustered structure of GTSP, and incorporates randomization to improve the exploration of the solution space. Its flexibility and robust performance on various GTSP instances have established GLNS as a solid heuristic framework and provide a strong foundation for further extensions and hybridizations, which this thesis seeks to develop [3].

3.2 GLNS Algorithm Description

Let us provide a pseudocode for the GLNS Algorithm 1, as presented in [3]. The algorithm starts with an initial feasible tour (line 2) and iteratively applies removal and insertion operators to explore the solution space. For each removal phase, a random number of vertices N_r is selected, where N_r is drawn from the range $\{1, \ldots, N_{\max}\}$. A removal heuristic R and an insertion heuristic I are chosen (line 5) using adaptive selection weights, which are updated based on the historical performance of each heuristic (line 17). The removal heuristic removes N_r vertices from the tour (line 8), and the insertion heuristic reinserts them (line 9) while ensuring feasibility, specifically that exactly one vertex per cluster is selected. The algorithm maintains the best solution found across iterations, progressively improving the quality of the tour (line 12). The trials (line 1) are of two types: an initial descent and a series of warm restarts. Each phase terminates when no improvement has been observed for a predefined number of iterations, as reflected in line 16.

GLNS supports three operational modes, *Fast*, *Medium*, and *Slow*, which adjust the algorithm's search depth and computational effort. These modes primarily control the total number of trials and the number of iterations allowed per trial.

Moving on, we summarize the GLNS **unified insertion** operator. This heuristic provides a flexible and generalized framework for reinserting vertices into a partial GTSP tour. The GTSP instance is defined over a partitioned vertex set $P_V = \{V_1, V_2, \ldots, V_m\}$, where each V_i is a cluster, and the objective is to construct a tour that includes exactly one vertex from each cluster. Given a partial tour T that currently includes a subset of clusters $P_T \subset P_V$, the operator aims to reinsert one of the remaining unvisited clusters $V_i \in P_V \setminus P_T$.

For each such candidate cluster V_i, the algorithm calculates the minimum insertion cost d_i, defined as the shortest distance from any vertex in V_i to any point in the current tour T. Then a cluster V_i is probabilistically selected according to an unnormalized probability mass function that is controlled by a parameter λ. When λ lies between 0 and 1, the selection process favors clusters with smaller d_i values, corresponding to nearest insertions. If λ is greater than 1, the algorithm instead favors clusters with larger d_i

Algorithm 1: GLNS Algorithm (based on [3])

Input: A GTSP instance
Output: A GTSP tour
for $i = 1$ *to num_trials* **do**
  $T \leftarrow initial\_tour(G, P_V)$;
  $T_{best} \leftarrow T$;
  repeat
    Randomly choose removal heuristic R and insertion heuristic I from weighted data;
    Randomly choose number of vertices to remove N_r;
    $T_{new} \leftarrow T$;
    Remove N_r vertices from T_{new} using R;
    Insert vertices into T_{new} using I;
    if $w(T_{new}) < w(T_{best})$ **then**
      Optimize T_{new} vertex selections;
      $T_{best} \leftarrow T_{new}$;
    if $accept(T_{new}, T)$ **then**
      $T \leftarrow T_{new}$;
    Update heuristic performance records;
  until *stop criterion is met* ;
  Update heuristic selection weights;
return T_{best} *with minimal* $w(T_{best})$;

values, corresponding to the farthest insertions. When λ is equal to 1, all remaining clusters are treated uniformly, resulting in a random selection. This mechanism allows for a tunable balance between exploitation and exploration during tour construction.

The insertion is also crucial for constructing the **initial tour**, as the initial tour is built either completely randomly or by first selecting a random starting vertex and then iteratively applying the unified insertion heuristic until all clusters have been included in the tour.

Next we will present two key **removal operators** used in GLNS, which this thesis further hybridizes: the *Unified Worst Removal* and the *Distance Removal* heuristics. Both of these removal heuristics operate by first sorting the nodes in the tour according to a specific criterion and then selecting which node to remove using a semi-randomized procedure. This selection process is governed by the parameter λ in the same way as it is used during the insertion operator; this controls the balance between random and deterministic removal. With some probability, the algorithm performs a purely random selection (if a simulated random draw exceeds the accumulated weights), while otherwise it selects a node based on the weighted ranking derived from the sorting criterion.

The **Unified Worst Removal** heuristic aims to eliminate vertices that most negatively impact the current tour cost. Specifically, it considers the cost change when removing a vertex v_{remove} as:

$$c(v_{-1}, v_{\text{remove}}) + c(v_{\text{remove}}, v_{+1}) - c(v_{-1}, v_{+1}), \tag{3.1}$$

where v_{-1} and v_{+1} are the neighbors of v_{remove} and $c(\cdot, \cdot)$ denotes the cost of the path between two tour vertices. The idea is to identify vertices whose deletion most improves the tour, effectively maximizing the benefit derived from this marginal cost formula.

The **Distance Removal** heuristic is motivated by the idea that vertices close to each other may belong to the same problematic region of the solution. The procedure begins by randomly selecting an initial vertex to remove and adding it to both the removal set and the seed set. In subsequent iterations, additional vertices are selected for removal based on their proximity to a randomly chosen vertex from the seed set.

In addition to removal and insertion operations, GLNS incorporates two local improvement subroutines: *Re-Opt* and *Move-Opt* (line 11 in Algorithm 1). The *Re-Opt* method re-optimizes the vertex selection within each cluster while keeping the cluster order fixed. However, in our implementation, we employ the fast variant of GLNS, which does not utilize the *Re-Opt* method. The Move-Opt method seeks to improve the tour by optimizing the order of clusters: it randomly selects a vertex from the tour, removes it, and reinserts a vertex from the same cluster with minimized insertion cost. Both subroutines contribute to further refining the GTSP solution.

3.3 Gutin Neighborhood Removal Operator

In our modification of the GLNS algorithm, we introduce a novel removal heuristic inspired by the *Gutin neighborhood*, originally proposed by Gutin et al. [9] for the classical Traveling Salesman Problem. While the Gutin neighborhood has also been applied in the context of the Generalized Traveling Salesman Problem (GTSP) by Schmidt and Irnich [1], their approach integrates it within an Iterated Local Search (ILS) framework, where it is used as a large neighborhood search mechanism that jointly reorders clusters and reselects representatives. In contrast, our approach adapts the Gutin neighborhood as a *removal operator* within the metaheuristic paradigm of GLNS [3].

The core idea is to define a subset $Z \subset \{1, 2, \ldots, N\}$ of positions in the tour to be removed, such that for all $i, j \in Z$ with $i \neq j$, the positions x_i and x_j are nonadjacent in the tour $x = (x_1, x_2, \ldots, x_N, x_1)$. By ensuring non-adjacency among selected positions, hole insertions can be tracked individually, supporting the possibility of locally optimal solutions.

We implement three removal operators to construct the subset Z. Algorithm 2 outlines the method originally proposed in [1]. Here the operator initializes $Z = \emptyset$ (Line 1) and iterates through each tour position $i = 1, \ldots, N$ (Line 2), where we assume $i_N = i_0$ to close the tour. For each i, if the previous position $i - 1 \in Z$, we skip i to preserve the non-adjacency requirement (Line 3). Otherwise, we include i in Z with a probability of 0.5, simulating a fair coin flip (Lines 5–7). Additionally, to maintain non-adjacency over the entire tour, we ensure that the first and last positions are not both included in Z (Lines 8–9).

The other two operators, which we introduce in this work, adapt the selection logic from existing GLNS removal heuristics, namely, Unified Worst Removal and Distance Removal, while enforcing the additional constraint that no two adjacent positions in the tour may be included in Z.

Algorithm 2: Gutin Neighborhood Removal Operator

Input: Tour $x = (x_1, x_2, \ldots, x_N, x_1)$
Output: Subset $Z \subset \{1, 2, \ldots, N\}$ of non-adjacent positions

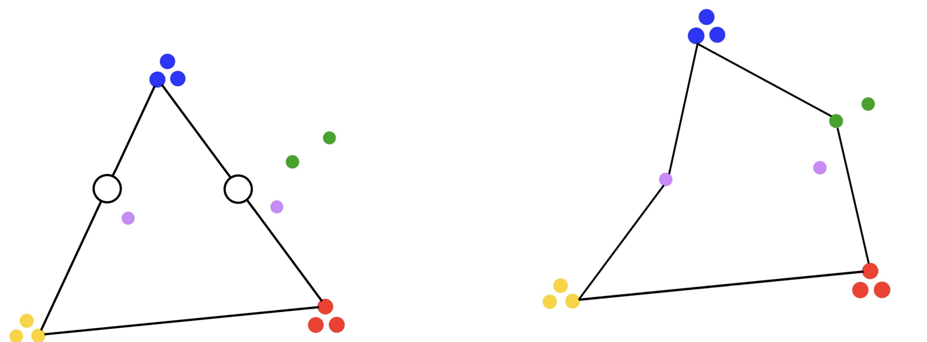
1 $Z \leftarrow \emptyset$;
2 **for** $i = 1$ *to* N **do**
3   **if** $(i - 1) \in Z$ **then**
4   continue;
5   Flip a fair coin;
6   **if** *heads* **then**
7         $Z \leftarrow Z \cup \{i\}$;
8 **if** $1 \in Z$ *and* $N \in Z$ **then**
9   Remove N from Z;
10 **return** Z

The second operator, *Gutin Distance Removal*, adapts the principle of the Distance Removal heuristic to the Gutin neighborhood structure. The operator begins by randomly selecting a seed position $i_0 \in \{1, \ldots, N\}$, which is added to Z. It then proceeds iteratively to select additional positions for removal based on their proximity to any position already in Z, using a distance-based probability distribution biased by a parameter λ. Crucially, each candidate is only added to Z if it is non-adjacent to all current elements of Z, thereby preserving the required Gutin structure. The process continues until a predefined number of non-adjacent positions have been selected or no valid candidates remain.

The third operator, *Gutin Worst Removal*, mirrors the Unified Worst Removal strategy but filters for non-adjacency during the selection process. For each vertex in the tour, the operator computes its removal cost impact using the same marginal cost formula as in (3.1). Vertices are ranked by this cost in descending order, and selections are made using a λ-biased randomized process. As with the previous operator, a candidate is accepted into Z only if it is non-adjacent to all previously selected positions. This variant ensures that highly disruptive vertices are removed while maintaining the structural integrity required for Gutin based hole insertions.

3.4 Hungarian Algorithm as Repair Operator

Following the Gutin Neighborhood-based removal operators, we introduce a novel repair operator that leverages the Hungarian algorithm to optimally reinsert removed vertices. This operator is usable exclusively after the application of a Gutin based removal method, capitalizing on the structured nature of the removed subset Z, where all positions are pairwise nonadjacent.



(a) : GTSP tour with two holes.

(b) : Complete GTSP tour repaired via hungarian aglorithm.

Figure 3.1: Application of the Hungarian algorithm as a repair operator for a GTSP tour. (a) A partial GTSP tour with two missing clusters, resulting in two holes in the tour. (b) Optimal reinsertion of the removed clusters into the two available gaps, as computed by the Hungarian algorithm.

The Hungarian algorithm, a classical method for solving the assignment problem in polynomial time, is employed to find an optimal matching between removed positions in the tour and representative vertices of the corresponding clusters [10]. In this context, the assignment problem is defined as follows:

Let $Z \subset \{1, 2, \ldots, N\}$ denote the set of tour positions selected for removal. For each $i \in Z$, let C_i be the cluster originally visited at position i. The goal is to reassign each position $i \in Z$ to a possibly new cluster C_i, and to select a vertex $v \in C_i$ to insert at position i such that the total insertion cost is minimized.

To efficiently determine the best way to reinsert the removed vertices, we model the problem as an assignment problem, where each removed position must be matched to a cluster representative minimizing the overall insertion cost.

This formulation naturally leads to the construction of a cost matrix $A = (a_{ij})$, where each element a_{ij} represents the minimum insertion cost of a vertex from cluster C_j into the position $i \in Z$ in the tour. Let $x = (x_1, x_2, \ldots, x_N, x_1)$ denote the current tour, where each $x_k \in V$ is a selected vertex from some cluster.The cost $c(u, v)$ denotes the travel cost between vertices u and v in the underlying graph. The insertion cost is computed as:

$$a_{ij} := \min_{v \in C_j} \left(c(x_{i-1}, v) + c(v, x_{i+1}) \right),$$

where x_{i-1} and x_{i+1} are the vertices immediately before and after position i in the tour (with circular indexing such that $x_0 = x_N$ and $x_{N+1} = x_1$).

Once the cost matrix is constructed, the Hungarian algorithm is applied to compute a minimum-cost perfect matching, thereby selecting an optimal representative $v_j \in C_j$ for each position $i \in Z$. This repair operator benefits from the non-adjacency structure of Z, which allows the reassignment to be handled as a clean assignment problem without interference from overlapping segments in the tour.

Chapter 4

Shortest Paths Finding in a Polygonal Domain

This chapter provides an overview of established and modern techniques for computing shortest paths in a polygonal domain, with a focus on both single-pair and all-pairs scenarios. We further introduce and analyze the Euclidian Hub Labeling (EHL) algorithm, and its adaptation for scalable all-pairs pathfinding. The performance and practicality of these approaches are evaluated using the TřiVis library as a benchmark reference.

4.1 Overview of Approaches for Single-Pair Shortest Path in a Polygonal Domain

The problem of finding the shortest path between two points in a polygonal domain involves determining a path of minimum Euclidean length that lies entirely within the free space, i.e., it does not intersect the interior of any polygonal obstacle.

Any optimal path in such a domain is composed of straight-line segments between mutually visible points and may include bends only at the convex vertices of the obstacles. This property enables the construction of a *visibility graph*, where the nodes correspond to convex vertices in the domain, and the edges connect pairs of mutually visible vertices.

Classic algorithms for shortest path computation, such as Dijkstra's algorithm [11] and its heuristic variant, A* [12], are applicable once the visibility graph has been established. Dijkstra's algorithm guarantees optimality by systematically exploring paths in order of increasing cost. However, it can become computationally expensive in dense or large graphs. The A* algorithm improves on this by incorporating a heuristic function, typically the Euclidean distance to the goal, that guides the search more directly towards the target, potentially reducing the number of explored nodes.

While both Dijkstra's algorithm and A* are quite effective for computing shortest paths once the visibility graph is available, the primary computational challenge in polygonal domains lies in the efficient construction of the visibility graph itself. This can be achieved through approaches such as Polyanya [2] or TřiVis [5].

4.2 Overview of Approaches for All-Pairs Shortest Path in a Polygonal Domain

Classical algorithms such as the Floyd–Warshall and Johnson's algorithm are traditionally employed for scenarios where shortest paths between all pairs of points are needed. The Floyd–Warshall algorithm computes all-pairs shortest paths by iteratively improving estimates of the distance between each pair using dynamic programming [13]. Johnson's algorithm, on the other hand, improves performance by applying Dijkstra's algorithm from each vertex after a reweighting step to handle negative edge weights [14].

However, these algorithms assume a precomputed graph with known edge weights, which is a limitation in polygonal domains where visibility between points must be determined dynamically. As such, a significant portion of the computational effort is devoted to constructing the visibility graph and integrating query points into it. To address this, preprocessing techniques such as visibility graphs or shortest path maps are often used to model the domain before applying classical all-pairs algorithms.

4.3 TřiVis: Description and Approach

Let us sum up the TřiVis library, a versatile, reliable, and high-performance tool for computing visibility in a polygonal domain, developed by Mikula, Kulich, and Přeučil [5]. This library serves as the baseline against which the output of this part of the thesis will be evaluated.

TřiVis is a C++ library designed for efficient computation of visibility in polygonal environments, with particular emphasis on robotics and related fields. At its core, TřiVis utilizes the Triangular Expansion Algorithm (TEA), which operates on a triangulated representation of the environment. Given a query point inside the polygonal space, the TEA recursively explores adjacent triangles visible from the point, progressively constructing the visibility polygon by maintaining and updating restricted view cones.

Among its various features, TřiVis provides functionality for visibility computation, point location, pathfinding, and graphical representations. For this thesis, the most relevant component is its all-pairs shortest path finder, which enables efficient evaluation of traversability and connectivity within complex polygonal domains.

4.4 EHL Description

The Euclidian Hub Labeling (EHL) algorithm, introduced in [4], is originally designed for fast single-pair shortest path queries within polygonal domains. It achieves this by leveraging extensive offline preprocessing to drastically accelerate online query times. In the preprocessing phase, EHL constructs a visibility graph over convex vertices using a search strategy similar to the one employed in TřiVis. Hub labels are then computed over this visibility graph to enable fast distance queries. Once preprocessing is complete, the shortest path computation only links the query points to the precomputed structure and evaluates distances through the relevant hub labels.

In this thesis, we extend the EHL approach to support all-pairs shortest path computation. This adaptation forms a central contribution of our work, enabling evaluation of connectivity and distances across the entire polygonal domain.

4.4.1 Graph of Vertices Creation

The first step in EHL is to construct a visibility graph within the polygonal domain. To achieve this, EHL uses the Polyanya algorithm, defined by Cui, Harabor, and Grastien [2], which functions in a way similar to the method used in TřiVis. This procedure creates a visibility graph $G = (V, E)$, where V is the set of convex vertices in the domain, and E is the set of edges connecting all pairs of vertices that are mutually visible.

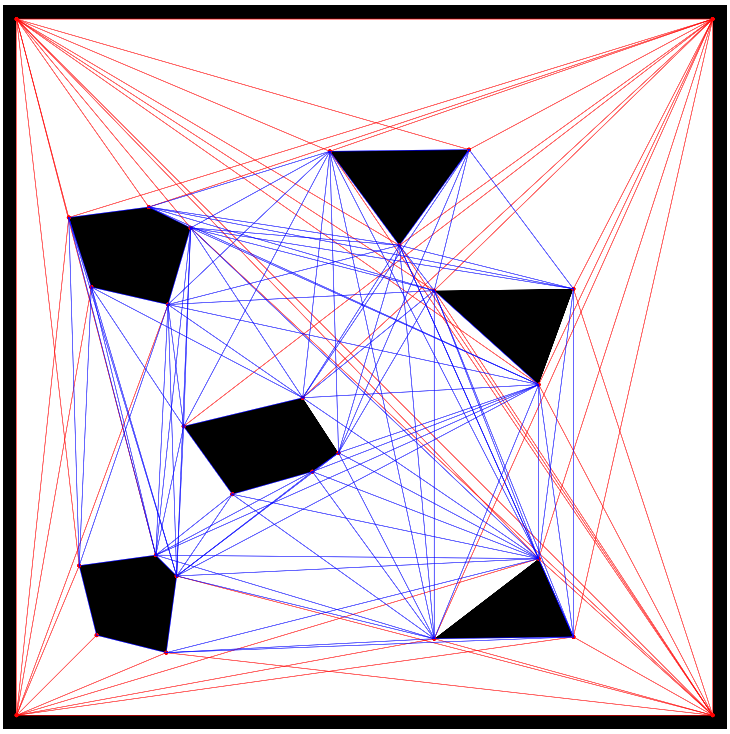


Figure 4.1: Visibility graph constructed over a polygonal domain. Edges between convex vertices are shown in blue, while red edges indicate connections involving at least one concave vertex, which may be omitted from the final visibility graph.

4.4.2 Graph Hub Labeling

Once the visibility graph is constructed, the next step is to compute hub labels for each vertex. The hub labeling technique allows for fast shortest path queries by storing precomputed distance information from each vertex to a carefully chosen subset of other vertices, called hubs.

Formally, let V be the set of vertices in the visibility graph and let d denote the shortest path distance function. For each vertex $v \in V$, let $L(v) \subseteq V \times \mathbb{R}$ denote the set of hub labels assigned to v, where each label is a pair $(h, d(v,h))$ consisting of a hub vertex $h \in V$ and the precomputed shortest path distance $d(.,.)$ from v to h. The hub labeling must satisfy the *cover property*:

$$\forall u, v \in V,\ \exists w \in L(u) \cap L(v) \text{ such that } d(u,v) = d(u,w) + d(w,v)$$

In other words, to compute the shortest path between any two vertices u and v, it is sufficient to examine the distances to their shared hubs and return the one with minimal total cost.

To create the hub labels, EHL uses the SHP algorithm presented by Li et al. [15]. Additionally, EHL integrates four pruning rules to enhance efficiency by reducing the number of stored labels. These pruning rules eliminate hubs that cannot contribute to any shortest path, based on properties such as minimum and maximum possible path lengths to a hub, and the identification of *non-taut regions*, areas from which a vertex occludes any traversable region, rendering them irrelevant for shortest path computations.

As stated previously, the efficiency of shortest path queries using hub labeling is directly influenced by the size of the intersection $L(v_s) \cap L(v_t)$, where v_s and v_t are the source and target vertices, respectively. The query time complexity is proportional to the number of common hubs in this intersection, as the shortest path is computed as:

$$d(v_s, v_t) = \min_{w \in L(v_s) \cap L(v_t)} (d(v_s, w) + d(w, v_t))$$

4.4.3 Polygonal Plane Bucketing

To efficiently connect arbitrary query points to the visibility graph, EHL employs a preprocessing stage known as *polygonal plane bucketing* to further accelerate the connection of query points to the preprocessed visibility graph.

Importantly, these vertices are not stored as raw coordinates. Instead, each vertex stored in a bucket includes its precomputed hub labels. This enables the algorithm to quickly retrieve not only which vertices might be used to connect a query point to the visibility graph, but also the full label information required for shortest path computations.

This approach partitions the domain into a set of rectangular regions, or *buckets*, each being filled with the hub labels of visible vertices from any point of the region. This includes vertices that may not be directly visible from every location inside the bucket but are visible from at least some part of it. The underlying rationale is that, via precomputing visible vertices, their searching can be skipped in the online shortest path search.

During query processing, the algorithm identifies the bucket containing the source and target points. Then it evaluates the cost of connecting these points to the stored vertices within the bucket, accounting for both the local connection cost and the precomputed label distances. If a vertex in the bucket is not directly visible from a query point (but is visible from another part of the bucket), it is still considered in the search for the cost of some additional computations.

A critical parameter affecting both performance and storage is the size of the buckets used in the visibility graph. Larger buckets reduce memory consumption but increase the chance that the associated graph vertex is not directly visible from the query point, leading to costly distance recalculations. Conversely, smaller buckets improve accuracy and visibility coverage but drastically inflate memory usage, with storage requirements growing quadratically. Even for moderately sized maps, this can result in hundreds of megabytes of precomputed data.

4.4.4 EHL Shortest Path Search

The shortest path between a source query point p_s and a target query point p_t is computed using the preprocessed bucket and hub label structures. First, the algorithm identifies the buckets containing p_s and p_t. Each bucket stores hub labels associated with visibility graph vertices that are at least partially visible from within the bucket.

For each visible vertex v_s in the source bucket and v_t in the target bucket, EHL evaluates all common hubs $w \in L(v_s) \cap L(v_t)$. The total path cost through each such triple (v_s, w, v_t) is calculated as the sum of:

1. the connection cost from p_s to v_s,
2. the precomputed shortest path $d(v_s, w) + d(w, v_t)$ through the common hub w,
3. the connection cost from v_t to p_t.

The algorithm selects the combination with the lowest total cost as the shortest path. To improve efficiency, EHL applies early termination rules. If the precomputed cost between v_s and v_t via any hub w exceeds the best total path cost found so far, that hub is pruned from further consideration. This selective pruning significantly reduces unnecessary computations, particularly in complex environments.

4.5 EHL All-Pairs Path Finding Adjustment

The straightforward application of the EHL method to all-pairs shortest path computation suffers from a major inefficiency related to the repeated connection of query points to the visibility graph. Specifically, for a set of N query points, the naive approach issues $N(N-1)$ shortest path queries, where each query independently computes the connection of both source and target points to the graph. Although the shortest path distances between the graph vertices themselves are precomputed during the EHL preprocessing phase, this redundancy in connecting points to the graph persists and significantly increases the total runtime.

To address this, we developed a novel caching mechanism that decouples the point-to-graph connection computations from individual queries. For each query point, the algorithm computes and stores its visible graph vertices along with the corresponding connection costs and associated hub labels only once. This data structure is then reused across all subsequent shortest path queries involving that point, eliminating the need for repeated visibility checks and distance computations between the point and the graph.

The caching structure was implemented with attention to efficient storage and fast lookup, balancing memory overhead against computational savings. Empirical evaluation on GPPC-2014 benchmark maps [16] showed that this optimization reduces the total query processing time by approximately a factor of three for datasets containing more than 100 query points.

This adjustment extends the applicability of EHL to batch shortest path computations in polygonal domains by significantly mitigating the computational bottleneck associated with repeated point-to-graph connections. It thus enables more scalable and practical use of hub labeling techniques for all-pairs shortest path queries in geometric environments.

Chapter 5

Experiments

In this chapter, we present experimental evaluations addressing both major components of this thesis. First, we compare the performance of the adjusted EHL method with the TřiVis library on all-pairs shortest path queries within polygonal domains. Next, we assess the new Gutin based operators for the GTSP, comparing them with existing methods. Finally, we demonstrate a combined use case involving both contributions, applied to the Watchman Route Problem in polygonal environments.

5.1 EHL vs. Trivis: Performance

To evaluate the performance of the adjusted EHL approach compared to the TřiVis library, we conducted a series of experiments across multiple polygonal domain maps from the GPPC-2014 [16]. All experiments were executed using C++ and compiled with the `-O3` flag on an machine with 16 GB of RAM. The availability of 16 GB memory was essential, as EHL wouldn't converge on large maps with small bucket sizes when less memory was available.

EHL was tested with three bucket size configurations (side lengths of 4, 16, and 64 units). These configurations represent a trade-off between preprocessing cost, query efficiency, and memory usage. In contrast, TřiVis performs all path computations online and does not require preprocessing.

Query Type	Třivis	EHL 4	EHL 16	EHL 64
Preprocessing	0.0	13.5	3.8	2.5
1000 Points	9.3	3.8	4.6	11.4
5000 Points	114.0	97.5	115.6	296.0
10 000 Points	497.2	390.4	462.6	1184.9
20 000 Points	2841.7	1560.0	1868.9	4760.0

Table 5.1: Execution Time in Seconds on `AcrosstheCape`

Query Type	Třivis	EHL 4	EHL 16	EHL 64
Preprocessing	0.0	6.2	1.7	1.1
1000 Points	6.2	3.4	4.3	12.5
5000 Points	132.0	87.2	111.1	307.2
10 000 Points	754.0	349.5	445.4	1235.7
20 000 Points	5589.9	1405.9	1777.0	4930.5

Table 5.2: Execution Time in Seconds on `Aftershock`

Query Type	Třivis	EHL 4	EHL 16	EHL 64
Preprocessing	0.0	2.5	0.8	0.6
1000 Points	3.5	3.1	3.7	8.0
5000 Points	77.0	81.0	94.3	195.1
10 000 Points	441.0	327.2	378.4	777.4
20 000 Points	3063.5	1306.0	1514.1	3115.2

Table 5.3: Execution Time in Seconds on `Archipelago`

Query Type	Třivis	EHL 4	EHL 16	EHL 64
Preprocessing	0.0	14.6	4.8	3.3
1000 Points	13.5	3.7	4.4	11.6
5000 Points	136.4	94.6	109.3	277.8
10 000 Points	528.8	381.4	436.3	1125.1
20 000 Points	2694.0	1528.4	1744.1	4474.5

Table 5.4: Execution Time in Seconds on `ArcticStation`

Query Type	Třivis	EHL 4	EHL 16	EHL 64
Preprocessing	0.0	17.1	5.0	3.4
1000 Points	13.1	3.9	5.0	12.7
5000 Points	148.0	99.4	123.1	297.9
10 000 Points	621.0	399.7	488.2	1192.7
20 000 Points	3487.5	1609.0	1963.2	4740.5

Table 5.5: Execution Time in Seconds on `BlastFurnace`

To complement the detailed numerical results presented in the tables, we also provide graphical representations in Figures 5.1- 5.5 of the execution times across different query points for each polygonal domain. These graphs incorporate preprocessing times for the EHL configurations, offering a clear visual comparison of the performance trade-offs between the TřiVis library and the various EHL bucket size settings.

Overall, the adjusted EHL approach outperforms TřiVis in terms of execution time for sufficiently large query sets, particularly when an appropriate bucket size is selected. In all tested environments, adjusted EHL with buckets of size 4 consistently achieves lower query times than TřiVis once the number of query points exceeds a few thousand.

The selection of the size of the bucket plays a critical role in balancing preprocessing time, query speed, and memory usage. Smaller bucket sizes (e.g., 4 units) yield the most accurate visibility connections and thus fastest query times, but at the cost of increased preprocessing time and memory footprint.

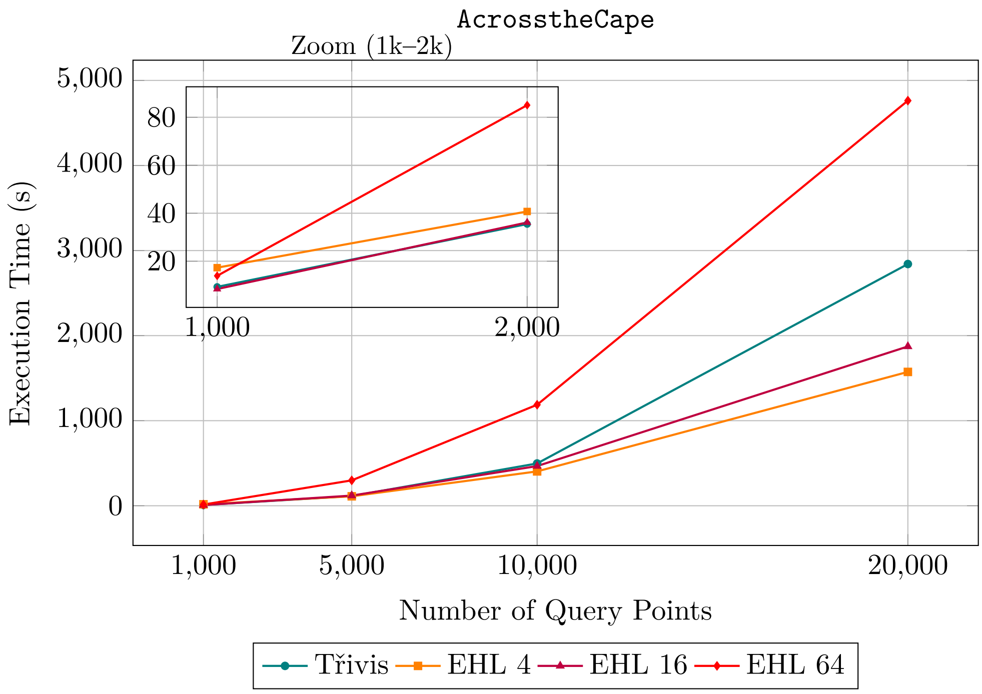


Figure 5.1: Execution Time vs. Number of Query Points on `AcrosstheCape`, with Zoom-In on 1k–2k Region (Including Preprocessing)

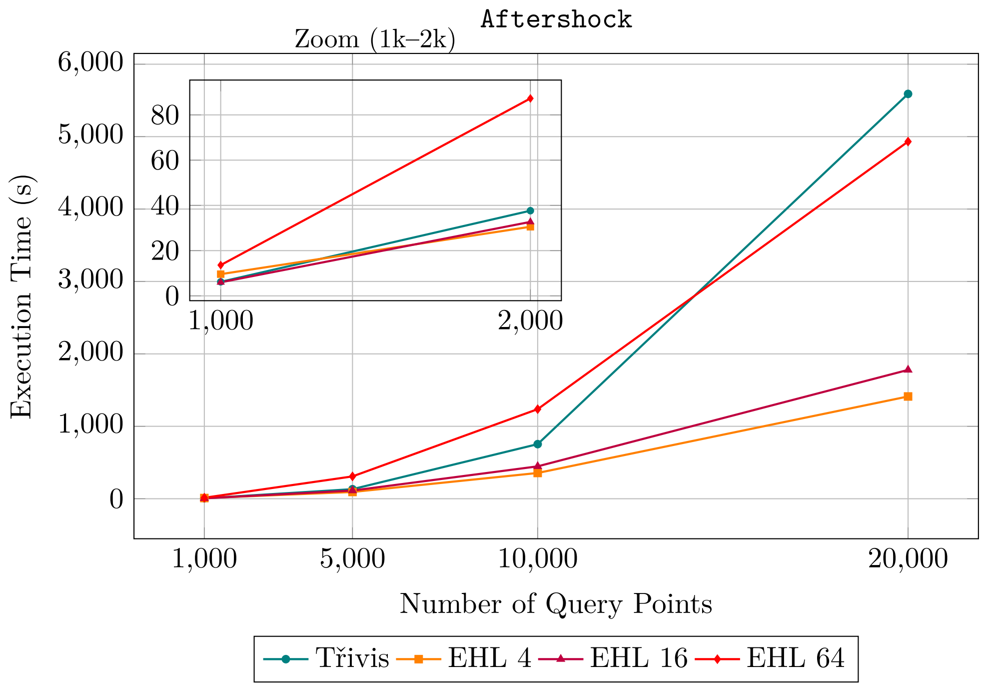


Figure 5.2: Execution Time vs. Number of Query Points on `Aftershock`, with Zoom-In on 1k–5k Region (Including Preprocessing)

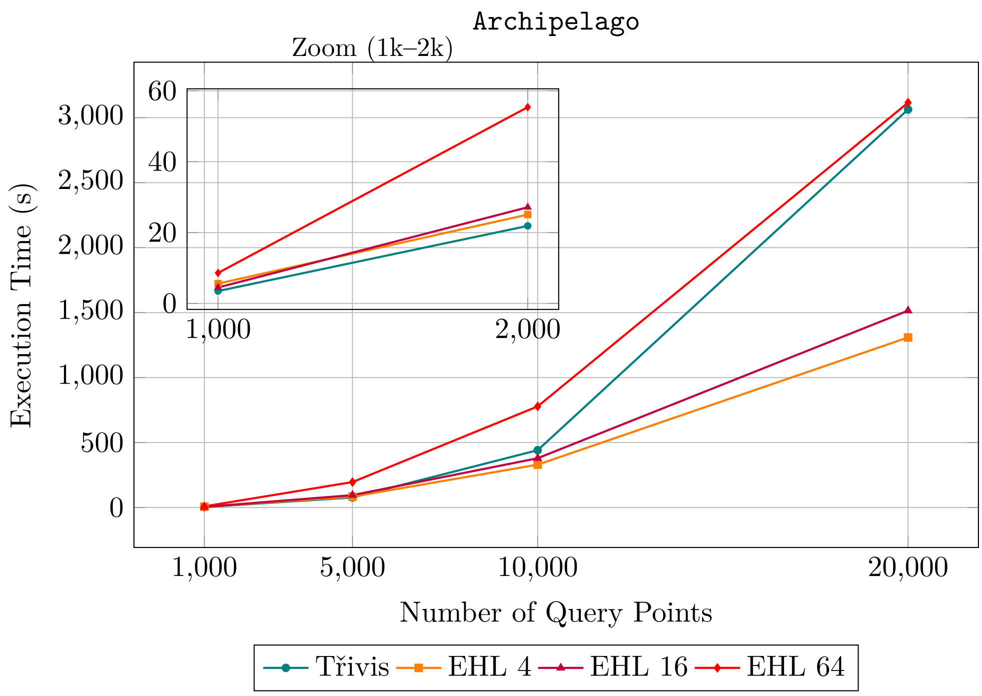


Figure 5.3: Execution Time vs. Number of Query Points on `Archipelago` (Including Preprocessing)

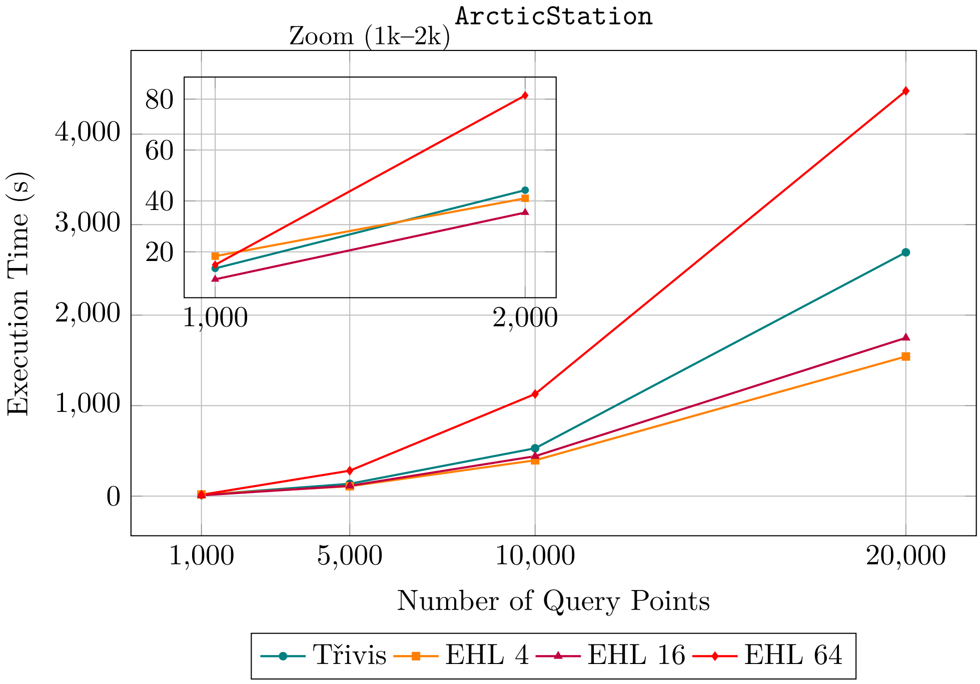


Figure 5.4: Execution Time vs. Number of Query Points on `ArcticStation` (Including Preprocessing)

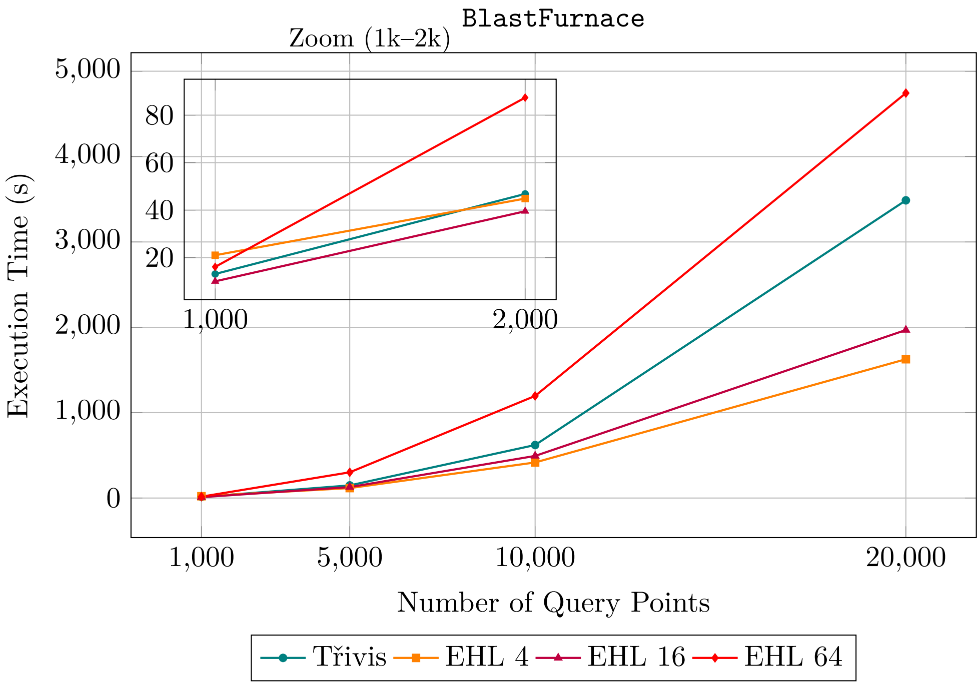


Figure 5.5: Execution Time vs. Number of Query Points on `BlastFurnace` (Including Preprocessing)

Despite some speed advantages, we ultimately adopt TřiVis for further use due to its superior robustness and versatility. In particular, EHL was observed to produce incorrect results or fail to converge in very detailed polygonal maps, where obstacles were separated by small distances, indicating a lack of dependability in challenging scenarios. Moreover, our evaluated instances contain about 1,000 query points, which is a scale at which TřiVis remains competitive with, or even outperforms, the adjusted EHL in both accuracy and performance.

5.2 Gutin Operators for GTSP

To evaluate the proposed Gutin based removal operators and the Hungarian repair heuristic, we ran experiments on instances from GTSP-LIB [17]. Each instance name is prefixed with its number of clusters and suffixed with its number of vertices. All experiments used optimized C++ code with the `-O3` flag and were run on a laptop with an M2 chip and 8 GB RAM.

Due to the inherent nondeterminism of the GLNS algorithm, multiple independent runs are necessary to obtain statistically meaningful results. While we would ideally perform a larger number of repetitions to better represent the stochastic behavior, the high computational cost, especially for large instances with over 200 clusters, limited us to 10 independent runs per configuration. We report aggregated statistics across these runs.

For the Hungarian repair operator, we employed the implementation provided in Google OR-Tools [18], which was selected due to its faster runtime performance compared to other available open-source alternatives. The OR-Tools library provides a robust and well-optimized solver for the linear assignment problem.

All tests employed the "fast" mode of GLNS due to runtime limitations; medium and slow modes were too time-consuming. Though the slower configurations may yield marginally better solutions, their impractical runtimes preclude their use in this study.

As an initial diagnostic step, we tested the GLNS algorithm in a restricted configuration using only Gutin based removal operators (i.e., no traditional GLNS removals) alongside the Hungarian algorithm repair. The goal was to examine the performance of our methods in isolation. These tests were performed using small-scale instances with fewer than 100 vertices, which are solvable in seconds using the GLNS metaheuristic.

The figures present the average percentage deviation from the best-known cost over time for three solver variants. The `orig` configuration corresponds to the original GLNS algorithm, employing the original repair heuristic and three of its default removal operators. The `mix` variant combines the original GLNS removal operators with our proposed Gutin based removal strategy, using only the Hungarian repair heuristic. Lastly, the `gutin` variant exclusively utilizes the Gutin based removal operator, with the Hungarian algorithm serving as the sole repair mechanism.

Each line represents the mean percentage deviation from the best-known solution over time (in milliseconds), with 95% confidence intervals. Narrower bands indicate lower variance.

In the Figures 5.6 and 5.7, the x-axis represents time in milliseconds, while the y-axis denotes the average percentage above the best-known cost observed across all test instances. The best known cost is defined as the lowest solution value obtained across all solver configurations and runs, serving as a common baseline for comparison.

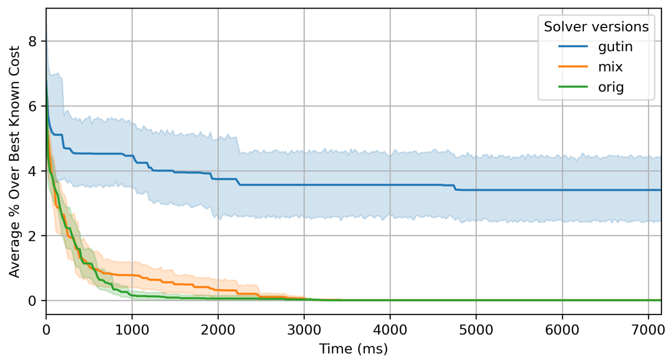


Figure 5.6: GTSP Result: 80rd400 (Gutin-only Heuristic)

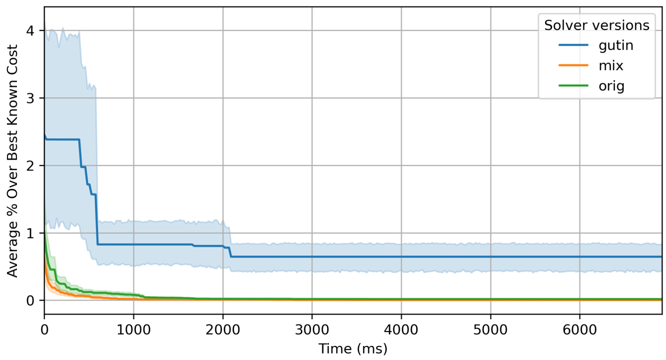


Figure 5.7: GTSP Result: 84fl417 (Gutin-only Heuristic)

From the performance plots, it is evident that the `gutin` configuration, relying solely on Gutin based removal operators, tends to be trapped in local optima. This premature convergence results in final solutions that are several percentage points worse, on average, than those obtained by the `orig` configuration. In contrast, the standard GLNS approach, which incorporates a broader diversity of removal and repair strategies, consistently achieves or closely approximates the best-known solutions across most instances. This suggests that Gutin based operators' lack of diversification limits their effectiveness in reaching globally optimal solutions without complementary mechanisms.

5.2.1 Performance on Large-Scale Instances (200+ Nodes)

Based on the previously presented findings from smaller instances, we limited further evaluation to three solver configurations that consistently yielded competitive results: the `orig` baseline, a configuration combining all GLNS and gutin removal operators with the Hungarian insertion heuristic being used solely after Gutin removals (`hung`), and a variant using both standard and Gutin based removal operators with a mixture of insertion heuristics (`mix`). In all operators, we adopted the same λ-based probabilistic cluster selection system as described by the original GLNS authors

Since performance differences among these variants were minimal on smaller instances, we focused subsequent experiments on larger instances comprising more than 200 clusters, selected from the extended GTSP-LIB dataset [17].

The plots follow the same structure as in previous analyses: each line represents the average performance across 10 independent runs per configuration, with the shaded area indicating the 95% confidence interval. The x-axis denotes the elapsed time in milliseconds, while the y-axis shows the average percentage above the best-known cost. Due to time constraints, all runs were terminated after 500,000 milliseconds.

The following Figures 5.8, 5.9, 5.10 and 5.11 illustrate representative results obtained on four large-scale instances.

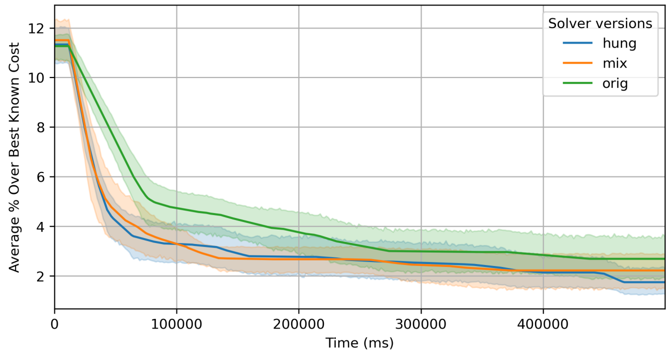


Figure 5.8: GTSP Result: 2000C10k

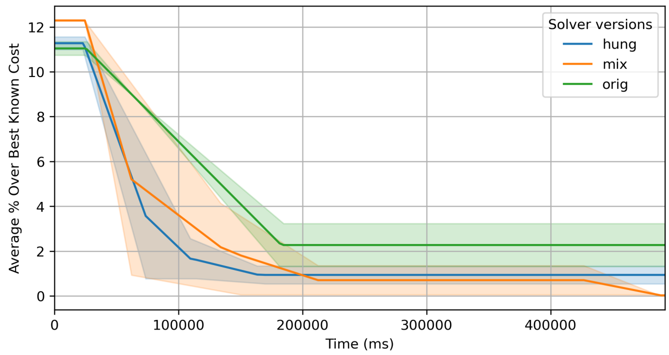


Figure 5.9: GTSP Result: 2370rl11849

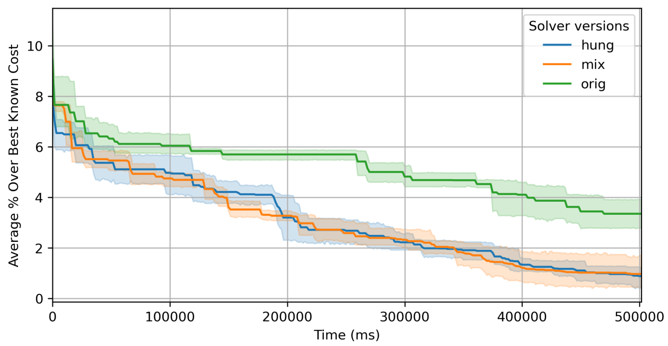


Figure 5.10: GTSP Result: 464u2319

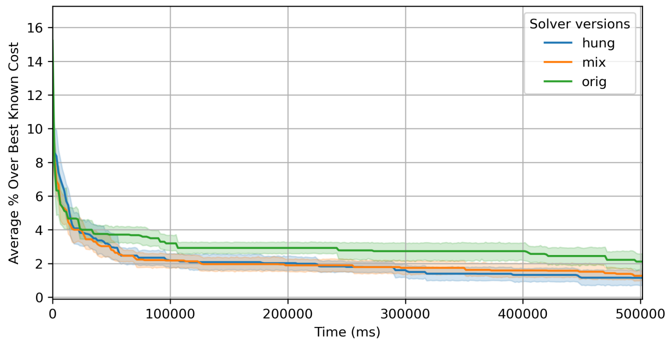


Figure 5.11: GTSP Result: 759fl3795

The results indicate that both hybrid Gutin based configurations usually outperform the standard GLNS variant across all time horizons. Specifically, the `mix` variant exhibits the most consistent improvement, achieving better solution quality by a margin of several percentage points on average at nearly every recorded time point. This suggests that the integration of Gutin based removals with diverse insertion strategies enhances both the convergence speed and final solution quality on large-scale instances. To provide a comprehensive overview of performance, we include a summary table reporting the average final solution cost percentage over the best-known solution, along with the corresponding standard deviation across all runs.

Additionally, the Table 5.6 presents the improvement Δ, defined as the difference in average solution cost between the Mix Insert and Hungar Insert heuristics compared to the GLNS standard. To conclude, the last row of the table reports the average performance in each criterion across all tested instances. All tests were terminated after reaching the 500,000 ms time limit.

Instance	**orig**		**mix**			**hung**		
	Avg	Std	Avg	Std	Δ	Avg	Std	Δ
2370rl11849	2.27	0.96	0.03	0.03	2.24	0.94	0.40	1.33
464u2319	3.35	0.67	0.97	0.72	2.38	0.89	0.48	2.46
479pr2392	3.69	1.14	1.80	1.03	1.89	2.33	0.66	1.36
2000C10k	2.69	1.38	2.22	1.13	0.47	1.75	0.88	0.94
1183rl5915	1.64	0.71	2.20	0.74	-0.56	0.59	0.60	1.05
759fl3795	2.11	0.75	1.27	0.48	0.84	1.15	0.74	0.96
331d1655	6.30	1.83	0.79	0.42	5.51	0.91	0.38	5.39
Average	3.15	1.06	1.33	0.65	1.82	1.22	0.59	1.93

Table 5.6: Average and Standard Deviation of Minimal Percent Over Best for Various Labels, Including Improvement Δ Compared to GLNS Standard

5.3 Gutin Operators for GTSP in a Polygonal Domain

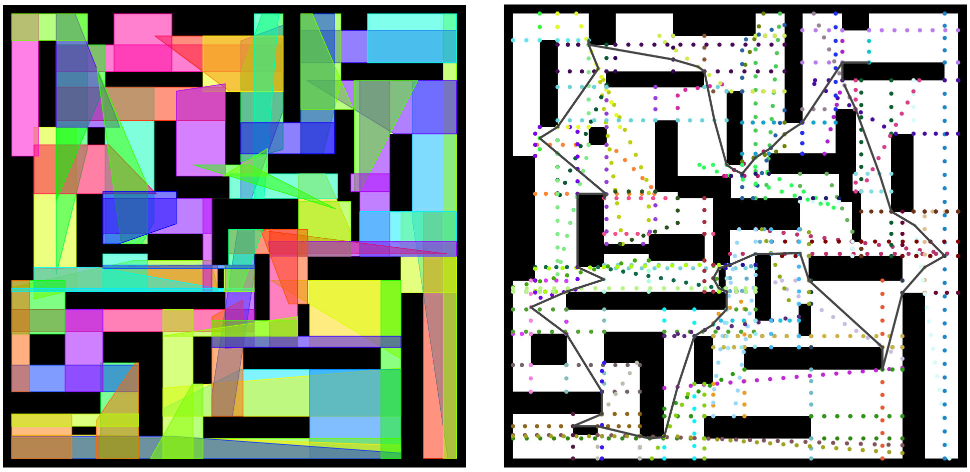
To demonstrate the practical applicability of our approach, we adapt the Generalized Traveling Salesman Problem (GTSP) formulation to solve the Watchman Route Problem under constrained visibility assumptions.

In this setting, the environment is decomposed into a collection of convex polygons, each serving as a distinct *visibility region*. The key idea is that the watchman's route is considered complete if it includes at least one point from each region. Due to the convexity of these regions, any point within a region has direct line of sight to all other points inside it. Thus, visibility within each region is inherently guaranteed by its geometric structure. To incorporate a practical constraint, we make the maximal Euclidean distance between any two points within a region the same as the the agent's visibility range.

Crucially, the visibility regions are defined such that their union completely covers the environment. As a result, ensuring that the watchman's path intersects at least one point in every region guarantees full visibility of the entire map.

To model the problem as a GTSP instance, we discretize each visibility region by sampling points along its boundary, forming the node set for the GTSP. Each region corresponds to a distinct GTSP *cluster*, and the objective is to calculate the shortest possible path that visits at least one node of each cluster. The required cost matrix is constructed using the shortest path distances computed via the TřiVis library [5], which accounts for obstacles in the environment.

Figure 5.12 presents an illustrative example of the pipeline under full (infinite) visibility. In Figure 5.12a, the environment is partitioned into convex visibility regions, assuming unlimited observation range. These regions are then discretized into sample points, as shown in Figure 5.12b, where the resulting GTSP solution path is also overlaid. In this setting, the process involved a total of 1389 vertices and 79 GTSP sets, with an execution time of 5395 ms using the `mix` mode, including 897 ms for all-pairs shortest path computation.

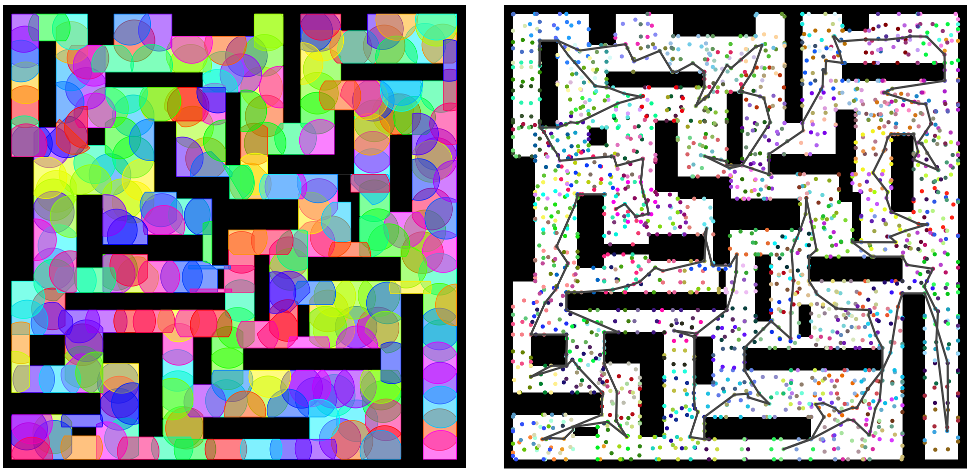


(a) : Original visibility regions (full visibility)

(b) : GTSP solution path (full visibility)

Figure 5.12: Region generation and GTSP-based path planning using full visibility range.

Figure 5.13 illustrates the same methodology applied in a reduced visibility range. The limited observation distance results in the generation of smaller convex visibility regions, as depicted in Figure 5.13a. After discretization, the corresponding GTSP solution is shown in Figure 5.13b. This scenario involved 2386 vertices and 278 GTSP sets, and the execution time using the `mix` mode was 20003 ms, including 2422 ms for all-pairs shortest path computation.



(a) : Original visibility regions (reduced range)

(b) : GTSP solution path (reduced range)

Figure 5.13: Region generation and GTSP-based path planning using reduced watchman visibility.

Chapter 6

Conclusions

This thesis has addressed two aspects related to the application of the Generalized Traveling Salesman Problem (GTSP) in polygonal domains with obstacles.

First, the adaptation and extension of the Euclidean Hub Labeling (EHL) algorithm for efficient all-pairs shortest path computations in polygonal environments was investigated. The modified EHL approach was benchmarked against TřiVis library, demonstrating competitive performance in query-intensive scenarios when appropriate preprocessing parameters are selected. However, limitations were observed in the original EHL implementation in terms of robustness and convergence, especially on complex maps with narrow obstacle separations, which constrains the practical applicability of EHL in some cases.

Second, new heuristic operators based on the Gutin neighborhood were integrated into the GLNS metaheuristic framework for GTSP. These operators focused on removal strategies within the destroy and repair approach, complemented by the Hungarian algorithm for repair. Experimental results on GTSP-LIB benchmark instances indicate that while Gutin-based removals alone tend to get trapped in local optima, their combination with traditional GLNS operators improves solution quality and convergence speed on large-scale instances. This hybrid approach consistently outperforms the original GLNS baseline in terms of solution cost within comparable computational budgets.

Finally, the integration of both shortest path computations and GTSP solving was demonstrated through the Watchman Route Problem in polygonal domains. The workflow involved discretizing visibility regions into clusters and generating cost matrices using TřiVis shortest path queries, followed by GTSP optimization with the enhanced GLNS solver. This pipeline highlights the feasibility of applying the developed methods to practical geometric routing problems with visibility constraints.

Overall, the work contributes both algorithmic enhancements for GTSP metaheuristics and insights into the computational trade-offs of shortest path preprocessing in polygonal domains.

Future work can focus on improving the robustness and efficiency of the EHL approach. In particular, integrating EHL with faster and more reliable pathfinding methods, such as TřiVis, could improve its practical applicability in complex environments.

For the GLNS solver, further research could explore more effective combinations of heuristic operators. An additional direction involves designing a better cost function for vertex removal in the Gutin neighborhood style, with one that would better complement the non-adjacency requirement.

Bibliography

[1] J. Schmidt and S. Irnich, “New neighborhoods and an iterated local search algorithm for the generalized traveling salesman problem,” *EURO Journal on Computational Optimization*, vol. 10, p. 100029, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2192440622000053?via%3Dihub

[2] M. Cui, D. D. Harabor, and A. Grastien, “Compromise-free Pathfinding on a Navigation Mesh,” in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 2017, pp. 496–502. [Online]. Available: https://doi.org/10.24963/ijcai.2017/70

[3] S. L. Smith and F. Imeson, “GLNS: An effective large neighborhood search heuristic for the Generalized Traveling Salesman Problem,” *Computers & Operations Research*, vol. 87, pp. 1–19, 2017. [Online]. Available: https://doi.org/10.1016/j.cor.2017.05.010

[4] J. Du, B. Shen, and M. A. Cheema, “Ultrafast Euclidean Shortest Path Computation Using Hub Labeling,” in *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 12 417–12 426, article 1394. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/26463

[5] J. Mikula, M. Kulich, and L. Přeučil, “TřiVis: Versatile, Reliable, and High-Performance Tool for Computing Visibility in Polygonal Environments,” in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024. [Online]. Available: https://arxiv.org/html/2410.08752v1

[6] B. Shen, M. A. Cheema, D. D. Harabor, and P. J. Stuckey, “Euclidean Pathfinding with Compressed Path Databases,” in *Proceedings of the*

Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI'20), 2021, pp. 4229–4235, article 584. [Online]. Available: https://www.ijcai.org/proceedings/2020/0584.pdf

[7] M. Fischetti, J. J. S. González, and P. Toth, "A Branch-And-Cut Algorithm for the Symmetric Generalized Traveling Salesman Problem," *Operations Research*, vol. 45, no. 3, pp. 378–394, 1997. [Online]. Available: http://www.jstor.org/stable/172016

[8] C. Noon, "A Lagrangian Based Approach for the Asymmetric Generalized Traveling Salesman Problem," *Operations Research*, vol. 39, no. 4, pp. 623–639, 1991. [Online]. Available: https://doi.org/10.1287/opre.39.4.623

[9] G. Gutin, A. Yeo, and A. Zverovitch, "Exponential Neighborhoods and Domination Analysis for the TSP," in *The Traveling Salesman Problem and Its Variations*, G. Gutin and A. P. Punnen, Eds. Boston, MA: Springer, 2006, pp. 223–256. [Online]. Available: https://www.academia.edu/16499482/Exponential_Neighborhoods_and_Domination_Analysis_for_the_TSP

[10] H. W. Kuhn, "The Hungarian Method for the Assignment Problem," *Naval Research Logistics Quarterly*, vol. 2, 1955. [Online]. Available: https://www.math.toronto.edu/mccann/1855/KuhnNRL55.pdf

[11] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959. [Online]. Available: https://hal.science/hal-03171590/file/dijkstra1959groupefmr.pdf

[12] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[13] Martin Mareš and T. Valla, *Průvodce labyrintem algoritmů*, 2nd ed. Praha: CZ.NIC, z. s. p. o., 2022. [Online]. Available: https://www.nic.cz/edice/28/pruvodce-labyrintem-algoritmu/

[14] J. Erickson, "All-Pairs Shortest Paths," https://jeffe.cs.illinois.edu/teaching/algorithms/book/09-apsp.pdf, lecture notes, University of Illinois.

[15] Y. Li, L. H. U, M. L. Yiu, and N. M. Kou, "An experimental study on hub labeling based shortest path algorithms," *Proceedings of the VLDB Endowment*, vol. 11, no. 4, pp. 445–457, 2017. [Online]. Available: https://doi.org/10.1145/3186728.3164141

[16] N. R. Sturtevant, D. Harabor, P. J. Stuckey, A. Felner, S. Koenig, H. Ma *et al.*, "The Grid-Based Path Planning Competition: 2014 Entries and Results," in *Proceedings of the Eighth Annual Symposium on Combinatorial Search (SoCS 2015)*, 2015. [Online]. Available: https://webdocs.cs.ualberta.ca/~nathanst/papers/GPPC-2014.pdf

[17] S. L. Smith, "GTSP-LIB: Generalized Traveling Salesman Problem Library," https://ece.uwaterloo.ca/~sl2smith/GLNS/, accessed: May 14, 2025.

[18] Google, "Or-tools/orttools/algorithms/hungarian.h at stable · google/or-tools," https://github.com/google/or-tools/blob/stable/ortools/algorithms/hungarian.h, 2015, accessed: May 13, 2025.

Appendix A

Code Availability

Adjusted EHL

The modified version of the EHL algorithm can be accessed at:

`https://gitlab.ciirc.cvut.cz/holejste/finding-all-pairs-shortest-paths`

Instructions for building and running the code are provided in the repository's `README` file.

GLNS with Custom Neighborhood Operators

The GLNS implementation with Gutin based Neighborhood operators is available at:

`https://gitlab.ciirc.cvut.cz/holejste/glns-with-gutin-neighborhood-operators_my_project`

Instructions for building and running the code are provided in the repository's `README` file.