Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

METAHEURISTICS FOR THE WATCHMAN ROUTE PROBLEM

Lubor Šída

Master's Thesis

Supervisor: RNDr. Miroslav Kulich, Ph.D.

Study program: Cybernetics and Robotics

May 2025

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Šída Lubor** Personal ID number: **492375**

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

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Master’s thesis details

Master’s thesis title in English:

Metaheuristics for the Watchman Route Problem

Master’s thesis title in Czech:

Metaheuristiky pro problém hledání cesty hlídače

Guidelines:

The Watchman Route Problem is the problem of finding the shortest closed collision-free route to be traversed by a mobile agent such that it sees the whole environment in which it operates. A standard solution approach to the WRP is decoupling, which splits the problem into two separate subproblems that are solved independently: the Art Gallery Problem and the Travelling Salesman Problem. However, despite both parts being solved optimally, decoupling does not always provide solutions of sufficient quality. The student will study a weakly coupling approach to the WRP generating a large set of sensing location candidates and finding a shortest possible path across a subset of the candidates completely covering the environment. This will be done in the following steps:
1. Get acquainted with the Watchman Route Problem (WRP), the Generalized Travelling Salesman Problem (GTSP), the Set Covering Problem (SCP), and state-of-the-art metaheuristics for these problems.
2. Realize/employ a method for generating a large set of sensing location candidates.
3. Formulate the WRP as a combination of the GTSP and SCP.
4. Design and realize a method for the formulated problem.
5. Evaluate experimentally properties of the developed method. Describe and discuss the obtained results.

Bibliography / sources:

1] Guanghui Lan, Gail W. DePuy, Gary E. Whitehouse, An effective and simple heuristic for the set covering problem, European Journal of Operational Research, Volume 176, Issue 3, 2007, Pages 1387-1403, ISSN 0377-2217, https://doi.org/10.1016/j.ejor.2005.09.028.
[2] Reyes, V., Araya, I. A GRASP-based scheme for the set covering problem. Oper Res Int J 21, 2391–2408 (2021). https://doi.org/10.1007/s12351-019-00514-z
[3] OR-Library: https://people.brunel.ac.uk/~mastjjb/jeb/orlib/scpinfo.html
[4] Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., Rei, W. (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. Operations Research, 60(3), 611-624. https://doi.org/10.1287/opre.1120.1048
[5] https://github.com/bruscalia/tsp-grasp?tab=readme-ov-file
[6] 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.
[7] 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.

Name and workplace of master's thesis supervisor:

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

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

Date of master's thesis assignment: **30.01.2025** Deadline for master's thesis submission: __________

Assignment valid until: **20.09.2026**

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

prof. Mgr. Petr Páta, Ph.D.
Dean's signature

III. Assignment receipt

The student acknowledges that the master'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 master'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): Šída Lubor
Personal number: 492375
Programme name: Cybernetics and Robotics

declare that I have elaborated the master's thesis entitled

Metaheuristics for the Watchman Route Problem

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 23.05.2025

Bc. Lubor Šída
....
student's signature

Abstract

This work addresses the Watchman Route Problem (WRP) with limited visibility, an optimization task from computational geometry, where the goal is to find the shortest path that ensures full visibility coverage of a known environment. With motivation in autonomous mobile robotics, this task represents real-world applications such as surveillance and inspection. The WRP is approached by decomposing it into two classical problems from combinatorial optimization: the Set Cover Problem (SCP) for selecting guard locations, and the Generalized Traveling Salesman Problem (GTSP) for optimizing the resulting tour. The environment is represented using both discrete and continuous polygonal domain segmentations, enabling flexible and accurate modeling of visibility constraints. Candidate guard locations are generated via a visibility-based heuristic, and a GRASP-based approach is employed to solve the SCP introducing novel weighting functions and insertion rules to better reflect spatial and geometric considerations of the overarching task in the SCP formulation. The selected guard positions are then connected using a TSP solver, and further refined through large neighborhood search techniques. The entire algorithm is implemented in C++ and validated through experimental evaluation.

Keywords: Watchman Route Problem, Computational Geometry, Combinatorial Optimization, Metaheuristics, Path Planning

Abstrakt

Práce se zabývá problémem hledání cesty hlídače (WRP) s limitovanou viditelností. Jedná se o optimalizační problém z oblasti výpočetní geometrie a cílem úlohy je najít nejkratší cestu známým prostředím tak, aby její projití garantovalo vizuální pokrytí celé oblasti. Úloha modeluje řadu aplikací z oblasti autonomní mobilní robotiky, ve kterých je potřeba kontrolovat či hlídat známé území. WRP je v rámci práce řešeno rokladem problému na dva klasické problémy z oblati kombinatorické optimalizace: Set Cover Problem (SCP) s účelem výběru vhodných klíčových pozic cesty a Generalized Traveling Salesman Problem (GTSP) pro vytvoření a následnou optimalizaci cesty mezi těmito body. Prostředí je reprezentováno dvěma způsoby – diskrétní aproximací a úplným rozdělením na kontinuální oblasti viditelnosti. Potenciální pozice pro vytvoření pokrytí prostředí jsou generovány existující heuristikou a formulavaná instance SCP je řešena přístupem založeným na GRASP metaheuristice s navržením nových váhových funkcí a pravidel pro konstrukci řešení s motivací lépe reprezentovat geometrický a prostorový charakter úlohy. Cesta skrz vybrané pozice je nalezena za použití TSP řešiče a dále optimalizována za využití large neighborhood search technik. Celý algoritmus je implementován v C++ a jeho funkčnost byla experimentálně ověřena.

Klíčová slova: Problém hledání cesty hlídače, výpočetní geometrie, kombinatorická optimalizace, metaheuristiky, plánování cest

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor, RNDr. Miroslav Kulich, Ph.D., whose expertise and patience provided me with great guidance throughout the course of working on this thesis. I am grateful to my family and friends for supporting me during this time, and I thank my good friend and roommate, Adam, who repeatedly provided me with valuable academic advice. A special thanks goes to my partner Lenka, who has been with me through it all.

Contents

1 Introduction

1.1 Motivation

Path planning is a fundamental component of autonomous mobile robotics, enabling agents to navigate through their environment in an efficient manner. The general goal is to create an optimized and collision-free path; however, the specific character of the application has important implications for the task itself. There are a wide variety of limitations and obstacles to be addressed when dealing with path planning. In so-called offline problems, the mobile agent has full knowledge of its environment in advance, as is often the case in industrial settings such as factories, warehouses, or other structured indoor spaces. In such situations, the focus fully shifts from exploration to optimization, aligning closely with classical problems in operations research and computational geometry, as efficient path planning often presents challenges closely studied in these fields.

In operations research, path planning can be viewed through the lens of combinatorial optimization problems such as the Traveling Salesman Problem (TSP) [1], Vehicle Routing Problem (VRP) [2], or variants thereof, where the objective is to minimize cost functions such as distance, time, or energy while satisfying constraints like capacity or time windows. Sometimes, the challenges that arise require additional spatial reasoning and overlap with problems in computational geometry. These problems are often NP-hard, making finding exact optimal solutions too challenging of a task for all but the smallest instances, hence creating the need for heuristic or approximation-based methods.

In [3], the authors conduct a survey looking at various methods used in robot path planning, focusing on heuristic approaches and comparing them to more classic methods, noting the recent development and successful employment of techniques such as neural networks, genetic algorithms, and ant colony optimization. Such methods have also been used in the field of vehicle routing to great success [4] [5] [6].

The Watchman Route Problem (WRP) [7] occupies a similar space in this field. It is a well-known and studied optimization problem from computational geometry, where the objective is to find the shortest route through a known environment such that the entire environment is visible from at least one point on the path. In autonomous robot path planning, it holds its significance for tasks requiring full area coverage through visibility, such as surveillance, inspection, or monitoring, and models the challenge of computing the shortest route accounting for geometric constraints and sensor limitations.

1.2 Contribution

This work focuses on solving the Watchman Route Problem with limited visibility through the use and modifications of known heuristics. The problem is reformulated into separate subproblems that are solved sequentially.

First, the WRP is reinterpreted in terms of visibility coverage, where the objective is to determine a set of guarding locations from which an entire environment can be observed. To generate candidate guard positions, an existing visibility-based heuristic was used, which produces a representative set of potential coverage points.

A custom polygonal domain *discretization* method was implemented, offering a simplified yet flexible representation of the environment. This technique allows control over the complexity of the problem instance while still reflecting essential geometric features. In parallel, a *continuous* segmentation of the polygonal domain was designed and implemented. This process involved computing all distinct regions of the environment defined by overlaps in visibility regions among the candidate guard locations. This was achieved through modifying a plane sweep algorithm, handling each point where the visibility regions intersect while keeping track of currently active polygons, enumerating each overlapping area, as it represents a real portion of the map. The result is an algorithm which can compute all distinct overlapping areas of generic polygons on a 2-dimensional plane.

Using the potential guarding locations and both the discrete and continuous representations, the visibility coverage component of the WRP was *formulated as a Set Cover Problem (SCP)* instance. To better reflect domain-specific characteristics, several custom functions were introduced to determine the weights of the instance sets, encoding spatial and geometric factors in the SCP formulation. The SCP was then solved using a GRASP-based metaheuristic, for which novel insertion rules were developed to guide the construction of candidate solutions.

Following the selection of a proper set of guard locations, the routing component of the problem was addressed, firstly interpreting the solution of the Set Cover Problem as a complete graph, with nodes representing chosen guard locations and edge weights equal to geometric distances between them in a specific environment. An initial route was computed using a third-party TSP solver. The final solution is further improved by set-wise optimization and route compression using a large neighborhood search technique, again considering all the potential guard locations and treating the whole problem as a Generalized Traveling Salesman (GTSP) instance.

The whole algorithm is implemented in C++ using open-source libraries.

1.3 Structure

In chapter 2, the Watchman Route Problem is formally introduced, alongside its variants and existing nomenclature as presented in current literature. A brief report on state-of-the-art methods for solving the WRP is included. The problem is broken down into a combination of the Set Cover Problem and (generalized) Traveling Salesman Problem, providing a formal definition, theoretical background, and current methods for solving both mentioned problems. Special attention is given to GRASP-based approaches for solving the SCP and a large neighborhood search method for the GTSP, which are used in the following chapters.

The proposed solution is described in chapter 3. It describes the method used for generating redundant number of potential guard locations and the designed methods for obtaining the segmentation of the environment in two different ways, which serve as a basis for formulating an SCP instance representing a WRP. Furthermore, it explains the choices behind custom weight functions for the Set Cover Problem and the proposed insertion rules for the GRASP-based scheme used to solve the instances. It also includes a description of experiments realized to test the proposed algorithm.

Chapter 4 includes a description of experiments performed to test the proposed algorithm and their results, while chapter 5 delves deeper into the strengths and weaknesses of the work, highlighting the space for possible improvements.

2 Background

2.1 Watchman Route Problem

The Watchman Route Problem (WRP) is an optimization problem defined as follows: Find the shortest possible path (cycle) within a polygonal domain[1] such that every point in the said domain is seen from at least one point on the path [7].

A further generalization of the problem introduces a finite visibility radius d for the watchman, which is referred to as $d-$WRP in the literature. However, unlike in the original case with infinite visibility, where surveying the boundary $\bar{P}$ ensured full coverage of the environment P, this no longer holds with limited visibility. For this reason, some authors make a distinction between *d-Watchman Route Problem* (the goal si to survey the boundary $\bar{P}$) and *d-Sweeper Route Problem* (both the boundary $\bar{P}$ and the entire P must be visible from the route) [9].

With motivation from the field of mobile robotics, the authors of [10] also introduce a non-zero radius r, representing the physical size of a mobile autonomous agent performing the surveillance task, referring to the problem as $rd-$Watchman Route Problem ($rd-$WRP).

This work focuses on the $d-$WRP with the objective of surveilling the entire area of a known polygonal domain P.

[1] *Polygonal domain*, P, having n vertices and h holes, is a closed, multiply-connected region whose boundary is a union of n line segments, forming $h+1$ closed (polygonal) cycles. (A simple polygon is a polygonal domain with $h = 0$.) [8]

2.1.1 Solving the WRP

If the polynomial domain P is rectilinear (all sides meet at right angles) and without holes, this problem can be solved in $\mathcal{O}(n)$ computational time [11]. For a general polygon without holes, [12] proposes an $\mathcal{O}(n^2)$ algorithm.

For a general polygonal domain with holes, WRP is NP-hard as proved in [13]. The authors of [14] provide a polynomial-time approximation algorithm with an approximation factor of $\mathcal{O}(log^2 n)$. Here, the approach is to partition the problem into *outer* WRP and *inner* WRP, where a solution to *Outer* WRP surveils the entire area of P outside of the route and serves as an initial input for *Inner* WRP, which seeks to minimize the route while simultaneously surveilling all of P which lies inside the route. Similarly, the authors of [15] split the problem into two – proposing to solve *Art Gallery Problem* first, obtaining the minimum number of guarding locations to cover the entirety of P, and then solving *Traveling Salesman Problem*, finding the shortest route to visit all guarding locations found in the previous step. This approach, where the problem is partitioned into two easier problems that are then solved separately, is called *decoupling* and is often used to solve general $(d-)$WRP.

Techniques outside of decoupling are presented by the authors of [16], where a self-organizing map (SOM) schema is proposed. A more novel approach, which also avoids the need for decoupling, is presented in [10], where the authors cover the environment by a set of convex polygons, each polygon being completely visible from any of its interior points or any point on its border. By visiting each of such polygons, the theoretical watchman can surveil the entire environment.

2.2 Decoupling the WRP into Set Cover and Traveling Salesman Problem

A decoupling approach of choice in this work is treating the WRP as a combination of the Set Cover Problem and Traveling Salesman Problem. The following chapters introduce the SCP and present an overview of state-of-the-art methods that have been successfully used to solve it. Chapter 2.2.2 provides a proper schema how to perform this decoupling.

2.2.1 Set Cover Problem

The Set Cover Problem (SCP) is a well-known combinatorial optimization problem. The general formulation is as follows: Given a set of elements (*universe* U), and a set of subsets (*collection* C), whose union equals U, find the smallest sub-collection of C such that its union is U. The SCP is NP-hard [17] and has been used to formulate and help solve many different problems [18], [19].

The SCP can be formulated as a following integer linear program [20]:

$$\begin{array}{rll}
\texttt{minimize} & \sum_{c \in C} x_c & \\
\texttt{subject to:} & \sum_{c: e \in c} x_c \geq 1 & \texttt{for each } e \in U \\
\texttt{variables:} & x_c \in \{0, 1\} & \texttt{for each } c \in C
\end{array}$$

The *weighted* or *non-unicost* Set Cover Problem introduces a slightly modified objective: instead of minimizing the number of selected subsets, each set is given a weight/cost w_c, and the objective function then minimizes $\sum_{c \in C} w_c \cdot x_c$. In practical applications, it is convenient to represent SCP instances using an incidence matrix $A \in \{0, 1\}^{|U| \times |C|}$, where $A_{e,c} = 1$ if element e is covered by subset c. The ILP formulation then becomes:

$$\begin{array}{rll}
\texttt{minimize} & \sum_{c \in C} w_c \cdot x_c & \\
\texttt{subject to:} & \sum_{c \in C} A_{e,c} \cdot x_c \geq 1 & \texttt{for each } e \in U \\
\texttt{variables:} & x_c \in \{0, 1\} & \texttt{for each } c \in C \\
\texttt{parameters:} & w_c \in \mathbb{R} & \texttt{for each } c \in C \\
 & A_{e,c} \in \{0, 1\} & \texttt{for each } e \in U,\ c \in C
\end{array}$$

When using this representation, it is easier to think of the subsets c of collection C as columns and the elements e of the *universe* U as rows of the matrix A, as they are represented by those respective dimensions.

The SCP has been successfully solved by GRASP-based [21] algorithms. Authors of [22] focus on the unicost SCP. Transforming the SCP into a *Maximum Satisfiability* (MAX-SAT) Problem, multiple different procedures are tested and evaluated. As presented, a GRASP-based approach showed very promising results. In [23], a modified Meta-RaPS (*Meta-heuristic for Randomized Priority Search*) meta-heuristic introduces an advance during the construction phase: the candidates are sorted by a wider selection of priority rules (with simple greedy being one of them) and only *%priority* of the time, the top candidate is selected, with (100-*%priority*) times being replaced by a random candidate.

More recently, the authors of [24] have used this aforementioned selection of varying priority functions as proposed in Meta-RaPS, while creating a new set of rules for iterative column instantiation. Firstly, the rows are sorted by the number of columns that cover them in ascending order. Then, during each iteration, given a specific priority value of all columns $f(c)$, choose column c to be added to the solution following one of these rules:

- randomly with probability proportional to $-f(c)$

- minimizing $f(c)$

with the first rule followed with a probability of $1 - \frac{1}{\#cols}$ and the second rule taking effect with the probability of $\frac{1}{\#cols}$. Here, $\#cols$ is equal to the number of columns that can cover the current uncovered row.

2.2.2 Decoupled problem formulation

The Watchman Route Problem can be decoupled into the Set Cover Problem and the Traveling Salesman Problem using the following schema:

- Let S be a segmentation of the polygonal environment P. $s_i \cap s_j = \emptyset$ for each i, j, $i \neq j$ and $\bigcup_{s \in S} s = S$
- Let G be a set of potential guards, which represent sensing locations along a route through the environment.
- Let M be a set of subsets of G, where each $m_i \in M$, $m_i \subseteq G$ is a subset that expresses which guards can cover the area s_i.

Then it is possible to formulate an instance of a Set Cover Problem, where *universe* U is represented by the segmentation S and *collection* C is represented by *visibility mapping* M.

A solution to such a formulated problem is a set of guards (sensing locations) which, when all visited in an arbitrary order, guarantee to create a route which is a feasible solution to the WRP. Obviously, the objective when constructing this route is to minimize the distance traveled, which leads to the second separate optimization problem, Traveling Salesman.

2.3 Formulating the WRP as a Generalized Traveling Salesman Problem

Similarly, considering the abundance of potential guard locations and the need to cover an entire physical area segmented into clusters with spatial relevance, the WRP can be viewed through the lens of finding the shortest route in a complete, partitioned graph. This problem is known as Generalized Traveling Salesman and is described in this chapter.

2.3.1 Generalized Traveling Salesman Problem

The *Generalized Traveling Salesman Problem* (GTSP), sometimes also referred to as *Set Traveling Salesman Problem* is a generalization of TSP, the arguably most famous problem in combinatorial optimization. In GTSP, the input is a complete directed graph with vertices V, edges E and edge weights w; $G = (V,\ E,\ w)$ with its n vertices partitioned into m disjoint sets. The goal is to find the minimal closed route (cycle) that visits each set exactly once (sometimes at least once). Since GTSP is a direct generalization of TSP, it is also NP-hard, and there exists no approximation algorithm for the general problem. For a special case where the edge weights are metric, an approximation algorithm with a factor of $\mathcal{O}(\log^2 n \cdot \log\log n \cdot \log m)$ is presented in [25].

In [26], the authors present a metaheuristic algorithm for symmetric GTSP based on *iterative local search*, creating three new GTSP neighborhoods to be used during variable neighborhood descent, all of which allow simultaneous modifications of the sequence of the sets and the selection of vertices per set. In addition, these are accompanied by neighborhood exploration algorithms that run in worst-case polynomial time.

Large Neighborhood Search (GLNS) heuristic for the GTSP

An *adaptive large neighborhood search* heuristic for the GTSP is proposed by authors of [27]. In an adaptive large neighborhood search, an initial solution is randomly generated, and later iteratively improved by removing vertices that are part of the solution and inserting different vertices, with the number of removed vertices chosen randomly, all while following specially designed removal/insertion rules. Said rules are the key component of this algorithm. Acceptance of the modified solution is based on a standard simulated annealing criterion as proposed in [28].

Given a graph $G = (V,\ E,\ w)$, partition $P = \{V_1, \ldots, V_m\}$ and a partial tour $T = (V_T,\ E_T)$, denote the sets already visited by T as P_T. Then, the framework for the overall insertion heuristic is as described in algorithm 1:

Algorithm 1: Framework of insertion heuristics. Taken from [27].

Input: A GTSP instance $(G,\ P_V)$ and a partial tour $T = (V_T,\ E_T)$ of G
Output: An updated partial tour T that visits one additional set
1 Pick a set V_i in $P_V \setminus P_T$
2 Find an edge $(x, y) \in E_T$ and vertex $v \in V_i$ that minimizes $w(x, v) + w(v, y) - w(x, y)$
3 Delete the edge (x, y) from E_T, add edges $w(x, v)$ and $w(v, y)$ to E_T and add v to V_T
4 **return** T

There are four different insertion heuristics that can be followed to pick a set V_i in step 2 (as published in [29]). Those are:

1. **Nearest:**

$$V_i^* = \arg\min_{V_i \in P_V \setminus P_T} \min_{u \in V_T} \operatorname{dist}(V_i, u)$$

2. **Farthest**:

$$V_i^* = \arg\max_{V_i \in P_V \setminus P_T} \min_{u \in V_T} \operatorname{dist}(V_i, u)$$

3. **Random:** pick a set V_i uniformly randomly from $P_V \setminus P_T$.

4. **Cheapest:**

$$\arg\min_{V_i \in P_V \setminus P_T} \min_{v \in V_i,\, (x,y) \in E_T} \{w(x,v) + w(v,y) - w(x,y)\}$$

with distance between set V_i and vertex $u \notin V_i$ defined as:

$$\operatorname{dist}(V_i, u) = \min_{v \in V_i} \{\min \{w(v,u),\ w(u,v)\}\} \tag{2.1}$$

In GLNS [27], the authors propose a *unified insertion heuristic*, combining all four of the aforementioned variants into a single rule. First, given a partial tour T, a distance between T and each $V_i \in P_V \setminus P_T$ is computed as

$$d_i = \operatorname{dist}(V_i, T) = \min_{u \in V_T} \operatorname{dist}(V_i, u) \tag{2.2}$$

where $\operatorname{dist}(V_i, u)$ is the same as in equation (2.1). Denoting $\mathcal{L}$ as the number of sets yet to be visited by tour T and given a parameter $\lambda \in [0, \infty)$, a set for the insertion heuristic is chosen as described in algorithm 2. The choice of

Algorithm 2: Set selection for the unified insertion heuristic. Taken from [27].

Input: A GTSP instance $(G,\ P_V)$, a partial tour $T = (V_T,\ E_T)$ of G, $\lambda \in [0, \infty)$
Output: A set $V_i \in P_V \setminus P_T$
1 Randomly select $k \in \{1, \ldots, \mathcal{L}\}$ following unnormalized probability mass function $[\lambda^0, \lambda^1, \ldots, \lambda^{\mathcal{L}-1}]$
2 Pick the set $V_i \in P_V \setminus P_T$ with the kth smalles distance d_i to the tour.
3 **return** V_i

parameter λ gives control over preferred insertion heuristic.

2.3.2 WRP formulation using GTSP

Following the same schema as described in Chapter 2.2.2, upon obtaining segmentation S of the polygonal domain, a set of potential guard locations G and a visibility mapping M, it is possible to express an instance of WRP as a GTSP instance.

Here, the guards are equal to the vertices V of a complete directed graph, the paths between the guards being the edges E and the weights w being the lengths of such paths. A slight complication occurs when formulating the partition of vertices into sets, as in GTSP, the sets are disjoint, meaning that a single vertex can not be assigned to multiple sets. Given the character of the original problem, this does not hold for set covering – a single guard can fully surveil multiple segments of S, just like a subset of a *collection* can contain multiple elements of the *universe* in SCP. However, the transformation of a GTSP with intersecting sets into one with disjoint sets is quite straightforward, as described in [30]:

- Let $G = (V, E, w)$ be a complete graph with clusters (sets) $C_0, C_1, \ldots, C_m$. These clusters might intersect.
- The vertices in V are denoted by numbers $0, 1, \ldots, n$
- For all $x = 1, \ldots, n$, $S_x = \{i \in \mathbb{Z};\ i \in [1, m]\,;\ \ x \in C_i\}$; S_x is a set of indices of all clusters containing x
- for all $x = 1, \ldots, n$, let $q_x = |S_x|$

Then it is possible to create a graph $G' = (V', E', w')$ with $m+1$ non-intersecting clusters $C'_0, C'_1, \ldots, C'_m$. For each vertex x in V, V' will contain q_x *copies* of x denoted $u_{x,i}$ for all $i \in S_x$, with $u_{x,i}$ in C'_i. The weights in w' equal to:

1. 0 for all edges $(u_{x,i}, u_{x,j})$ for each $x = 1, \ldots, n$ and all $i, j \in S_x$
2. $w_{x,y}$ for all edges $(u_{x,i}, u_{y,j})$ for each distinct $x, y = 1, \ldots, n$ and all $i \in S_x,\ j \in S_y$

This leads to an increase in problem size, but such formulated problem can be an input into any existing GTSP solver.

3 Proposed Solution

This chapter describes the proposed approach for solving the d-WRP. Firstly, an existing heuristic is used to generate a set of potential guard locations G in a known environment. The output of this step is a set of coordinates which guarantee a full coverage of the environment. The selected heuristic is used in such a manner that there is an overabundance of generated locations so that it is possible to find good quality solution during the next steps. An overview of this heuristic is in Chapter 3.1.

The environment is then represented in two independent ways. Chapter 3.2 describes a simple discrete approximation of arbitrary resolution, which provides an imperfect representation while offering a full control over the problem size. Chapter 3.3 proposes an algorithm which, when provided with visibility fields of potential guard locations, enumerates all distinct segments which are results of the overlapping fields of view of each guard in set G. This continuous representation is complete and allows for the generation of a proper solution for the Watchman Route problem, but leads to a greater problem size.

Both approaches are used when obtaining a visibility mapping M of the set of guards and the environment, which allows for the formulation of the SCP instance. See Chapter 2.2.2 for a proper definition of visibility mapping M and a schema on how the problem is formulated. This enables the use of an SCP solver to obtain a selection of guard locations providing full coverage of the environment while minimizing a specified cost of the selected guards.

Chapter 3.4 is dedicated to the part of the framework where the SCP is solved, describing the GRAPS-based metaheuristic used and custom cost functions and construction insertion rules. Those were designed with spatial-based reasoning to further increase the quality of the solution found with respect to the original problem; finding the optimal route through the environment.

Chapter 3.5 describes how the problem can also be solved as an instance of the Generalized Traveling Salesman Problem. GLNS, a large neighborhood search heuristic for the GTSP, described in Chapter 2.3.1, is used as a technique for further optimization of the solution found using the decoupled approach.

A diagram depicted in Figure 3.1 shows the entire solution framework.

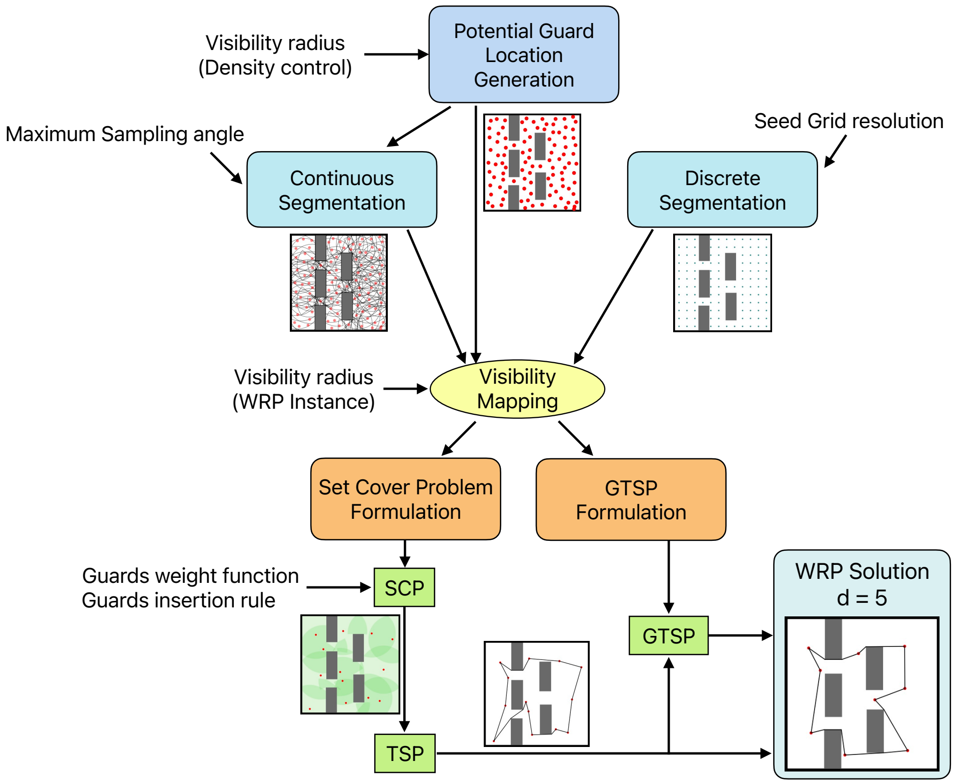


Figure 3.1: Flowchart of the entire solution framework

3.1 Generating potential guard locations

The heuristic used to generate potential guard locations is taken from [31]. Here, the authors formulate the Omnidirectional Sensor Placement Problem (OSPP), seeking to place a minimum number of static point sensors in a continuous 2D environment to achieve a user-defined coverage of the area. Importantly, the problem setting is compatible with the one discussed in this work, defining both *unlimited* $\text{Vis}_\infty(g)$ and *limited-range visibility model* $\text{Vis}_d(g)$ within a polygonal domain $\mathcal{W}$ with holes. The definitions of visibility models are stated in Equations

(3.1) and (3.2) and are adopted for the purpose of this work.

$$\mathrm{Vis}_{\infty}(g) := \{p \in \mathcal{W};\ \overline{gp} \subset \mathcal{W}\} \tag{3.1}$$

$$\mathrm{Vis}_{d}(g) := \{p \in \mathcal{W};\ \overline{gp} \subset \mathcal{W} \wedge ||g-p|| \leq d\} \tag{3.2}$$

Where $\overline{gp}$ is a line segment connecting g and p and $||.||$ is the Euclidean norm.

The proposed framework is a class of *hybrid accelerated refinement* (HAR) heuristics. This technique utilizes a set of different sensor-placement methods and refines their output by efficiently filtering out redundant sensor locations. The core of this algorithm is iteratively choosing the guard location which covers the largest uncovered area of the environment and eliminating guard locations which become obsolete in the process.

In the context of generating potential guard locations for the WRP route, the output of this method is a "well behaved" candidate for several reasons:

1. The generated guard locations guarantee full coverage of the environment.
2. The spatial distribution of the generated locations is approximately uniform across the defined region.
3. Visibility radius of the sensors is an input parameter, allowing fully custom sampling factor (setting the visibility radius lower increases the number of generated locations and vice versa).

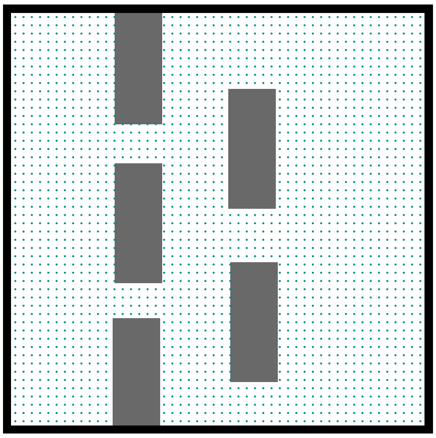
Increasing the number of potential guarding locations expands the overall problem size, resulting in greater computational complexity. However, it also improves the upper bound of the quality of the achievable solution by offering more flexibility in optimization.

To ensure a big enough set of potential guarding locations, input visibility radius for the heuristic was set to values below 1 meter, with all further experiments being conducted with visibility radius $d \geq 2$ meters.

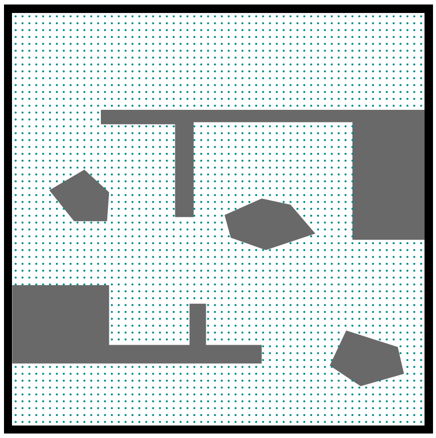
3.2 Discrete environment representation

For ease of implementation and fully customizable problem size, it is desirable to represent the environment as a grid of points (further referred to as *seeds*) of predetermined resolution. Note that such a representation does not represent the environment in its entirety; however, as proved by later experiments, with high enough grid resolution, it rarely leads to leaving any areas uncovered on simpler maps. The resolution of the point grid is an input parameter of the algorithm. Figure 3.2 depicts the principle of discrete environment representation.

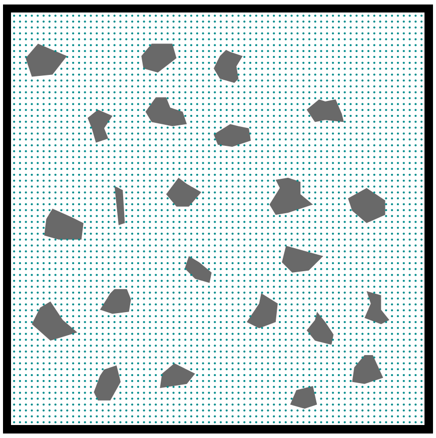
The visibility mapping M is obtained by determining whether the seed s_i is visible from the guard g_i for each pair $(g_i,\ s_i)$. This step has a computational complexity of $\mathcal{O}\left(|S| \cdot |G|\right)$.



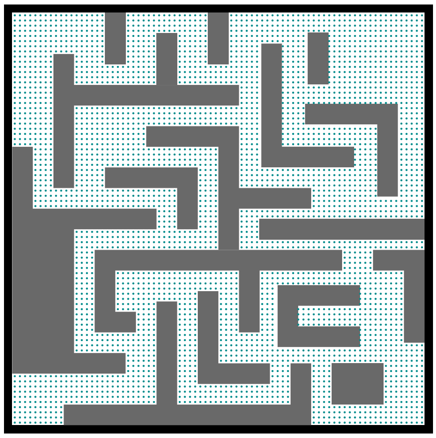
(a) map: Gaps, seed grid resolution: 50



(b) map: Complex2, seed grid resolution: 60



(c) map: Potholes, seed grid resolution: 70



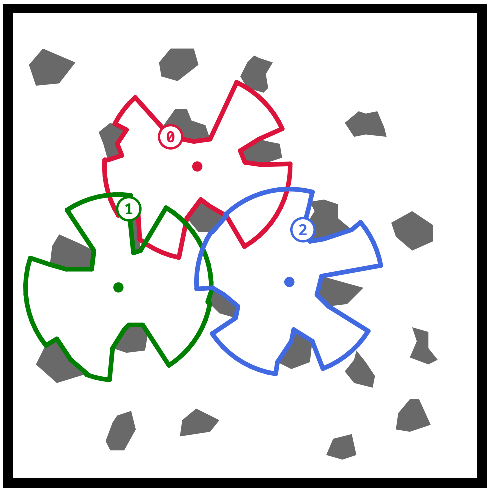
(d) map: Maze, seed grid resolution: 80

Figure 3.2: Example of discrete environment representation with different seed grid resolution on four different maps, each representing an environment of 20×20 meters.

3.3 Continuous environment representation

In order to formulate a continuous and complete visibility mapping M for an instance of SCP given a set of potential guarding locations, it is necessary to enumerate all present disjoint areas which are created by overlapping visibility fields of said guards. A simple representation of the task can be seen in Figure 3.3.

Without any obstacles present, the field of view of each guard is circular; however in the general case (considering obstacles in the environment and its border), it consists of a combination of radial sections and line segments. For this reason, the entire visibility region is converted into a polygon. The maximum sampling angle for conversion to a polygon can be arbitrarily chosen and is an

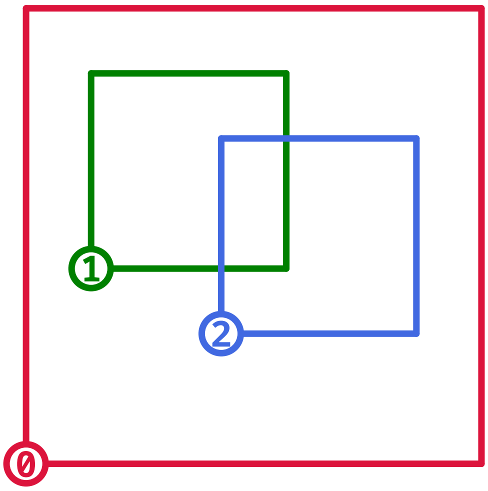



```
VISIBILITY MAPPING:
M = {
{0},
{1},
{2},
{0,1},
{0,2},
{1,2}
}
6 segments found.
```

Figure 3.3: Visibility fields of three guards on map Potholes.

input parameter of the algorithm. The problem now becomes: How to find all different overlapping regions of n non-convex polygons in a 2D plane?

It is apparent that a brute force solution will not suffice, because it would require to check an existence of an overlap of all m-tuples of n polygons for $m = 1, \ldots, n$. Furthermore, simply enumerating all present overlaps in such manner does not account for situations, where an overlap of m polygons is fully inside another area, say a different polygon. Such overlapping must be filtered out, as it would represent a non-existing space in the environment. An illustration of this issue is depicted in Figure 3.4.




```
VISIBILITY MAPPING:
M = {
{0},
{0,1},
{0,2},
{0,1,2}
}
4 segments found.
```

Figure 3.4: Overlap of polygons 1 and 2 is fully inside polygon 0, this area should only be reported once, as an intersection of all three.

3.3.1 Adaptation of the Plane-Sweep Algorithm

The chosen approach is based on designing an augmentation of the *Plane-Sweep algorithm* for finding all intersections of line segments in a 2-dimensional plane as first described in [32] and later published in [33]. Here, the input is a set of n line segments $S = \{s_1, \ldots, s_n\}$ and the algorithm finds all their intersections while avoiding checking all segments pairwise. The idea behind the approach is that it is only necessary to check for an intersection of two line segments if they are "next to each other" on the 2D plane. The algorithm moves a theoretical *sweep line* in a predetermined direction, visiting all *event points* (start and end points of line segments, as well as segment intersections) while keeping track of currently active segments and their order.

This paradigm can be used to solve a problem formulated in Chapter 3.3. With a known structure of geometrically ordered line segments, it is possible to keep track of presently active *polygons* defined by said line segments. Overlapping areas of active polygons are then detected at each event point by interpreting the line segment ordering as described in Algorithm 6.

First, all line segment start and end points are stored in a *balanced binary search tree* (BBST) structure called the *event queue*, with custom geometric ordering. The usual way to order event points is primarily *up-down* and *left-to-right* in the case where two points are next to each other on the horizontal axis. During the execution of the algorithm, the found intersections will be inserted into this structure as new entries, which is the reason to use a BBST. See Algorithm 3.

Algorithm 3: $FindIntersections(S)$. Inspired by [33].

Input: A set of n line segments $S = \{s_1, \ldots, s_n\}$
Output: All intersections of segments in S

$Q \leftarrow \{\}$; `// event queue`
$\mathcal{T} \leftarrow \{\}$; `// sweep-line status`
$\mathcal{M} \leftarrow [\,]$; `// map of segments beginning points`
for each segment $s \in S_A$ **do**
  $Q.insert(s_p)$
  $M[s_p] \leftarrow s$
  $Q.insert(s_q)$; `//` s_p `and` s_q `are endpoints of` s
while Q *is not empty* **do**
  $p \leftarrow pop(Q)$
  $HandleEventPoint(p, \mathcal{T})$

Active line segments are also stored in a BBST structure, called the *sweep-line status*. Here, line segments are ordered primarily by the coordinates of their intersection with the sweepline. In case of an intersection of two or more segments (where the coordinates of their intersection with the sweepline is the same), their position in the structure is determined by the angle between the segment and the sweep-line. A visualization of this ordering is shown in Figure 3.5.

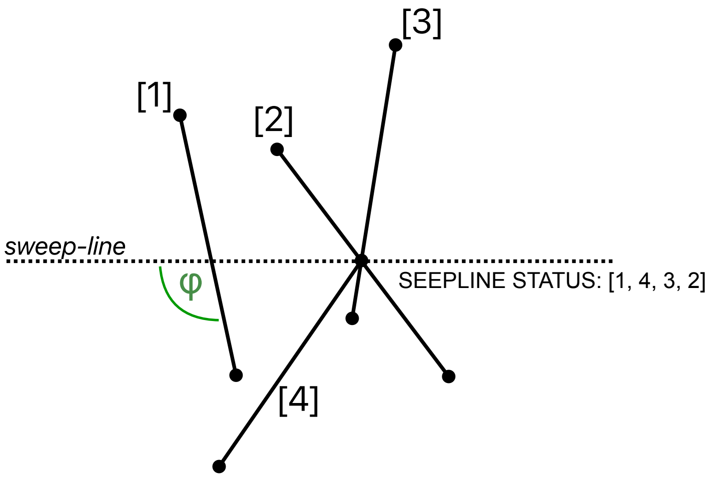


Figure 3.5: Segments ordering example.

The core procedure of the original algorithm is described in full detail in Algorithm 4. The function *ReportCurrentOverlaps*$(p, \mathcal{T})$ on line `1` is the designed augmentation and is described in the following chapter. In the original algorithm, at each event point p, all segments that start, end or go through p are collected and inserted into three separate sets. The segments that start in p are present in $M[p]$, and the other segments are present in $\mathcal{T}$. Finding a specific element in a BBST has a complexity of $\mathcal{O}(logn)$, and all segments that go through p are stored next to each other in the sweep-line status structure $\mathcal{T}$. Then, if there is more than one segment in union of all named sets, an intersection must be present. This is where the algorithm detects and reports the intersections.

The segments ending in p must then be removed from $\mathcal{T}$, and the segments beginning in p must be inserted into $\mathcal{T}$. Segments that have p in their interior might change their order, so they are removed and reinserted into the structure (they will be sorted upon reinsertion). Set operations on lines `8` and `9` in Algorithm 4 are responsible for this process.

Next up, one of the following situations is currently at hand: either all segments handled in this event point p ended in p, or there are segments which contain p and exist below the sweep-line. In the former case, the algorithm must find the segments which are directly to the left and to the right of p and check whether those intersect below the sweep-line; if they do, their intersection must be inserted into Q as a new event (lines `10` to `13` of Algorithm 4). In the latter case, there are segments in p which "continue" below the sweep-line. They are either newly inserted into $\mathcal{T}$ or just changed their position in $\mathcal{T}$ due to being present in an intersection. The leftmost and rightmost of these segments are checked for intersection with their left and right neighbor in $\mathcal{T}$, respectively. Again, if such intersections exist, they are inserted into Q. See lines `14` to `20` of Algorithm 4 and a helper method $FindNewEvent(s_l, s_r, p)$ provided in Algorithm 5.

It is important to note a limitation of the original plane-sweep algorithm for intersection detection: it assumes that there are no segments that overlap, neither partially nor completely. Due to the geometric character of the original problem, this limitation cannot be unaddressed; visibility fields of guards are often shaped by nearby obstacles or environment borders, which results in many overlapping line segments. This issue is depicted in Figure 3.6. In the augmented version of the algorithm, overlapping segments are inserted next to each other in the tree structure in an arbitrary order, and the handling procedure described in Algorithm 6 checks whether segments active in the current event point overlap.

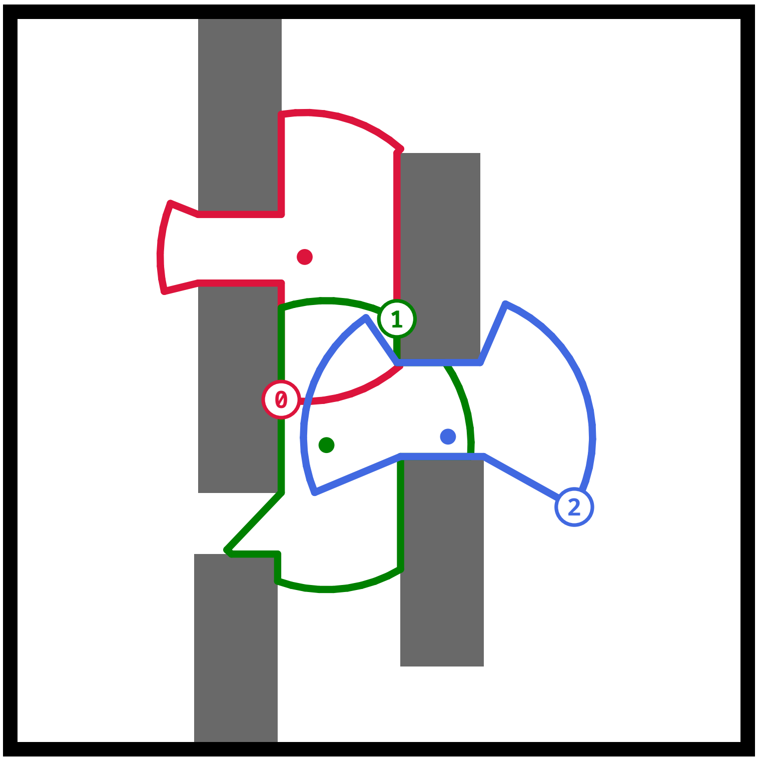


Figure 3.6: Line segments defining visibility fields of three guards fully or partially overlap on obstacle borders.

Algorithm 4: $HandleEventPoint(p, \mathcal{T})$. Inspired by [33].

Input: Event point p, sweep-line status $\mathcal{T}$

$ReportCurrentOverlaps(p, \mathcal{T})$; `// augmentation`
$U(p) \leftarrow$ all segments in $\mathcal{T}$ that begin in p
$C(p) \leftarrow$ all segments in $\mathcal{T}$ that have p in their interior
$L(p) \leftarrow$ all segments in $\mathcal{T}$ that end in p
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
    Report p as an intersection.
    Report $U(p)$, $L(p)$ and $C(p)$
$\mathcal{T}.remove(L(p) \cup C(p))$
$\mathcal{T}.insert(U(p) \cup C(p))$
if $U(p) \cup C(p) = \emptyset$ **then**
    $s_l \leftarrow$ first segment to the left of p
    $s_r \leftarrow$ first segment to the right of p
    $FindNewEvent(s_l, s_r, p)$
else
    $s' \leftarrow$ the leftmost segment in $U(p) \cup C(p)$
    $s_l \leftarrow$ the segment to the left of s' in $\mathcal{T}$
    $FindNewEvent(s_l, s', p)$
    $s'' \leftarrow$ the rightmost segment in $U(p) \cup C(p)$
    $s_r \leftarrow$ the segment to the right of s'' in $\mathcal{T}$
    $FindNewEvent(s'', s_r, p)$

Algorithm 5: $FindNewEvent(s_l, s_r, p)$. Inspired by [33].

$c \leftarrow$ intersection of s_l and s_r
if c exists **then**
    if $(c_y < p_y)$ **or** $(c_y = p_y$ **and** $c_x > p_x)$ **then**
        `// c is below or to the right of` p
        $Q.insert(c)$

Algorithm Augmentation

As already mentioned in Chapter 3.3.1, the idea behind the plane-sweep augmentation is this: If, at each event point p, the algorithm keeps track of all currently active segments in their geometric order, and these segments define the boundaries of closed polygons in the plane, it is possible to enumerate all currently active polygons and report their overlaps below the sweepline. This process is described in Algorithm 6 and Figure 3.7 visually shows what the output of $ReportCurrentOverlaps(p)$ is.

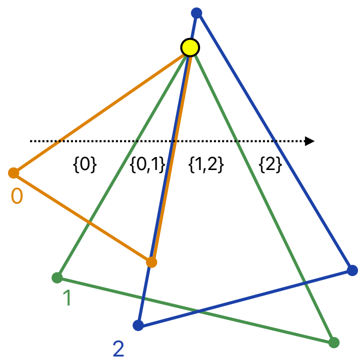


Figure 3.7: At event point p (yellow), the *ReportCurrentOverlaps*(p) function reports all areas below p in terms of polygon overlaps. The dotted arrow symbolizes that the procedure moves from left to right, as it iterates over currently active segments in this order.

Algorithm 6: *ReportCurrentOverlaps*($p, \mathcal{T}$)

Input: Event point p, sweep-line status $\mathcal{T}$

Output: All overlapping areas of polygons which are active below p

$P_A \leftarrow \{\}$; `// set of open polygons`

$\varphi_{now} \leftarrow -1$

$x_{now} \leftarrow -1$

$P_B \leftarrow \{\}$; `/* Set of polygons which have a defining segment with` φ_{now}`,` x_{now} `*/`

for each segment $s \in S_A$ **do**

  $\varphi_s \leftarrow$ angle between the sweep-line and s

  $x_s \leftarrow x$ coordinate of the intersection of s with the sweep-line

  if $(\varphi_s = \varphi_{now}$ **and** $x_s = x_{now})$ **or** $\varphi_{now} = x_{now} = -1$ **then**

    $P_B.insert(PolygonID(s))$

  else

    $P_A \leftarrow (P_A \cup P_B) \setminus (P_A \cap P_B)$

    if P_A *is not empty* **then**

      Report P_A

    $\varphi_{now} \leftarrow \varphi_s$

    $x_{now} \leftarrow x$

    $P_B.clear()$

    $P_B.insert(PolygonID(s))$

Here are some of the more crucial parts of the procedure: a structure that keeps track of the polygons which are open in each iteration is denoted P_A. All segments which are currently active at event point p are checked in their geometric order (as they are stored in the sweepline status, so there is no need to re-sort). The `for` loop starting at line `6` iteratively collects all line segments with the same angle φ_s and same intersection with the sweep-line x_s and stores their respective polygon IDs in P_B. When the angle or intersection of the current segment is different from those of the previous segments (`else` clause on line `11`), it is necessary to check which polygons are being opened and closed by line segments stored in P_B up until now. The set operation on line `12` is responsible for this step: $P_A \cap P_B$ defines all polygons which are present in both sets, meaning those were open before this iteration (they are in P_A), and are being closed in this iteration (the part of P_B which is also in P_A). $P_A \cup P_B$ contains all polygons that were open before (P_A), are now being opened or closed (P_B). By performing set subtraction $(P_A \cup P_B) \setminus (P_A \cap P_B)$, the resulting set will only contain polygons which:

1. Were open in the previous iteration and are not being closed in the current iteration.

2. Were not open in the previous iteration and are being open now.

The contents of the open polygon set P_A can now be reported, as there is a physical area in the environment where all these polygons overlap. After this step, set P_B is emptied, because from this iteration onward, it will be keeping track of segments with new values of φ_s and x_s.

3.4 Solving the decoupled WRP

With a set of potential guarding locations and visibility mapping obtained as described in the previous chapter, it is possible to formulate an instance of the Set Cover Problem (see Chapter 2.2.2) and find a solution using any SCP solver. This chapter describes the methods used to achieve this goal, namely the GRASP heuristic and domain-specific details which were employed to better serve the original purpose of the problem.

3.4.1 GRASP

The Greedy Randomized Adaptive Search Procedure (GRASP), introduced in [21], builds a randomized solution using a greedy strategy, then refines it through local search. This process repeats until a chosen stopping criterion is met (see Algorithm 7).

Algorithm 7: $GRASP(I)$. Inspired by [21].

Input: Problem Instance I
Output: Best found solution of instance I.
$bestSolution \leftarrow$ None
while *Stopping criterion not met* **do**
  $solution \leftarrow ConstructGreedyRandomizedSolution(I)$
  $solution \leftarrow ImproveByLocalSearch(I, solution)$
  if *costOf(solution)* $<$ *costOf(bestSolution)* **then**
    $bestSolution \leftarrow solution$
return $bestSolution$

3.4.2 Using GRASP for the Set Cover Problem

For the clarity of the rest of this chapter, consider that an instance I of the SCP consists of:

- Number of elements n
- Number of subsets m
- Adjacency matrix $A \in \{0,1\}^{n \times m}$, where each row represents an element and each column represents a subset. If $A_{ij} = 1$, then element (row) i is covered by subset (column) j.
- Vector $\mathbf{c}$ of length m of costs associated with the subsets. $\mathbf{c}_j \in \mathbb{R}$ is the cost of including subset (column) j in the solution.

Therefore, when talking about a specific SCP instance, all the information above is available.

The construction phase for the SCP is designed as follows: first, for each column j, calculate how many uncovered rows are covered by it. Denote this number as u_j. Then for each column, calculate its score $e_j = \frac{u_j}{1+c_j}$ and keep note of the lowest e_{min} and highest e_{max} scores. Construct the candidate list by putting columns with $e_j > e_{min} + \alpha(e_{max} - e_{min})$ on it. $\alpha \in [0,1]$ is a parameter that controls the greediness of this process – the higher the value of α, the better the score is required for the column to be put on the candidate list. Then select one of the columns on the candidate list by a chosen method, here referred to as the "insertion rule". Mark all rows covered by the selected guard. Repeat this process until the solution is feasible, that is, all rows are covered. See Algorithm 8. An overview of proposed insertion rules is described in the following chapters.

Algorithm 8: *scpConstructInitialSolution*(I, α)

```
Input: Problem Instance I
Output: Initial greedy randomized solution
solution ← {}
covered ← {}
nCovered ← 0
while nCovered < n do
    e_min ← ∞
    e_max ← −∞
    CL ← {}
    for each column c_j do
        u_j ← number of uncovered rows which c_j covers
        e_j ← u_j/(1 + c_j)
        if e_j < e_min then
            e_min ← e_j
        if e_j > e_max then
            e_max ← e_j
    for each column c_j do
        if e_j > e_min + α(e_max − e_min) then
            CL.add(e_j)
    k ← InsertionRule(CL)
    solution.insert(k)
    Update covered, nCovered according to the chosen column k
return solution
```

The local search operators later used during the improvement phase are **1-0, 1-1 and 2-1 exchange**.

1-0 Exchange

The purpose of 1-0 exchange is to remove useless columns from the solution. All columns selected during the initial construction process are checked to see whether they offer "unique" coverage, that is, if they cover a row which no other column in the solution covers. Columns not meeting this condition are removed from the solution. The pseudocode is in Algorithm 9.

Algorithm 9: 1-0 Exchange for local search

Input:
Problem Instance I
Solution $\mathcal{X}$, containing indices of columns which are part of the solution
Map T, where $T[j]$ is the set of rows covered by column j,
Vector $\mathbf{v}$, where $\mathbf{v}[i]$ is the number of columns in the solution which cover row i
Output: Improved solution $\mathcal{X}$

for each $j \in \mathcal{X}$ **do**
  is_useless $\leftarrow$ `true`
  for each $i \in T[j]$ **do**
    if $\mathbf{v}[i] \leq 1$ **then**
      is_useless $\leftarrow$ `false`
  if *is_useless* **then**
    for each $i \in T[j]$ **do**
      $\mathbf{v}[i] \leftarrow \mathbf{v}[i] - 1$
    remove j from $\mathcal{X}$
return $\mathcal{X}$

1-1 Exchange

1-1 exchange iteratively attempts to improve the current solution by replacing one column in the solution with another that is not currently selected. For each column in the solution, it identifies rows that would become uncovered if that column were removed. It then looks for an unused column that has a lower cost and can cover all of those critical rows. If such a column is found, the algorithm performs the exchange, updates the row coverage counts, and restarts the process. This continues until further improving exchanges can not be made. The 1-1 exchange is decsribed in detail in Algorithm 10.

Algorithm 10: 1-1 Exchange for local search

Input:
Problem Instance I
Solution $\mathcal{X}$, containing indices of columns which are part of the solution
Map T, where $T[j]$ is the set of rows covered by column j,
Vector $\mathbf{v}$, where $\mathbf{v}[i]$ is the number of columns in the solution which cover row i
Output: Improved solution $\mathcal{X}$

```
improved ← true;
while improved do
    improved ← false;
    for each j_out ∈ 𝒳 and not improved do
        toCover ← {};
        for each i ∈ T[j_out] do
            if v[i] ≤ 1 then
                toCover ← toCover ∪{i};
        for j_in ← 0 to m − 1 and not improved do
            if c[j_in] − c[j_out] < 0 and j_in ∉ 𝒳 and T[j_in] ⊇ toCover then
                for each i ∈ T[j_out] do
                    v[i] ← v[i] − 1;
                for each i ∈ T[j_in] do
                    v[i] ← v[i] + 1;
                𝒳 ← 𝒳 \ {j_out} ∪ {j_in};
                improved ← true;
return 𝒳
```

2-1 Exchange

2-1 Exchange works very similar as 1-1 exhange, except that this time, instead of removing a column and seeing whether another cheaper column can replace it while making the solution feasible, columns are removed in pairs and replaced by a single column which has a lower cost than the two removed combined, while covering the missing rows, if such column exists. The process repeats until no further improving exchanges are possible.

Algorithm 11: 2-1 Exchange for local search

Input:
Problem Instance I
Solution $\mathcal{X}$, containing indices of columns which are part of the solution
Map T, where $T[j]$ is the set of rows covered by column j
Vector $\mathbf{v}$, where $\mathbf{v}[i]$ is the number of columns in the solution which cover row i

Output: Improved solution $\mathcal{X}$

```
improved ← true;
while improved do
    improved ← false;
    for each distinct pair j1, j2 ∈ X and not improved do
        v' ← copy of v;
        toCover ← {};
        for each i ∈ T[j1] do
            v'[i] ← v'[i] − 1;
        for each i ∈ T[j2] do
            v'[i] ← v'[i] − 1;
        for each row i such that v'[i] ≤ 0 do
            toCover ← toCover ∪ {i};
        for j_in ← 0 to m − 1 and not improved do
            if c[j_in] − c[j1] − c[j2] < 0 and j_in ∉ X and T[j_in] ⊇ toCover then
                for each i ∈ T[j_in] do
                    v'[i] ← v'[i] + 1;
                X ← X \ {j1, j2} ∪ {j_in};
                v ← v';
                improved ← true;
return X
```

3.4.3 Determining guard cost

A crucial part of an SCP instance is the vector of costs $\mathbf{c}$. As a reminder, in the decoupled formulation, columns (subsets) of the SCP instance are potential guarding locations on a 2 dimensional map. For the purposes of obtaining better results in the later task of finding a shortest route through the environment, the costs should reflect some sort of geometrical relevancy of each guard. In total, three different cost functions for potential guards were designed:

- **Unicost** — the cost of all guards is set to 1. This means that during the construction of the candidate list, all guards are equally likely to be put on the list.

- **Base** — the cost of each guard $c(g_j)$ is calculated as the sum of distances from this guard to all other potential guards in the environment.

$$c(g_j) = \sum_{k \in G} \mathrm{dist}(g_j, g_k) \tag{3.3}$$

- **Quadratic** — the cost of each guard $c(g_j)$ is calculated as the sum of distances from this guard to all other potential guards in the environment, squared:

$$c(g_j) = \left(\sum_{k \in G} \mathrm{dist}(g_j, g_k) \right)^2 \tag{3.4}$$

The idea behind tying the cost of each guard to its distance from all other guards on the map is to prioritize guard locations that are in the middle of larger areas and suppress those located near concave corners. During preprocessing of each SCP instance, guard costs are normalized to the interval $[0, 1]$. This normalization is primarily driven by technical constraints: as the problem size grows (with more guards and larger maps), costs can increase without a guaranteed upper bound, potentially causing variable overflow during runtime.

3.4.4 Custom insertion rules for the initial solution construction

As described in Algorithm 8, after constructing *candidate list* CL, one of the columns on the list must be selected and inserted into the solution. The proposed insertion rules are:

- **Random choice** — randomly select a guard from the candidate list CL.
- **MinMinPath** — for each guard on the candidate list $g_c \in CL$ and each distinct pair of guards that are already part of the solution $g_j,\ g_k \in \mathcal{X}$, find the shortest path $g_j \to g_c \to g_k$. Then, choose the the guard with the *minimal* shortest path.

$$g^\star = \operatorname*{argmin}_{g_c \in CL} \min_{g_j, g_k \in S} \left(\mathrm{dist}(g_j, g_c) + \mathrm{dist}(g_c, g_k) \right) \tag{3.5}$$

- **MaxMinPath** — for each guard on the candidate list $g_c \in CL$ and each distinct pair of guards that are already part of the solution $g_j,\ g_k \in \mathcal{X}$, find the shortest path $g_j \to g_c \to g_k$. Then, choose the the guard with the *maximal* shortest path.

$$g^\star = \operatorname*{argmax}_{g_c \in CL} \min_{g_j, g_k \in S} \left(\mathrm{dist}(g_j, g_c) + \mathrm{dist}(g_c, g_k) \right) \tag{3.6}$$

3.5 Solving the WRP as a Generalized Traveling Salesman Problem

Using the schema described in Chapter 2.3.2, the Watchman Route Problem is also formulated and solved as a GTSP instance. The solver used for this purpose is a *large neighborhood search* heuristic published in [26], further referred to as GLNS. A short overview of this method is provided in Chapter 2.3.1.

A note on the language used in this chapter: while in the Set Cover + TSP decoupled approach it is convenient to think of each guard as covering some distinct areas on a map ("guard i covers areas x, y, z"), in GTSP, it makes more sense to think of guards as vertices which belong to clusters ("guard i belongs to clusters x, y, z"). Furthermore, using the term "cluster" instead of "set" for the GTSP should decrease the undesired confusion between the two approaches. For the purpose of solving the Watchman Route Problem, those formulations mean the same thing.

As already mentioned in Chapter 2.3.2, formulating the WRP as a GTSP instance can not be done without expanding the instance due to the natural characteristic of individual guard locations (vertices) being part of multiple clusters, which is resolved by creating duplicate vertices for each cluster they are present in and setting an edge of weight 0 between the duplicates. Figure 3.8 shows a visual representation of the process on a minimal example with vertices `1, 2, 3` and clusters `A, B, C, D, E`.

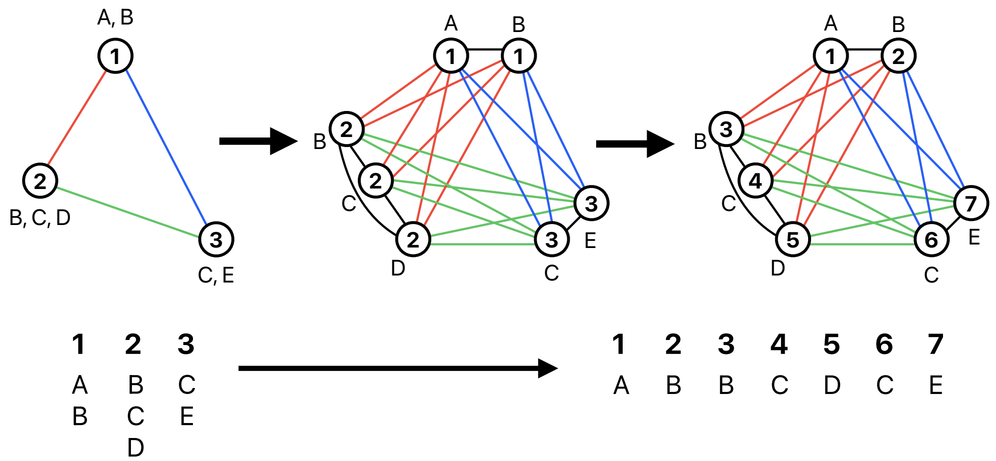


Figure 3.8: A visual representation of formulating a GTSP instance with vertices which belong to multiple sets as a one-in-a-set instance.

At the beginning of each GLNS run, the solver constructs a random initial solution. At first, two random vertices from different clusters are selected and then the solver inserts new vertices into the solution until all clusters are represented in the solution. This phase of the algorithm can be substituted by supplying the solver with a result of the decoupled approach and building the initial solution from it. This process is complicated a little by the need to reformulate the found SCP solution such that it fits the expanded instance. This process is described in Algorithm 12.

Algorithm 12: Transformation of a solution of the decoupled approach to a solution of an expanded GTSP instance

Input:
Tour through the original graph T
Expanded instance of the original graph with map C, where $C[i]$ is a cluster that vertex i in the expanded graph belongs to
Map D, where $D[j]$ is a set of indices expressing which vertices in an expanded graph are duplicates of vertex j
Output:
Tour through the expanded graph T_{exp} which is equivalent to the input tour in the original graph

$T_{exp} \leftarrow [\,]$
$T_{tmp} \leftarrow [\,]$
$C_{tmp} \leftarrow [\,]$
$visited \leftarrow \{\}$
for each $j \in T$ **do**
    for each duplicate $d \in D[j]$ **do**
        $T_{tmp}.push(d)$
        $C_{tmp}.push(C[d])$
for each $k \in T_{tmp}$ **do**
    if $C_{tmp}[k] \notin visited$ **then**
        $T_{exp}.push(k)$
return T_{exp}

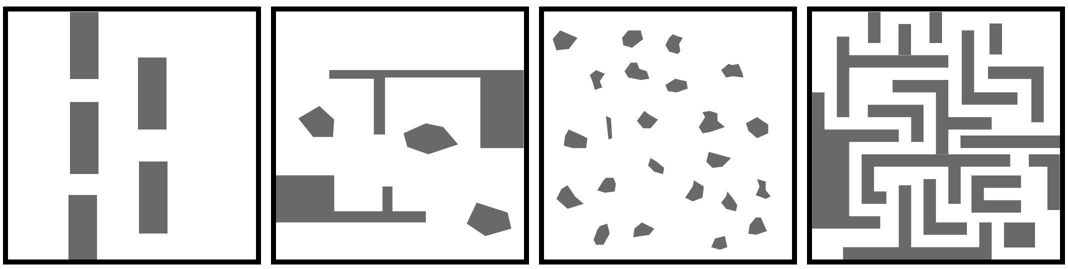
3.6 Implementation details

The whole algorithm was implemented in C++. For computing visibility regions, fundamental geometric operations and visualization, `TřiVis` [34] was used. The algorithm implemented to obtain continuous environment representation relies on the use of the GMP [35] library for multi-precision arithmetic. The Traveling Salesman instances were all solved using the LKH3 [36] solver. The GLNS solver used is originally written by authors of [26] in Julia. For compatibility with the rest of the work, a C++ implementation provided by [37] was used.

4 Results

4.1 Technical setting

All further tests were performed on one or multiple of the following maps: Gaps, Complex2, Potholes, and Maze, as they are displayed on Figure 4.1. All maps have been scaled so that they represent a 20×20 meters environment. All other variables and input parameters used to run specific tests are always provided with the results in the following chapters. All experiments were performed on an Apple MacBook Pro, with an Apple M3 Pro 12-core CPU, 18 GB of RAM, and running MacOS 15.3.2.



(a) Gaps **(b)** Complex2 **(c)** Potholes **(d)** Maze

Figure 4.1: Maps used in further experiments.

4.2 SCP Approach

4.2.1 Parameter testing

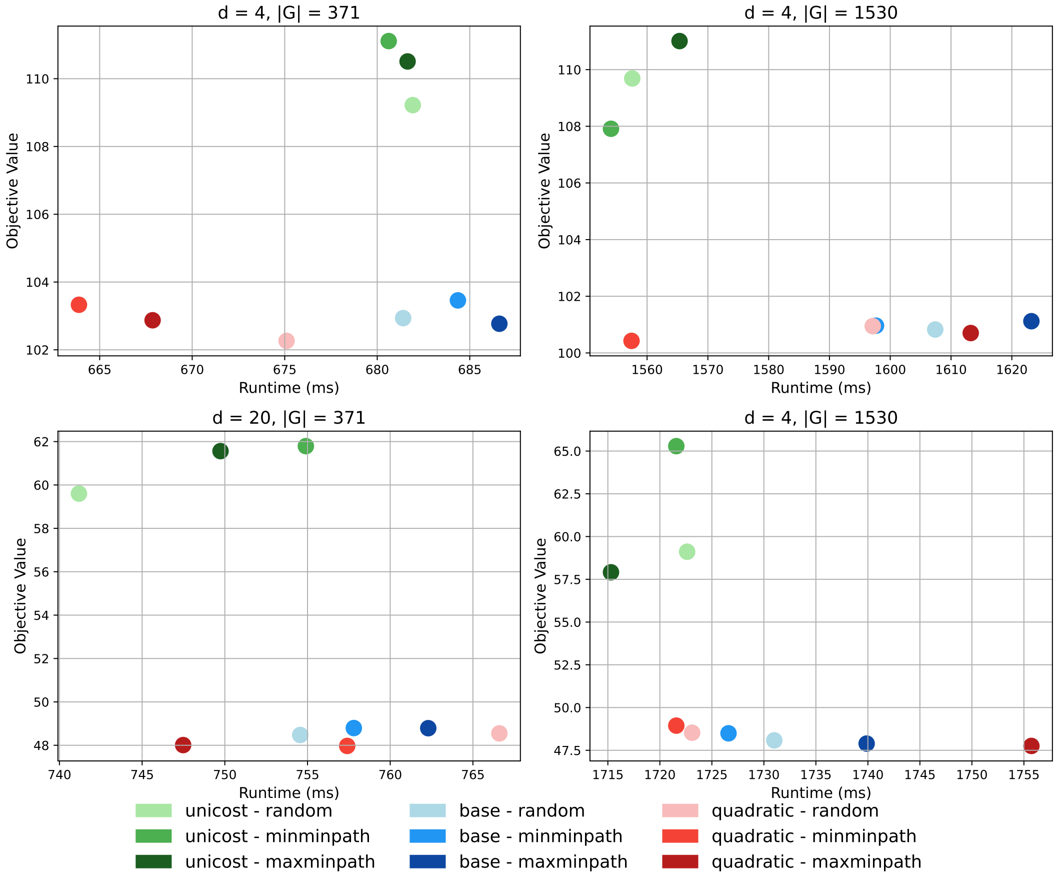


Figure 4.2: Results of the decoupled approach: average performance of different combinations of SCP metrics and insertion rules. Map: Complex2.

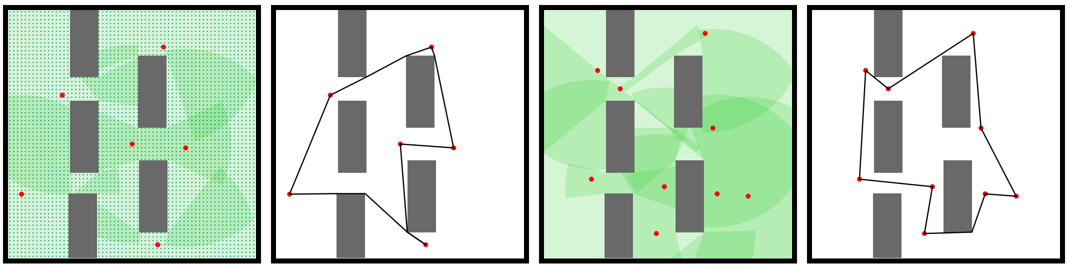
The proposed metaheuristics for the SCP were tested on all 4 maps mentioned in Chapter 4.1. On all maps, each combination of SCP metric and SCP insertion rule were tested with visibility radii $d = 4$ and $d = 20$ and with two different sets of potential guard locations, with one being significantly larger than the other. Therefore, 36 different configurations were tested on each map.

Each configuration was run 50 times. The results were very similar across all 4 maps; therefore, only the results from one map (Complex2) are shown in this chapter. Table 4.1 contains the average result of each configuration, showing the average runtime and lengths of the routes found. Figure 4.2 offers a visualization that focuses on the different combinations of SCP metrics and insertion rules. These tests have shown that the **unicost** metric is significantly worse then the other two, while not offering a compensation in terms of runtime, which is very similar across all 9 configurations. The other two guard metrics, **base** and **quadratic** provide higher quality solutions and are very comparable in terms of both objective value and runtime, with **quadratic** having a slight edge. The insertion rules seems to have little to no effect on the quality of the solution. Due to these results, further experiments will be conducted using the **quadratic** SCP metric with varying insertion rules.

4.2.2 Influence of continuous segmentation on performance

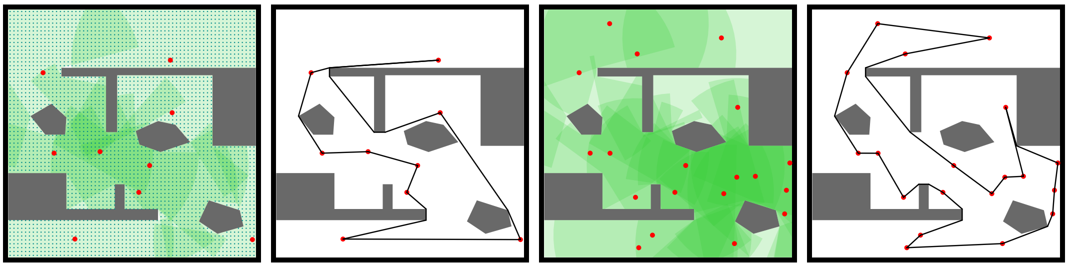
Unlike discrete environment segmentation, which offers complete control over problem size in the form of a definitive upper bound on the number of distinct areas to be covered, the magnitude of continuous segmentation is a direct result of the set of guard locations. Furthermore, the visibility radius chosen in any specific instance has a significant impact on the number of disjoint overlapping areas.

In all of the experiments conducted, choosing to solve the continuous variant led to a significant increase in the problem size — and since the visibility mapping is a key component of both the decoupled and the GTSP formulation, both of the approaches were impacted. Adding to that, when formulating the GTSP instance, the problem size increases once more, due to the duplication process of vertices which belong to more than one set (see Chapter 2.3.2). This significant increase in problem size leads to a notably worse performance when generating a solution using the decoupled approach. Table 4.2 shows the results of a comparative analysis of the discrete and continuous methods on the four maps with 3 different visibility radii. Figures 4.3, 4.4, 4.5 and 4.6 provide a visual comparison of results obtained using both methods.



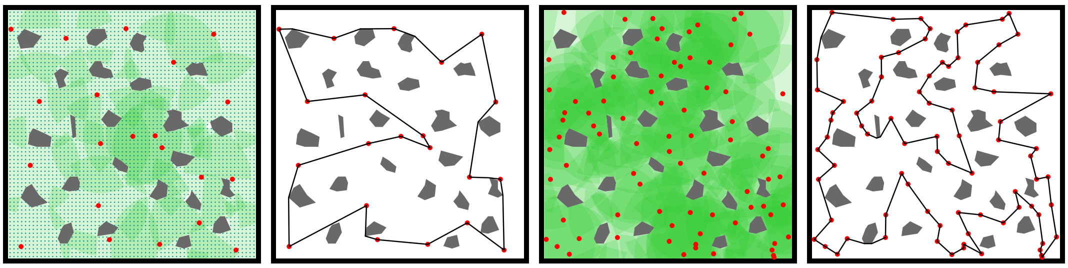
(a) (Discrete) Coverage **(b)** (Discrete) Route = 51.8 m **(c)** (Continuous) Coverage **(d)** (Continuous) Route = 52.5 m

Figure 4.3: Results of the decoupled approach: example solution found using discrete **(a, b)** and continuous **(c, d)** representation. Map: Gaps, $d = 8$.



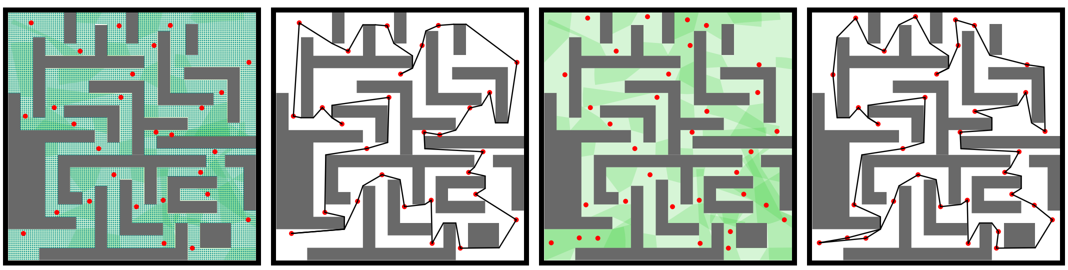
(a) (Discrete) Coverage
(b) (Discrete) Route = 85.0 m
(c) (Continuous) Coverage
(d) (Continuous) Route = 96.1 m

Figure 4.4: Results of the decoupled approach: example solution found using discrete **(a, b)** and continuous **(c, d)** representation. Map: Complex2, $d = 8$.



(a) (Discrete) Coverage
(b) (Discrete) Route = 96.3 m
(c) (Continuous) Coverage
(d) (Continuous) Route = 149.1 m

Figure 4.5: Results of the decoupled approach: example solution found using discrete **(a, b)** and continuous **(c, d)** representation. Map: Potholes, $d = 4$.



(a) (Discrete) Coverage
(b) (Discrete) Route = 127.0 m
(c) (Continuous) Coverage
(d) (Continuous) Route = 133.7 m

Figure 4.6: Results of the decoupled approach: example solution found using discrete **(a, b)** and continuous **(c, d)** representation. Map: Maze, $d = 6$.

Map: Complex2, Seed grid resolution: 64							
d	|G|	Metric	Insertion	N of areas	Tour len.	Runtime (ms)	Route (m)
4	371	base	maxmin	257	15.8	686.60	102.7692
			minmin	257	16.2	684.36	103.4583
			random	257	15.8	681.40	102.9351
		quadratic	maxmin	257	15.8	667.86	102.8719
			minmin	257	15.9	**663.88**	103.3291
			random	257	15.6	675.10	**102.2639**
		unicost	maxmin	257	16.7	681.60	110.5094
			minmin	257	17.1	680.62	111.1104
			random	257	16.7	681.92	109.2209
	1530	base	maxmin	3273	16.1	1623.22	101.1195
			minmin	3273	16.0	1597.66	100.9632
			random	3273	16.0	1607.40	100.8261
		quadratic	maxmin	3273	16.0	1613.20	100.7023
			minmin	3273	15.9	1557.40	**100.4258**
			random	3273	16.0	1597.12	100.9466
		unicost	maxmin	3273	17.0	1565.30	111.0076
			minmin	3273	16.8	**1554.06**	107.9125
			random	3273	16.7	1557.56	109.6880
20	371	base	maxmin	2191	5.1	762.30	48.7875
			minmin	2191	5.1	757.80	48.7939
			random	2191	5.1	754.56	48.4743
		quadratic	maxmin	2191	5.1	747.48	48.0090
			minmin	2191	5.0	757.40	**47.9693**
			random	2191	5.2	766.60	48.5478
		unicost	maxmin	2191	5.5	749.70	61.5609
			minmin	2191	5.7	754.90	61.7898
			random	2191	5.5	**741.18**	59.6016
	1530	base	maxmin	2439	5.3	1739.88	47.8932
			minmin	2439	5.2	1726.60	48.4936
			random	2439	5.3	1731.00	48.0797
		quadratic	maxmin	2439	5.3	1755.72	**47.7544**
			minmin	2439	5.1	**1721.58**	48.9427
			random	2439	5.3	1723.10	48.5281
		unicost	maxmin	2439	5.5	1715.30	57.9104
			minmin	2439	5.6	**1721.58**	65.2869
			random	2439	5.5	1722.62	59.1121

Table 4.1: Average tour length, runtime in milliseconds and route length of the solution obtained using the decoupled approach. Each row shows average result from 50 trial runs.

Map	Seg. Type	|G|	d	Areas	Tour len.	Runtime (ms)	Route (m)
Complex2	continuous	99	4	1528	51.8	633.40	134.8029
	discrete	99	4	790	38.2	634.40	123.2538
	continuous	99	8	3687	36.2	690.90	111.4411
	discrete	99	8	1119	21.1	637.60	94.2552
	continuous	99	12	6321	39.4	726.60	111.8430
	discrete	99	12	1406	16.3	646.70	80.9070
Gaps	continuous	101	4	1839	38.4	646.60	110.0638
	discrete	101	4	1014	36.6	651.50	106.8854
	continuous	101	8	3221	22.4	668.50	77.1829
	discrete	101	8	1033	17.9	630.10	69.3639
	continuous	101	12	4715	20.7	680.90	72.0364
	discrete	101	12	914	13.7	644.00	64.7700
Maze	continuous	155	4	1172	64.9	675.10	165.3090
	discrete	155	4	616	59.2	648.20	156.8880
	continuous	155	8	1659	60.1	691.20	156.1777
	discrete	155	8	608	48.6	657.80	149.1679
	continuous	155	12	1676	57.9	680.50	157.0555
	discrete	155	12	589	47.2	644.60	147.6373
Potholes	continuous	150	4	6117	106.7	1211.30	172.1007
	discrete	150	4	1921	46.0	670.00	120.3842
	continuous	150	8	36737	125.1	5795.10	187.2574
	discrete	150	8	3066	34.6	695.40	100.2785

Table 4.2: Results of the decoupled approach: average tour length, runtime and route length on all 4 maps with 3 different visibility radii — comparison of continuous (max sampling angle: 0.1 rad) and discrete (seed grid resolution: 64) environment representation. Each row shows average result from 10 trial runs.

4.3 GTSP approach

4.3.1 Instance size

The increase in instance size when formulating the one-in-a-set GTSP is noticable. In some cases, the expanded instance reaches such a high number of vertices that solving the instance becomes technicaly challenging (note that an instance with n vertices requires access to a $n \times n$ matrix of edge weights, and while this matrix can be reduced to half this size for symmetrical instances, it is often not enough).

The final number of vertices is a result of many factors: For the discrete case, seed grid resolution plays a role, as this parameter is directly responsible for the number of clusters in the graph. For both the discrete and continuous case, the obtained visibility coverage is dependent on the chosen visibility radius, and while usually higher values lead to the creation of more distinct areas, this is not always guaranteed. This happens when the visibility radius is so high that entire areas defined by obstacles can be seen by many guards (an extreme example of this is an instance where the environment is a convex polygon with no holes and $d = \infty$. In this case, no matter how many guards are inside this environment, there is only one distinct area — the entire domain, and its visible by all guards). However, higher visibility radius always leads to the need to create more duplicates of vertices in the orignal problem, because a guard with greater visibility radius will see more areas, therefore being a member of more clusters. For the stated reasons, the geometric character of the environment has a non-trivial effect on the final problem size.

Map	Seg. Type	$\lvert G \rvert$	d	Areas	N of vertices
Complex2	continuous	99	4	1528	14012
	discrete	99	4	790	7463
	continuous	99	8	3687	78416
	discrete	99	8	1119	23251
Gaps	continuous	101	4	1839	18960
	discrete	101	4	1014	10036
	continuous	101	8	3221	74002
	discrete	101	8	1033	25202
Maze	continuous	155	4	1172	8433
	discrete	155	4	616	4380
	continuous	155	8	1659	15462
	discrete	155	8	608	5617

Table 4.3: Instance size of the formualated one-in-a-set GTSP problem for different maps and visibility radii. Discrete representation was obtained using seed grid resolution = 64 and continuous using maximum sampling angle = 0.1 rad.

4.3.2 GLNS with initial route

Combining both approaches as proposed in Chapter 3.5 promises a better starting solution for the GLNS compared to random initial construction. This idea proves to be correct for larger instances (tens of thousands of vertices). While the initial solution gives GLNS a head start in larger instances, it does not guarantee faster time to convergence nor better final solution. Figures 4.7, 4.9, 4.11 show the comparison of evolution of the route length for a run using pure GLNS (with random initial solution) and GLNS with initial solution taken from the decoupled approach.

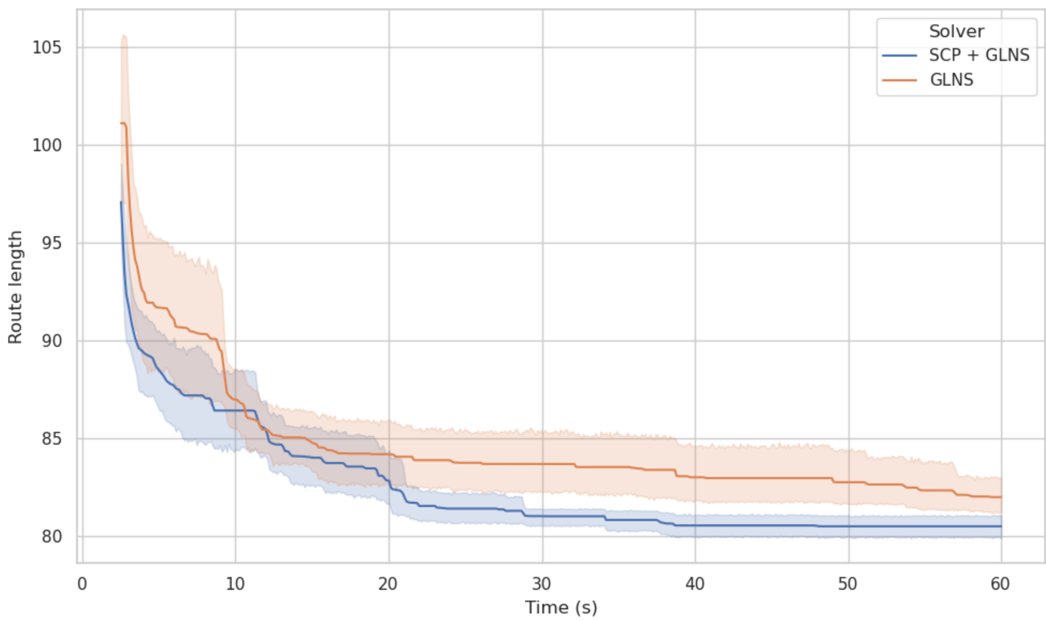
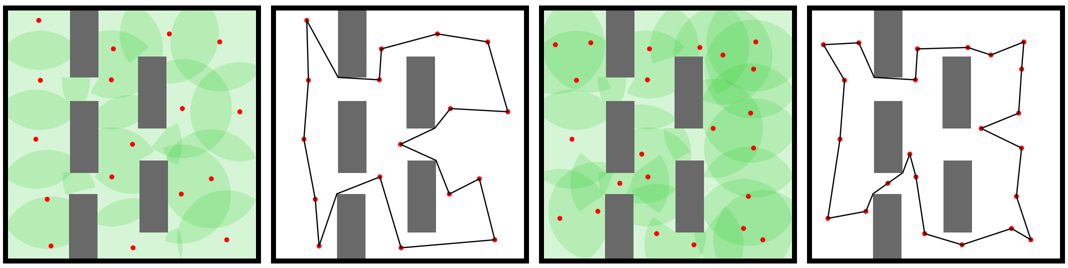


Figure 4.7: Comparison of quality of solution found by pure GLNS and SCP + GLNS over time, results of 10 runs with 95% confidence interval. Map: Gaps, $d = 4$, $|G| = 100$, continuous representation with max sampling angle = 0.1 rad.



(a) (SCP) Coverage

(b) (SCP) Route = 89.4 m

(c) (GTSP) Coverage

(d) (GTSP) Route = 80.4 m

Figure 4.8: Coverage + route obtained using SCP **(a, b)** and coverage + route obtained using GTSP with provided initial solution **(c, d)**. Map: Gaps, $d = 4$, $|G| = 100$, continuous representation with max sampling angle = 0.1 rad.

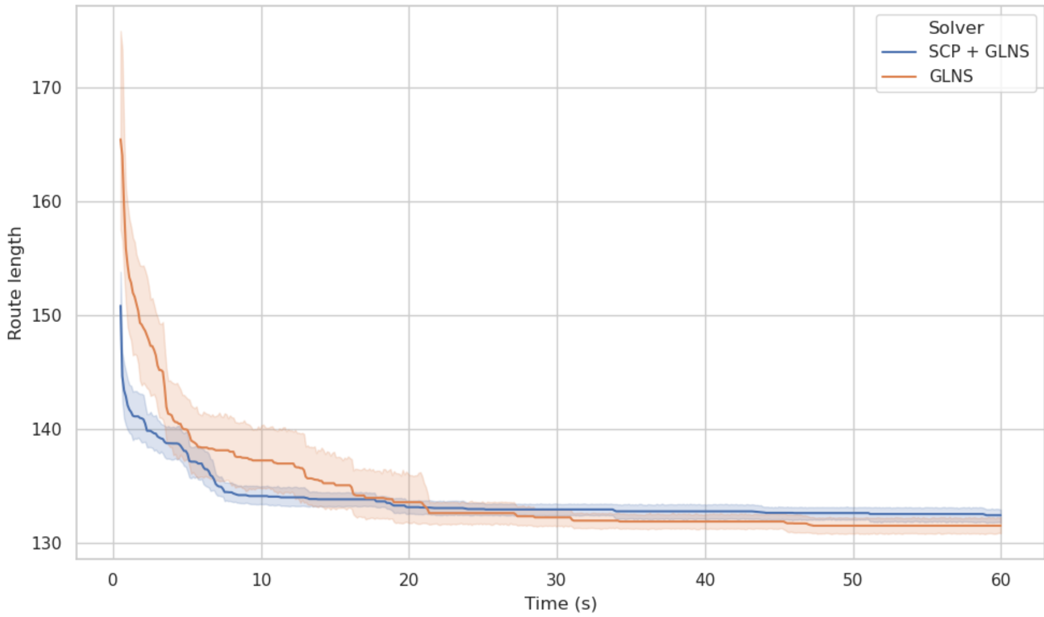
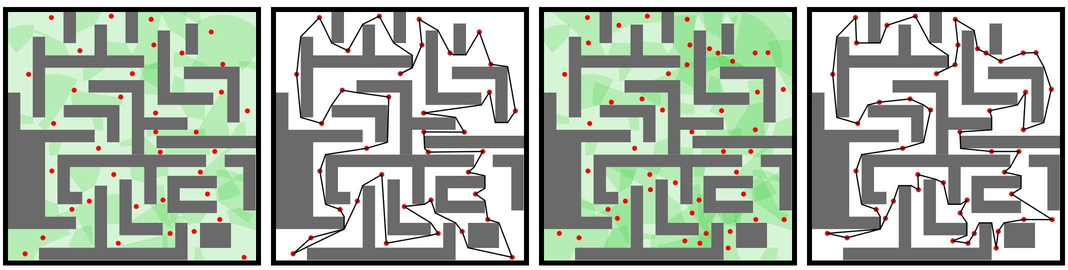


Figure 4.9: Comparison of quality of solution found by pure GLNS and SCP + GLNS over time, results of 10 runs with 95% confidence interval. Map: Maze, $d = 4$, $|\mathrm{G}| = 155$, continuous representation with max sampling angle = 0.1 rad.



(a) (SCP) Coverage

(b) (SCP) Route = 142.3 m

(c) (GTSP) Coverage

(d) (GTSP) Route = 131.4 m

Figure 4.10: Coverage + route obtained using SCP **(a, b)** and coverage + route obtained using GTSP with provided initial solution **(c, d)**. Map: Maze, $d = 4$, $|\mathrm{G}| = 155$, continuous representation with max sampling angle = 0.1 rad.

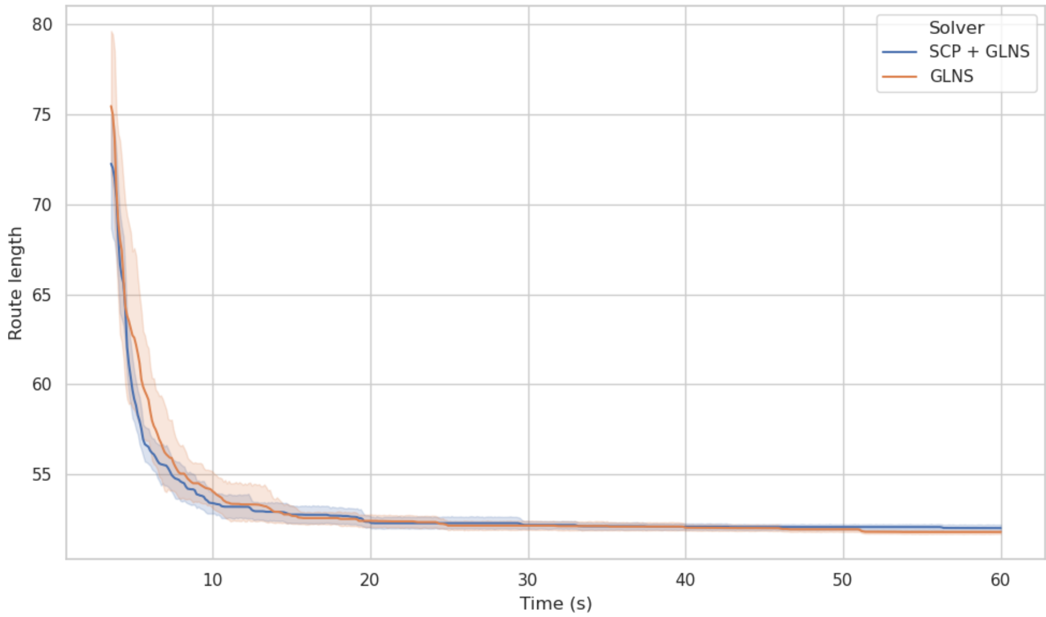
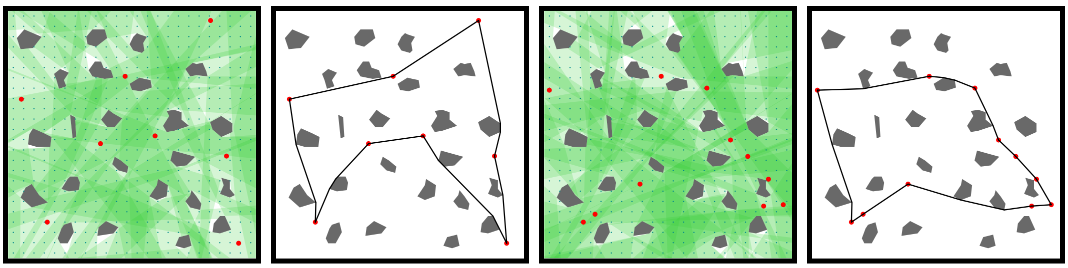


Figure 4.11: Comparison of quality of solution found by pure GLNS and SCP + GLNS over time, results of 10 runs with 95% confidence interval. Map: Potholes, $d = 40$, $|G| = 150$, discrete representation with seed grid resolution = 24.



(a) (SCP) Coverage **(b)** (SCP) Route = 68.7 m **(c)** (GTSP) Coverage **(d)** (GTSP) Route = 52.9 m

Figure 4.12: Coverage + route obtained using SCP **(a, b)** and coverage + route obtained using GTSP with provided initial solution **(c, d)**. Map: Potholes, $d = 40$, $|G| = 150$, discrete representation with seed grid resolution = 24.

4.4 Key insights

Conducted experiments proved correctness of the proposed and implemented algorithm. When using the approximate discrete representation, the algorithm offers a flexible framework for generating watchman routes which handles large instances fast, but might provide solutions which do not cover the environment fully. The quality of the solutions found is not optimal and rarely comes close to being such, as when used in combination with a further optimization technique (GLNS), better solutions are found.

Discrete environment representation was used mainly for algorithm parameter optimization. Custom weight functions for setting costs of potential guards were shown to have a huge effect on the quality of the solution. The chosen metric values guards which are near centers of larger areas. Insertion rules for the initial solution construction did not seem to affect solution quality greatly. It is suspected that the local search procedure homogenizes solutions constructed using different insertion rules.

Continuous environment representation leads to a notable increase in problem size, which sometimes has a negative effect on the quality of found solutions. This might be mitigated by further optimization. Solutions found using the decoupled approach were used as an initial solution for the formulated GTSP problem, which was solved using the GLNS heuristic. For larger instances (tens of thousands of vertices), this proved to increase the performance of GLNS, as the solver began with a higher quality initial solution compared to a random one it would normally produce at the beginning of the run.

5 Discussion

5.1 Conclusion

In this thesis, the Watchman Route Problem with limited visibility was introduced and formulated as a combination of the Set Cover Problem (SCP) and Generalized Traveling Salesman Problem (GTSP). Using this decoupled formulation, a heuristic-based approach for solving the WRP was presented.

The SCP formulation was realized by deploying a prior heuristic to generate a set of potential guard locations which guarantee full coverage and designing two methods for creating a representation of the environment. While one of such methods provides a discrete, approximate environment representation and allows for fully customizable instance size, the other offers a complete, continuous segmentation of the environment into distinct areas of visibility created by potential guard locations. A GRASP-based metaheuristic was designed and implemented to solve such formulated instance of the SCP, with the final solution — a route through the environment — found using a third-party TSP solver.

Furthermore, the problem was reformulated as one-in-a-set Generalized Traveling Salesman Problem for the purposes of further optimization using GLNS, a state-of-the-art large neighborhood search technique. The proposed and implemented method was experimentally validated and key outcomes of its use were presented.

5.2 Future outlook

A major setback of the proposed framework lies within the final problem size when using the complete, continuous domain representation. An abundance of potential guarding locations placed in a non-trivial environment with multiple obstacles results in a large number of distinct segments with unique visibility coverage. This has two main problematic consequences: first, the SCP solver is less effective in providing a good quality solution, and second, the one-in-a-set GTSP instance formulated in such cases becomes so large, it imposes serious technical challenges for practical applications.

The first problem might be solved (or at least partially mitigated) by employing a more sophisticated heuristic for solving the SCP. A possible step could be to implement more advanced local search operators during the GRASP procedure.

A broader solution would be to design and use a method to systematically identify and eliminate segments, which could be ignored without losing the full environment representation (some segments might always end up covered if, say, other segments next to them are). With some further augmentation of the algorithm used to obtain this representation, it would also be possible to compute an area of each found segment and remove those which are smaller than a specified threshold (or removing segments until a specified total threshold is reached).

Furthermore, the performance of the algorithm used for obtaining the continuous representation could be improved by using a dynamic programming technique during the overlap detection step, as it would remove the need to traverse the entire sweep-line at each iteration, and instead reuse already computed results, for the price of greater space complexity.

References

[1] G. Dantzig, R. Fulkerson, and S. Johnson. “Solution of a Large-Scale Traveling-Salesman Problem”. In: *Journal of the Operations Research Society of America* 2.4 (1954), pp. 393–410.

[2] G. B. Dantzig and J. H. Ramser. “The Truck Dispatching Problem”. In: *Management Science* 6.1 (1959), pp. 80–91.

[3] Thi Thoa Mac et al. “Heuristic approaches in robot path planning: A survey”. In: *Robotics and Autonomous Systems* 86 (2016), pp. 13–28.

[4] Thibaut Vidal et al. “A Hybrid Genetic Algorithm for Multidepot and Periodic Vehicle Routing Problems”. In: *Operations Research* 60 (June 2012), pp. 611–624.

[5] Wan Othman et al. “Solving Vehicle Routing Problem using Ant Colony Optimisation (ACO) Algorithm”. In: *International Journal of Research and Engineering* 5 (Nov. 2018), pp. 500–507.

[6] Hongguang Wu and Yuelin Gao. “An ant colony optimization based on local search for the vehicle routing problem with simultaneous pickup–delivery and time window”. In: *Applied Soft Computing* 139 (2023), p. 110203.

[7] Tetsuo Asano, S.K. Ghosh, and Thomas Shermer. “Visibility in the plane”. In: *Handbook of Computational Geometry* (Jan. 2000), pp. 829–876.

[8] Joseph Mitchell. “Geometric Shortest Paths and Network Optimization”. In: *Handbook of Computational Geometry* (Jan. 2000), pp. 635–667.

[9] Simeon Ntafos. “Watchman routes under limited visibility”. In: *Comput. Geom. Theory Appl.* 1.3 (Mar. 1992), pp. 149–170.

[10] Jan Mikula and Miroslav Kulich. “Towards a Continuous Solution of the d-Visibility Watchman Route Problem in a Polygon With Holes”. In: *IEEE Robotics and Automation Letters* 7.3 (2022), pp. 5934–5941.

[11] Wei-Pang Chin and Simeon Ntafos. “Shortest watchman routes in simple polygons”. In: *Discrete Comput. Geom.* 6.1 (Dec. 1991), pp. 9–31.

[12] Xuehou Tan and Tomio Hirata. “Constructing shortest watchman routes by divide-and-conquer”. In: *Algorithms and Computation*. Ed. by K. W. Ng et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 68–77.

[13] Adrian Dumitrescu and Csaba D. Tóth. “Watchman tours for polygons with holes”. In: *Computational Geometry* 45.7 (2012), pp. 326–333.

[14] Joseph S. B. Mitchell. “Approximating watchman routes”. In: *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA ’13. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2013, pp. 844–855.

[15] Eli Packer. “Computing Multiple Watchman Routes”. In: *Experimental Algorithms*. Ed. by Catherine C. McGeoch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 114–128.

[16] Jan Faigl and Libor Přeučil. “Inspection planning in the polygonal domain by Self-Organizing Map”. In: *Applied Soft Computing* 11.8 (2011), pp. 5028–5041.

[17] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. First Edition. W. H. Freeman, 1979.

[18] Kamesh Munagala et al. “The Pipelined Set Cover Problem”. In: *Database Theory - ICDT 2005*. Ed. by Thomas Eiter and Leonid Libkin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 83–98.

[19] Joaquín Bautista and Jordi Pereira. “Modeling the problem of locating collection areas for urban waste management. An application to the metropolitan area of Barcelona”. In: *Omega* 34.6 (2006). Reverse production systems, pp. 617–629.

[20] Vijay V. Vazirani. *Approximation algorithms*. Berlin, Heidelberg: Springer-Verlag, 2001.

[21] Thomas Feo. “Greedy Randomized Adaptive Search Procedures”. In: *Journal of Global Optimization* 6 (Mar. 1995), pp. 109–133.

[22] Joaquín Bautista and Jordi Pereira. “A GRASP algorithm to solve the unicost set covering problem”. In: *Computers & Operations Research* 34.10 (2007), pp. 3162–3173.

[23] Guanghui Lan, Gail W. DePuy, and Gary E. Whitehouse. “An effective and simple heuristic for the set covering problem”. In: *European Journal of Operational Research* 176.3 (2007), pp. 1387–1403.

[24] Victor Reyes and Ignacio Araya. “A GRASP-based scheme for the set covering problem”. In: *Operational Research* 21.4 (Dec. 2021), pp. 2391–2408.

[25] Naveen Garg, Goran Konjevod, and R. Ravi. “A Polylogarithmic Approximation Algorithm for the Group Steiner Tree Problem”. In: *Journal of Algorithms* 37.1 (2000), pp. 66–84.

[26] Jeanette Schmidt and Stefan Irnich. “New neighborhoods and an iterated local search algorithm for the generalized traveling salesman problem”. In: *EURO Journal on Computational Optimization* 10 (2022), p. 100029.

[27] Stephen L. Smith and Frank Imeson. “GLNS: An effective large neighborhood search heuristic for the Generalized Traveling Salesman Problem”. In: *Computers & Operations Research* 87 (2017), pp. 1–19.

[28] David Pisinger and Stefan Ropke. “A general heuristic for vehicle routing problems”. In: *Computers & Operations Research* 34.8 (2007), pp. 2403–2435.

[29] Daniel J. Rosenkrantz, Richard E. Stearns, and Philip M. Lewis II. “An Analysis of Several Heuristics for the Traveling Salesman Problem”. In: *SIAM Journal on Computing* 6.3 (1977), pp. 563–581.

[30] Yao-Nan Lien, Eva Ma, and Benjamin W.-S. Wah. “Transformation of the generalized traveling-salesman problem into the standard traveling-salesman problem”. In: *Information Sciences* 74.1 (1993), pp. 177–189.

[31] Jan Mikula and Miroslav Kulich. *Omnidirectional Sensor Placement: A Large-Scale Computational Study and Novel Hybrid Accelerated-Refinement Heuristics*. 2025. arXiv: `2410.08784 [cs.RO]`. URL: `https://arxiv.org/abs/2410.08784`.

[32] Bentley and Ottmann. “Algorithms for Reporting and Counting Geometric Intersections”. In: *IEEE Transactions on Computers* C-28.9 (1979), pp. 643–647.

[33] Mark de Berg et al. *Computational geometry: algorithms and applications*. Berlin, Heidelberg: Springer-Verlag, 1997.

[34] Jan Mikula, Miroslav Kulich, and Libor 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)*. IEEE, Oct. 2024, pp. 10503–10510.

[35] The GMP development team. *GMP: The GNU Multiple Precision Arithmetic Library*. GNU Project. 2024. URL: `https://gmplib.org/`.

[36] Keld Helsgaun. *An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems: Technical report*. Roskilde Universitet, Dec. 2017.

[37] Jan Mikula. *GLNS for GTSP*. 2025. URL: `https://gitlab.ciirc.cvut.cz/mission-planning/glns-for-gtsp`.