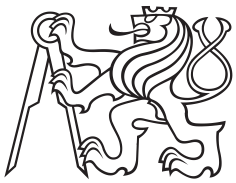
Master Thesis



Czech
Technical
University
in Prague

A coupled approach to Watchman Route Problem

Štefan Trusina

Supervisor: RNDr. Kulich Miroslav, Ph.D.
May 2022

Acknowledgements

I would like to thank my supervisor RNDr. Miroslav Kulich, Ph.D. for his guidance, patience and good advice. I also thank Ing. Jan Mikula for his help and useful insights, as well as for providing datasets for this thesis.

Declaration

I hereby declare that I have completed this thesis on my own and that all the used sources are included in the list of references, in accordance with the Methodological instructions on ethical principles in the preparation of university theses.

Prague, May 20, 2022

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 20. května 2022

Abstract

The Watchman Route Problem (WRP) is the problem of finding a closed minimal-length path which, if followed by a robot, enables it to see the whole environment. A decoupled approach to WRP consists of first partitioning the environment into such areas that, if all are visited by the robot, it sees the whole environment, and of finding a minimal-length closed path that visits all areas. This thesis deals with the second part of this decoupled approach. The problem is called the Travelling Salesman Problem with Neighborhoods (TSPN). The neighborhoods in this thesis are polygoncircles. A polygoncircle is a geometrical area constructed by clipping half-planes from some circle. The GLNS heuristic algorithm, originally developed to solve the Generalized Travelling Salesman Problem (GTSP), was modified for polygoncircles. For this purpose, the Touring Polygon Problem was also modified and a new algorithm called Point – Polygoncircle – Point was developed. For some two points and a polygoncircle, this algorithm finds a point on the polygoncircle that is closest to the two points. The GLNS algorithm was further modified to work in an environment with obstacles. The algorithm was tested on several maps and for several visibility ranges of the robot. The generated paths were compared with paths generated by a similar algorithm developed by J. Mikula and M. Kulich. Our implementation mostly produced comparable or slightly worse results.

Keywords:

TSPN, WRP, TPCP, GLNS,
Polygoncircle, Polygonal domain

Supervisor:

RNDr. Kulich Miroslav, Ph.D.

Abstrakt

Problém hledání cesty hlídače (WRP) je problém hledání uzavřené cesty minimální délky, kterou když robot projede, uvidí celé prostředí. Sdružený přístup k WRP spočívá zaprvé v rozdělení prostředí na takové plochy, které když robot všechny projede, uvidí celé prostředí, zadruhé v nalezení cesty minimálí délky, která prochází přes všechny plochy. Tato práce se věnuje druhé části sdruženého přístupu k WRP. Tento problém se nazývá Problém obchodního cestujícího se sousedstvími. V rámci této práce jsou sousedství tvořena polygonkruhy, tedy geometrickými plochami, které vznikly ořezáním kruhu polorovinami. Heuristický algoritmus GLNS, který byl původně vyvinutý pro řešení Problému obecného obchodního cestujícího, byl upraven pro polygonkruhy. Pro polygonkruhy byl též upraven Problém procházení polygonů (Touring Polygon Problem) a vyvinut nový algoritmus nazvaný Bod – polygonkruh – bod. Pro dva libovolné body a polygonkruh najde tento algoritmus takový bod na polygonkruhu, který je těmto dvěma nejbližší. GLNS algoritmus byl dále upraven tak, aby fungoval i v prostředích s překážkami. Algoritmus byl testován na několika mapách a na několika poloměrech viditelnosti. Nalezené cesty byly následně porovnány s cestami nalezenými podobným algoritmem vyvinutým J. Mikulou a M. Kulichem. Výsledky našeho algoritmu vycházely srovnatelné nebo o něco horší.

Klíčová slova:

TSPN, WRP, TPCP, GLNS,
Polygonkruh, Polygonální doména

Překlad názvu:

Sdružený přístup k problému hlídače

Contents

Figures

Tables

Algorithms

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Trusina** Jméno: **Štefan** Osobní číslo: **487622**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Umělá inteligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Sdružený přístup k problému hlídače

Název diplomové práce anglicky:

A coupled approach to Watchman Route Problem

Pokyny pro vypracování:

1. Get acquainted with the Close Enough Travelling Salesman Problem (CETSP) and state-of-the-art metaheuristics for routing problems.
2. Design and realize a solver for finding the shortest path between two points and a generalized polygon, i.e. a polygon with some edges substituted by circular arcs (point-generalized polygon-point, PGP).
3. Modify the method described in [1] for the CETSP based on a combination of a point-circle-point solver and the GLNS solver [2] to work with generalized polygons.
4. Given a polygon with holes, design and implement an algorithm that generates a set of generalized polygons with a specified maximal size, which completely covers the polygon.
5. Combine the implemented methods to solve the Watchman Route Problem.
6. Evaluate experimentally properties of the extended algorithm. Describe and discuss the obtained results.

Seznam doporučené literatury:

[1] Lukáš Fanta: The Close Enough Travelling Salesman Problem in the polygonal domain, diplomová práce, FEL, ČVUT v Praze, 2021, https://dspace.cvut.cz/handle/10467/96747
[2] Stephen L. Smith, Frank Imeson,GLNS: An effective large neighborhood search heuristic for the Generalized Traveling Salesman Problem, Computers & Operations Research, Volume 87, Pages 1-19, ISSN 0305-0548, doi:10.1016/j.cor.2017.05.010, 2017.
[3] Gendreau M., Potvin JY. (eds) Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol 272. Springer, Cham, 2019

Jméno a pracoviště vedoucí(ho) diplomové práce:

RNDr. Kulich Miroslav, Ph.D. CIIC ČVUT v Praze

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **28.01.2022** Termín odevzdání diplomové práce: **20.05.2022**

Platnost zadání diplomové práce: **30.09.2023**

RNDr. Kulich Miroslav, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

 © ČVUT v Praze, Design: ČVUT v Praze, VIC

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Chapter 1

Introduction

This thesis introduces several heuristic algorithms that can be used for solving the watchman route problem (WRP). Assume a mobile robot that sees with its sensors a certain portion of the environment around itself, where this environment is of a finite size and known in advance. The WRP is the problem of finding a closed minimal-length path which, if followed by the robot, enables the robot to see the whole environment.

Several variants of this problem exist. In this version of the WRP, the robot is represented by a point, the portion of the environment that it sees in an open space without obstacles is a circle of radius d (visibility radius) with the center coinciding with the position of the robot, and the environment is represented in 2D as a polygon with polygonal holes. Such environment is called a polygonal domain. A polygonal domain can be visualized as a map with polygonal border and polygonal obstacles (Fig. 1.1).

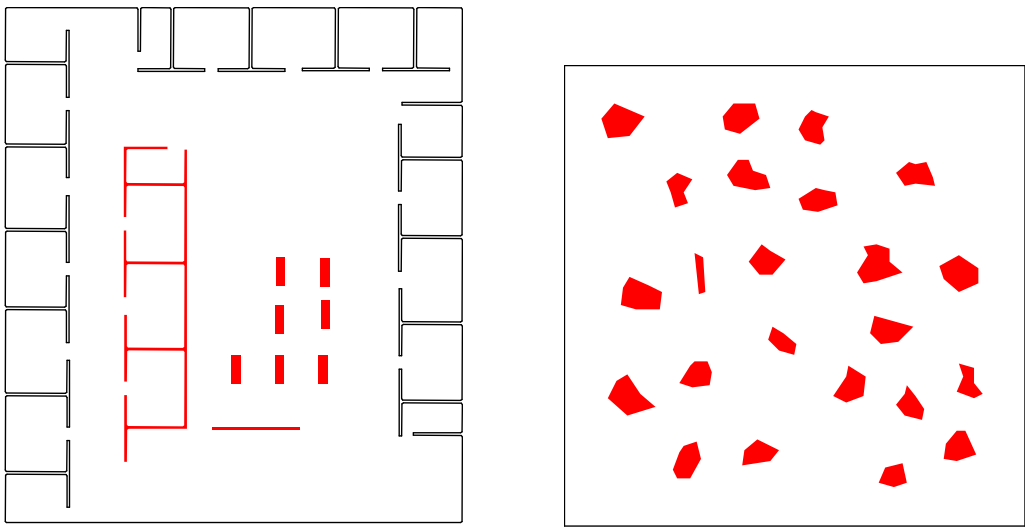


Figure 1.1: Examples of maps with polygonal border and polygonal obstacles.

The WRP models many real-life problems. Consider the case of a museum night security guard. He knows the environment of the museum and must do periodic

rounds around the whole building. It is desirable to perform these rounds in an effective manner, that is, the rounds should cover the whole building but they should be as short as possible.

In this thesis, the WRP is not solved as a whole. Instead, it is divided into two subproblems. This thesis describes an algorithm that solves the second of these subproblems.

In the next chapter, the WRP will be formally defined. It will be shown how the WRP can be divided into two subproblems. The second of these subproblems is the so-called Travelling Salesman Problem with Neighborhoods (TSPN).

The third chapter describes several heuristic algorithms that together can solve the TSPN. The algorithms are described in their basic form, that is, for environments without obstacles. At the end of the chapter, a modification for environments with obstacles is described.

In the fourth chapter, the algorithm is evaluated. Its performance is compared with a similar algorithm known to produce very good results. Pictures showing paths found on several maps are also presented.

Chapter 2

Problem Specification

The Watchman Route Problem is the problem of finding a closed path such, that by following it, the robot sees the whole environment. The shape of a closed path is a closed polyline, which can be represented by a sequence of vertices $\{P_1, P_2, \ldots, P_n\}$.

A formal definition of the problem follows: A polygonal domain $\mathcal{W}$ is given. The task is to find the shortest closed path π represented by a sequence of points $\mathcal{S}_{path} = \{P_1, P_2, \ldots, P_n\}$ such that it lies inside $\mathcal{W}$ and for all points $Q \in \mathcal{W}$, there exists a point $P_x \in \pi$ such that the line-segment QP_x lies inside $\mathcal{W}$ and its length is smaller or equal to d. The existence of point P_y for every point P_x means that all points in $\mathcal{W}$ will be seen from at least one point on route π.

The algorithms described in this text assume a decoupled approach to the WRP. The decoupled approach splits the WRP into two subproblems:

- The algorithm finds a set of areas whose union gives $\mathcal{W}$. Each one of these areas must satisfy the following requirement: The robot must see the whole area from any location on that area.
- The Travelling Salesman Problem with Neighborhoods (TSPN) is solved on the set of the found areas. A solution of the TSPN is a closed path represented by a sequence of points $\mathcal{S}_{path} = \{P_1, P_2, \ldots, P_n\}$ such that for each area q, at least point from the sequence $\mathcal{S}_{path}$ lies on q.

The requirement that the whole area is visible to the robot from all its points implies that the area must be convex and the distance between any two points must be smaller than visibility radius d.

The areas used in the algorithms described in this thesis we call polygoncircles. A polygoncircle is a geometrical shape that can be constructed by clipping half-planes from an original circle. More formally, a polygoncircle is a set of points $P_{pc} = [x_{pc}, y_{pc}]$ defined by a set of inequalities:

- The inequality $(x_{pc} - x_{center})^2 + (y_{pc} - y_{center})^2 \leq r^2$ represents the original circle with center in point $P_{center} = [x_{center}, y_{center}]$ and with radius r.

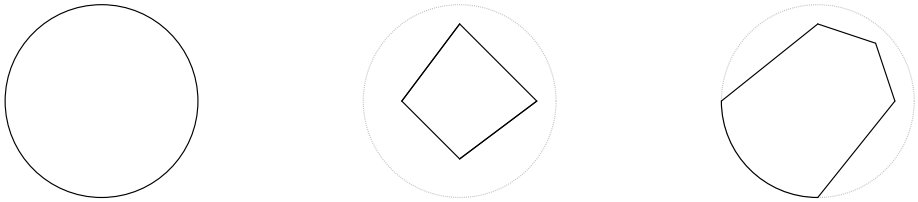
- The inequalities $ax_{pc} + by_{pc} \leq c$ represent the half-planes clipped from the original circle.

From the definition, it is clear that polygoncircles are convex.

The distance between any two points in any polygoncircle on $\mathcal{W}$ must be smaller than the visibility radius d. That means that in the context of this thesis, $r = \frac{1}{2}d$.

Thus, polygoncircle is a geometrical shape that satisfies the requirements when using a decoupled approach to solve WRP.

If no halfplane is clipped from the original circle, the polygoncircle is a circle (Fig. 2.1a). If a polygoncircle was clipped in such a way that its border has no circular parts, it is a polygon (Fig. 2.1b). If a polygoncircle is neither a circle, nor a polygon, it is a proper polygoncircle (Fig. 2.1c). The border of each proper polygoncircle has both circular and linear parts (Fig. 2.2).



(a) : Circle. **(b) :** Polygon. **(c) :** Proper polygoncircle.

Figure 2.1: Polygoncircle types.

A circular part of the border of a polygoncircle we call c-section. Each part of the border of the original circle of a polygoncircle that is not c-section we call p-section.

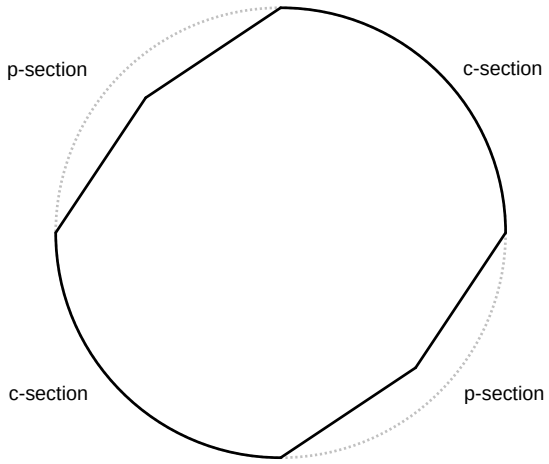


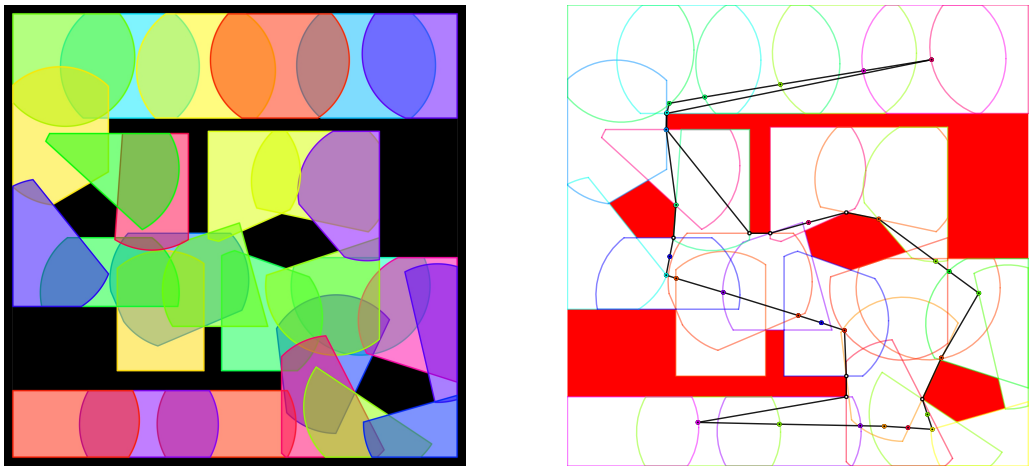
Figure 2.2: c-sections and p-sections.

If a set of polygoncircles with $r = \frac{1}{2}d$, which cover $\mathcal{W}$ is found, every closed path π that has at least one common point with every polygoncircle from that set enables the robot to see the whole polygonal domain $\mathcal{W}$.

The problem of finding such closed path π that traverses through every polygoncircle, while minimizing its length, is called the Travelling Salesman Problem with Neighborhoods (TSPN).

The TSPN is usually presented as solving instances consisting of circles in an environment without obstacles. However, the redefinition of the TSPN problem to solving instances consisting of polygoncircles in an environment with obstacles is straightforward.

There exist various approaches to solving TSPN. The approach described in this text uses a modified version of the GLNS heuristic [2], which was originally developed for solving the generalized travelling salesman problem (GTSP).



(a) : Finding a set of covering areas. **(b) :** Solving TSPN.

Figure 2.3: Illustration of the decoupled approach to WRP using polygoncircles.

Chapter 3

Solution Description

This chapter describes the algorithms that are used to solve the TSPN on a set of polygoncircles. Given a polygonal domain $\mathcal{W}$ and a set of polygoncircles that cover $\mathcal{W}$, finding a good-quality solution to TSPN means also finding a good-quality solution to WRP. The algorithms are first described in their basic form for maps without obstacles.

First, the Point – Polygoncircle – Point (PPCP) algorithm is described. It is used to find point P_{opt} on a polygoncircle p such that the sum of distances from P_{opt} to two other fixed points is minimal.

Second, the Touring – Polygoncircle – Problem (TPCP) algorithm is presented. The TPCP algorithm optimizes a closed path on a sequence of polygoncircles, while keeping the order of polygoncircles fixed. To optimize the path, it uses the PPCP algorithm.

Third, a description of the GLNS algorithm is given. This algorithm uses PPCP and TPCP algorithms to find an optimal path through a set of polygoncircles while the order of polygoncircles is not fixed.

After describing the algorithms in their basic form, some changes are proposed to solve TSPN instances on maps with obstacles.

3.1 Point – Polygoncircle – Point (PPCP)

The PPCP problem can be formulated in the following way: Assume a polygoncircle p and points A and B. Find point P^* on p such that it minimizes the sum of distances between points A and P^* and between points B and P^*. More formally:

$$P^* = \arg\max_{P \in p} \{dist(P, A) + dist(P, B)\}$$

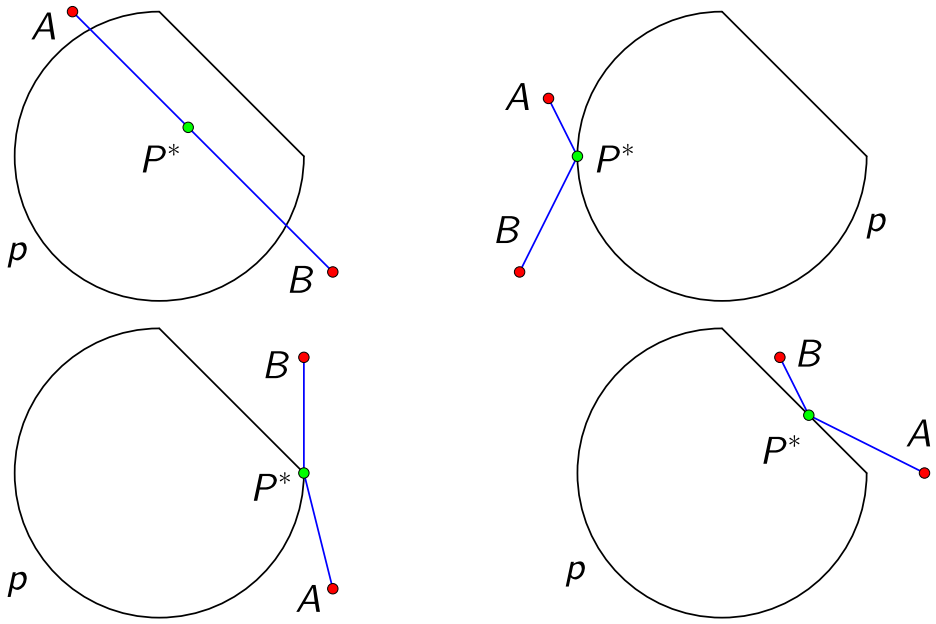


Figure 3.1: PPCP examples.

3.1.1 Geometrical Approach to PPCP

The set of points Q such that $dist(Q, A) + dist(Q, B) = k$, where k is constant, is an ellipse with focal points in A and B. Call this ellipse $\epsilon(k)$.

By setting $k = dist(AB)$, $\epsilon(k)$ degenerates to line-segment AB. By gradually increasing k, the circumference of $\epsilon(k)$ increases (Fig. 3.2).

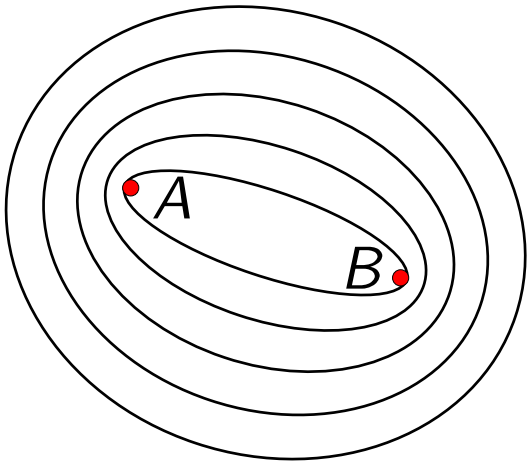


Figure 3.2: Ellipses with the same focal points but different sizes.

Assume a polygoncircle p such that line-segment AB does not intersect p. Call k^* the smalles value of k such that the intersection of $\epsilon(k)$ and p is non-empty. The intersection when $k = k^*$ contains just a single point P^*. This point is the

solution of the PPCP problem instance consisting of points A, B and polygoncircle p (Fig. 3.3).

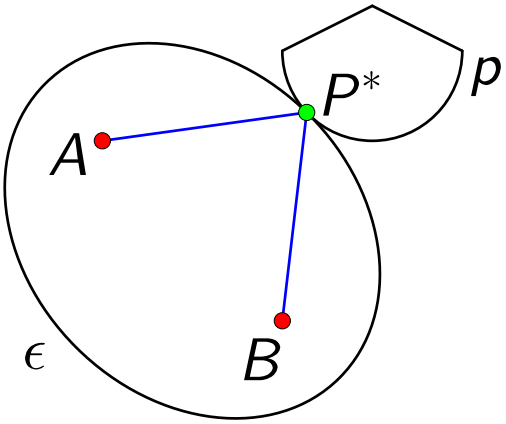


Figure 3.3: The smallest ellipse intersecting a polygoncircle.

The important property of an ellipse ϵ with focal points F_1 and F_2 is that for all points $Q \in \epsilon$, the normal at point Q bisects the angle $\measuredangle F_1 Q F_2$ (Fig. 3.4). That implies that a ray from one focal point is reflected by ellipse ϵ to the second focal point.

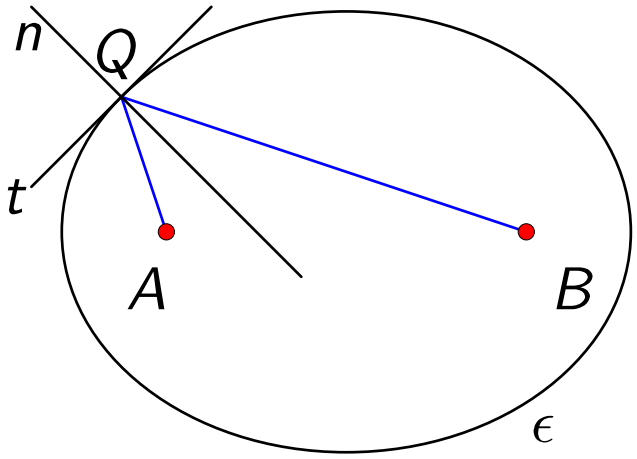


Figure 3.4: A ray from one focal point is reflected to the second focal point.

To build an algorithm which solves the PPCP problem, several subproblems of geometrical nature need to be solved. In the following sections these subproblems will be described.

3.1.2 Point – Circle – Point (PCP)

Assume a circle c and points A and B, where A and B are both outside c and line-segment AB does not intersect c. The PCP is the problem of finding point P^* on c such that it minimizes the sum of distances between points A and P^* and between points B and P^*. Formally in mathematical notation:

$$P^* = \arg\min_{P \in c} \{dist(P, A) + dist(P, B)\}$$

It can be seen that the PCP problem is an instance of the PPCP problem because any circle is an instance of polygoncircle.

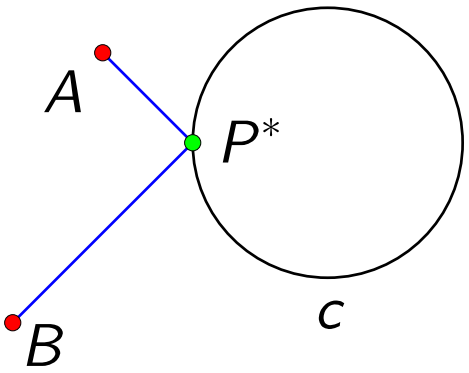


Figure 3.5: PCP is an instance of PPCP.

Call ϵ the smallest ellipse with focal points in A and B such that it has a non-empty intersection with c. This intersection contains just a single point P^*. The PCP problem is equivalent to the problem of finding P^*.

Point P^* is the point where circle c and ellipse ϵ touch. In this point the tangent of c is equal to the tangent of ϵ. The PCP problem can therefore be equivalently formulated in the following way: Find point P^* on c such that a ray from A is reflected in P^* to B (Fig. 3.3).

The solution of this problem is described in detail in L. Fanta's master thesis [3]. It is not trivial and requires solving a fourth-degree equation.

3.1.3 Point – Line-segment – Point

Assume a line-segment l with end-points C and D and points A and B, where A and B both lie on one side of line CD. The task is to find point $P^* \in l$ such that it minimizes the sum of distances between points A and P^* and between points B and P^*. In mathematical notation:

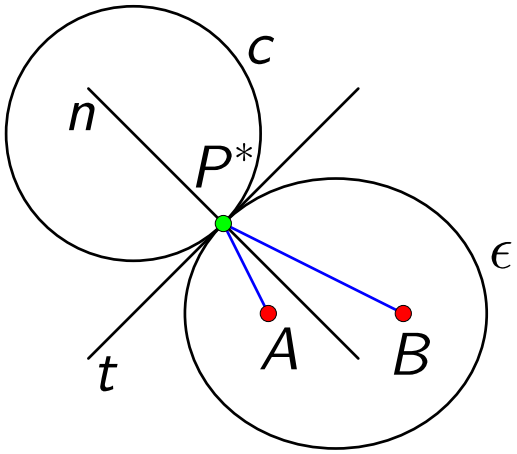


Figure 3.6: Ellipse and circle touch in optimal point.

$$P^* = \arg\max_{P \in l} \{dist(P^*, A) + dist(P^*, B)\}$$

Call ϵ the smallest ellipse with focal points in A and B such that it has a non-empty intersection with l. This intersection contains just a single point P^*. Either l is tangent to ϵ (Fig. 3.7a) or ϵ touches l in one of its endpoints (Fig. 3.7b). If l is tangent to ϵ, P^* is the point where a ray from A reflects to B.

Figure 3.7: Two cases of Point – Line-segment – Point.

Therefore the Point – Line-segment – Point problem can be equivalently formulated as: Find point $P^* \in l$ such that the angles $\measuredangle AP^*C$ and $\measuredangle BP^*D$ are equal and if no such point exists, choose P^* from points C and D.

First, the space is transformed in such a way that point C moves to the origin and line-segment l lies on the x-axis. This transformation consists of one translation and one rotation. Points A and B are also transformed. The transformed points will be called A', B', C', D' and their coordinates $[a'_x, a'_y]$, $[b'_x, b'_y]$, $[c'_x, c'_y]$, $[d'_x, d'_y]$. The transformed line-segment l will be called l'. The optimal point in the transformed space will be called P' and its coordinates $[p'_x, p'_y]$.

Due to the space transformation, the task is significantly simplified. Since the optimal point P' lies on x-axis, $p'_y = 0$ and P' can be found by computing p'_x (Fig. 3.8). Due to the properties of similar triangles, the following equation can be made:

$$\frac{a'_x - p'_x}{a'_y} = \frac{b'_x - p'_x}{b'_y}$$

The solution is:

$$p'_x = \frac{a'_x b'_y - a'_y b'_x}{a'_y - b'_y}$$

If $0 \leq p'_x \leq d'_x$ then the final value of p'_x was found. If $p'_x < 0$, p'_x is assigned value 0. If $d'_x < p'_x$, p'_x is assigned value d'_x. After d'_x is determined, P' is known. The inverse transformation applied to P' yields the optimal point P^*.

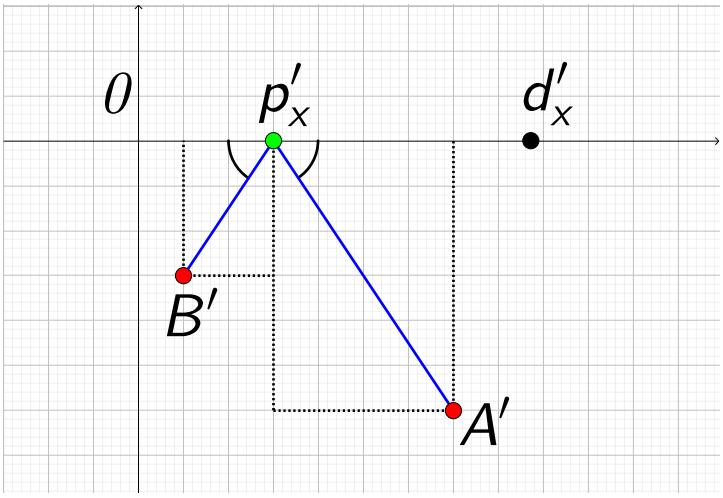


Figure 3.8: Point C moves to the origin and line-segment l lies on the x-axis.

3.1.4 Point – Polyline – Point

The Point – Polyline – Point problem is similar to the Point – Line-segment – Point problem but the optimal point P^* lies on a polyline (Fig. 3.9).

The solution can be found by sequentially solving the Point – Line-segment – Point problem on individual line-segments of the polyline. The optimal point P^* is chosen from the optimal points found on individual line-segments.

The Point – Polygon – Point problem where P^* lies on a polygon can be solved by the Point – Polyline – Point algorithm because a polygon can be treated as a closed polyline.

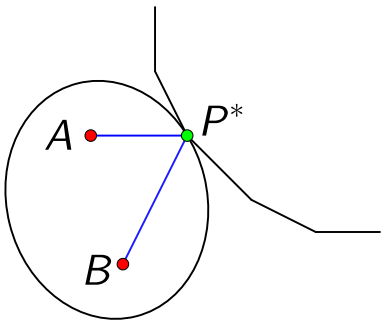


Figure 3.9: The Point – Polyline – Point problem is solved by sequentially solving the Point – Line-segment – Point problem.

3.1.5 Main Algorithm

The PPCP algorithm has a tree structure. The individual PPCP problem instances are first classified into categories and then finally solved in the leafs of the tree. Each individual PPCP problem instance consists of points A and B and a polygoncircle p. The polygoncircle p was created by clipping half-planes from an original circle c.

In the first branching of the algorithm tree, the instances are divided into four categories:

- A and B both lie inside c.
- A lies inside c and B lies outside c.
- A lies outside c and B lies inside c.
- A and B both lie outside c.

The other important criterion is whether the polygoncircle p is circle, polygon or proper polygoncircle.

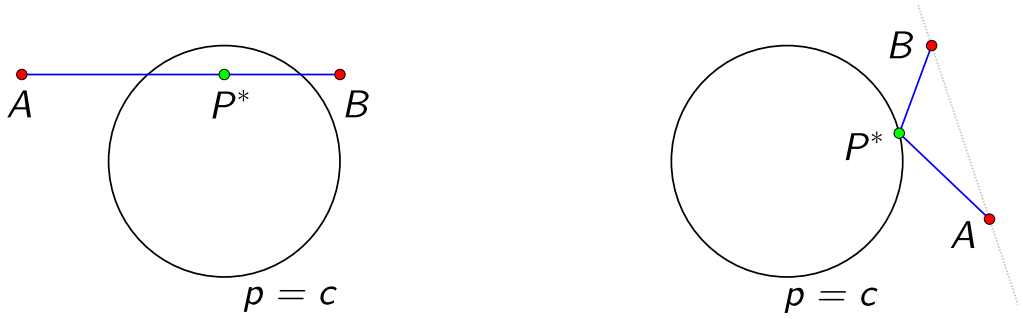
In the following sections, the working of the PPCP algorithm will be demonstrated on several examples. These examples should provide a good idea of how the algorithm works.

Example 1

Polygoncircle p is a circle and points A and B are both outside circle c.

If the intersection of line-section AB and circle c is non-empty, P^* can be any point in that intersection (Fig. 3.10a).

If the intersection of line-section AB and circle c is empty, the PCP algorithm is used to find P^* (Fig. 3.10b).



(a): (b):

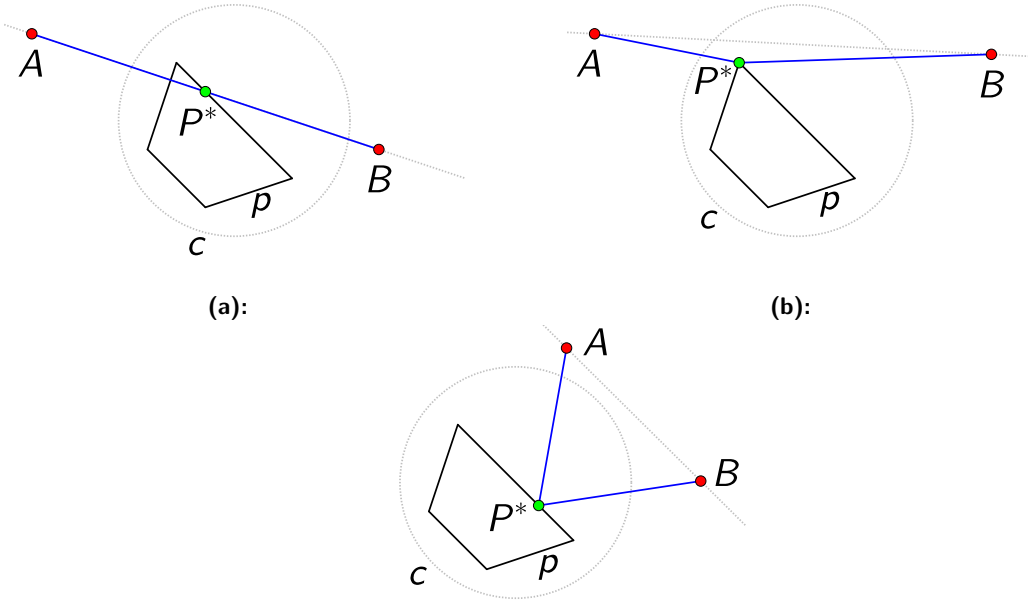
Figure 3.10: Example 1.

Example 2

Polygoncircle p is a polygon and points A and B are both outside circle c.

If the intersection of line AB and circle c is non-empty, the intersection of line-segment AB and polygon p is checked. If it is non-empty, P^* can be any point from that intersection (Fig. 3.11a). If it is empty, P^* is found using the Point – Polygon – Point algorithm (Fig. 3.11b).

If the intersection of line AB and circle c is empty, P^* is found using the Point – Polygon – Point algorithm (Fig. 3.11c).



(a): (b):

(c):

Figure 3.11: Example 2.

Example 3

Polygoncircle p is a polygon and points A and B are both inside circle c.

If the intersection of line AB and the border of polygon p is non-empty, the two points in the intersection are named C and D. If the intersection of line-segments AB and CD is non-empty, P^* can be any point from that intersection (Fig. 3.12a). If the intersection is empty, P^* is found using the Point – Polygon – Point algorithm (Fig. 3.12b).

If the intersection of line AB and the border of polygon p is empty, the Point – Polygon – Point algorithm is used to find P^* (Fig. 3.12c).

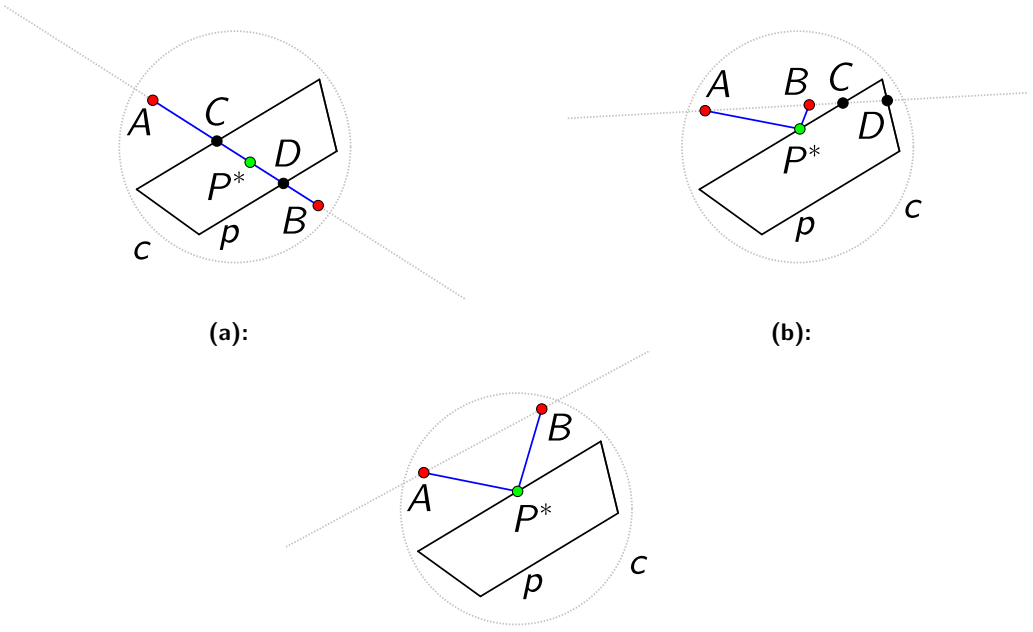


Figure 3.12: Example 3.

Example 4

Polygoncircle p is a proper polygoncircle, points A and B are both inside circle c. The intersection of line AB and circle c is a line-segment bounded by points C and D.

If points C and D both lie on c-section of c, P^* can be any point from line-segment AB (Fig. 3.13a).

If point C lies on c-section and point D lies on p-section of p, the intersection point of line AB and the polyline associated with point D is found and called E. If the intersection of line-segments AB and CE is non-empty, P^* can be any

point from the intersection (Fig. 3.13b). If the intersection is empty, P^* is found using the Point – Polyline – Point algorithm (Fig. 3.13c).

The case when point C lies on p-section and point D lies on c-section of p is symmetric to the previous case.

If points C and D both lie on different p-sections of p, the intersection points of line AB and the two polylines associated with points C and D are found and called E and F. If the intersection of line-segments AB and EF is non-empty, P^* can be any point from the intersection (Fig. 3.13d). If the intersection is empty, P^* is found using the Point – Polyline – Point on the two polylines associated with points C and D. P^* is chosen from the two optimal points found on the two polylines (Fig. 3.14a).

If points C and D both lie on the same p-section of p, the intersection of line-segment AB and the polyline associated with points C and D is checked. If the intersection is non-empty, any point from that intersection can be returned as P^* (Fig. 3.14b). If the intersection is empty, the Point – Polyline – Point algorithm is used to find P^* (Fig. 3.14c).

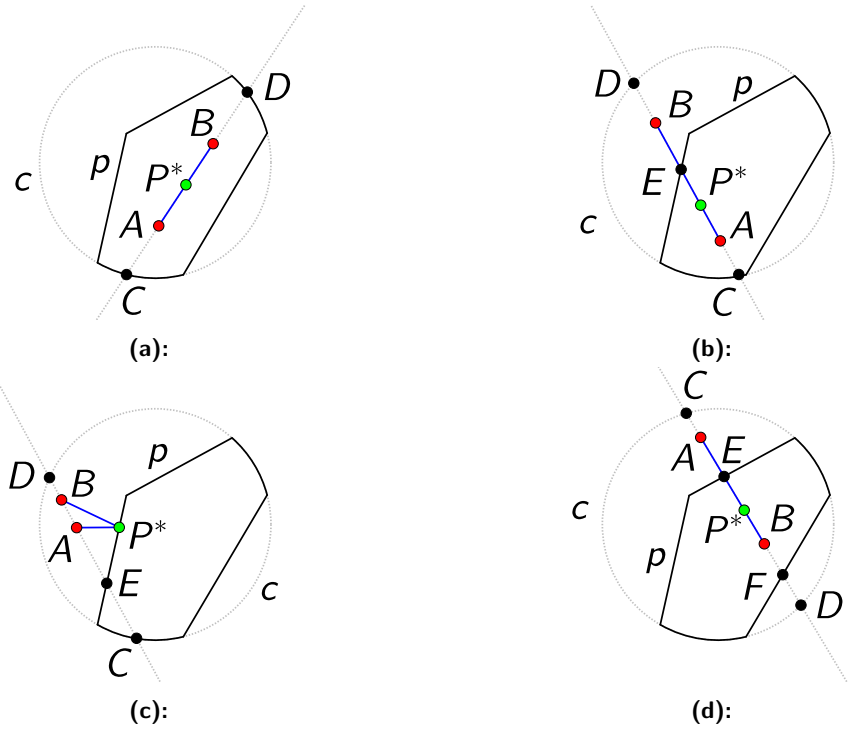


Figure 3.13: Example 4, first four cases.

Example 5

Polygoncircle p is a proper polygoncircle, points A and B are both outside circle c and line-segment AB does not intersect c. The PCP algorithm is used to find point C on circle c.

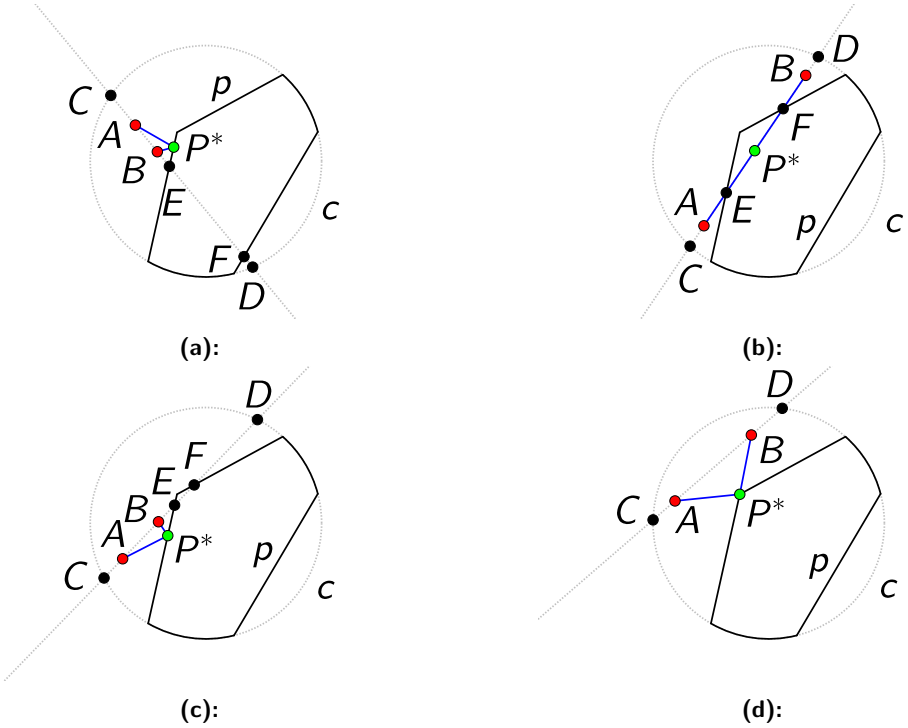


Figure 3.14: Example 4, last four cases.

If point C lies on c-section of p, C is returned as the optimal point P^* (Fig. 3.15a).

If point C lies on p-section of p, P^* lies on the polyline associated with point C and the Point – Polyline – Point algorithm can be used to find P^* (Fig. 3.15b).

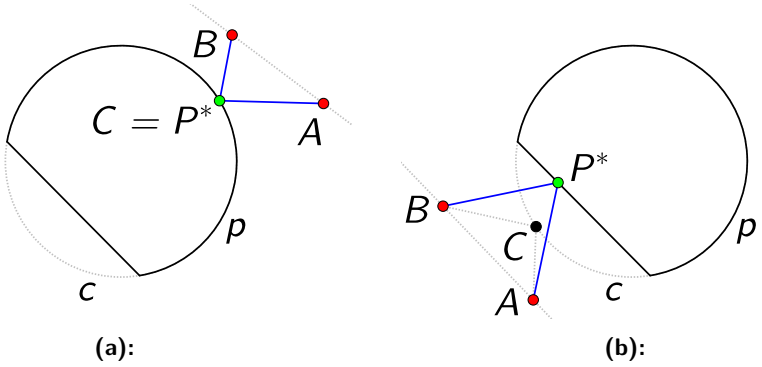


Figure 3.15: Example 5.

3.2 Touring Polygoncircle Problem (TPCP)

The Touring Polygoncircle Problem (TPCP) is the problem of finding a closed path on a given sequence of polygoncircles with the minimal length. More formally, for a given fixed sequence of polygoncircles $\mathcal{S}_{polyc} = \{p_1, p_2, \ldots, p_n\}$ find a closed

path $\mathcal{S}_{path} = \{P_1, P_2, \ldots, P_n\}$ such that the length of $\mathcal{S}_{path}$ is minimal. A closed path is defined by a sequence of points where every point P_i lies on polygoncircle p_i. The length of a closed path is defined in the following way:

$$len(\mathcal{S}_{path}) = \sum_{n=1}^{n} dist(P_i, P_{(i+1)mod(n)})$$

We used an iterative approach to solve the TPCP. It is described by Algorithm 1. This algorithm is inspired by the Touring Polygon Problem (TPP) algorithm [4], which solves a similar problem, but instead of polygoncircles, it optimizes a closed path on a fixed sequence of polygons.

The input of the algorithm is a fixed sequence of polygoncircles $\mathcal{S}_{polyc}$. The algorithm's output is a sequence of points representing an optimized closed path $\mathcal{S}_{path}$.

Algorithm 1: TPCP

Input : Sequence of polygoncircles $\mathcal{S}_{polyc}$
Output : Optimal closed path $\mathcal{S}_{path}$

```
1 P ← init_points(S_polyc);
2 while stopping criteria not met do
3     for i ← 1 to n do
4         A ← predecessor(P_i);
5         B ← successor(P_i);
6         P_i ← find_optimal_point(A, B, p_i);
```

The algorithm starts by initializing the closed path $\mathcal{S}_{path}$ (Line 1). This is done by randomly choosing a point on every polygoncircle. The main part of the algorithm consists of two nested cycles. The algorithm is stopped after the stopping criteria are met. The inner cycle iterates through points in $\mathcal{S}_{path}$ (Line 3) and each point is updated using the PPCP algorithm (Line 6). More precisely, in the ith iteration of the inner cycle, the point P_i is updated by applying the PPCP algorithm to points $P_{(i-1)mod(n)}$ and $P_{(i+1)mod(n)}$ and the polygoncircle p_i.

There are two stopping conditions that can stop the execution of the algorithm. The first condition occurs if some number of iterations of the outer cycle is reached. The second condition occurs if the difference between the lengths of the closed path $\mathcal{S}_{path}$ in the previous iteration and in the current iteration is smaller than some small value ϵ.

The stopping criteria ensure that the algorithm always stops and also that if changes in the length of $\mathcal{S}_{path}$ between subsequent iterations become small, the algorithm does not continue.

3.3 GLNS for TSP with Polygonal Neighborhoods

This chapter deals with solving the TSPN problem, for which we modified the GLNS solver [2]. The GLNS solver was originally developed for solving the exactly-one-in-a set Generalized Travelling Salesman Problem (GTSP), which is a generalization of the classic Travelling Salesman Problem (TSP).

In the GTSP, a set of vertices is partitioned into disjoined sets and the goal is to find a closed path of minimal length that passes through at least one point in each set from the partition.

The advantage of the GLNS solver is that it can easily be modified to solve the TSPN.

The GLNS solver works under the framework of Adaptive Large Neighborhood Search (ALNS). The basic idea of ALNS is simple. First, an initial solution is constructed by some initialization procedure. Then, the solution is iterativerly partially destroyed and repaired. If in some iteration i a better solution is found, that solution is accepted and the destroy/repair loop continues until some stopping criterion is met.

Two types of heuristics are used in the destroy/repair loop. In each iteration a removal heuristic is chosen to partially destroy the current solution and an insertion heuristic is chosen to repair the solution.

Each removal and insertion heuristic has an associated weight and in each destroy/repair iteration, the heuristics are chosen according to their weights. The weights are updated online during the execution of the algorithm in such a way that a successful heuristic will have a greater probability of being chosen in the future.

3.3.1 Algorithm Description

The structure of the GLNS solver is described by Algorithm 2. This algorithm uses a data structure called tour, which describes both a sequence of points and a corresponding sequence of polygoncircles.

Tour $\mathcal{T}$ is a pair $(\mathcal{S}_{path}, \mathcal{S}_{polyc})$, where $\mathcal{S}_{path}$ is a sequence of points defining a closed path and $\mathcal{S}_{polyc}$ is the corresponding sequence of polygoncircles.

Like in TPCP algorithm, the input is a sequence of polygoncircles $\mathcal{S}_{polyc}$. The output is a tour $\mathcal{T}_{lowest}$ describing an optimal closed path and the corresponding permutation of the input sequence of polygoncircles $\mathcal{S}_{polyc}$.

The main body of the algorithm consists of three nested loops. Each iteration of the first loop is called a cold restart. The number of cold restarts is determined by the parameter N_{cold}. In each cold restart, a tour is initialized and saved as $\mathcal{T}_{best}$. Then, unless it is the first cold restart, the weights assigned to removal and insertion heuristics are updated. The second loop follows.

The iterations in the second loop are called warm restarts and their number is determined by the parameter N_{warm}. At the beginning of each warm restart, the parameter $\mathcal{T}_{current}$ is initialized with the current value of $\mathcal{T}_{best}$. The third loop follows.

The third loop does not have a fixed number of iterations. In each iteration of the third loop, the number N_r of vertices to remove by removal heuristic is uniformly randomly selected. The removal heuristic R and insertion heuristic I are selected randomly using the selection weights. N_r vertices and polygon circles are removed by R from $\mathcal{T}_{current}$ and the partially destroyed $\mathcal{T}_{current}$ is then rebuilt by adding N_r polygoncircles and vertices using I. If the new tour is accepted, the $\mathcal{T}_{current}$ will hold this new tour. If not, $\mathcal{T}_{current}$ will have the old value. If the new tour $\mathcal{T}_{current}$ is better than $\mathcal{T}_{best}$, $\mathcal{T}_{current}$ is optimized and $\mathcal{T}_{best}$ is updated. The shortest tour from $\mathcal{T}_{best}$ found in all cold restarts will be optimized and returned by the algorithm as the resulting tour $\mathcal{T}_{lowest}$.

Algorithm 2: GLNS

Input : Sequence of polygoncircles $\mathcal{S}_{polyc}$
Output : Optimal tour $\mathcal{T}_{lowest}$

```
1  T_lowest ← init_tour(S_polyc);
2  for i ← 1 to N_cold do
3      T_best ← init_tour(S_polyc);
4      if i > 1 then
5          Update selection weights;
6      for j ← 1 to N_warm do
7          T_current ← T_best;
8          while stopping criterion not met do
9              From {1, ..., N_max} uniformly randomly select the number N_r of
                vertices to remove;
10             Select a removal heuristic R and an insertion heuristic I using
                the selection weights;
11             Create a copy of T_current called T_new;
12             Use R to remove N_r vertices and polygoncircles from T_new;
13             For each removed polygoncircle, use I to add that polygoncircle
                and a point on it to T_new;
14             if accept_tour(T_new, T_current) then
15                 T_current ← T_new;
16             if len(T_current) < len(T_best) then
17                 optimize(T_current);
18                 T_best ← T_current;
19     if len(T_best) < len(T_lowest) then
20         T_lowest ← T_best;
21 optimize(T_lowest);
```

3.3.2 Removal Heuristics

Three types of removal heuristics are used to remove polygoncircles and points from $\mathcal{T}_{new}$ in the remove/insert cycle:

- Segment removal
- Distance removal
- Worst removal

There is only one segment removal heuristic but the other two types are further parametrized by parameter λ.

The segment removal heuristic consists of uniformly randomly choosing an index i of some vertex in $\mathcal{T}_{new}$ and then erasing a continuous segment of N_r vertices and the associated polygoncircles from $\mathcal{T}_{new}$ starting from index i.

The distance removal heuristic starts by randomly uniformly choosing an index i of some vertex in $\mathcal{T}_{new}$. That vertex P_i is erased from $\mathcal{T}_{new}$ along with the corresponding polygoncircle p_i. A loop with $N_r - 1$ iterations follows. In each iteration, the remaining M vertices are sorted by their distance from P_i in ascending order. The parameter λ is used to construct an unnormalized discrete probability mass function $[\lambda^0, \lambda^1, \ldots, \lambda^{M-1}]$. After normalization, this function is used to randomly select a vertex from the sorted sequence of vertices. The selected vertex is erased from $\mathcal{T}_{new}$ along with the corresponding polygoncircle. Next iteration follows.

The worst removal heuristic also uses the parameter λ to construct a probability mass function to choose the vertex that will be removed from $\mathcal{T}_{new}$. It consists of a loop with N_r iterations. In each iteration, the vertices in $\mathcal{T}_{new}$ are sorted by their removal cost value in ascending order and then a vertex and the corresponding polygoncircle is chosen and removed using the probability mass function $[\lambda^0, \lambda^1, \ldots, \lambda^{M-1}]$. The removal cost value of vertex P_i is computed in the following way. The predecessor of P_i in $\mathcal{T}_{new}$ is named P_j and its follower P_k. The removal cost of P_i is then computed as $dist(P_i, P_j) + dist(P_i, P_k) - dist(P_j, P_k)$.

3.3.3 Insertion Heuristics

Two types of insertion heuristics are used to repair $\mathcal{T}_{new}$ in the remove/insert cycle:

- Cheapest insertion
- Unified insertion

Assume a partial tour $\mathcal{T} = (\mathcal{S}_{path}, \mathcal{S}_{polyc})$, where $\mathcal{S}_{path} = \{P_1, P_2, \ldots, P_m\}$ and $\mathcal{S}_{polyc} = \{p_1, p_2, \ldots, p_m\}$. Call $\mathcal{T}'$ the set of all polygoncircles not in $\mathcal{T}$. An insertion heuristic chooses a polygoncircle $p_{in} \in \mathcal{T}'$ that will be inserted into $\mathcal{T}$,

computes point $P_{opt} \in p_{in}$ using PPCP algorithm and inserts polygoncircle p_{in} and point P_{opt} into a position in $\mathcal{T}$ called emplace index.

The emplace index $i_{emplace}$ is the position in $\mathcal{T}$ that minimizes the insertion cost of point P_{opt}. The insertion cost c_{insert} of point P_{opt} that is inserted between a pair of neighboring points P_j and P_k in partial tour $\mathcal{T}$ is defined in the following way:

$$c_{insert}(P_{opt}, P_j, P_k) = dist(P_{opt}, P_j) + dist(P_{opt}, P_k) - dist(P_j, P_k)$$

The procedure that searches for an optimal index $i_{emplace}$ is called emplace point heuristic. This procedure optimizes the insertion cost c_{insert} for a given polygoncircle p_{in}. It therefore optimizes $i_{emplace}$ and $P_{opt} \in p_{in}$ at the same time:

$$P_{opt}, i_{emplace} = \underset{P \in p_{in}, i \in \{1,\dots,m\}}{\arg\min} \{c_{insert}(P, P_{(i-1)mod(m)}, P_i)\}$$

The cheapest insertion heuristic consists of a loop with N_r iterations. In each iteration p_{in}, P_{opt} and $i_{emplace}$ are chosen so as to minimize c_{insert}:

$$p_{insert}, P_{opt}, i_{emplace} = \underset{p \in \mathcal{T}', P \in p, i \in \{1,\dots,m\}}{\arg\min} \{c_{insert}(P, P_{(i-1)mod(m)}, P_i)\}$$

The unified insertion heuristic is parametrized by λ. It again consists of a loop with N_r iterations. In each iteration a p_{in} is chosen from polygoncircles in $\mathcal{T}'$ and then P_{opt} and $i_{emplace}$ are found by emplace point heuristic. The procedure for choosing p_{in} sorts polygoncircles in $\mathcal{T}'$ according to their distance from $\mathcal{T}$. The distance of a polygoncircle p from tour $\mathcal{T}$ is defined as:

$$dist(p, \mathcal{T}) = \min_{p_i \in \mathcal{T}} dist(centr(p), centr(p_i))$$

The polygoncircle p_{in} is chosen from the sorted sequence of polygoncircles using a discrete probability mass function $[\lambda^0, \lambda^1, \dots, \lambda^{M-1}]$.

3.3.4 Tour Optimization

Apart from remove and insert heuristics, the GLNS algorithm uses two optimization techniques that can fix some parts of the solution in a short amount of time. These two techniques are called Re-Opt and Move-Opt.

In Re-Opt heuristic, the positions of points in a tour $\mathcal{T}$ are optimized while the order of polygoncircles remains the same. The TPCP algorithm is used to find the optimal positions of points on polygoncircles in $\mathcal{T}$.

The Move-Opt heuristic optimizes the order of polygoncircles in $\mathcal{T}$. First, a point P_i is uniformly randomly selected and removed from $\mathcal{T}$ together with the

associated polygoncircle p_i. Then a new point $P_{opt} \in p_i$ along with position $i_{emplace}$ that minimize the insertion cost c_{insert} are computed by emplace point heuristic and p_i and P_{opt} are inserted into $\mathcal{T}$ at position $i_{emplace}$. This procedure is repeated N_{move} times.

3.3.5 Tour Initialization

At the beginning of the GLNS algorithm, an initial tour must be created. In this version of GLNS algorithm for polygoncircles, the initialization can be done in two ways:

- Random initialization
- Random insertion initialization

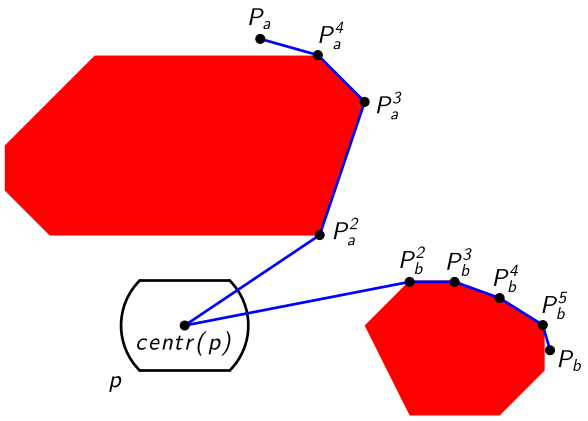
The random initialization first creates a random sequence of all polygoncircles $\mathcal{S}_{polyc}$ and then the path $\mathcal{S}_{path}$ is constructed by randomly choosing a point on every polygoncircle in $\mathcal{S}_{polyc}$.

The random insertion initialization starts by uniformly randomly choosing two polygoncircles p_1 and p_2. The associated points $P_1 \in p_1$ and $P_2 \in p_2$ are also selected randomly. A partial tour $\mathcal{T}$ is created by setting $\mathcal{S}_{polyc} = \{p_1, p_2\}$ and $\mathcal{S}_{path} = \{P_1, P_2\}$. The remaining polygoncircles are added to $\mathcal{T}$ in a loop where in each iteration a polygoncircle $p_i \notin \mathcal{T}$ is uniformly randomly selected. After p_i is selected, a point $P_{opt} \in p_i$ along with position $i_{emplace}$ that minimize the insertion cost c_{insert} are computed by emplace point heuristic and p_i and P_{opt} are inserted into $\mathcal{T}$ at position $i_{emplace}$. The loop continues until $\mathcal{T}$ contains all polygoncircles.

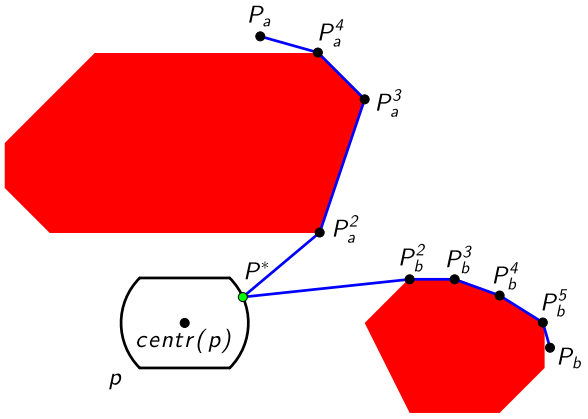
3.4 Environment with Obstacles

When working in an environment with polygonal obstacles, the Euclidean metric cannot be used to compute distances between points. Instead, the distance between points P_i and P_j is defined as the length of the shortest collision-free path with end-points in P_i and P_j. A collision-free path between points P_i and P_j is a polyline defined by a sequence of points $\{P_1, P_2, \ldots, P_n\}$, where $P_1 = P_i$, $P_n = P_j$ such that it does not intersect any polygonal obstacle.

Working in an environment with obstacles requires changes in the PPCP algorithm. Assume a polygoncircle p and points P_a and P_b. The version of PPCP algorithm for an environment with obstacles computes the optimal point $P^* \in p$ in the following way. The shortest collision-free paths between points $centr(p)$ and P_a and between points $centr(p)$ and P_b are computed (Fig. 3.16a). These paths are represented as sequences of points $\{P_a^1, P_a^2, \ldots, P_a^m\}$ and $\{P_b^1, P_b^2, \ldots, P_b^n\}$ where $P_a^1 = centr(p)$, $P_a^m = P_a$, $P_b^1 = centr(p)$, $P_b^m = P_b$. The optimal point P^* is found by using the original PPCP algorithm to solve the instance of the PPCP problem consisting of the polygoncircle p and points P_a^2 and P_b^2 (Fig. 3.16b).



(a):



(b):

Figure 3.16: PPCP in an environment with obstacles.

When using the GLNS algorithm in an environment with obstacles, the tour $\mathcal{T}_{lowest}$ must be reconstructed at the end of the algorithm. Without the reconstruction, there is no guarantee that the line-segment between consecutive points in $\mathcal{T}_{lowest}$ does not intersect an obstacle. The reconstruction consists of finding the shortest collision-free path between each pair of consecutive points in $\mathcal{T}_{lowest}$. The path defined by the reconstructed tour $\mathcal{T}_{lowest}$ is guaranteed to be collision-free.

The Fade2D library [5] was used to generate meshes on polygonal maps with obstacles and the Polyanya solver [6] was used for finding the shortest collision-free paths between points in environments with polygonal obstacles using meshes generated by Fade2D library.

Chapter 4

Results

The heuristic algorithms described in previous chapters were used to solve such instances of the TSPN, where the set of polygons cover the polygonal map. Solving the TSPN on such instances also gives solution to the WRP. The algorithms were implemented in C++ using the C++ 17 standard.

Several polygonal domains were used to experiment with the algorithms: complex2 (Fig. 4.2), jf-jh (Fig. 4.3), jf-pb2 (Fig. 4.4), jf-ta2 (Fig. 4.5), potholes (Fig. 4.6), warehouse2 (Fig. 4.7). For each of these domains, paths were found for multiple visibility radii. On Figure 4.2, the computed paths are shown along with polygoncircles.

The files containing map representations and sets of polygoncircles that were used in tests and experiments were kindly provided by Jan Mikula. In an article published by Jan Mikula and Miroslav Kulich [1], a procedure generating a set of polygons that cover a map is described. In the context of that article, sets of polygons and not polygoncircles are generated. However, a very similar procedure is used for generating sets of polygoncircles covering some map.

The results obtained from solving the TSPN on maps jf-jh, jf-ta2, jf-pb2 and visibility radii 2, 3, 4, 5, 10 were compared with results in an article published by J. Mikula and M. Kulich [1]. They also use decoupled approach but partition the map into polygons, not polygoncircles. Then they discretize the problem by sampling the polygons and solve the GTSP on the obtained samples.

The experiments were executed within the same computational environment as in [1] using a single core of the Intel Core i7-6700 CPU (3.40 GHz), 16 GB of RAM, and running Ubuntu 20.04. The results can therefore be directly compared.

Two parametrizations of the proposed algorithm were used for experiments in [1]. The parametrization named Trade-off provides a trade-off between solution quality and runtime. The parametrization named Best aims at finding the best quality solutions regardless of runtime. The results of both parametrizations along with the results of our experiments are in Table 4.1). Our experiments were given as input the running times of the TSPN part of the Best parametrization. If the time limit was reached, our algorithm did not continue with the next cold restart

and stopped executing. Each combination of map and visibility radius was solved 20 times.

In the table, d stands for visibility radius. The values of l_{ref} are the reference path lengths taken from [7]. The experimental results in the table are given relative to these reference values. PDM means PD(l_{mean}) and PDB means PD(l_{best}), where l_{mean} is the mean solution and l_{best} is the best solution from all runs of the algorithm for some combination of map and d. PD(l) is the percent deviation from the reference path length l_{ref} and is computed as $100(l - l_{ref})/l_{ref}$. In experiments done in [1], in each run of the algorith, a new partition of the map is constructed with possibly different number of polygons. $\bar{n}$ is the mean number of polygons from all solutions. In our experiments, the set of polygoncircles for one combination of map and d was fixed. n is therefore the number of polygons. $\bar{t}$ is the average time.

The paths found by our algorithm are almost always better than paths found by the Trade-off parametrization. However, our algorithm's $\bar{t}$ is almost always bigger. On the other hand, the paths found by the Best parametrization are always better, but its $\bar{t}$ is almost always bigger, because it was used as time-limit to our algorithm.

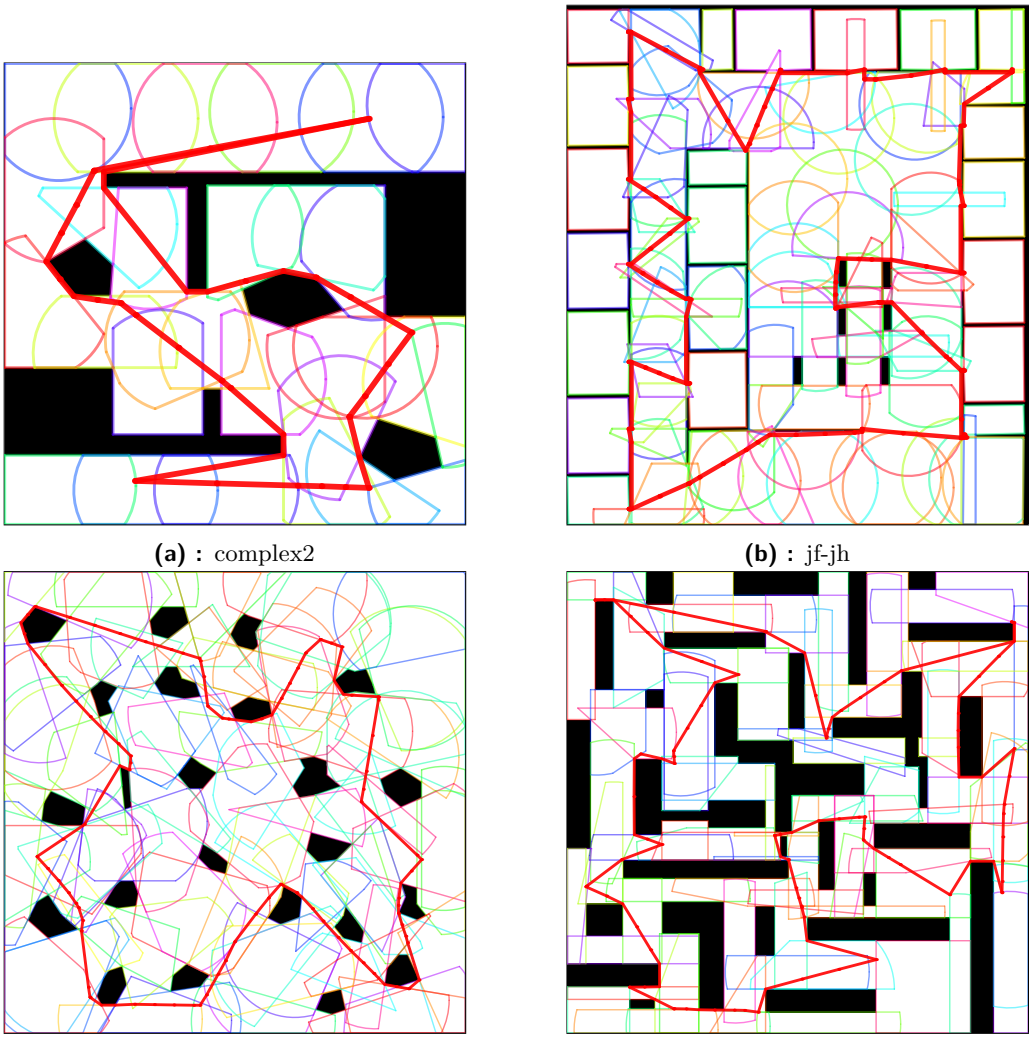
The GLNS algorithm can be used in multiple modes. It uses many parameters that affect its execution. In our experiments, the Fast mode was used. Other modes are not practical for large numbers of polygoncircles because of very long execution times. The Fast mode is characterized by a small number of cold and warm restarts. It also uses Re-Opt heuristic only at the end of the algorithm, before the final path is returned.

For instances with a high number of polygoncircles, it is not immediately clear why the quality of solution is not better than the Best parametrization of algorithm in [1]. But since we use continuous optimization using PPCP and in [1] polygoncircles are sampled, it could be expected that at least in instances with fewer polygoncircles, our solutions should be consistently at least slightly better. However, this is not the case and a further investigation into this matter should be made.

The PPCP algorithm was thoroughly tested on millions of randomly generated instances. The points found by the PPCP algorithm were compared with the points found by a sampling method: The border of a polygoncircle was sampled by a large number of points. The best sample was compared with the point computed by the PPCP algorithm. If the distance between the two points is small, the computed point is considered to be correct.

The TPCP algorithm is simple and there is not much room for tuning.

The GLNS algorithm, on the other hand, is very complex and parametrized by many parameters. It is probable that here, many improvements can be made. It would be useful to create other modes that would be experimentaly found to work well with polygoncircles.



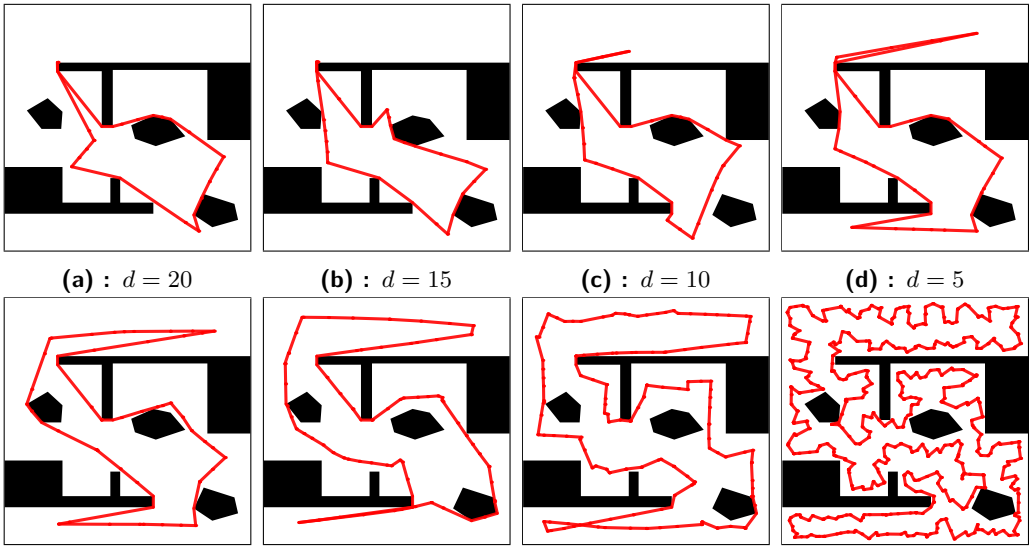
(a) : complex2 **(b) :** jf-jh

(c) : potholes **(d) :** warehouse2

Figure 4.1: Examples of solutions to WRP.

Map	d	l_{ref}	Trade-off [1]				Best [1]				Our results			
			$\bar{n}$	PDM	PDB	$\bar{t}$	$\bar{n}$	PDM	PDB	$\bar{t}$	n	PDM	PDB	$\bar{t}$
jh	2	295.5	353	-24.1	-26.1	2.4	343	-33.0	-34.5	131.2	367	-23.2	-24.6	28.3
	3	215.5	191	-33.1	-36.7	1.9	173	-42.4	-43.2	112.3	194	-36.1	-37.2	6.2
	4	207.9	123	-46.7	-49.4	1.6	117	-49.0	-49.6	59.8	133	-46.0	-47.6	2.0
	5	204.3	93	-50.5	-51.5	1.5	84	-51.5	-52.2	25.6	95	-50.1	-51.3	1.1
	10	194.6	62	-51.6	-52.8	1.4	60	-51.4	-53.3	13.7	71	-50.5	-51.3	1.8
ta	2	427.0	502	-15.5	-19.2	2.2	489	-26.8	-28.7	175.5	519	-20.9	-21.9	70.6
	3	335.6	247	-19.9	-25.1	1.5	239	-26.9	-28.5	122.6	266	-24.5	-25.3	11.4
	4	291.3	156	-23.8	-26.7	1.3	150	-28.1	-29.9	109.6	162	-24.9	-25.6	2.3
	5	256.8	110	-24.1	-25.6	1.3	105	-27.9	-29.3	48.3	113	-25.8	-26.4	1.0
	10	216.9	52	-35.2	-36.9	1.1	50	-34.9	-37.4	8.8	53	-35.1	-35.5	0.2
pb	2	919.0	1089	-3.5	-7.7	9.9	1064	-16.9	-18.2	452.2	1136	-12.3	-13.0	474.1
	3	781.6	492	-9.1	-10.3	3.6	456	-13.7	-14.3	172.7	485	-12.3	-12.5	92.5
	4	721.9	300	-10.4	-12.1	2.0	284	-12.6	-13.3	125.4	287	-12.2	-12.4	14.3
	5	687.2	212	-11.7	-12.5	1.6	207	-12.8	-13.4	112.3	205	-11.9	-12.1	5.7
	10	615.9	96	-17.8	-18.8	1.3	94	-19.6	-20.0	28.3	89	-17.7	-18.2	0.4

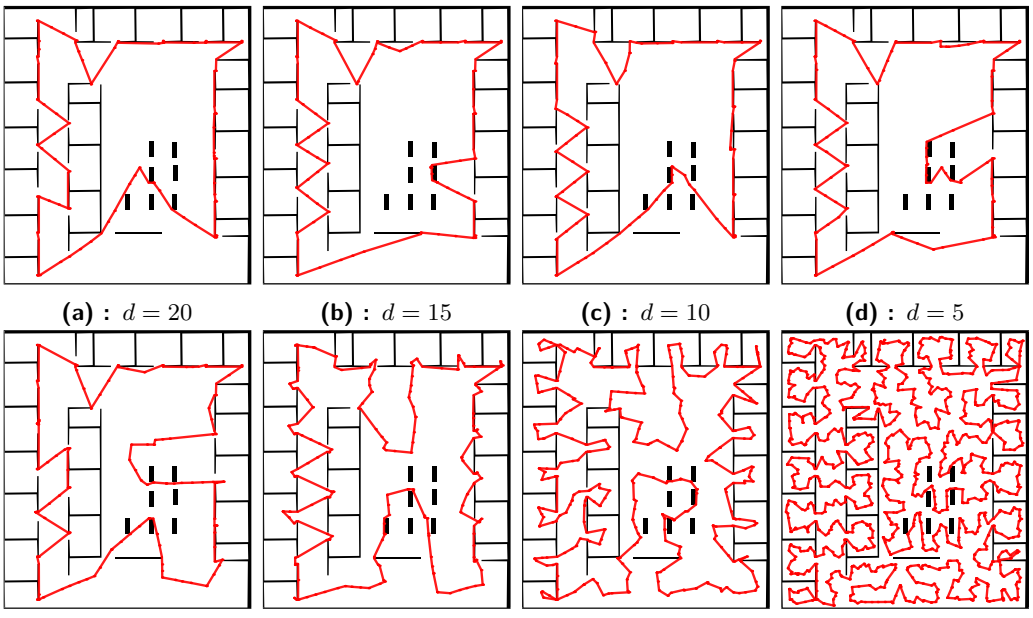
Table 4.1: Comparision of our experimental results with results of a similar algorithm described in [1]



(a) : $d = 20$ **(b) :** $d = 15$ **(c) :** $d = 10$ **(d) :** $d = 5$

(e) : $d = 4$ **(f) :** $d = 3$ **(g) :** $d = 2$ **(h) :** $d = 1$

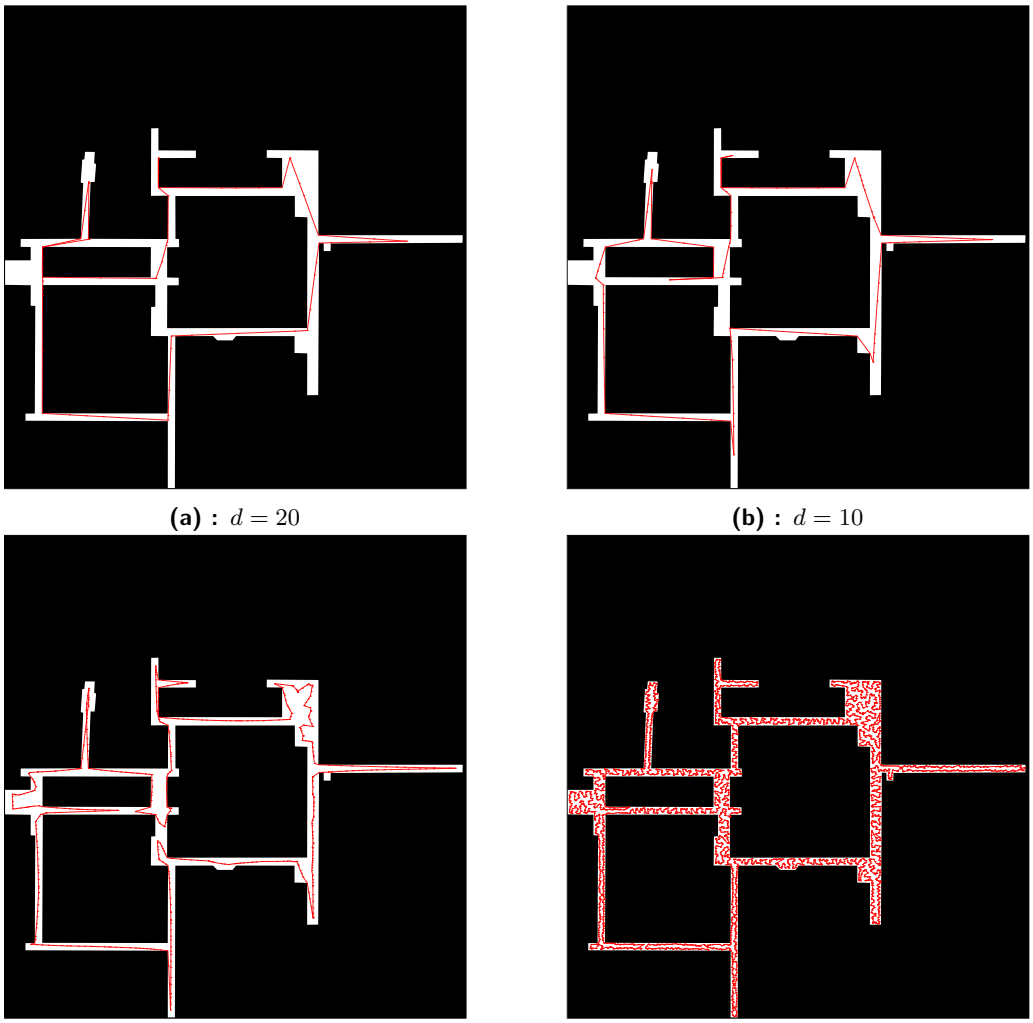
Figure 4.2: Different values of d in complex2.



(a) : $d = 20$ **(b) :** $d = 15$ **(c) :** $d = 10$ **(d) :** $d = 5$

(e) : $d = 4$ **(f) :** $d = 3$ **(g) :** $d = 2$ **(h) :** $d = 1$

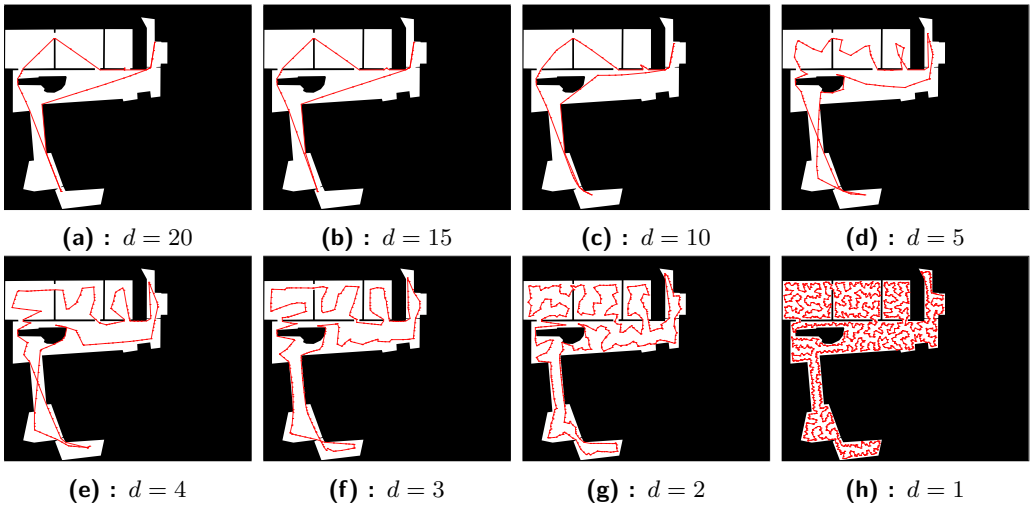
Figure 4.3: Different values of d in jf-jh.



(a) : $d = 20$ **(b) :** $d = 10$

(c) : $d = 3$ **(d) :** $d = 1$

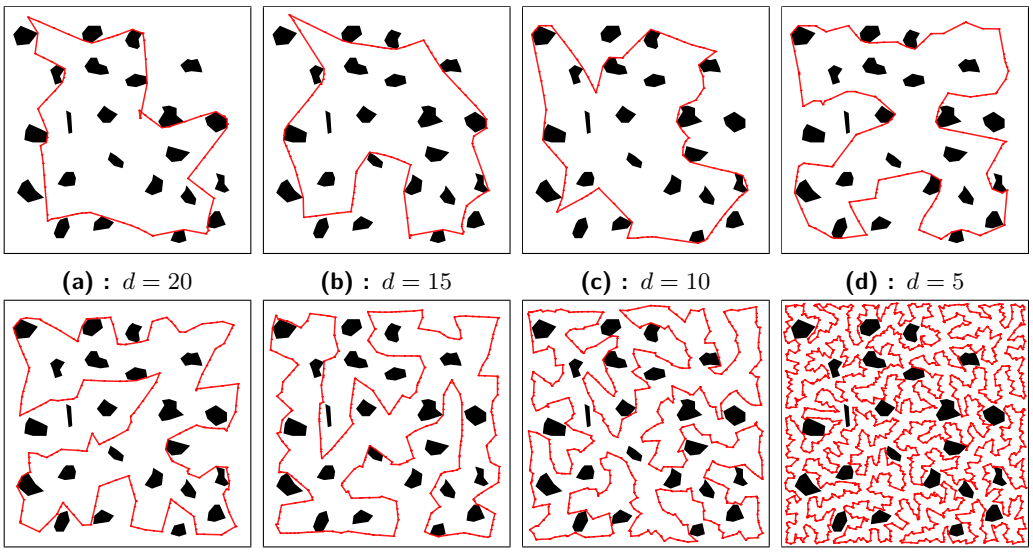
Figure 4.4: Different values of d in jf-pb2.



(a) : $d = 20$ **(b) :** $d = 15$ **(c) :** $d = 10$ **(d) :** $d = 5$

(e) : $d = 4$ **(f) :** $d = 3$ **(g) :** $d = 2$ **(h) :** $d = 1$

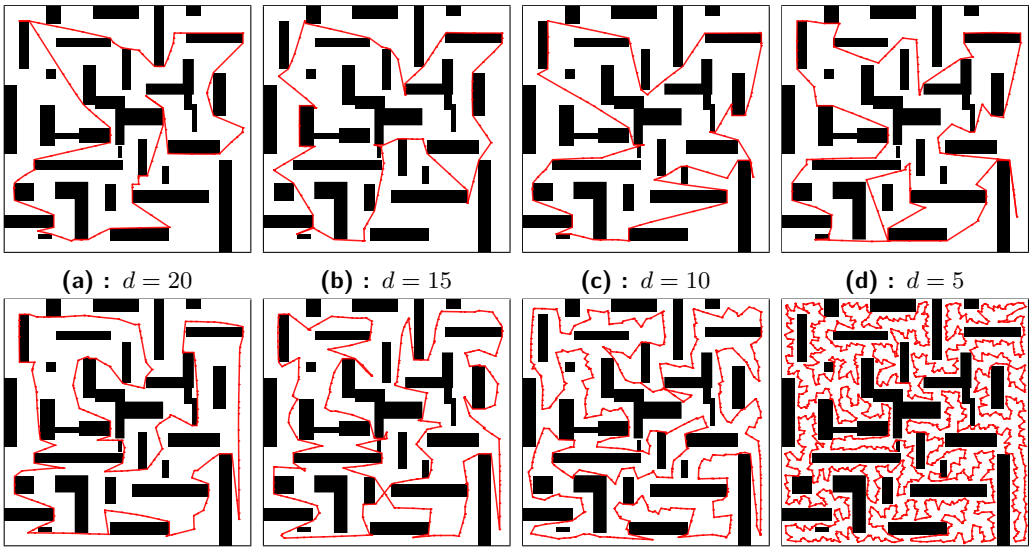
Figure 4.5: Different values of d in jf-ta2.



(a) : $d = 20$ **(b)** : $d = 15$ **(c)** : $d = 10$ **(d)** : $d = 5$

(e) : $d = 4$ **(f)** : $d = 3$ **(g)** : $d = 2$ **(h)** : $d = 1$

Figure 4.6: Different values of d in potholes.



(a) : $d = 20$ **(b)** : $d = 15$ **(c)** : $d = 10$ **(d)** : $d = 5$

(e) : $d = 4$ **(f)** : $d = 3$ **(g)** : $d = 2$ **(h)** : $d = 1$

Figure 4.7: Different values of d in warehouse2.

Appendix A

Bibliography

[1] J. Mikula and M. Kulich, "Towards a continuous solution of the d-visibility watchman route problem in a polygon with holes," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 5934–5941, 2022.

[2] S. Smith and F. Imeson, "Glns: An effective large neighborhood search heuristic for the generalized traveling salesman problem," *Computers & Operations Research*, vol. 87, 05 2017.

[3] L. Fanta, "The Close Enough Travelling Salesman Problem in the polygonal domain," Master's thesis, CTU in Prague, 2021.

[4] M. Dror, A. Efrat, A. Lubiw, and J. Mitchell, "Touring a sequence of polygons," *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*, 04 2003.

[5] Geom Software, "Fade2d (1.99)." `https://www.geom.at/fade2d/html/index.html`, 2022.

[6] M. Cui, D. Harabor, and A. Grastien, "Compromise-free pathfinding on a navigation mesh," pp. 496–502, 08 2017.

[7] J. Faigl and L. Přeučil, "Inspection planning in the polygonal domain by self-organizing map," *Applied Soft Computing*, vol. 11, no. 8, pp. 5028–5041, 2011.

Appendix B

Contents of the attached CD

File	Description
`code.zip`	An archive containing `C++` source files along with data files used for testing and generating images.
`text.zip`	An archive containing the text of the thesis.
`results.zip`	An archive containing generated images of maps, polygons and found paths.