ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Vidašič** Jméno: **Jan** Osobní číslo: **457120**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Problém obchodního cestujícího se sousedstvími

Název diplomové práce anglicky:

Travelling Salesman Problem with Neighborhoods

Pokyny pro vypracování:

1. Seznamte se s problémem obchodního cestujícího se sousedstvími
(TSPN - Travelling Salesman Problem with Neighborhoods).
2. Seznamte se s metaheuristikami pro řešení směrovacích problémů.
3. Navrhněte a implementuje metaheuristický algoritmus pro řešení TSPN.
Inspirujte se zejména [2].
4. Navrhněte/diskutujte rozšíření realizovaného algoritmu pro aplikace v
mobilní robotice, např. pro úlohu inspekce prostředí, případně uvažujte
prostředí s překážkami
5. Experimentálně vyhodnoťte vlastnosti implementovaného algoritmu.
Popište a diskutujte dosažené výsledky.

Seznam doporučené literatury:

[1] Gendreau M., Potvin JY. (eds) Handbook of Metaheuristics.
International Series in Operations Research & Management Science, vol
272. Springer, Cham, 2019
[2] Stephen L. Smith, Frank Imeson, GLNS: An effective large
neighborhood search heuristic for the Generalized Traveling Salesman
Problem, Computers & Operations Research, Volume 87, 2017
[3] David Woller, Hledání zdrojů gama záření, Search for sources of
gamma radiation, Diplomová práce, České vysoké učení technické v
Praze, 2019
[4] Pan, Xiuxia, Fajie Li, and Reinhard Klette. Approximate shortest path
algorithms for sequences of pairwise disjoint simple polygons. Department
of Computer Science, University of Auckland, 2010.

Jméno a pracoviště vedoucí(ho) diplomové práce:

RNDr. Miroslav Kulich, Ph.D., inteligentní a mobilní robotika CIIRC

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

Datum zadání diplomové práce: **11.02.2020** Termín odevzdání diplomové práce: **22.05.2020**

Platnost zadání diplomové práce: **30.09.2021**

RNDr. Miroslav Kulich, 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

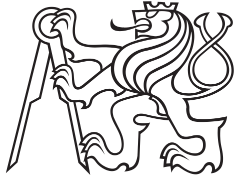
III. PŘEVZETÍ ZADÁNÍ

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

Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computers

Travelling Salesman Problem with Neighborhoods

Bc. Jan Vidašič

Supervisor: RNDr. Miroslav Kulich, Ph.D.

May 2020

Acknowledgements

I would like to thank my supervisor, RNDr. Miroslav Kulich Ph.D., for his willingnes and support.

Declaration

I declare that the presented work was written independently and that I have listed all used sources of information as per the methodical instructions about following of the ethical principles in preparation of university thesis.

Prague, 21. May 2020

Abstract

This thesis explores the possibility of transforming the metaheuristic algorithm GLNS, used for General Travelling Salesman Problem (GTSP), to instead solve the version of Travelling Salesman Problem with Neighborhoods (TSPN) where the neighborhoods are simple, possibly intersecting, polygons. Two algorithms are proposed and implemented, each utilizing GLNS in a different way. Both also make use of an algorithm solving the unconstrained version of Touring Polygons Problem (TPP). The second proposed algorithm is additionally equipped to handle a case, when there are simple polygonal obstacles between the neighborhoods. This is made possible using a visibility graph.

Keywords: TSPN, GLNS, TPP, Travelling Salesman Problem with Neighborhoods, Touring Polygons Problem

Supervisor: RNDr. Miroslav Kulich, Ph.D.
Czech Technical University in Prague
Czech Institute of Informatics, Robotics, and Cybernetics
Jugoslávských partyzánů 1580/3
16000 Prague 6

Abstrakt

Práce se zabývá využitím metaheuristického algoritmu GLNS, používaného k řešení problému obecného obchodního cestujícího, k řešení upraveného problému obchodního cestujícího se sousedstvími. Tato úprava spočívá v tom, že sousedstvími jsou pouze nedegenerované mnohoúhelníky, jež se mohou i překrývat. V rámci práce jsou navrhnuty a implementovány dva algorithmy, které využívají původní nebo modifikovaný algoritmus GLNS. Dále je v obou také využit algoritmus pro řešení úlohy průchodu mnohoúhelníky. Druhý navrhnutý algoritmus je schopný řešit i instance, kde jsou mezi sousedstvími překážky ve tvaru nedegenerovaných mnohoúhelníků. Využívá k tomu datové struktury, která se nazývá graf viditelnosti.

Klíčová slova: Obchodní cestující se sousedstvími

Překlad názvu: Problém obchodního cestujícího se sousedstvími

Contents

1 Introduction **1**

2 Definitions **3**
2.1 Generalized travelling salesman problem . . . 3
2.2 Travelling salesman problem with neighborhoods . . . 3
2.3 Touring polygons problem . . . 3
2.4 Thesis problem formulation . . . 4
2.4.1 $\text{TSPN}_{disjoint}$ problem formulation . . . 5
2.4.2 $\text{TSPN}_{intersect}$ problem formulation . . . 5
2.4.3 $\text{TSPN}_{obstacles}$ problem formulation . . . 5
2.4.4 Thesis problems overview . . . 6

3 Methods **7**
3.1 Floating TPP algorithm . . . 7
3.2 GLNS algorithm . . . 11
3.2.1 Solver framework . . . 12
3.2.2 Insertion heuristics . . . 13
3.2.3 Bounding insertions before evaluation . . . 14
3.2.4 Removal heuristics . . . 14
3.2.5 Local optimizations . . . 15
3.2.6 Modes of operation . . . 15
3.2.7 Implementation details . . . 16
3.3 GLNSC . . . 16
3.3.1 Floating TPP modifications . . . 16
3.3.2 GLNS modifications . . . 19
3.3.3 Implementation change of insertions . . . 21
3.3.4 Parameters . . . 21
3.4 GLNS-TPP . . . 22
3.4.1 GLNS modification . . . 23
3.4.2 TPP modifications . . . 24
3.4.3 Optimization of TPP on $\text{TSPN}_{obstacles}$ . . . 25
3.4.4 Possible future improvements . . . 26
3.4.5 Parameters . . . 26

4 Experimental results **29**
4.1 Methodology . . . 29
4.1.1 Instances . . . 29
4.1.2 Measurements . . . 31
4.2 GLNSC . . . 32
4.2.1 Insertion change . . . 32
4.3 GLNS-TPP . . . 32
4.3.1 TPP change . . . 33
4.3.2 $\text{TSPN}_{obstacles}$ . . . 34
4.4 GLNSC and GLNS-TPP comparison . . . 38
4.4.1 Influence of the amount of edges . . . 38
4.4.2 $\text{TSPN}_{disjoint}$ and $\text{TSPN}_{intersect}$ instances . . . 39

5 Conclusion **43**

Bibliography **45**

A Contents of the attached CD **47**

Figures

2.1 Example instances of the thesis problems 4

3.1 Illustration of the floating TPP for convex polygons (Algorithm 1). 8
3.2 Finding the closest point on a line segment in regards to the points x and y (Algorithm 3). 11
3.3 Comparison of choosing the boundary point and the point in the middle in Algorithm 5. 18
3.4 $x \in P$ and $y \in P$, therefore Algorithm 5 selects the point in the middle of (x, y). But P is not convex and such point is outside of its bounds. The optimal point p is on the frontier of P and therefore can be found by Algorithm 3. 19
3.5 General steps of GLNS-TPP. . . . 23
3.6 Process of finding the optimal point in the TSPN$_{obstacles}$ instances (Algorithm 6). 26
3.7 Example of a solution that is not optimal found by Algorithm 6. . . . 27

4.1 Examples of the solved TSPN$_{disjoint}$ instances. 30
4.2 Examples of the solved TSPN$_{intersect}$ instances. 30
4.3 Examples of the solved TSPN$_{obstacles}$ instances (red polygons represent the obstacles). 31
4.4 Influence of the GLNS mode setting on the runtime. 33
4.5 Solved instance of potholes_12 by the original and modified TPP. . . . 34
4.6 Average duration of a single TPP iteration on the *potholes_* instances. 36
4.7 Average number of TPP iterations on the *potholes_* instances. 36
4.8 Weight reduction by TPP iteration. 37
4.9 Amount of obstacles influence . . 38
4.10 Time dependence on the amount of edges of the polygons 39
4.11 Time dependence on the amount of polygons of GLNSC and GLNS-TPP 41

Tables

2.1 Problems overview 6

3.1 GLNSC parameters 22
3.2 GLNS-TPP parameters 27

4.1 GLNSC insertion change time and solution cost comparison. 32
4.2 Influence of the GLNS mode setting on the solution cost. 33
4.3 GLNS-TPP TPP change time and solution cost comparison. 34
4.4 GLNS-TPP time performance on $\text{TSPN}_{obstacles}$ instances. 35
4.5 GLNS-TPP solution costs on $\text{TSPN}_{obstacles}$ instances. 35
4.6 GLNS-TPP time dependency on the amount of obstacles. 37
4.7 GLNS-TPP solution costs dependency on the amount of obstacles. 38
4.8 Influence of the number of edges (per polygon) on the runtime of GLNSC and GLNS-TPP 39
4.9 Comparison of GLNSC and GLNS-TPP times on $\text{TSPN}_{disjoint}$ (density_) and $\text{TSPN}_{intersect}$ (ngons_, potholes_) 40
4.10 Comparison of GLNSC and GLNS-TPP solution costs on $\text{TSPN}_{disjoint}$ (density_) and $\text{TSPN}_{intersect}$ (ngons_, potholes_) 40

Chapter 1
Introduction

The thesis aims to design and develop an algorithm that finds the shortest tour visiting all given polygons and that starts and ends in the same polygon. This is a variant of the so called Traveling Salesman Problem with Neighborhoods (TSPN) and is an extension of the well known NP-hard Traveling Salesman Problem (TSP) with the difference that each vertex of the tour can be moved within a given neighborhood. It is a problem with vast applications in the motion planning tasks in robotics and logistics.

This thesis presents two different algorithms for the solution of TSPN, where neighborhoods are simple, not necessarily disjoint, polygons. One of the algorithms is also modified to handle a case, where there are simple polygonal disjoint obstacles between the neighborhoods that the tour has to avoid passing through. The core of both algorithms is a very effective heuristic solver for the Generalized Traveling Salesman Problem (GTSP) called GLNS, tailored for the addressed problem.

The first algorithm, we call GLNSC, is a modification of GLNS which works with polygons, instead of sets of points.

The second one - GLNS-TPP, works in two phases. In the first phase the TSPN problem is transformed into a GTSP problem by splitting the input polygons into triangular meshes and using the points in the center of the generated triangles as the input for GLNS. The order of the polygons found by GLNS is then used in the second phase which utilizes an algorithm for the so called Touring Polygons Problem (TPP). The algorithm finds a tour with the shortest length between the polygons with a fixed order. This algorithm is also capable of working with obstacles between the neighborhoods utilizing a data structure called visibility graph.

The purpose of Chapter 1 is to provide a general overview of what this thesis is about. Chapter 2 defines the existing problems that are connected to this work such as TSPN and TPP, and most importantly specifies the formulation of the problem this thesis is trying to solve. Chapter 3 first describes the GLNS solver of GTSP and then discusses both proposed algorithms. Chapter 4 presents experimental results with a comparison of both algorithms on maps without obstacles and the performance of the second algorithm on the maps with obstacles. Chapter 5 serves as an overview of the work done and describes possible future direction and improvements of the algorithms.

Chapter 2
Definitions

First, this chapter defines the problems this thesis builds upon - GTSP, TPP and TSPN. Then, the thesis problems themselves are defined. Both the GTSP and TPP problems are introduced here, because the algorithms used to solve them are utilized in some form in both algorithms presented in this thesis. TSPN is then the basis for the thesis problems formulation.

2.1 Generalized travelling salesman problem

The exactly-one-in-set GTSP is a well known NP-hard problem that can be described as follows [4].

Given a complete weighted graph $G = (V, E, w)$ on n vertices and a partition of V into m sets $P_v = \{V_1, \ldots, V_m\}$, where $V_i \cap V_j = \emptyset$ for all $i \neq j$ and $\cup_{i=1}^{m} V_i = V$, find a cycle in G that contains exactly one vertex from each set $V_i, i \in \{1, \ldots, m\}$ and has a minimum length.

2.2 Travelling salesman problem with neighborhoods

TSPN was first defined in [1]. We assume a salesman that wants to meet some buyers. Each buyer specifies a compact set in a plane, his neighborhood, where he is willing to meet. These sets can be of arbitrary shape, for example a disk or a polygon. The sets are also allowed to intersect. The salesman aims to find a tour of the shortest length that intersects the neighborhoods of all the buyers and returns to the initial point of departure.

TSPN is very similar to GTSP, the only difference being that the neigborhoods of the GTSP instance consist of points instead of continuous regions.

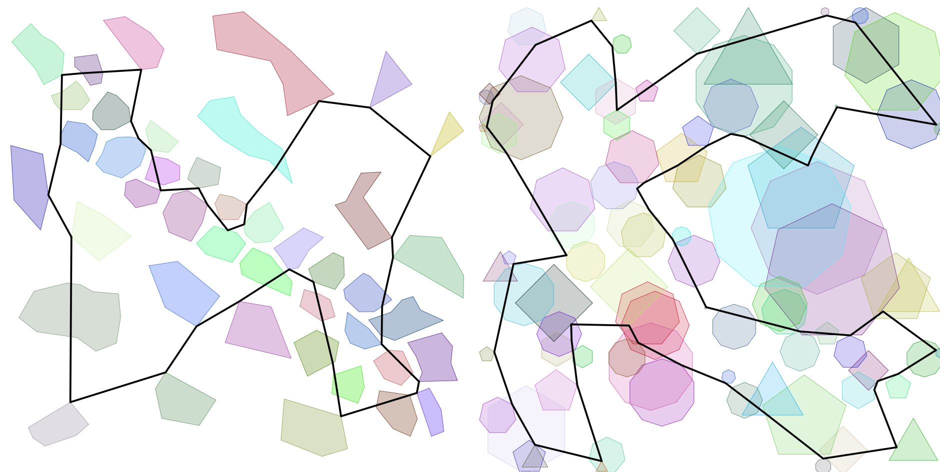
2.3 Touring polygons problem

TPP first appeared in paper [2]. The basic idea is that we assume a starting point s, endpoint q and an ordered set of simple polygons. We want to find the shortest path that starts at point s, visits all polygons from the set in

the specified order, and finishes at point q. The unconstrained version of this problem, where the points s and q are not specified, is called floating TPP. Both the TPP and the floating TPP problems also belong to the category of NP-hard problems.

2.4 Thesis problem formulation

This thesis aims to solve three different problems. They are in essence just restricted or updated versions of TSPN defined in Section 2.2. They build on top of each other and are progressively harder to solve as each of them adds some constraint on top of its predecessor. Their example instances can be found in Figure 2.1.



(a) : Solved TSPN$_{disjoint}$ instance

(b) : Solved TSPN$_{intersect}$ instance

(c) : Solved TSPN$_{obstacles}$ instance (red polygons are obstacles)

Figure 2.1: Example instances of the thesis problems

2.4.1 TSPN$_{disjoint}$ problem formulation

The first problem of this thesis, lets call it TSPN$_{disjoint}$, is the base problem of this thesis upon which the remaining problems are built. In an instance of TSPN$_{disjoint}$ the neighborhoods are allowed to be only simple polygons, whose convex hulls do not intersect with any other polygon (or its convex hull) of the instance. A more formal definition of TSPN$_{disjoint}$ follows.

Assume an instance $I = \{P\}$, where

- $P = \{P_i \mid i \in < 1, m >\}$
- P_i is a simple polygon for all i
- $C(P_i) \cap C(P_j) = \emptyset, \forall i, j$, where $C(\cdot)$ is a convex hull

The solution to a TSPN$_{disjoint}$ instance is a tour $T = (p_1, p_2, \ldots, p_m)$ visiting each polygon P_i exactly once with a minimum tour length $w_T = \sum_{i=1}^{m} d_e(p_i, p_{i+1})$, where d_e is the length of a straight line segment between the two consecutive points of the tour T. For a point p_i to visit polygon P_i the point must be inside P_i, that is $p_i \in P_i$.

2.4.2 TSPN$_{intersect}$ problem formulation

The second problem addressed in this thesis, lets call it TSPN$_{intersect}$, is similar to TSPN$_{disjoint}$ problem, with the exception that the polygons are allowed to intersect. It should be noted that an algorithm solving a TSPN$_{intersect}$ instance is also capable of solving a TSPN$_{disjoint}$ instance. A more formal definition of TSPN$_{intersect}$ follows.

Assume an instance $I = \{P\}$, where

- $P = \{P_i \mid i \in < 1, m >\}$
- P_i is a simple polygon for all i

The solution to a TSPN$_{intersect}$ instance is a tour $T = (p_1, p_2, \ldots, p_m)$ visiting each polygon P_i at least once with a minimum tour length $w_T = \sum_{i=1}^{m} d_e(p_i, p_{i+1})$.

2.4.3 TSPN$_{obstacles}$ problem formulation

The third addressed problem, lets call it TSPN$_{obstacles}$, is identical to the TSPN$_{intersect}$ problem, but introduces obstacles as an additional constraint. The tour of the salesman must not intersect any obstacle on the way around the neighborhoods. The obstacles must be simple polygons and have to be disjoint. Additionally the obstacles are not allowed to intersect with the neighborhoods that are to be visited. It should be noted that an algorithm solving a TSPN$_{obstacles}$ instance is also capable of solving TSPN$_{intersect}$ instance, as a TSPN$_{intersect}$ instance is a TSPN$_{obstacles}$ instance without obstacles. A more formal definition of TSPN$_{obstacles}$ follows.

Assume an instance $I = \{P, O\}$, where

- $P = \{P_i \mid i \in < 1, m >\}$
- $O = \{O_j \mid j \in < 1, n >\}$
- P_i is a simple polygon for all i
- O_j is a simple polygon for all j
- $O_k \cap O_l = \emptyset, \forall\ k \neq l$
- $O_v \cap P_w = \emptyset, \forall\ v, w$

The solution to a TSPN$_{obstacles}$ instance is a tour $T = (p_1, p_2, \dots, p_m)$ visiting each polygon P_i at least once with a minimum tour length $w_T = \sum_{i=1}^{m} d_e(p_i, p_{i+1})$, where d_e is the length of a shortest polyline l_i between the two consecutive points of the tour T, that does not intersect any obstacle O_j.

2.4.4 Thesis problems overview

Table 2.1 overviews the differences between the problems and shows which of the proposed algorithms (GLNSC and GLNS-TPP) is able to solve which problem.

Problem	Regions intersect	Obstacles	GLNSC solves	GLNS-TPP solves
TSPN$_{disjoint}$	No	No	Yes	Yes
TSPN$_{intersect}$	Yes	No	Yes	Yes
TSPN$_{obstacles}$	Yes	Yes	No	Yes

Table 2.1: Problems overview

Chapter 3

Methods

3.1 Floating TPP algorithm

Both GLNSC and GLNS-TPP incorporate a TPP solver in some form. This section thus describes the used algorithm for the floating version of TPP.

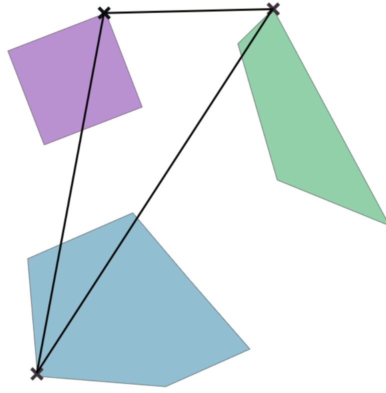
The algorithms solving TPP generally fall to the category of rubberband algorithms introduced in [3]. The reason for this name is that they work by 'tightening' the solution. While there exists a variety of algorithms solving different versions of TPP, the algorithm chosen for this thesis, due to its rather simple implementation, comes from the paper [10]. First, the pseudocode of the algorithm for disjoint convex polygons is presented (Algorithm 1) along with its visualization in Figure 3.1. Then the general version working with simple polygons whose convex hulls do not intersect is described (Algorithm 2).

Algorithm 1: Floating TPP algorithm for convex polygons

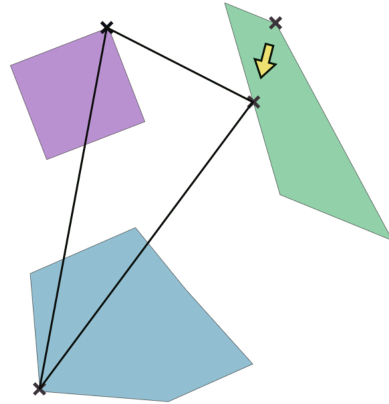
Data: A sequence of k pairwise disjoint simple polygons $P_0, P_1, \ldots, P_{k-1}$ in a plane π; an accuracy constant $\epsilon > 0$

Result: A sequence $< p_0, p_1, \ldots, p_k >$ (where $p_k = p_0$) which visits polygon P_i at p_i in the given order $i = 0, 1, \ldots, k-1$, and finally $i = 0$ again

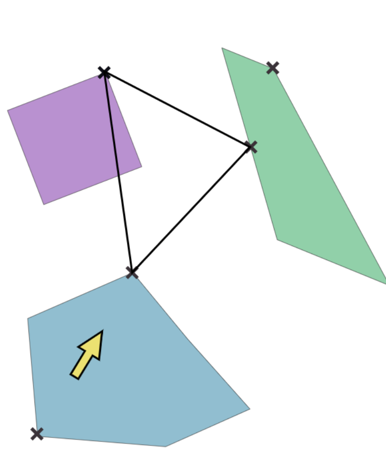
```
1  For each i ∈ {0, 1, ..., k − 1}, let initial p_i be a vertex of P_i;
2  L_0 = ∞;
3  L_1 = Σ_{i=0}^{k−1} d_e(p_i, p_{i+1});
4  while L_0 − L_1 >= ε do
5  |   for i = 0, 1, ..., k − 1 do
6  |   |   Let p_{i−1}, p_{i+1} and P_i be the input of Algorithm 3, the output
   |   |    is a point q_i ∈ ϑP_i;
7  |   |   Replace p_i by q_i;
8  |   end
9  |   L_0 = L_1;
10 |   L_1 = Σ_{i=0}^{k−1} d_e(p_i, p_{i+1});
11 end
12 Return < p_0, p_1, ..., p_{k−1}, p_k >;
```



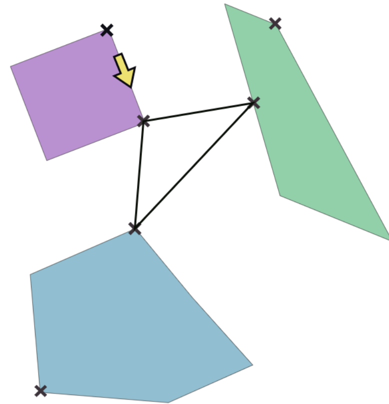
(a) : The tour is initialized by choosing a random vertex of each polygon and the length of the current tour L_0 is computed.



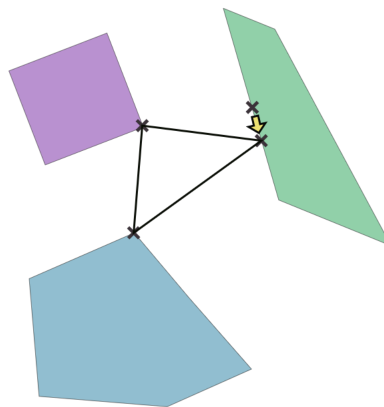
(b) : The first point is optimized by Algorithm 3 in the first iteration of the inner loop on the line 5.



(c) : The second point is optimized.



(d) : The final point was optimized. The algorithm now checks, whether the length of the new tour L_1 is smaller than L_0 by at least ϵ.



(e) : The second iteration of the main loop improves only one more point. The third iteration will be the last one, as it won't be able to shorten the tour anymore.

Figure 3.1: Illustration of the floating TPP for convex polygons (Algorithm 1).

The Algorithm 1 takes a sequence of pairwise disjoint simple polygons P and an accuracy constant ϵ as an input and returns an approximate shortest path consisting of a sequence of points that tour the polygons in the given order. At the beginning of the algorithm, a random vertex of each polygon is selected to form the initial solution (1) and the lengths L_0 and L_1 are set (lines 2 and 3). L_0 represents the path length of the points from the previous iteration, while L_1 is the path length of the current iteration. The main cycle of the algorithm (line 4) terminates when L_1 was shortened by less than ϵ with respect to L_0. The inner cycle (line 5) goes through the current tour from the beginning and for each point p_i finds a point $q_i \in \vartheta P_i$ on the frontier of the polygon P_i that minimizes the distance $d_e(p_{i-1}, q_i) + d_e(q_i, p_{i+1})$. In other words, the inner cycle replaces each point p_i of the tour with a point on the frontier of its parent polygon P_i that minimizes the distance to the preceding p_{i-1} and succeeding p_{i+1} point in the tour. This is accomplished by Algorithm 3.

Algorithm 2: Floating TPP algorithm for not necessarily convex polygons

Data: A sequence of k simple polygons $P_0, P_1, \dots, P_{k-1}$ such that convex hulls $C(P_0), C(P_1), ..., C(P_{k-1})$ are pairwise disjoint and an accuracy constant $\epsilon > 0$

Result: A sequence $< p_0, p_1, \dots, p_k >$ (where $p_k = p_0$) which visits polygon P_i at p_i in the given order $i = 0, 1, \dots, k-1$, and finally $i = 0$ again

1. For $i \in \{0, 1, \dots, k-1\}$, apply the Melkman algorithm for computing $C(P_i)$;
2. Let $C(P_0), C(P_1), ..., C(P_{k-1})$ be the input of Algorithm 1 for computing an approximate shortest route $< p_0, p_1, \dots, p_{k-1}, p_k >$;
3. For $i = 0, 1, \dots, k-1$ use Algorithm 3 with points p_{i-1}, p_{i+1} and polygon P_i as arguments to find a point $q_i \in \vartheta P_i$. Update p_i by letting p_i be q_i;
4. Let $P_0, P_1, \dots, P_{k-1}$ be the input of Algorithm 1, and points p_i as obtained from line 3 be the initial points in line 1 of Algorithm 1 for computing an approximate shortest route $< p_0, p_1, \dots, p_{k-1}, p_k >$;
5. Return $< p_0, p_1, \dots, p_{k-1}, p_k >$;

Algorithm 2 starts by computing a convex hull $C(P)$ of each input polygon (line 1). These convex hulls are then used as the input for Algorithm 1 to calculate an approximate shortest route consisting of the points p (line 2). Because the points p are on the frontier of the convex hulls $C(P)$ and therefore some point p_i could be outside of P_i, the inner cycle of Algorithm 1 is used once (line 3) to get the points p back within the frontier of P. Algorithm 1 is then called again using the points p instead of the initial points to get the final approximate shortest route.

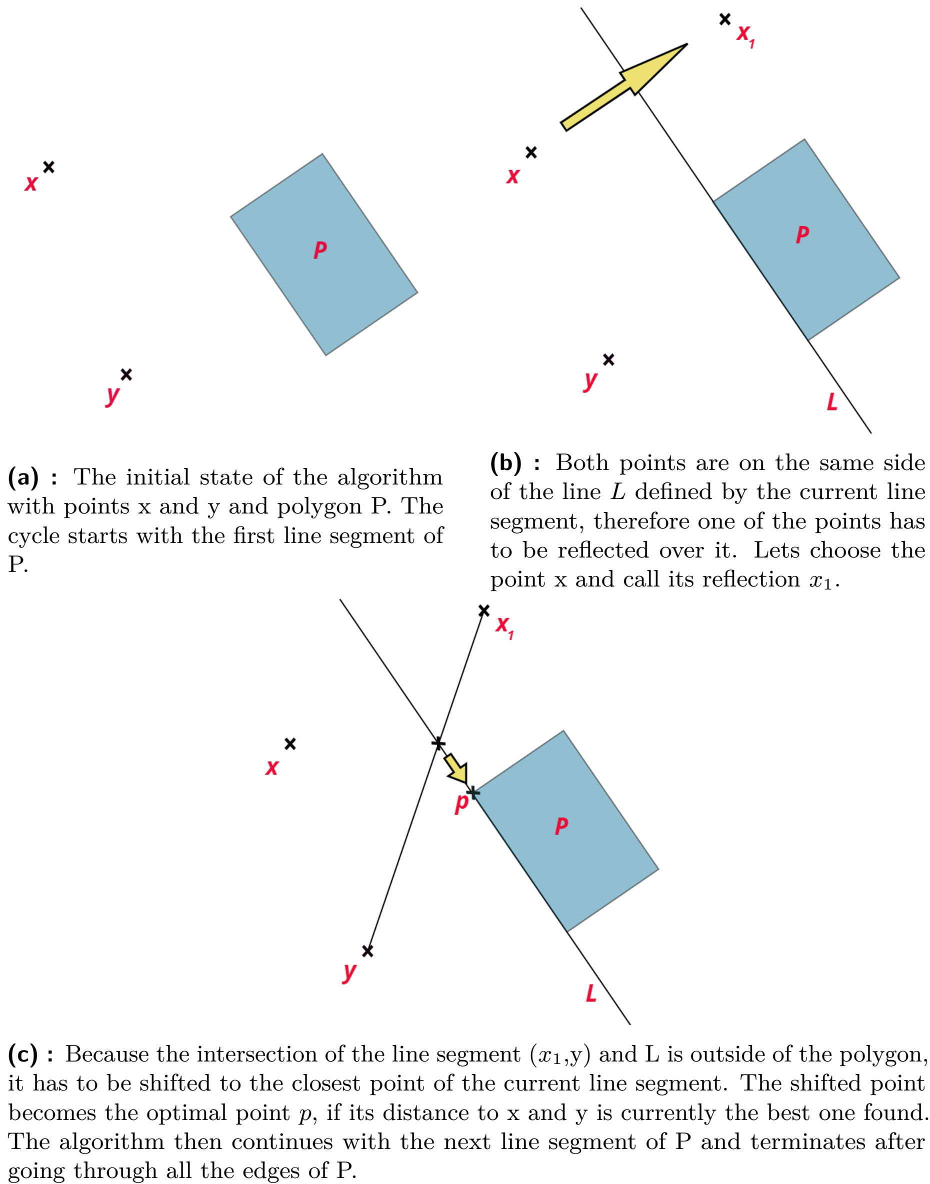
Algorithm 2 provides an $(1 + (L_2 - L_1)/L)$ - approximate solution for the floating TPP problem, where the input polygons P are not necessarily convex.

Here L means the length of the optimal path, L_1 is the length of the path obtained in line 2 and L_2 is the length of the final path in line 4. Time complexity in practice, as shown in the original paper [10], has an upper bound of $\mathcal{O}(n^2)$.

Algorithm 3: Optimal connecting point on the frontier of a polygon

```
   Data: Point x, Point y, Polygon P
   Result: Optimal point p in P, such that min{dist(p,x) + dist(p,y)}
1  Point p;
2  minDist = ∞;
3  k = number of vertices in P;
4  for i = 0 to k do
5  |   Line L;
6  |   if i < k - 1 then
7  |   |   L is defined by the line segment L_S = (P[i], P[i+1]);
8  |   end
9  |   else
10 |   |   L is defined by the line segment L_S = (P[i], P[0]);
11 |   end
12 |   if Point x and y on the same side of line L then
13 |   |   Reflect either x or y over the line L;
14 |   end
15 |   Find the intersecting point j of the line segment (x, y) and line L;
16 |   if j ∉ L_S then
17 |   |   Shift j to the boundary point of L_S that is closest to j;
18 |   end
19 |   tmpDist = dist(x,j) + dist(y,j);
20 |   if tmpDist < minDist then
21 |   |   minDist = tmpDist;
22 |   |   p = j;
23 |   end
24 end
25 return p;
```

Algorithm 3 finds the point p on the frontier of the input polygon P, that minimizes the distance to the input points x and y. The main loop (line 4) iterates over the line segments L_S of P and for each one of them computes a point j_i closest to x and y, such that $j_i \in L_S$. The point j_i with minimum distance $d = dist(x, j_i) + dist(y, j_i)$ is selected as p (line 22) and returned as the output of the algorithm (line 25). The point j_i is found by expanding L_S into a line L and locating the intersection of the line segment (x, y) and L (line 15). If both x and y are on the same side of L, one of them has to be first reflected over L (line 13). If the intersection is inside L_S, it becomes j_i. Otherwise j_i is one of the two boundary points of L_S closest to the intersection (line 17). An example of an iteration of the main loop, where the reflection is needed, is displayed in Figure 3.2.



(a) : The initial state of the algorithm with points x and y and polygon P. The cycle starts with the first line segment of P.

(b) : Both points are on the same side of the line L defined by the current line segment, therefore one of the points has to be reflected over it. Lets choose the point x and call its reflection x_1.

(c) : Because the intersection of the line segment (x_1,y) and L is outside of the polygon, it has to be shifted to the closest point of the current line segment. The shifted point becomes the optimal point p, if its distance to x and y is currently the best one found. The algorithm then continues with the next line segment of P and terminates after going through all the edges of P.

Figure 3.2: Finding the closest point on a line segment in regards to the points x and y (Algorithm 3).

3.2 GLNS algorithm

GLNS is a solver for the exactly one-in-a-set Generalized Travelling Salesman Problem (GTSP) as defined in Section 2.1. It currently belongs to the state-of-the-art heuristics for solving GTSP together with the GLKH [5] and GK [6] solvers. GLNS was introduced in paper [4] together with experimental results showing that it is competitive with the best known algorithms, often finding better solutions given the same amount of time.

The general framework under which GLNS operates is called Adaptive Large Neighborhood Search (ALNS). ALNS was first introduced in [8] and

subsequently defined more generally in [9]. The main idea of ALNS and therefore GLNS is simple. First, an initial solution is found and then it is iteratively destroyed and repaired. If a better solution than the current best one is found, it is accepted. The search is finished when a terminating condition is met. ALNS uses two different sets of heuristics for removals and insertions of the solution. Each heuristic holds a weight representing how successful the heuristic is in improving the solution and this weight is updated as the algorithm runs, thus the word adaptive in ALNS. The more successful heuristics are selected more often then the less successful ones. GLNS also regularly locally optimizes the solution.

The following sections give a simplified description of the original GLNS algorithm that is needed to understand the changes that were made in the algorithms implemented in this thesis. More information can be found in the original paper [4]. An emphasis is placed upon the description of the insertion and removal heuristics and also local optimizations, as these methods were modified in Algorithm 3.3.

3.2.1 Solver framework

This section offers a pseudocode and high-level description of the GLNS algorithm.

The GLNS algorithm, shown in pseudocode in Algorithm 4 works in iterations. Each iteration i starts by constructing a random initial tour T (line 2) and making it a baseline best tour $T_{best,i}$ for this iteration (line 3). After that, the following cycle follows (line 4). A single removal heuristic R and insertion heuristic I are selected first, according to their current weight (line 5). Next, a number N_r is uniformly randomly selected from between $1, \ldots, N_{max}$ (where N_{max} is a parameter of the algorithm), marking a number of vertices that are going to be removed (line 6). After creating a copy T_{new} of the current tour (line 7), we remove N_r vertices from T_{new} using the selected removal heuristic R (line 8). Then, using the selected insertion heuristic I, we insert a vertex into T_{new} for each removed vertex, so that the tour T_{new} visits all sets again (line 9). The tour T_{new} is then locally optimized using methods the paper called ReOpt and MoveOpt. These two methods try to re-optimize the order of the sets and the vertex in each set (line 10). If the length of the tour T_{new} is smaller than the current best tour of the iteration $T_{best,i}$, then T_{new} becomes $T_{best,i}$ (line 11). The tour T_{new} is accepted or declined as a new tour T for this iteration based on the standard simulated annealing criterion (line 14). The stopping criteria of each iteration have two phases - initial descent and several warm restarts (line 18). The first phase, the initial descent, terminates after a fixed amount of non-improving iterations. In the second phase, each warm restart starts with the best solution found, but a lower simulated annealing temperature. Each warm restart ends after a several non-improving iterations.

Algorithm 4: GLNS(G,P_v)

Data: A GTSP instance (G, P_v)
Result: A GTSP tour on G

```
1  for i = 1 to num_trials do
2      T ← initial_tour(G, P_v);
3      T_best,i ← T;
4      repeat
5          Select a removal heuristic R and insertion heuristic I using the
             selection weights;
6          Select the number of vertices to remove, N_r, uniformly
             randomly from 1, ..., N_max;
7          Create a copy of T called T_new;
8          Remove N_r vertices from T_new using R;
9          For each of the N_r sets not visited by T_new, insert a vertex
             into T_new using I;
10         Locally re-optimize T_new;
11         if w(T_new) < w(T_best,i) then
12             T_best,i ← T_new;
13         end
14         if accept(T_new, T) then
15             T ← T_new;
16             Record improvement made by R and I;
17         end
18     until stop criterion is met;
19     Update selection weights based on improvements of each heuristic
         over trial;
20 end
```

3.2.2 Insertion heuristics

The four insertion heuristics that GLNS uses are described in this section - nearest, farthest, random and cheapest insertion.

Each of these heuristics takes as an input a GTSP instance (G, P_V) and a partial tour $T = (V_T, E_T)$ of graph $G = (V, E, w)$. $P_V = \{V_1, \ldots, V_m\}$ is a partition of V. The partial tour T is a cycle in G such that each set in the partition P_V is visited at most once. The sets that are visited by the partial tour T are denoted $P_T \subseteq P_V$. The insertion heuristic then takes this partial tour T, inserts a vertex from an unvisited set into it (according to the heuristic-specific rules) and returns a tour that visits one more set than the input tour.

The heuristics use the set-vertex distance for their decision making about which set to insert (V_i in $P_V \backslash P_T$). The set-vertex distance is computed for each set V_i, $i \in \{1, \ldots, m\}$ and vertex $u \in V \backslash V_i$ at the beginning of the algorithm and is defined as:

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

Each heuristic has a specific way it selects the set to visit according to this distance. They work as follows:

I. **Nearest Insertion** selects the set V_i, that contains a vertex v which is at a minimum distance to any vertex on the partial tour T.

$$argmin\ min\ dist(V_i, v); V_i \in P_V \backslash P_T, v \in V_T$$

II. **Farthest Insertion** selects the set V_i, that contains a vertex v which is at a maximum distance to any vertex on the partial tour T.

$$argmax\ min\ dist(V_i, v); V_i \in P_V \backslash P_T, v \in V_T$$

III. **Random Insertion** selects the set V_i uniformly randomly from $P_V \backslash P_T$.

IV. **Cheapest Insertion** selects the set V_i that contains a vertex v that minimizes the insertion cost.

$$argmin\ min\{w(x, v)+w(v, y)-w(x, y)\}; V_i \in P_V \backslash P_T, v \in V_i, (x, y) \in E_T$$

3.2.3 Bounding insertions before evaluation

GLNS uses a bounding technique for speeding up insertions. It uses precomputed distances $dist(V_i, u)$ between each $V_i \in P_V \backslash P_T$ and each vertex $u \in V_T$. Prior to checking the insertion cost for each vertex $v \in V_i$ in each edge $(x, y) \in E_T$ the lower bound is computed

$$lb = dist(V_i, x) + dist(V_i, y) - w(x, y).$$

If lb is greater or equal to the minimum insertion cost for a vertex $v \in V_i$ so far, then the insertion costs for edge (x, y) do not have to be computed as they cannot be smaller then the lower bound lb.

3.2.4 Removal heuristics

GLNS uses three different removal heuristics - worst, distance and segment removal. Each of these heuristics removes N_r vertices from a tour $T = (V_T, E_T)$ using a heuristic-specific strategy.

Worst Removal

The goal of the worst removal heuristic is to remove the vertex, that results in the biggest reduction in the length of the tour. Given a partial tour T we remove the vertex v_j that maximizes the removal cost

$$r_j = w(v_{j-1}, v_j) + w(v_j, v_{j+1}) - w(v_{j-1}, v_{j+1}).$$

Distance Removal

Distance removal aims to remove vertices that are close to each other. Given a complete tour T a random vertex is removed from T and added to a set $V_{removed}$. Then an iteration follows, where a seed vertex v_{seed} is uniformly randomly selected from $V_{removed}$ and for each $v_j \in V_T$ the removal cost r_j is computed as

$$r_j = min\{w(v_{seed}, v_j), w(v_j, v_{seed})\}.$$

The vertex with minimal r_j is removed. This process is repeated until N_r number of vertices is removed.

Segment Removal

Segment removal removes a continuous segment of the size N_r from the tour T. Given a complete tour T with vertices $V_T = \{v_1, \ldots, v_m\}$, a vertex v_j is uniformly randomly selected and then the vertices $v_j, v_{j+1}, \ldots, v_{j+N_r-1}$ are removed from the tour T.

3.2.5 Local optimizations

Prior to evaluating the acceptance condition in an iteration of GLNS, the new tour is locally optimized. Two techniques are used for this optimization and they are called ReOpt and MoveOpt.

ReOpt

The goal of ReOpt is to optimize the choice of a vertex in each set, keeping the set ordering fixed. This is achieved by creating a directed acyclic graph (DAG) containing all vertices of the GTSP graph. The shortest path between each vertex of the first set of the tour and its copy is then computed in this DAG and the minimal shortest path becomes the optimized tour.

MoveOpt

MoveOpt is a special case of GLNS where only one set is removed and then reinserted again with a minimum insertion cost. Given a complete tour $T = (V_T, E_T)$ a random vertex v_j is selected and removed from the tour. A vertex from the same set V_j with a lowest insertion cost $w(x, u) + w(u, y) - w(x, y)$ for all $u \in V_j$ and all $(x, y) \in E_T$ is reinserted afterwards. MoveOpt is repeated N_{move} (parameter of GLNS) times.

3.2.6 Modes of operation

GLNS has three general settings - fast, medium and slow. For detailed information please see the original paper. Generally, each of these settings dictates how many total iterations will happen through setting the stopping and acceptance criteria and the maximum number of vertices that can be

removed from the tour using the removal heuristics. The only important note in regards to this thesis is that the fast setting does not use the cheapest insertion heuristic and the ReOpt optimization.

3.2.7 Implementation details

GLNS was originally written in the Julia language. This thesis uses a C++ implementation of the algorithm that was created as a part of the thesis by David Woller [7].

3.3 GLNSC

GLNSC (GLNS for Continuous Neighborhoods) works by modifying the GLNS algorithm itself, but preserving the main structure and purpose of its operations. The modifications make use of the various calculations of analytical geometry, where for example the set-vertex distance is replaced by the point-to-polygon distance. The most notable changes were made in the insertion and removal heuristics and also in the local re-optimization methods. While GLNS works with sets consisting of discrete vertices whose distances can be precomputed, no such thing can be easily done in GLNSC as the points in the tour are constantly changing within their continuous regions (simple polygons), whenever a removal and insertion is done on the partial tour. This inability to precompute distances between the points significantly slows the algorithm down, as the point-to-polygon distances need to be constantly re-computed during the insertions and removals whenever a point in the tour shifts. What can be precomputed, though, are the polygon-to-polygon distances. These are then used for the lower bounding technique during insertions. The local re-optimization methods (MoveOpt and ReOpt) and cheapest insertion heuristic make use of Algorithm 2 for the floating Touring Polygons Problem described in Section 3.1, slightly modifying it to also work with intersecting polygons and thus allowing it to solve the $\mathrm{TSPN}_{intersect}$ problem instances on top of the $\mathrm{TSPN}_{disjoint}$ ones. While ReOpt is replaced by the entire algorithm for the floating TPP problem, the MoveOpt and cheapest insertion use just its subroutine. All modified operations are equivalent to the original GLNS with the exception of ReOpt, where it is only an approximation and not an exact solution. This is due to it being the floating TPP problem, which is NP-hard. All the modifications are explained in greater detail in the following sections.

3.3.1 Floating TPP modifications

The original implementation of the floating TPP algorithm is able to solve instances, where the convex hulls of the simple polygons in the tour are disjoint, thus it can be used for $\mathrm{TSPN}_{disjoint}$. If the convex hulls are not necessarily disjoint (as is the case of $\mathrm{TSPN}_{intersect}$), this algorithm has to be modified. The only change GLNSC applies in the intersecting case is

that Algorithm 2 now uses Algorithm 5 to find the optimal connecting point instead of Algorithm 3.

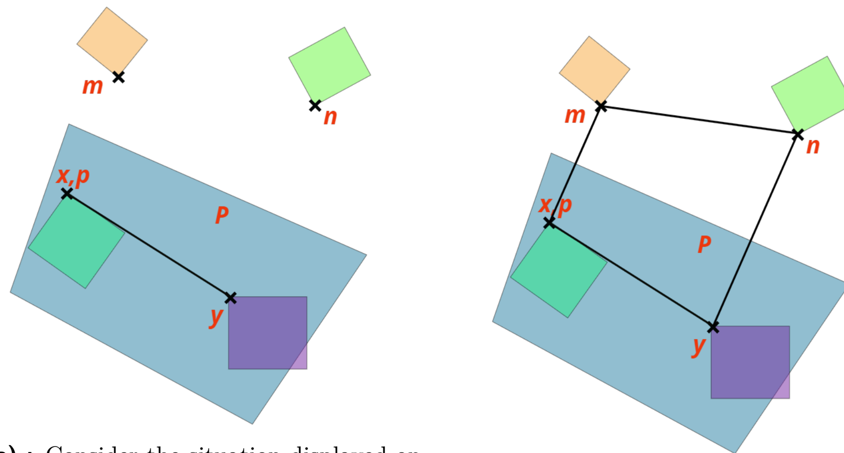
Algorithm 5: Optimal connecting point inside the polygon

Data: Point x, Point y, Polygon P

Result: Optimal point p in P, such that $min\{dist(p, x) + dist(p, y)\}$

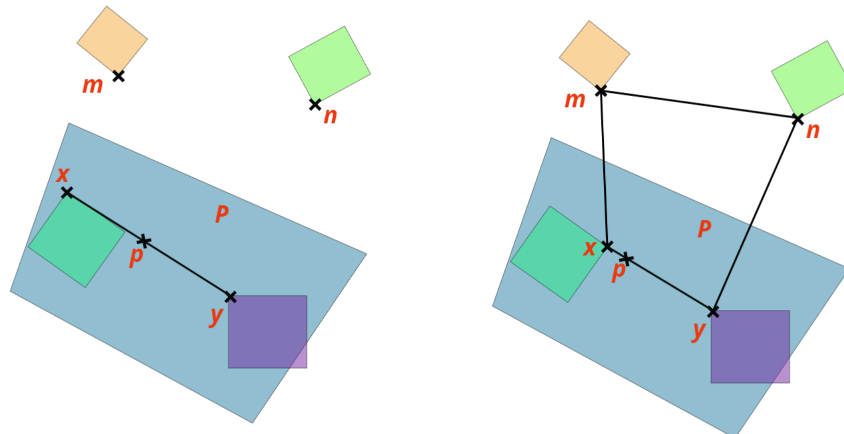
```
1 Point p;
2 if x ∈ P and y ∈ P then
3 |   p = point in the middle of the line segment (x, y);
4 |   if p ∈ P then
5 |   |   return p;
6 |   end
7 end
8 p is output of Algorithm 3 called with x,y and P as the arguments;
9 return p;
```

The ultimate reason why Algorithm 3 is used in the line 6 of the floating TPP algorithm is to find the point $p \in P$ that is closest to the points x and y. For TSPN$_{disjoint}$ the points x and y are guaranteed to be outside of P, therefore the point p must be on the frontier of P. In TSPN$_{intersect}$, in contrast, both x and y could be inside of P and in such case the optimal point p will lie on the line segment (x, y) connecting these two points and not on the frontier of P. That is why Algorithm 5 first checks, whether $x \in P$ and $y \in P$ (line 2). In case that both of them are inside of P, a point from the line segment (x, y) is chosen as p. Although any point from (x, y) could be taken, choosing the point in the middle returned noticeably better solutions than the points closer to the boundary or the boundary points x and y themselves. The reason for this is apparent, if for example the point x is selected as p, the next iteration will possibly not be forced to improve this point as the points p, x and the point preceding x lie on a single line segment. See Figure 3.3 for an example, where choosing the boundary point as p brings a worse final solution than the point in the middle. Finally, as the point p is the point in the middle of (x, y), it could be out of the bounds of its parent polygon, since the polygons in the TSPN$_{intersect}$ instance are not guaranteed to be convex. Therefore the line 4 checks if $p \in P$. If not, there has to be an intersection of (x, y) with the frontier of the polygon P and therefore the original Algorithm 3 can be used to find the optimal point p. Figure 3.4 provides an example of this situation.



(a) : Consider the situation displayed on the picture above. Because $x \in P$ and $y \in P$, the optimal connecting point in polygon P lies on the line segment (x, y). Algorithm 5 selects the point x as p.

(b) : TPP is unable to improve neither x nor p and has to terminate, returning the tour <m,x,p,y,n> as the shortest one.



(c) : Now consider the situation where the optimal point p was selected as the point in the middle of the line segment (x, y) by Algorithm 5.

(d) : In this case the next iteration of TPP was forced to improve the tour further.

Figure 3.3: Comparison of choosing the boundary point and the point in the middle in Algorithm 5.

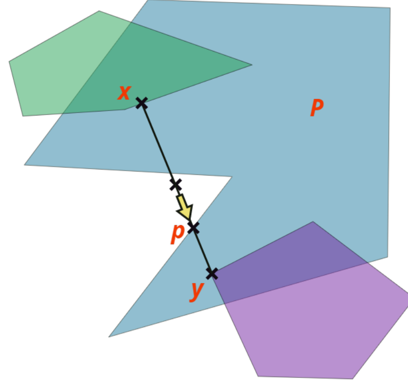


Figure 3.4: $x \in P$ and $y \in P$, therefore Algorithm 5 selects the point in the middle of (x, y). But P is not convex and such point is outside of its bounds. The optimal point p is on the frontier of P and therefore can be found by Algorithm 3.

3.3.2 GLNS modifications

Insertion heuristics

The insertion heuristics work as follows in GLNSC:

I. **Nearest Insertion** selects the polygon V_i, that contains a point v which is at a minimum distance to any point on the partial tour T.

$$argmin\ min\ dist(V_i, u); V_i \in P_V \backslash P_T, u \in V_T$$

II. **Farthest Insertion** selects the polygon V_i, that contains a point v which is at a maximum distance to any point on the partial tour T.

$$argmax\ min\ dist(V_i, u); V_i \in P_V \backslash P_T, u \in V_T$$

III. **Random Insertion** selects the polygon V_i uniformly randomly between the polygons that are not visited by partial tour T.

IV. **Cheapest Insertion** selects the polygon V_i that contains a point v that minimizes the insertion cost.

$$argmin\ min\{w(x, v) + w(v, y) - w(x, y)\}; V_i \in P_V \backslash P_T, v \in V_i, (x, y) \in E_T$$

As can be seen, the difference between the insertion heuristics of GLNS and GLNSC is only in semantics, the sets are polygons and vertices are points. What is different in GLNSC is how these individual heuristics calculate the distances and costs to find the minima and maxima to accomplish the same goal as their GLNS counterparts.

The first modification is to the set-vertex distance that the original GLNS precomputes for each vertex and set at the beginning of the algorithm and then

uses during insertions to choose a nearest, farthest and random set. No such thing can be done in GLNSC as the pool of the possible points for the partial tour T is infinite and not known at the beginning of the algorithm. This set-vertex distance is therefore replaced by a method calculating the minimum point-to-polygon distance. The point-to-polygon distance is determined by going through the line segments of the polygon, computing the point-to-segment distance and choosing the shortest one.

The second modification is to the cheapest insertion. The original algorithm takes the sets S that are not visited by the partial tour T and tries to place them between all the pairs of subsequent points $x \in T$, $y \in T$, calculating the insertion cost $\{w(x,v)+w(v,y)-w(x,y)\}$, where $v \in S_i$ such that the sum of distances of v to x and v to y is minimal. The set S_i with minimum insertion cost is selected. Choosing the optimal point v in GLNSC is more difficult than in original GLNS, because now there is an infinite number of potential points p in polygon V_i. The modification is that the cheapest insertion in GLNSC chooses the optimal point p in polygon V_i using Algorithm 5.

Removal heuristics

The differences in removal heuristics in GLNSC are again only in semantics, the sets are now polygons and the vertices are points. The only difference is in distance removal where the removal cost is now calculated as $r_j = w(v_{seed}, v_j)$ instead of $r_j = min\{w(v_{seed}, v_j), w(v_j, v_{seed})\}$, as the distance between the two points is the same.

Bounding insertions

Originally, this method used precomputed minimal set-vertex distances. This is equivalent to the minimal point-to-polygon distance, which, as was said before, can't be precomputed. Due to the point-to-polygon calculation being very computationally expensive, the lower bounding would instead of speeding up the insertions make them slower and the original point of lower bounding would be lost. Instead, the point-to-polygon distance is replaced by the polygon-to-polygon distance, which can be precomputed, because the polygons are known at the beginning of the algorithm. It should be noted, that this lower bound is generally larger than in the original GLNS and therefore allows more iterations that are not perspective through. Nevertheless, it is still beneficial as it trims the search space and avoids some unnecessary calculations of the optimal point in polygon, which is also expensive.

Prior to calculating the optimal point $v \in V_i$ (with respect to the points x,y in edge $(x, y) \in E_T$), $V_i \in P_V \setminus P_T$, and checking the insertion cost for each edge $(x, y) \in E_T$ the lower bound is computed

$$lb = dist(V_i, V_x) + dist(V_i, V_y) - w(x, y).$$

If lb is greater or equal to the minimum insertion cost for a point $v_i \in V_i$ so far, then the optimal point v and the insertion cost for edge (x, y) does not have to be computed as it cannot be smaller then the lower bound lb.

ReOpt

GLNS uses shortest path algorithm for DAG to calculate the exact solution, where the optimal point is chosen in each set to reduce the total length of the tour. ReOpt in TSPN$_{disjoint}$ and TSPN$_{intersect}$ problem instances and a complete tour T visiting the polygons in a given order equals to the NP-hard floating TPP problem defined in Section 2.3. Therefore, Algorithm 2 for solving floating TPP can be used, but with minor changes described in Section 3.3.1. The floating TPP algorithm is only an approximation, therefore ReOpt in GLNSC does not give the exact solution, unlike its GLNS counterpart.

As was said in Section 3.2.6, the fast setting does not use ReOpt in the original GLNS. In contrast, GLNSC runs it on each setting at the end of the algorithm to improve the best tour that was found.

MoveOpt

Given a complete tour $T = (V_T, E_T)$ a random point $v_j \in V_j$ is selected, removed from the tour, and then for each $(x, y) \in E_T$ an optimal connecting point u is found using the procedure described in Algorithm 5. The polygon V_j is then reinserted with the minimum insertion cost $w(x, u) + w(u, y) - w(x, y)$. MoveOpt is repeated N_{move} times.

3.3.3 Implementation change of insertions

After analyzing the runtime of GLNSC in a profiler, it turned out that the nearest and farthest insertions are a major bottleneck. It is due to the point-to-polygon computation being expensive operation that can't be precomputed. The following reformulation of the mentioned insertions allows the algorithm to use precomputed polygon-to-polygon distances and thus speeding the algorithm up significantly without sacrificing the quality of the solution in a meaningful way. Nevertheless, this change is a slight diversion from how the original GLNS works. This change will be analyzed in terms of the time improvement and solution quality in Section 4.2.1.

I. **Nearest Insertion** selects the polygon V_i, that contains a point v which is at a minimum distance to any polygon on the partial tour T.

$$argmin\ min\ dist(V_i, V_u); V_i \in P_V \backslash P_T, V_u \in V_T$$

II. **Farthest Insertion** selects the polygon V_i, that contains a point v which is at a maximum distance to any polygon on the partial tour T.

$$argmax\ min\ dist(V_i, V_u); V_i \in P_V \backslash P_T, V_u \in V_T$$

3.3.4 Parameters

Table 3.1 contains the parameters (aside from the map instance) that need to be set to run GLNSC.

Parameter	Values	Description
GLNS mode	Fast|Medium|Slow	Mode of operation of GLNS
ϵ	>0	Parameter of TPP (ReOpt).

Table 3.1: GLNSC parameters

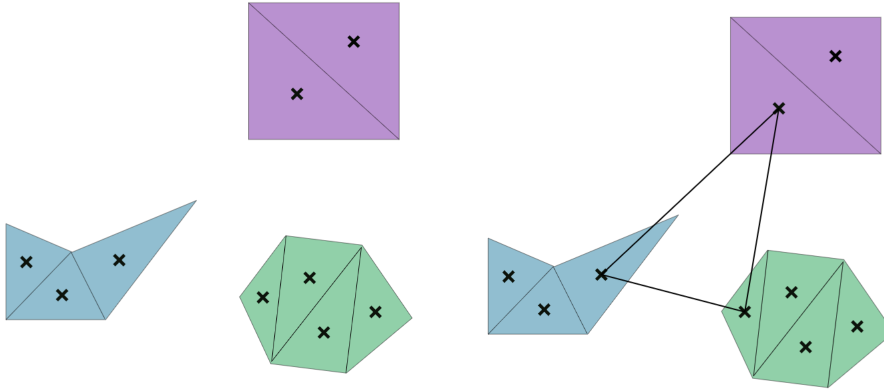
3.4 GLNS-TPP

GLNS-TPP solves all three problems defined in Section 2.4. While GLNSC modifies the GLNS methods with geometric operations such as the point-to-polygon distance, GLNS-TPP preserves the GLNS algorithm almost entirely. This is made possible by making the instance discrete by transforming each polygon into a triangular mesh and using a center point of each generated triangle as the input for GLNS. Each polygon is then represented by a set of points. It should be noted that the problem (aside from the obstacles in TSPN$_{obstacles}$) is thus converted into a GTSP instance instead of TSPN.

Lets first describe the inner working of the algorithm for an instance without obstacles. GLNS-TPP works in two phases. The first phase starts by creating a triangular mesh of each polygon, pre-computing the all-to-all distances between the center points of the triangles and running GLNS on these points. The solution that GLNS returns is a sequence of individual points, each representing a polygon of the instance, but this solution is of course not optimal for TSPN. Though it gives a good approximation of what the optimal order of the polygons in the tour should be. The second phase makes use of this order and employs the modified floating TPP algorithm described in Section 3.3.1, giving the output points and the order of polygons from the first phase as the input. The TPP algorithm's output in the second phase is then the final solution. A general overview of GLNS-TPP can be seen in Figure 3.5. These changes allow GLNS-TPP to solve the TSPN$_{disjoint}$ and TSPN$_{intersect}$ problem instances.

The general structure is the same for the case with polygonal obstacles as for an instance without obstacles. The algorithm still runs in two phases, but slight modifications have to be made to account for obstacles. These modifications utilize a data structure called visibility graph [12], that is filled with the environment obstacles and bounds and after that it is capable of calculating the shortest path distance between any points within the bounds of this environment. GLNS algorithm stays again intact thanks to the discrete space GLNS-TPP creates by splitting the points into a mesh. The pre-computation in the first phase calculates, instead of simple point-to-point distances, the all-to-all shortest paths using the Floyd-Warshall algorithm [13]. The second phase, the algorithm for the floating TPP, needs two additional changes from those described in Section 3.3.1. The first modification is rather straightforward. The calculation of the point-to-point distance is replaced by the shortest path utilizing the visibility graph in the same way as in the first phase. The second modification updates Algorithm 5 to account for the obstacles. More details will be presented in Section 3.4.2. Thanks to these

changes is GLNS-TPP then able to solve the TSPN$_{obstacles}$ problem instances.



(a) : First, the triangular mesh is created from the input polygons, the points in their center selected and the distances between them precomputed.

(b) : Then GLNS finds the approximate shortest GTSP tour among the sets (polygons) and their points.

(c) : Finally, TPP takes the tour found by GLNS and tries to optimize its points.

Figure 3.5: General steps of GLNS-TPP.

3.4.1 GLNS modification

This section describes the changes made to the GLNS algorithm. The changes will be divided by the problem type - TSPN$_{disjoint}$, TSPN$_{intersect}$ and TSPN$_{obstacles}$. All three problem types share the same modifications, only TSPN$_{obstacles}$ has additional changes on top.

TSPN$_{disjoint}$ and TSPN$_{intersect}$

GLNS-TPP transforms the polygons into a triangular mesh before the GLNS algorithm starts. The density of the mesh is controlled by a parameter ϕ. More triangles means more input points for GLNS, thus making it run slower, but with more accurate final solution. The point in the center of each triangle

is then taken and used as the input for GLNS.

TSPN$_{obstacles}$

After the polygons are split, the point-to-point distances need to be precomputed for each pair of the points. In TSPN$_{obstacles}$ the simple Euclidean distance between two points, which is used in the other two problem types, can't be used here due to the potential obstacle that might be between them. The solution for this is a data structure called visibility graph, that allows computation of the shortest path between any two points with respect to the environment obstacles. The precomputation therefore, instead of the Euclidean point-to-point distances, calculates the shortest paths between all pairs of points. GLNS-TPP uses the all-pairs shortest path Floyd-Warshall algorithm to accomplish this. GLNS is then able to run its course. The obstacles are accounted for in the precomputed point-to-point distances and GLNS does not need to take them into consideration anymore.

3.4.2 TPP modifications

After GLNS finishes, GLNS-TPP runs the floating TPP solver using the order of the polygons in the tour found by GLNS. Although Algorithm 2 takes only the order of the polygons as an input, it proved to provide the same or better results when the points of the tour were used as the input points on line 4 of Algorithm 2, skipping the first 3 lines of the algorithm entirely. The reasoning behind this change is simple. The output from GLNS is already a rough estimate of what the tour should be like, it just needs to be tightened. There is no need to go back to square one by trying to find the initial points on the convex hulls. The experimental results of this change will be shown in Section 4.3.1. The description of the TPP modifications by the problem type follows.

If the problem type is TSPN$_{disjoint}$, GLNS-TPP can just use Algorithm 2 for the floating TPP computation (without the first 3 lines as per the previous paragraph), as the convex hulls of the input polygons do not intersect.

If the problem type is TSPN$_{intersect}$, it also uses Algorithm 2 without the first 3 lines, but with a slight modification described in Section 3.3.1.

If the problem type is TSPN$_{obstacles}$, the situation becomes more complex. It is no longer correct to use Algorithm 5 in Algorithm 1 in the way it was used before. When looking for the optimal point $p \in P$ in regards to the points x and y, it is necessary to take obstacles into consideration. The polygon P might be possibly entirely hidden from the view of the points x and y. The idea behind the modification in such case is to find the vicarious points j and k on the obstacle(s) for both x and y and then calculating the optimal point p in regards to j and k instead. The pseudocode of the modified algorithm is shown in Algorithm 6, while Figure 3.6 illustrates its idea. It works by going through the vertices $v_i \in P$ and looking for the shortest polyline l in the visibility graph, that connects the point x to v_i and v_i to y. The points on such polyline directly before and after the point v_i are then used as the

vicarious points instead of x and y. If it is true for the polyline l, that its section from x to v_i or y to v_i is a straight line segment (no obstacle is in the way), than x or y are not replaced. After finding the vicarious points, the algorithm calls Algorithm 5 to find the optimal point on P between them. It should be noted, that while Algorithm 5 finds an exact solution, Algorithm 6 is just an approximation. See Figure 3.7 for an example of a solution that is not optimal. The exact algorithm would have an exponential time complexity, because it would need to try all the combinations of vertices on the obstacles (that are blocking the view for x or y) that have the view of the polygon P.

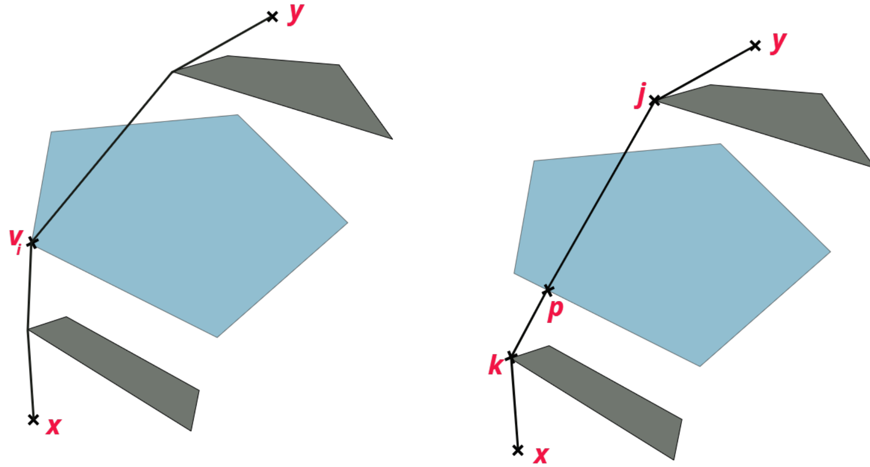
Algorithm 6: Optimal point in polygon through obstacles

```
   Data: Point x, Point y, Polygon P
   Result: Point o
 1 minLength = infinite;
 2 point j, k, o;
 3 for vertex v ∈ P do
 4 |   polyline L_x = shortest path from x to v in the visibility graph;
 5 |   polyline L_y = shortest path from y to v in the visibility graph;
 6 |   currentLength = length(L_x) + length(L_y);
 7 |   if currentLength < minLength then
 8 |   |   j = last point before v on the polyline L_x;
 9 |   |   k = last point before v on the polyline L_y;
10 |   |   minLength = currentLength;
11 |   end
12 end
13 Let j, k be the input points of Algorithm 5 and o be the output
     optimal point;
14 Return o;
```

3.4.3 Optimization of TPP on $\text{TSPN}_{obstacles}$

The main bottleneck of TPP in $\text{TSPN}_{obstacles}$ is Algorithm 5 that is being called by Algorithm 6, as it tries to find the optimal point on all edges of the input polygon. The distances have to be calculated using the shortest path in the visibility graph and that is an expensive operation.

The idea of the optimization is to work with the assumption, that the chosen best vertex of the polygon in Algorithm 6 is most likely around the area of the optimal point that is then supposed to be found by Algorithm 5. By passing this vertex to Algorithm 5 it is then possible to try only the edge preceding and succeeding this vertex in the input polygon for the selection of the optimal point. As can be seen in Figure 3.6, the optimized Algorithm 5 would look for the optimal point p on the neighboring edges of the vertex v_i and ignored the others. This change will be analyzed in Section 4.3.2.



(a) : First, the shortest polyline connecting a vertex v_i and the points x and y is found. The grey polygons are the obstacles.

(b) : Then the vicarious points j and k are used to find the optimal point p.

Figure 3.6: Process of finding the optimal point in the TSPN$_{obstacles}$ instances (Algorithm 6).

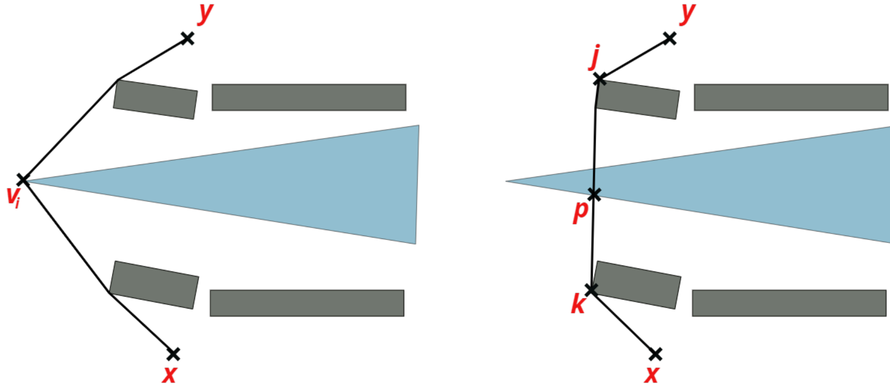
3.4.4 Possible future improvements

The creation of the triangular mesh and the subsequent selection of the points in their centers serve the purpose of an even discrete representation of the polygon. Given a polygon with a large number of edges this mesh consists of a large number of triangles. One of the speedup improvements would be an even pruning of the triangles to reduce the total number of generated points per polygon.

The library implementing the visibility graph this thesis uses does not employ the fastest possible algorithms. This has an impact on the initial precomputation phase and the TPP algorithm. More efficient alternatives are described in [12]. For the precomputation the Mitchell's algorithm [14] could be used, as the points for the shortest path queries are fixed (as opposed to TPP). The Mitchell's algorithm uses a data structure called the shortest path map (SPM). The construction of SPM has $O(n^{5/3+\epsilon})$ time complexity, whereas an efficient construction of the visibility graph has $O(n^2)$ time complexity. The subsequent shortest path queries over SPM then need $O(\log n)$ time, compared to $O(n^2)$ of the visibility graph.

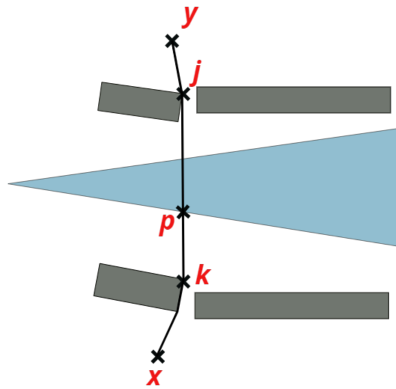
3.4.5 Parameters

Table 3.2 contains the parameters (aside from the map instance) that need to be set to run GLNS-TPP.



(a) : The shortest polyline connecting a vertex v_i and the points x and y is found.

(b) : Then the vicarious points j and k are used to find the point p. This solution is not optimal.



(c) : The optimal solution looks like this.

Figure 3.7: Example of a solution that is not optimal found by Algorithm 6.

<table>
<tr><th>Parameter</th><th>Values</th><th>Description</th></tr>
<tr><td>GLNS mode</td><td>Fast|Medium|Slow</td><td>Mode of operation of GLNS</td></tr>
<tr><td>ϵ</td><td>>0</td><td>Parameter of TPP.</td></tr>
<tr><td>ϕ</td><td>>0</td><td>Parameter of the algorithm generating the triangular mesh. Higher number means less triangles.</td></tr>
</table>

Table 3.2: GLNS-TPP parameters

Chapter 4
Experimental results

This chapter documents the performance of both GLNSC and GLNS-TPP on the problem instances of various sizes and types. It begins with an explanation of the statistical methods used to analyze the experimental results and the description of the created library of instances. Afterwards, each algorithm is examined individually. The final section comprises of the comparison of both algorithms on the problem instances that they are both able to solve.

4.1 Methodology

This section describes the instances and the methodology used to compare and analyze the results from the experiments.

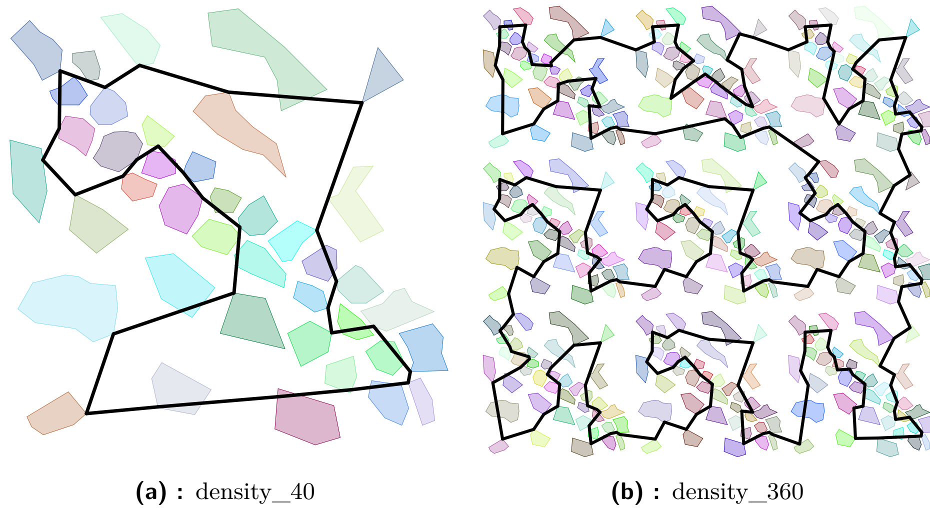
4.1.1 Instances

To test the performance of the implemented algorithms, a library of polygonal maps was created. For each problem type (TSPN$_{disjoint}$, TSPN$_{intersect}$ and TSPN$_{obstacles}$), a set of instances scaling in size is available. The instances always indicate the number of polygons they have at the end of their name, such as *density_400* with 400 polygons.

The instances whose name starts with *density_* and *ngons_* were all created from a base building block - their smallest instance (see Figure 4.1 and Figure 4.2). For example the base instance of *density_* is *density_40* and all the other instances are just increments of 40.

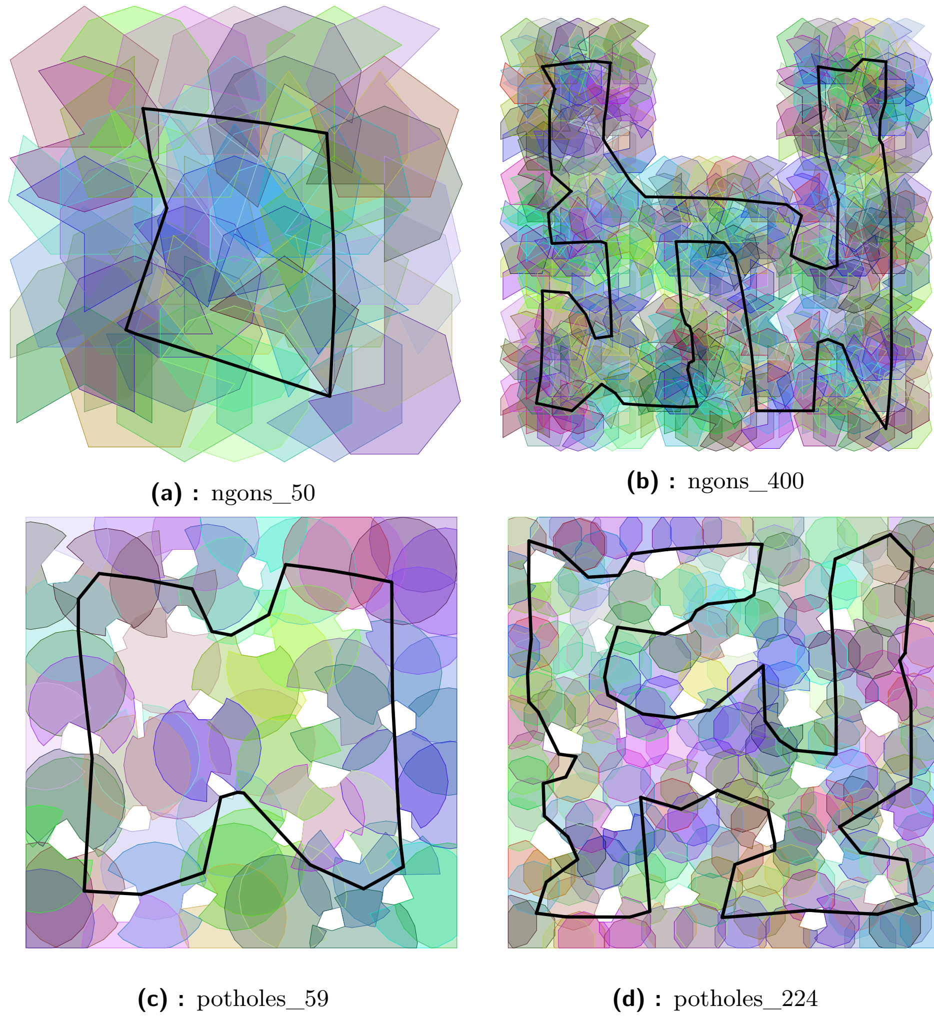
The instances starting with *potholes_* were created by having a map with 23 polygonal obstacles and generating the instance polygons around them to cover the map entirely. These instances are then used for both the TSPN$_{intersect}$ and TSPN$_{obstacles}$ problems, the obstacles are just not taken into consideration in the former. See *potholes_59* in Figure 4.2 (c) and *potholes_59* in Figure 4.3 (a).

The *obstacles_* instances (Figure 4.2) have a fixed number of polygons (32 to be specific) and a scaling number of obstacles. The number concluding the name now indicates the number of obstacles instead of polygons. They serve the purpose of testing the effect the number of obstacles has on the runtime in TSPN$_{obstacles}$.



(a) : density_40 **(b) :** density_360

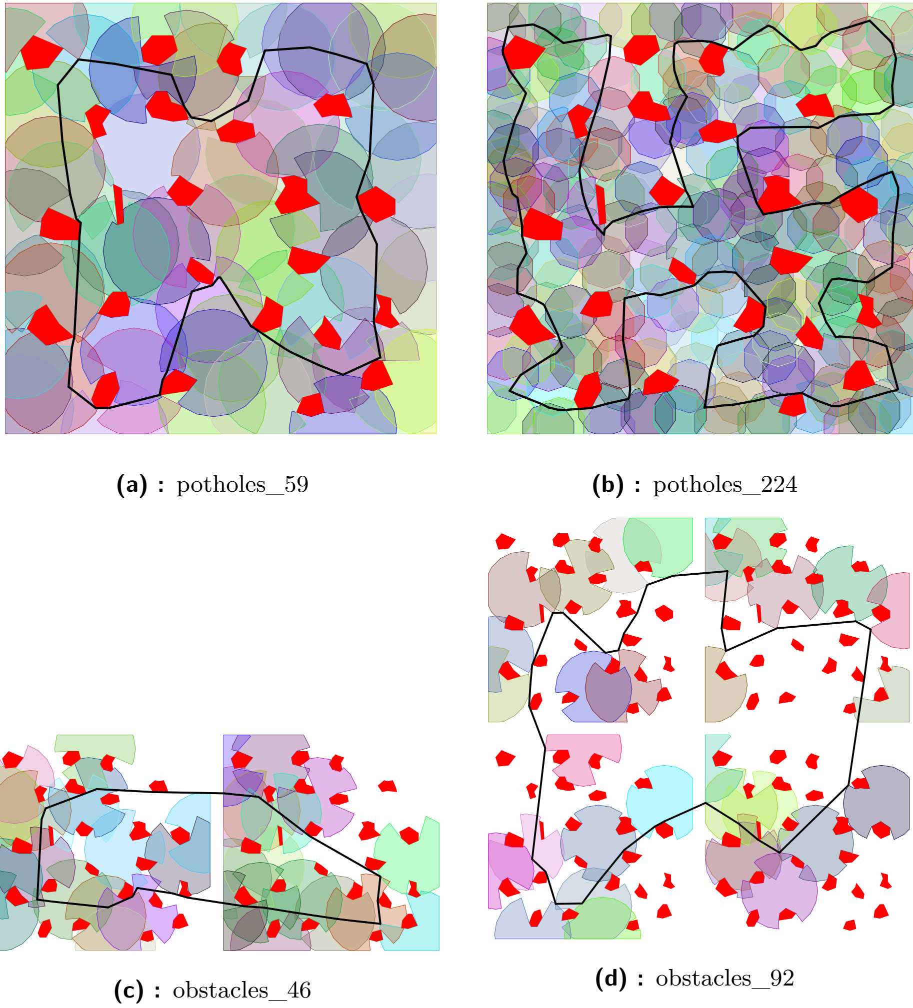
Figure 4.1: Examples of the solved TSPN$_{disjoint}$ instances.



(a) : ngons_50 **(b) :** ngons_400

(c) : potholes_59 **(d) :** potholes_224

Figure 4.2: Examples of the solved TSPN$_{intersect}$ instances.



(a) : potholes_59

(b) : potholes_224

(c) : obstacles_46

(d) : obstacles_92

Figure 4.3: Examples of the solved TSPN$_{obstacles}$ instances (red polygons represent the obstacles).

4.1.2 Measurements

Each algorithm was executed 50 times over each instance. The only exception are the instances consisting of 400 polygons as the time requirements would be too large, these instances are therefore solved only 5 times.

The data from the experiments were examined using the ministat tool [15]. The column with average in the tables means mean average and the data in this column are accompanied by the ± sign followed by the standard deviation σ. Whenever a speedup or solution cost comparison is made between the algorithms or the changes made to them, the 95% confidence interval is used to prove that there is a statistically significant difference. If there is a dash sign displayed in the comparison column, no statistically significant difference could be proved at 95% confidence. The numbers in the tables are always rounded to two decimal places, but the calculations use the original, not rounded, numbers.

For the experiments a machine with i7-3840QM CPU, 8GB RAM and Linux Ubuntu 18.04 OS was used.

4.2 GLNSC

The only feasible mode of GLNS is the fast mode. When trying to run the algorithm on the *potholes_12* instance with 12 polygons, the medium mode did not return a result even after 30 minutes. In contrast, the fast mode is able to solve this instance under a second. The medium and slow mode of GLNSC are therefore not considered in this chapter. Even though they would most likely return slightly better solutions, the time requirements are not worth it. For this reason the algorithm was executed in all the following sections in the fast GLNS setting. The ϵ for the ReOpt (TPP) method was always set to 0.01.

4.2.1 Insertion change

This section documents the results of the implementation changes proposed in Section 3.3.3. Table 4.1 contain comparison of the original and modified version of insertions in terms of the time performance and best tour solution cost found. Most of the instances that were used for this comparison are highly intersecting, as that is the type of the instance that should matter the most due to the nature of this change. The cost of the solution would need a more thorough analysis as it can't be concluded which version is better from the collected data. But the speedup is apparent and grows together with the size of the instance. Therefore this modification will be applied to GLNSC in all following sections of this chapter.

	Time (s)			Solution cost		
Problem name	Original insertions	Modified insertions	Speedup	Original insertions	Modified insertions	Improvement
potholes_12	0.58±0.09	0.39±0.05	32.47%±5.12%	19.97±6.29	20.14±6.12	–
potholes_24	2.86±0.38	1.31±0.17	54.23%±4.05%	63.29±0.88	63.42±1.13	–
potholes_33	7.35±1.04	2.55±0.30	65.32%±4.14%	75.84±1.56	74.76±1.01	1.44%±0.69%
potholes_59	54.77±7.85	11.32±1.41	79.34%±4.09%	104.56±0.66	104.67±0.78	–
ngons_50	10.65±1.40	3.71±0.59	65.16%±4.01%	4126.61±61.54	4140.96±106.11	–
ngons_100	154.48±23.66	34.56±5.15	77.63%±4.40%	8163.51±70.91	8188.57±81.38	–
density_80	55.67±6.88	13.59±1.43	75.58%±3.54%	13601.43±6.95	13601.07±7.05	–

Table 4.1: GLNSC insertion change time and solution cost comparison.

4.3 GLNS-TPP

Surprisingly, according to Table 4.2 the fast mode in GLNS-TPP returns overall better solutions than the other two modes, but the difference is only marginal. The difference would probably be most likely in favor of the slower modes on the bigger instances, but verifying this would take a lot of time as the slow mode's time requirements scale very fast with the instance size (see Figure 4.4). The other modes were therefore not considered in the rest of this chapter and GLNS-TPP was always executed in the fast GLNS setting.

Same as GLNSC, the parameter ϵ was set to 0.01 for TPP and ϕ was set to 10^6 to create the minimum possible amount of triangles.

	Solution cost by GLNS mode				
	Fast	Medium		Slow	
Problem name	Average	Average	Improvement (over Fast)	Average	Improvement (over Fast)
potholes_12	9.16±0.22	9.22±0.01	−0.68%±0.66%	9.23±0.01	−0.70%±0.66%
potholes_24	62.81±0.08	62.79±0.05	–	62.89±0.13	−0.13%±0.07%
potholes_33	74.15±0.45	74.46±0.38	−0.41%±0.22%	74.41±0.38	−0.34%±0.22%
potholes_59	104.11±0.04	104.13±0.25	−0.02%±0.01%	104.13±0.05	−0.02%±0.02%
ngons_50	4356.10±79.37	4365.11±70.85	–	4319.94±36.69	0.84%±0.57%
density_40	7098.89±0.00	7098.89±0.00	–	7098.89±0.00	–

Table 4.2: Influence of the GLNS mode setting on the solution cost.

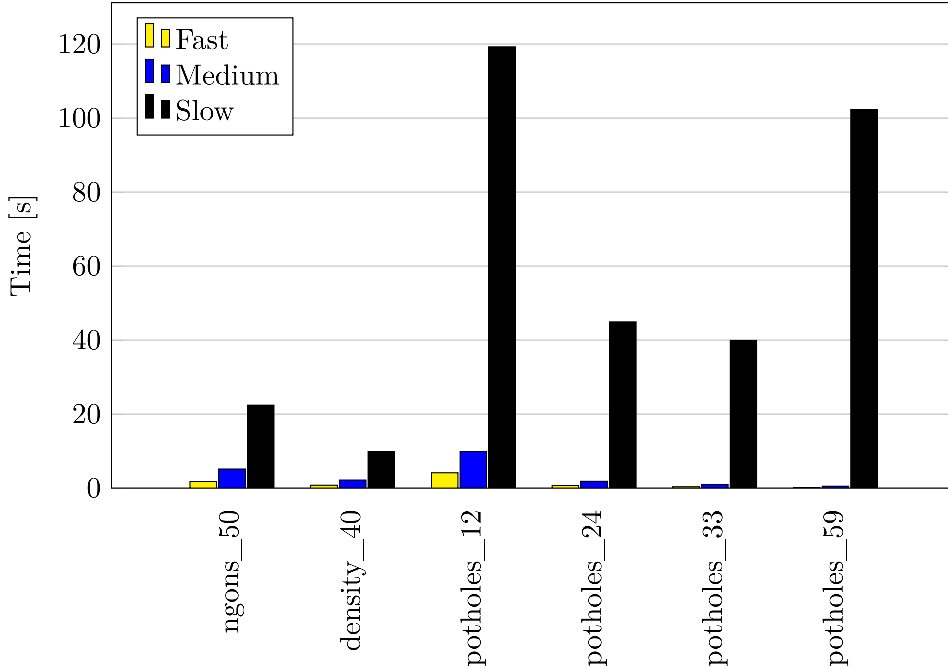


Figure 4.4: Influence of the GLNS mode setting on the runtime.

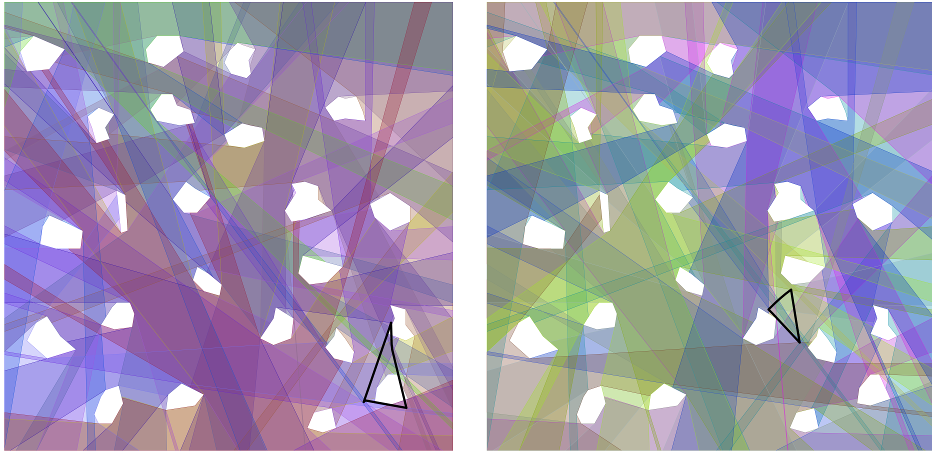
4.3.1 TPP change

This section analyzes the change from Section 3.4.2, where the first 3 lines of Algorithm 2 are skipped and the points found by GLNS are used as the initial points instead.

Problem name	Generated triangles	Time (s)		Solution cost		
		Original TPP	Modified TPP	Original TPP	Modified TPP	Improvement
potholes_12	988	0.06±0.00	0.06±0.00	18.16±4.47	9.14±0.25	49.67%±6.91%
potholes_24	826	0.27±0.03	0.25±0.03	65.00±1.03	62.80±0.06	3.39%±0.45%
potholes_33	857	0.58±0.08	0.55±0.07	77.34±2.07	74.11±0.47	4.18%±0.77%
potholes_59	1204	3.28±0.38	3.02±0.38	104.70±0.53	104.11±0.04	0.57%±0.14%
potholes_99	1472	15.64±2.10	14.73±1.83	139.46±2.29	136.77±0.71	1.92%±0.48%
ngons_50	258	1.53±0.20	1.49±0.21	4487.30±153.98	4364.50±92.21	2.74%±1.12%
ngons_100	516	14.31±1.70	13.43±1.74	8887.37±92.78	8660.54±84.47	2.55%±0.40%
ngons_150	774	55.74±6.81	58.10±7.41	13539.07±165.76	13054.55±105.24	3.58%±0.41%
density_40	188	0.77±0.11	0.85±0.20	7096.99±13.04	7094.63±24.19	–
density_80	376	6.64±0.84	6.16±0.71	13946.59±57.05	13938.26±49.75	–
density_120	564	23.48±3.02	23.3±2.84	20254.77±88.34	20250.29±85.08	–

Table 4.3: GLNS-TPP TPP change time and solution cost comparison.

As can be seen in Table 4.3, the runtime of GLNS-TPP is not and should not be affected. That is because even for the larger instances such as *ngons_150* with 150 polygons, the original version of the floating TPP takes just around 0.01 seconds of the total time. The solution cost shows noticeable improvements with 95% confidence on the $\text{TSPN}_{intersect}$ instances (*potholes_* and *ngons_*), while on the $\text{TSPN}_{disjoint}$ instances (*density_*) shows no statistically significant difference. This was expected, as the original floating TPP algorithm was made for the instances of the $\text{TSPN}_{disjoint}$ type. The most significant difference of nearly 50% was recorded on the *potholes_12* instance, where the polygons are of an unusual shape, spanning across the whole map in splinters. See Figure 4.5, where is this instance solved by the original and modified version of TPP.



(a) : Original TPP. **(b) :** Modified TPP.

Figure 4.5: Solved instance of potholes_12 by the original and modified TPP.

4.3.2 $\text{TSPN}_{obstacles}$

Two variants of GLNS-TPP are compared in this section. The first one is referred to as the optimized TPP or optimized GLNS-TPP and applies the change proposed in Section 3.4.3. The other version is the original unchanged

version and will be referred to as the original TPP or original GLNS-TPP. On top of the general performance on the instances with obstacles this section delves more into the detail of TPP. The reason is that while in the instances without obstacles does TPP take a negligible amount of the total time, in $\text{TSPN}_{obstacles}$ it turns into a bottleneck.

Performance analysis

Table 4.4 shows, that running TPP with $\epsilon = 0.01$ on the $\text{TSPN}_{obstacles}$ instances causes it, in the worst case, to take most of the total runtime. The optimized TPP is able to bring this time considerably down, but it comes at a price of a generally worse solution cost (Table 4.5). On instances with polygons that do not consist of very unusual shapes this solution cost is almost comparable with the original TPP. The most significant difference was on the *potholes_12* instance already shown in Figure 4.5 (but with added obstacles). The optimized TPP in this case was not able to improve the solution cost at all and always ended after the first iteration.

	Time (s)						
	Original TPP			Optimized TPP			
Problem name	Precompute	TPP	Total (incl. GLNS)	Precompute	TPP	Total (incl. GLNS)	TPP Speedup
potholes_12	15.85±0.14	52.89±14.85	69.04±14.87	18.33±1.10	3.95±0.16	22.62±1.21	92.53%±7.88%
potholes_24	10.43±0.15	32.31±6.87	43.25±6.87	11.22±0.17	19.86±5.37	31.63±5.34	38.52%±7.57%
potholes_33	11.33±0.29	50.42±9.43	62.57±9.44	11.97±0.32	8.78±5.00	21.66±4.98	82.58%±5.94%
potholes_59	23.20±0.21	49.41±8.65	75.89±8.70	24.37±1.31	27.05±6.40	55.29±7.05	45.26%±6.11%
potholes_99	37.03±0.33	83.28±23.35	135.76±23.83	40.13±1.87	37.09±18.59	95.89±19.52	55.47%±10.05%
potholes_224	88.00±1.34	136.34±61.13	463.55±64.10	90.61±2.28	31.82±26.80	388.80±40.52	76.67%±13.74%

Table 4.4: GLNS-TPP time performance on $\text{TSPN}_{obstacles}$ instances.

	Solution cost				
	Original TPP		Optimized TPP		
Problem name	After GLNS	After TPP	After GLNS	After TPP	Improvement (after TPP)
potholes_12	11.45±0.37	9.31±0.69	11.37±0.24	11.37±0.24	−22.13%±2.19%
potholes_24	68.40±0.00	64.76±0.71	68.40±0.00	64.97±0.66	–
potholes_33	80.88±0.00	74.77±0.07	80.88±0.00	76.86±1.46	−2.80%±0.55%
potholes_59	119.03±0.06	107.14±0.43	119.05±0.07	107.16±0.58	–
potholes_99	156.31±0.22	138.81±1.09	156.23±0.25	139.55±1.74	−0.53%±0.42%
potholes_224	235.34±0.78	199.64±1.70	235.81±1.08	205.94±4.42	−3.15%±0.67%

Table 4.5: GLNS-TPP solution costs on $\text{TSPN}_{obstacles}$ instances.

Figure 4.6 and Figure 4.7 show, that there are two reasons for the achieved TPP time reduction. The first one is that the optimized TPP brings down the time of each TPP iteration. The second one is the reduced average number of the total TPP iterations. Both are related to the reduced number of edges and solution paths the optimized TPP goes through.

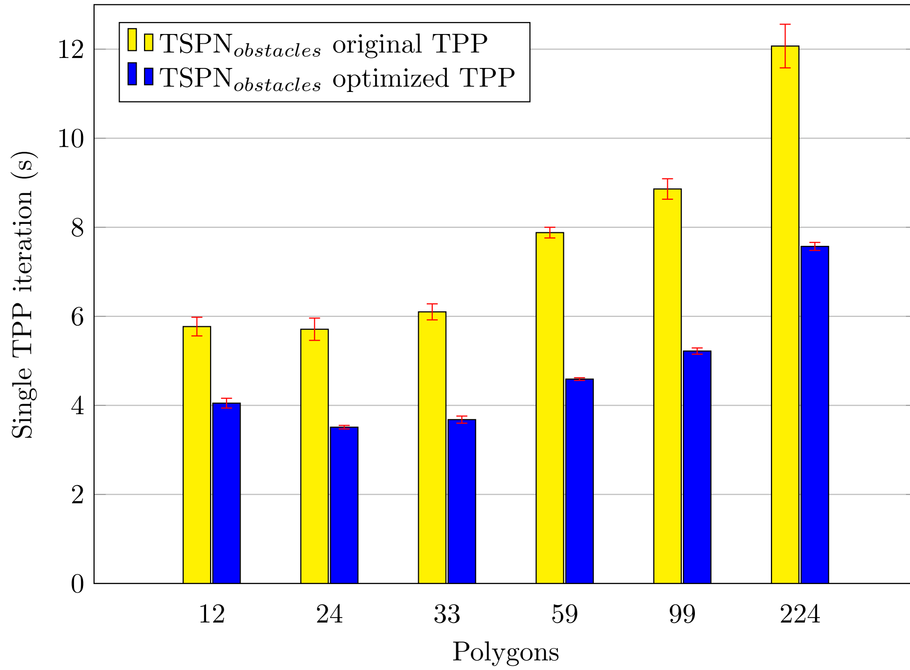


Figure 4.6: Average duration of a single TPP iteration on the *potholes_* instances.

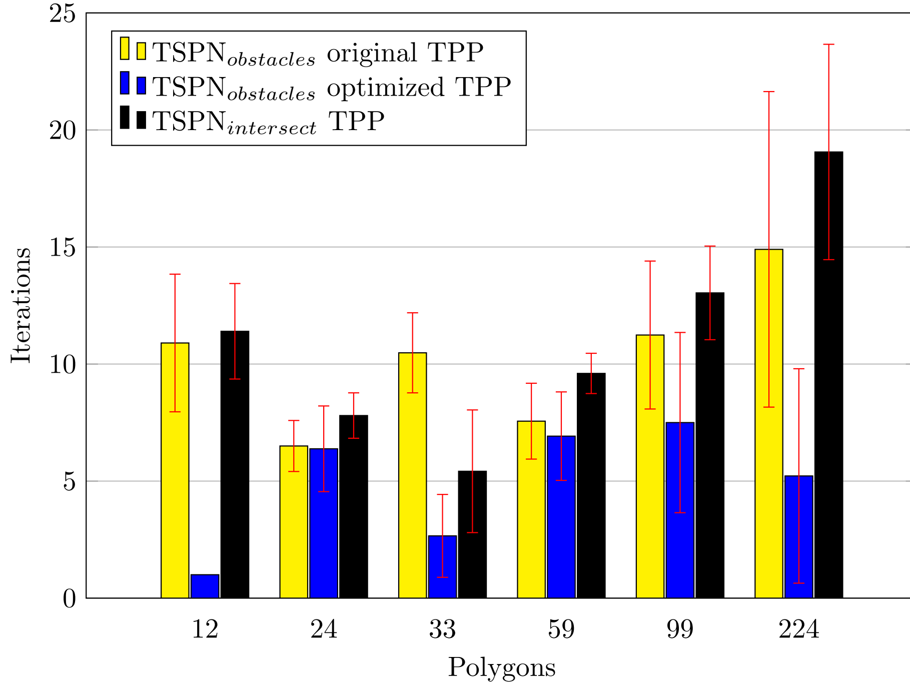
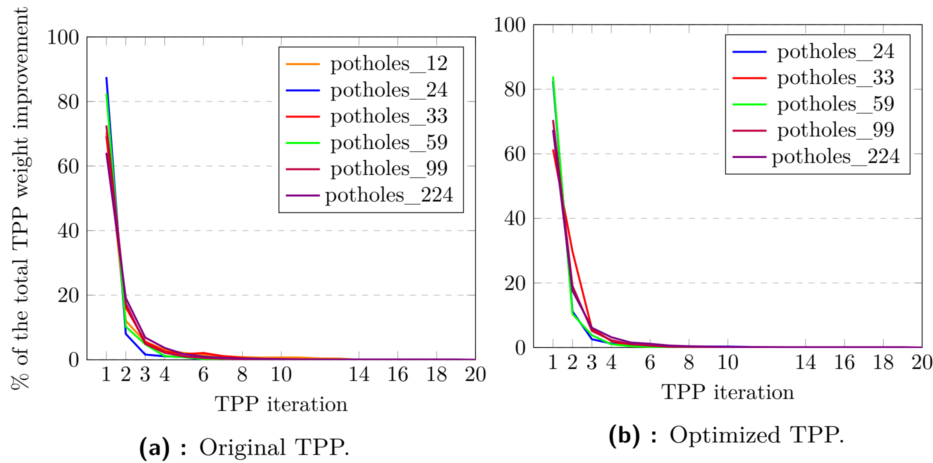


Figure 4.7: Average number of TPP iterations on the *potholes_* instances.

TPP iterations efficiency

Figure 4.8 shows, that the improvements made by the TPP iterations diminish very quickly. The biggest reductions in the tour length take place in the first three or four iterations. In all tested instances around 90% of the total tour length reduction by TPP took place in the first three iterations. Capping them at those levels therefore seems to be very desirable when trying to find a balance between the speed of the algorithm and the quality of the final solution.



(a) : Original TPP.

(b) : Optimized TPP.

Figure 4.8: Weight reduction by TPP iteration.

Influence of the amount of obstacles

To test the effect the growing number of obstacles has on the runtime of GLNS-TPP, the number of input polygons was of fixed size of 32. The number of possible TPP iterations was capped at 3 to utilize the knowledge gained from the previous section.

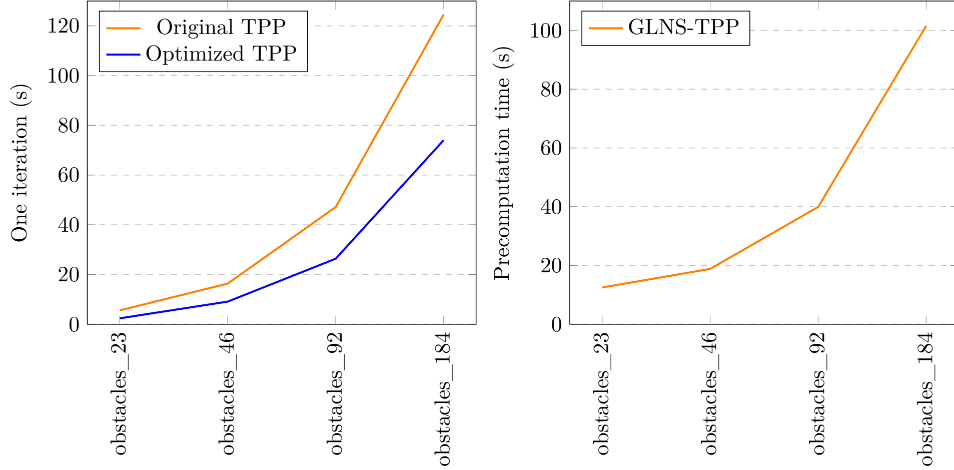
Table 4.6 and Table 4.7 show a significant slowdown of both precomputation time and TPP times when the amount of obstacles is doubled. Even the optimized TPP reaches around 70 seconds per TPP iteration on an instance with 184 obstacles. Figure 4.9 shows, that this dependence of the precomputation and TPP time on the amount of obstacles is polynomial.

	Time (s)						
	Original TPP			Optimized TPP			
Problem name	Precompute	TPP	Total (incl. GLNS)	Precompute	TPP	Total (incl. GLNS)	TPP Speedup
obstacles_23	11.46±0.21	19.28±0.40	31.60±0.49	11.94±1.05	9.62±1.77	22.45±2.23	50.10%±2.64%
obstacles_46	17.84±0.09	46.19±0.85	66.03±0.84	17.80±0.16	26.89±0.39	46.72±0.45	41.79%±0.57%
obstacles_92	37.25±0.14	132.59±3.37	177.69±3.42	37.40±0.10	74.12±6.01	119.44±6.02	44.10%±1.46%
obstacles_184	95.88±2.93	362.66±5.52	493.74±9.30	99.60±4.13	213.89±4.38	349.51±9.02	41.02%±1.77%

Table 4.6: GLNS-TPP time dependency on the amount of obstacles.

	Solution cost				
	Original TPP		Optimized TPP		
Problem name	After GLNS	After TPP	After GLNS	After TPP	Improvement (after TPP)
obstacles_23	80.27±0.02	75.25±0.39	80.27±0.02	75.48±0.55	−0.31%±0.25%
obstacles_46	133.27±0.01	127.25±0.21	133.27±0.02	127.27±0.27	–
obstacles_92	218.39±0.03	204.30±0.63	218.39±0.03	204.64±0.57	−0.16%±0.12%
obstacles_184	389.93±0.09	376.43±0.32	390.00±0.20	376.22±0.13	–

Table 4.7: GLNS-TPP solution costs dependency on the amount of obstacles.



(a) : Influence on the TPP iteration length **(b) :** Influence on the precomputation time

Figure 4.9: Amount of obstacles influence

4.4 GLNSC and GLNS-TPP comparison

Both algorithms were compared on their common problem instances (TSPN$_{disjoint}$ and TSPN$_{intersect}$). Additionally, the influence of the number of edges and polygons was examined.

4.4.1 Influence of the amount of edges

This section analyzes how the amount of edges of the polygons in an instance affects the runtime of the algorithms.

Table 4.8 shows, that the complexity of the input polygons plays a major role in the total runtime and that its rate of growth stably increases with the additions of further edges. The runtime almost doubles for both algorithms when going from 80-gons to 160-gons. But looking at Figure 4.10 it is apparent, that the time dependency on the amount of edges in polygons is linear. With GLNSC being more sensitive to the increase of the edges than GLNS-TPP. While in GLNSC this growth is caused by the increasing number of expensive geometric calculations, in GLNS-TPP the cause is the algorithm creating the triangular mesh, as it is forced to generate larger numbers of triangles to split the input polygons.

Problem name	Edges	GLNSC Time (s)	GLNS-TPP Generated triangles	GLNS-TPP Time (s)	Speedup
3gons_50	3	2.32±0.26	50	0.92±0.06	60.45%±3.23%
4gons_50	4	2.76±0.43	100	1.31±0.17	52.56%±4.66%
5gons_50	5	2.86±0.37	150	1.44±0.17	49.47%±3.88%
6gons_50	6	3.09±0.38	200	1.34±0.17	56.60%±3.74%
7gons_50	7	2.92±0.46	250	1.53±0.19	47.65%±4.79%
8gons_50	8	3.01±0.49	300	1.50±0.21	49.99%±4.97%
9gons_50	9	3.09±0.37	350	1.57±0.17	49.36%±3.68%
10gons_50	10	3.10±0.51	400	1.48±0.15	52.31%±4.79%
20gons_50	20	4.41±0.64	900	1.71±0.24	61.25%±4.38%
30gons_50	30	5.31±0.97	1400	1.97±0.28	63.00%±5.32%
40gons_50	40	6.87±0.93	1900	2.23±0.35	67.49%±4.07%
80gons_50	80	11.36±1.86	3900	3.67±0.56	67.70%±4.79%
160gons_50	160	21.48±3.59	7900	7.22±0.74	66.39%±4.79%

Table 4.8: Influence of the number of edges (per polygon) on the runtime of GLNSC and GLNS-TPP

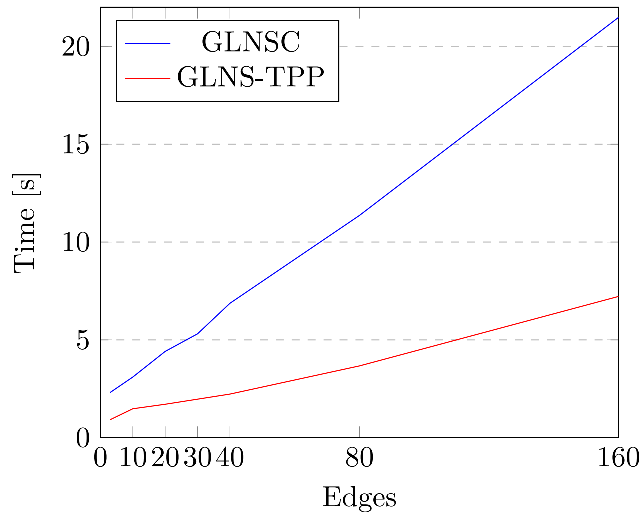


Figure 4.10: Time dependence on the amount of edges of the polygons

4.4.2 TSPN$_{disjoint}$ and TSPN$_{intersect}$ instances

Table 4.9 compares the algorithms in terms of speed on the instances of TSPN$_{disjoint}$ and TSPN$_{intersect}$ type. Both GLNSC and GLNS-TPP were set to their fastest possible settings. It is clear from the speedup column, that GLNS-TPP is significantly faster than GLNSC on all instances. This speed comes at a price of a slightly worse solution cost (Table 4.10), with the exception of some of the *potholes_* instances, especially *potholes_12* with around 54% improvement. This instance was previously shown in Figure 4.5 and this type of a map looks like a weak spot of GLNSC.

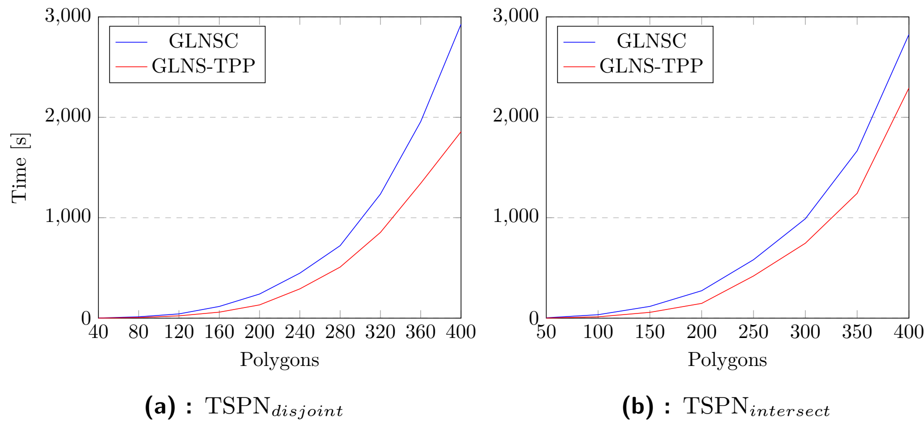
Problem name	GLNSC time (s)			GLNS-TPP time (s)			
	Min	Max	Average	Min	Max	Average	Speedup
density_40	1.02	1.96	1.50±0.20	0.54	1.34	0.85±0.20	43.43%±5.35%
density_80	10.49	17.00	13.59±1.43	4.85	7.87	6.16±0.71	54.71%±3.30%
density_120	35.53	53.00	42.59±3.98	18.41	32.53	23.30±2.84	45.29%±3.22%
density_160	93.74	151.61	116.68±13.65	46.87	77.49	59.94±8.22	48.63%±3.83%
density_200	167.80	298.34	240.30±26.63	104.09	185.03	132.18±15.03	44.99%±3.57%
density_240	423.24	463.17	448.31±15.50	259.58	322.13	291.73±26.99	34.93%±7.16%
density_280	574.84	810.54	721.86±89.18	457.90	607.19	508.77±58.44	29.52%±15.23%
density_320	1114.87	1327.53	1235.27±77.08	734.74	1043.56	853.81±144.68	30.88%±13.69%
density_360	1780.25	2125.37	1957.44±123.51	1224.95	1409.54	1342.79±69.57	31.40%±7.47%
density_400	2204.98	3229.25	2926.16±388.54	1474.39	2178.23	1856.53±255.34	36.55%±15.73%
ngons_50	2.85	5.92	3.71±0.59	1.14	2.15	1.49±0.21	59.80%±4.74%
ngons_100	23.45	47.26	34.56±5.15	9.76	17.25	13.43±1.74	61.13%±4.41%
ngons_150	80.54	152.20	117.33±14.00	43.66	77.52	58.10±7.41	50.48%±3.79%
ngons_200	222.64	356.12	273.37±29.10	119.51	196.97	147.18±16.63	46.16%±3.44%
ngons_250	516.48	627.05	582.03±45.60	384.10	468.06	420.02±28.45	27.84%±7.04%
ngons_300	757.54	1218.47	989.64±203.86	649.02	839.96	745.97±76.46	24.62%±22.69%
ngons_350	1545.81	1755.46	1665.82±90.11	1174.11	1341.22	1242.44±74.54	25.42%±7.24%
ngons_400	2642.20	3075.21	2824.64±211.18	2011.46	2708.17	2291.66±260.03	18.87%±12.23%
potholes_12	0.28	0.52	0.39±0.05	0.05	0.07	0.06±0.00	84.56%±3.66%
potholes_24	0.97	1.68	1.31±0.17	0.20	0.31	0.25±0.03	80.59%±3.66%
potholes_33	1.96	3.22	2.55±0.30	0.44	0.71	0.55±0.07	78.27%±3.37%
potholes_59	8.75	14.17	11.32±1.41	2.14	4.12	3.02±0.38	73.32%±3.63%
potholes_99	30.87	55.57	39.17±5.00	11.50	20.20	14.73±1.83	62.40%±3.82%
potholes_224	334.32	581.55	432.11±55.16	167.34	282.66	223.59±27.76	48.26%±4.01%

Table 4.9: Comparison of GLNSC and GLNS-TPP times on $\text{TSPN}_{disjoint}$ (density_) and $\text{TSPN}_{intersect}$ (ngons_, potholes_)

Problem name	GLNSC solution cost			GLNS-TPP solution cost			
	Min	Max	Average	Min	Max	Average	Improvement
density_40	6936.97	6937.99	6937.07±0.31	7006.68	7162.74	7094.63±24.19	−2.27%±0.10%
density_80	13595.90	13635.40	13601.07±7.05	13887.20	14047.50	13938.26±49.75	−2.48%±0.10%
density_120	20057.90	20174.70	20099.80±28.25	20159.80	20345.50	20250.29±85.08	−0.75%±0.13%
density_160	26629.40	26885.60	26768.01±64.82	26644.20	27715.20	27247.91±251.58	−1.79%±0.27%
density_200	35170.30	35648.80	35343.15±98.29	35254.80	36642.30	35902.81±349.61	−1.58%±0.29%
density_240	41953.20	42587.60	42218.14±231.55	42663.00	43270.50	42902.88±228.34	−1.62%±0.79%
density_280	48544.90	49127.10	48804.68±231.43	48972.90	49866.50	49470.66±354.93	−1.37%±0.90%
density_320	53128.10	53666.90	53424.02±214.06	53907.70	54591.60	35902.81±349.61	−1.30%±0.67%
density_360	59470.70	60491.40	60031.14±404.75	59868.20	61680.80	60895.70±678.06	−1.44%±1.36%
density_400	68401.00	68876.80	68630.93±170.41	69694.60	71304.40	70544.54±784.40	−2.79%±1.07%
ngons_50	4069.07	4609.69	4140.96±106.11	4152.37	4469.08	4364.50±92.21	−5.40%±0.95%
ngons_100	8082.05	8383.02	8188.57±81.38	8529.70	8891.75	8660.54±84.47	−5.76%±0.40%
ngons_150	12278.70	12897.00	12441.55±154.97	12817.10	13311.50	13054.55±105.24	−4.93%±0.42%
ngons_200	16170.00	16833.80	16407.12±172.16	16815.30	17996.00	17237.22±393.45	−5.06%±0.73%
ngons_250	22160.00	22565.50	22346.50±148.86	23392.40	24527.10	23847.25±467.98	−6.71%±2.11%
ngons_300	26362.10	26821.50	26571.92±205.69	27648.90	28365.10	28010.66±350.32	−5.42%±1.58%
ngons_350	30505.00	30653.80	30581.88±61.73	31695.00	32465.70	32093.58±351.28	−4.94%±1.20%
ngons_400	32768.10	34276.20	33306.02±588.62	33978.60	35884.90	34845.82±835.77	−4.62%±3.17%
potholes_12	8.65	38.16	20.14±6.12	8.36	9.24	9.14±0.25	54.61%±8.53%
potholes_24	62.75	67.22	63.42±1.13	62.75	63.05	62.80±0.06	0.98%±0.50%
potholes_33	73.36	76.55	74.76±1.01	73.55	75.12	74.11±0.47	0.86%±0.42%
potholes_59	104.03	107.91	104.67±0.78	104.06	104.16	104.11±0.04	0.54%±0.21%
potholes_99	132.93	149.11	135.39±2.74	135.31	138.14	136.77±0.71	−1.02%±0.59%
potholes_224	107.20	196.06	190.48±1.88	193.51	200.95	197.89±1.77	−3.68%±0.38%

Table 4.10: Comparison of GLNSC and GLNS-TPP solution costs on $\text{TSPN}_{disjoint}$ (density_) and $\text{TSPN}_{intersect}$ (ngons_, potholes_)

Figure 4.11 visualizes the dependency of GLNSC and GLNS-TPP on the number of instance polygons. This dependency is polynomial for both algorithms.



(a) : TSPN$_{disjoint}$ **(b) :** TSPN$_{intersect}$

Figure 4.11: Time dependence on the amount of polygons of GLNSC and GLNS-TPP

Overall, the solution cost difference of GLNS-TPP is almost negligible compared to the achieved speedup. Moreover, this speed difference buffer provides GLNS-TPP with the opportunity to lower the ϕ parameter, generate a larger number of triangles and thus possibly improve the solution cost. In this way, GLNS-TPP is more flexible. GLNSC does not have much room for a speedup apart from the further changes to the inner working of the underlying GLNS algorithm.

Chapter 5
Conclusion

The thesis addresses the restricted version of TSPN, where the neighborhoods are simple, possibly intersecting, polygons. Two algorithms, GLNSC and GLNS-TPP, were proposed and implemented, each of them capable of solving this problem. GLNS-TPP is also able to handle the case, where there are simple polygonal obstacles between the neighborhoods that the tour has to avoid passing through. The subset of this problem, where the polygons are convex, can be used for a practical task of the robot motion planning, where the robot has to explore the entire map. Because of the convexity of the polygons the robot has the certainty that upon visiting the polygon it can see its entire area. Therefore covering the map with such polygons and solving this version of TSPN guarantees the map has been fully seen by the robot.

Both proposed algorithms make use of the state of the art metaheuristic GTSP solver called GLNS and the algorithm for solving the floating TPP. GLNSC transforms the internal heuristics and methods of GLNS to work with simple polygons instead of discrete points. Some of these modifications use the methods from TPP. GLNS-TPP works in two phases. At the beginning of the first phase the polygons are split into a triangular mesh. The points in the center of these triangles are then used as the input for the original GLNS. The second phase uses the tour gained from the first phase, shortens it using the algorithm for the floating TPP and returns the final tour. For an instance with obstacles the TPP algorithm is slightly modified and the visibility graph is used to calculate the distances between the points.

For the experimental part of this thesis a library of polygonal maps was created to test the algorithms on. GLNSC and GLNS-TPP were evaluated both individually and against each other. The only feasible GLNS mode for both algorithms is the fast mode, as the others are too slow without bringing a meaningful improvement of the final solution. When set to their fastest settings, GLNS-TPP is significantly faster on all instances with the speedup being as much as 85% on smaller instances, but this percentage comes down with the growing number of polygons. The largest tested instance contained 400 polygons and it took GLNSC and GLNS-TPP around 2800 and 2200 seconds respectively to solve. GLNS-TPP found generally worse solutions by a few percents with the exception of instances with polygons of unusual shapes (such as long intersecting splinters) where it found better solutions by as

much as 55%. GLNS-TPP is more flexible than GLNSC through its ability to control how dense the triangular mesh is generated and thus further improve the final solution. GLNS-TPP was also individually tested on the instances with polygonal obstacles. TPP in this case takes usually the majority of the total time. Though it can be reduced by capping the number of the total iterations, as on all the tested instances the 90% of the total solution cost improvement took place in the first few TPP iterations.

In conclusion, both algorithms are capable of successfully solving the version of TSPN this thesis is motivated by, with GLNS-TPP being also able to account for the obstacles between the neighborhoods and avoid them throughout the tour. Overall, GLNS-TPP is faster, more flexible and has more potential for improvement than GLNSC. The main areas of the possible future improvements of GLNS-TPP lie in exploring different ways of splitting the polygons and using faster alternatives to the visibility graph.

Bibliography

[1] E.M. Arkin and R. Hassin. Approximation Algorithms for the Geometric Covering Salesman Problem. Discrete Applied Mathematics, 55(3), pages 197–218, 1994.

[2] M. Dror, A. Efrat, A. Lubiw, and J. Mitchell. Touring a sequence of polygons. In Proc. STOC, pages 473–482, 2003.

[3] Li F., Klette R. Rubberband Algorithms. In: Euclidean Shortest Paths. Springer, London, pages 53-89, 2011.

[4] Stephen L. Smith and Frank Imeson, GLNS: An effective large neighborhood search heuristic for the generalized traveling salesman problem, Computers and Operations Research 8, pages 1–19, 2017.

[5] K. Helsgaun, Solving the equality generalized traveling salesman problem using the Lin–Kernighan–Helsgaun algorithm, Mathematical Programming Computation 7 (3), pages 269–287, 2015.

[6] G. Gutin, D. Karapetyan, A memetic algorithm for the generalized traveling salesman problem, Natural Computing 9 (1), pages 47–60, 2010.

[7] David Woller, Hledání zdrojů gama záření, Search for sources of gamma radiation, Diplomová práce, České vysoké učení technické v Praze, 2019.

[8] S. Ropke, D. Pisinger, An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows, Transportation science 40 (4), pages 455–472, 2006.

[9] D. Pisinger, S. Ropke, A general heuristic for vehicle routing problems, Computers & Operations Research 34 (8), pages 2403–2435, 2007.

[10] Pan, Xiuxia & Li, Fajie & Klette, Reinhard. Approximate Shortest Path Algorithms for Sequences of Pairwise Disjoint Simple Polygons. Proceedings of the 22nd Annual Canadian Conference on Computational Geometry, CCCG, pages 175-178, 2010.

[11] Melkman. On-line construction of the convex hull of a simple polygon. Information Processing Letters vol.25, pages 11-12, 1987.

[12] Latombe, J.C., Robot Motion Planning, Springer. page 157, 1991.

[13] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L. Introduction to Algorithms (1st ed.), pages 558–565, 1990.

[14] Mitchell, J.S.B.: Shortest paths among obstacles in the plane. Int. J. Comput. Geom. Appl. 6, pages 309–332, 1996.

[15] Kamp, Poul-Henning. Ministat tool, Retrieved May 15, 2020, from https://www.freebsd.org/cgi/man.cgi?query=ministat

Appendix A

Contents of the attached CD

Filename or directory	Description
GLNSC	Source code of the GLNSC algorithm.
GLNS-TPP	Source code of the GLNS-TPP algorithm.
DataInstances	Polygonal library of instances.
Experiments	Data from the experiments.
DP.pdf	Text of this thesis.
readme.txt	Instructions for running the source code.