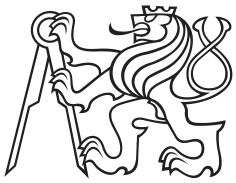
Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Traveling Salesman Problem with Overlapping Circle-polygonal Neighborhoods.

Bc. Lars Kahlert

Supervisor: Ing. Jan Mikula
Field of study: Cybernetics and robotics
May 2024

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Kahlert Lars** Personal ID number: **478066**

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

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Master’s thesis details

Master’s thesis title in English:

Traveling Salesman Problem with Overlapping Circular-polygonal Neighborhoods

Master’s thesis title in Czech:

Problém obchodního cestujícího s překrývajícími se kruhovo-polygonálními sousedstvími

Guidelines:

1. Familiarize yourself with the traveling salesman problem with continuous neighborhoods (TSPN) and related problems, such as the generalized traveling salesman problem (GTSP), close-enough traveling salesman problem (CETSP), touring polygons problem (TPP), and watchman route problem (WRP). Explore the existing techniques employed to solve these problems, focusing on their relevance in solving the TSPN. Motivate these problems in the context of possible robotic applications. [1-7]
2. Design and implement a solver to find the shortest path between a given point-neighborhood-point (PNP) triple in a 2D plane. The neighborhood should be a convex region with boundary edges consisting of either line segments or concentric circular arcs.
3. Extend the PNP solver to handle a polygonal environment with obstacles and verify the correctness of the implementation.
4. Utilize the PNP solver to implement the rubberband algorithm [7] for the touring neighborhoods problem (TNP), which is a TSPN variant with a fixed order of neighborhoods.
5. Employ the PNP/TNP solutions to design and implement a solution method for the TSPN while considering a limited computational time budget.
6. Conduct experimental evaluations of the TSPN method on instances from [1] and compare it to the existing method [1]. Describe and discuss the results obtained.

Bibliography / sources:

[1] Mikula, J., & Kulich, M. (2022). Towards a Continuous Solution of the d-Visibility Watchman Route Problem in a Polygon With Holes. IEEE Robotics and Automation Letters, 7(3), 5934–5941. https://doi.org/10.1109/LRA.2022.3159824
[2] Kulich, M., Vidašič, J., & Mikula, J. (2023). On the Travelling Salesman Problem with Neighborhoods in a Polygonal World. In Robotics in Natural Settings. CLAWAR 2022. Lecture Notes in Networks and Systems, 530, 334–345. https://doi.org/10.1007/978-3-031-15226-9_32
[3] Fanta, L. (2021). The Close Enough Travelling Salesman Problem in the polygonal domain. [Master's thesis, CTU FEE] https://dspace.cvut.cz/handle/10467/96747
[4] Faigl, J., & Přeučil, L. (2011). Inspection planning in the polygonal domain by Self-Organizing Map. Applied Soft Computing, 11(8), 5028–5041. https://doi.org/10.1016/j.asoc.2011.05.055
[5] Faigl, J. (2018). GSOA: Growing Self-Organizing Array - Unsupervised learning for the Close-Enough Traveling Salesman Problem and other routing problems. Neurocomputing, 312, 120–134. https://doi.org/10.1016/j.neucom.2018.05.079
[6] Smith, S. L., & Imeson, F. (2017). GLNS: An effective large neighborhood search heuristic for the Generalized Traveling Salesman Problem. Computers & Operations Research, 87, 1–19. https://doi.org/10.1016/j.cor.2017.05.010
[7] Dror, M., Efrat, A., Lubiw, A., & Mitchell, J. S. B. (2003). Touring a sequence of polygons. Proceedings of the Thirty-Fifth ACM Symposium on Theory of Computing - STOC ’03, 473–482. https://doi.org/10.1145/780611.780612

Name and workplace of master's thesis supervisor:

Ing. Jan Mikula Department of Cybernetics FEE

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

Date of master's thesis assignment: **01.09.2023** Deadline for master's thesis submission: **24.05.2024**

Assignment valid until: **16.02.2025**

Ing. Jan Mikula	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

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

.	
Date of assignment receipt	Student's signature

Acknowledgements

I am sincerely grateful to my supervisors, Ing. Jan Mikula and RNDr. Miroslav Kulich, Ph.D., for their valuable advice, endless patience, and exceptional mentorship during the creation of this thesis. Furthermore, I extend my heartfelt appreciation to my family and friends for their understanding, encouragement, and patience during this journey.

Declaration

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

In Prague, 24.05.2024

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

V Praze dne 24.05.2024

Lars Kahlert

Abstract

This thesis builds upon a heuristic solution framework for solving the Watchman Route Problem (WRP) in polygonal environments. The goal is to find the shortest route from which a robot with an omnidirectional sensor with limited visibility range can fully inspect a known 2D environment. The original framework generates a set of convex regions covering the full environment and obtains the solution by solving the traveling salesman problem with neighborhoods (TSPN) through discretization. This thesis proposes a solution that solves the TSPN without resorting to the time-consuming discretization step. We introduce a Point-Neighborhood-Point (PNP) optimization algorithm within a rubber-band (RB) algorithm for the local improvement of paths over a fixed order of neighborhood regions. The TSPN is solved by an Iterative Local Search (ILS) metaheuristic. Through experimental evaluation, a comparison with the original method is created, and we show that our approach can provide superior solutions within strict computational constraints.

Keywords:

Routing Problems;
Traveling Salesman Problem with Neighborhoods;
Metaheuristics;
Rubber-band algorithm;
Iterated Local Search

Supervisor:

Ing. Jan Mikula

IMR - Intelligent Mobile Robotics,
CIIRC, CTU in Prague,
Jugoslávských partyzánů 1580/3,
160 00 Praha 6, Dejvice,
Czech Republic

Abstrakt

Tato práce vychází z heuristického řešení hlídačova problém (WRP) v polygonálních prostředích. Cílem je najít nejkratší trasu, po které může robot s všesměrným senzorem s omezeným rozsahem viditelnosti plně prozkoumat známé 2D prostředí. Původní heuristika generuje množinu konvexních oblastí pokrývajících celé prostředí a řešení získá vyřešením problému putujícího obchodníka s okolím (TSPN) pomocí diskretizace. Tato práce navrhuje způsob, který řeší TSPN, aniž by se uchýlil k časově náročnému kroku diskretizace. Zavádíme optimalizační algoritmus typu bod-sousedství-bod (PNP) v rámci gumičkového algoritmu (RB) pro lokální zlepšování cest s pevným pořadím sousedství. TSPN je řešena metaheuristikou iterativního lokálního prohledávání (ILS). Prostřednictvím experimentálního vyhodnocení je vytvořeno srovnání s původní metodou a ukazujeme, že náš přístup může poskytnout lepší řešení v rámci přísných omezení výpočetního času.

Klíčová slova:

Směrovací problémy;
Problém obchodního cestujícího se sousedstvími;
Metaheuristiky;
Gumičkový algoritmus;
Iterované lokální prohledávání

Překlad názvu:

Problém obchodního cestujícího s překrývajícími se kruhovo-polygonálními sousedstvími

Contents

Figures

Tables

Algorithms

Chapter 1

Preliminaries

1.1 Introduction

Imagine a mobile robot tasked to collect sensory data, such as images of the environment or specific regions within it. Suppose the environment map is available to the robot, which may be the case when the robot operates in the environment regularly or is provided with some explicit information like a floor plan. In that case, we talk about the *mobile robot inspection* (MRI) task, as opposed to the exploration task, where the map is unknown. The established approach for addressing the MRI task from a planning perspective is through the *hierarchical planning paradigm*. This planning hierarchy comprises the *controller, motion planner*, and *task planner* arranged from bottom to top [1, 2]. On the lowest level is the *controller*, which controls the actuators to move the robot toward its goal. Above that is the *motion planner*, sometimes further separated into *local* and *global* components, which plan a collision-free route between two points and set the goal for the *controller*. The local planner operates at a higher resolution and refines the path generated by the global planner. Ultimately, the task planner is responsible for achieving the task goal, monitoring its progression, and ensuring overall efficiency. When efficiency is a primary concern, the task planner may incorporate an optimization module that operates on a simplified environment model to achieve global task efficiency.

In the context of the MRI, the optimization module typically takes the form of a routing problem solver. In its simplest form, when the MRI's goal is to inspect a predefined set of locations, the routing problem aligns with the well-known *traveling salesman problem* (TSP) [3] on the all-pairs shortest collision-free paths

graph of the set of locations. In a more complicated scenario, continuous regions of interest may be defined within the environment, or these regions may encompass the entire environment itself. When the environment model is a polygon with holes, and the sensory model is omnidirectional visibility with a limited range d, we obtain the *d-visibility watchman route problem* (d-WRP) [4].

Both WRP and d-WRP have been shown to be NP-hard for polygons with holes [5]. This leads to heuristic approaches being used in practice. Such as in Li et al. (2008) [6], or Danner and Kavraki (2000) [7] where an approximate algorithm is used to generate a solution in a viable time. In the recent work of Mikula and Kulich (2022) [8], the environment is first entirely covered in regions such that all points in a region are d-visible from one to another. Then, the proposed approach finds a watchman route touching a point from each region to guarantee that the whole environment is visible. Such a route is found on the discretized (sampled) regions and then improved in the continuous domain. Finding the shortest tour that touches at least one point from each of the given continuous regions (neighborhoods) is the *traveling salesman problem with neighborhoods* (TSPN) [9].

Mikula and Kulich (2022) [8] propose an approach where a solution to the TSPN is found in a discretization of the neighborhoods. In their approach, all neighborhoods of the TSPN are sampled, and then the shortest path touching at least one sample from each neighborhood has to be found. The problem of finding the shortest closed tour that visits at least one node from each predefined cluster of nodes is called the *generalized traveling salesman problem* (GTSP) [10] (see Sec. 1.2). Finding a solution on the discretized GTSP also solves the TSPN. In their work, Mikula and Kulich show that this discretization step can take up to 80% of the solution's computational time for instances with many regions. This thesis aims to avoid the discretization scheme involving the construction and solution of a GTSP instance and solve the TSPN directly, thus saving time in the initialization and having more time to optimize the solution.

The contributions of this thesis are as follows: i) The implementation of an algorithm to optimize the path length between a point-neighborhood-point (PNP) triple for neighborhoods with edges as line segments or concentric circular arches and the expansion of this algorithm to an environment with polygonal obstacles. Such algorithms have been shown to be beneficial for local path improvement algorithms and, before this thesis, were either for purely polygonal [11] or circular neighborhoods [12]. ii) Utilizing the PNP algorithm in a *rubber-band algorithm* (RB) [13] for the *touring neighborhood problem* (TNP) [14] which is a variant on the TSPN with a fixed order of neighborhoods. The new implementation of the RB algorithm is used to improve the solutions found by our proposed method and the reference method from [8], which previously used a RB that approximated neighborhoods with polygons. iii) Implementation of five parametrizations of

the reference method from [8] and evaluation of their solution quality over an extended time scale. iv) Implementation of the *iterative local search* (ILS) [15] metaheuristic for finding a solution directly on the TSPN instance, thus avoiding the discretization step from the conversion to GTSP. Comparing the solutions of our ILS implementation to the results from the improved reference method shows that we can generate better solutions in a limited time window. However, the reference provides better solutions over a larger time scale.

In Chapt. 2 our version of the TSPN problem is introduced, and the method from [8] upon which we want to improve is briefly described. Then, Chapt 3 describes our solution approach and the implemented algorithms. In Chapt. 4 we evaluate the reference method and our implemented algorithms and compare the results. The last Chapt. 5 contains the final remarks.

1.2 State of the Art

Close-Enough TSP (CETSP). The MRI task where specific positions are to be inspected is described as the well-known TSPN problem; if instead, it would be enough to get within a given distance of the inspection points, it can be described as the CETSP [16]. Faigl (2018) [17] uses an unsupervised learning process where positions in the environment are represented by artificial neural network nodes.

Watchman route problem (WRP). The *watchman route problem* first studied by Chin and Ntafos (1988) [18], considers the problem of finding the shortest route from which all points in a given space $\mathcal{W}$ are visible. The space $\mathcal{W}$ is generally a polygon with holes; see the example in Fig. 2.1a. A point p from $\mathcal{W}$ is considered *visible* from the path if a straight line from any point on q the path can be fully within $\mathcal{W}$ i.e. $\overline{pq} \subset \mathcal{W}$. One approach to the WRP is the *decoupling* approach presented by Packer (2008) [19] on the *multiple watchman routes problem* (MWRP). In this approach, a set of static guards is computed, and routes are built by splitting the minimum spanning tree. The resulting routes are optimized by substituting, removing, and adding vertices.

***d*-watchman route problem (*d*-WRP**. The d-WRP [4] considers the same problem of "seeing" the entire environment as WRP with the added constraint that the *visibility range* d is limited. This would be the MRI task of inspecting the whole environment with a robot with an omnidirectional sensor with a limited range. A point p from the environment $\mathcal{W}$ is considered *d-visible* from a point q of the path if it is true that $\overline{pq} \subset \mathcal{W} \wedge |pq| \leq d$. Danner and Kavraki (2000) [7] use a *decoupling* approach, where first guards with limited visibility are positioned

in a way that the full environment is visible. Then, the shortest path visiting all guards is found.

***d*-WRP unsupervised learning**. Faigl (2011) [20] deals with the d-WRP) using an artificial neural network called *self-organizing map*(SOM). The watchman route is represented as a ring of connected neuron weights evolving in $\mathcal{W}$. The approach starts with an initial tour from which only a part of the environment is seen. Then, the tour is iteratively expanded until the whole $\mathcal{W}$ is seen by drawing the neuron weights toward yet unseen parts of $\mathcal{W}$.

***d*-WRP towards a continuous solution**. Mikula and Kulich (2022) [8] propose a novel heuristic framework, for the d-WRP, i.e., limited visibility is considered. In the proposed framework, the environment $\mathcal{W}$ is entirely covered by regions $\mathcal{R}$[1] with such properties that a watchman route only needs to visit any point in each region to guarantee full visibility coverage of $\mathcal{W}$. This is a heuristic way of finding a solution to WRP using TSPN [9], where $\mathcal{R}$ are the neighborhoods in TSPN. Note that as this is a heuristic, an optimal solution is not guaranteed. In the next step, the neighborhood regions are discretized by sampling the free (not shared with $\mathcal{W}$) edges of each region, and together with the shortest path distance between all samples, a weighted graph is constructed. Then, the goal is to find a circuit (i.e., closed walk) that minimizes the distance while visiting a sample from each region, also known as GTSP [10]. The GTSP solution is then found using GLNS: *an effective large neighborhood search heuristic* by Smith and Imeson (2017) [21].

It can be generally expected that a solution found only on discrete samples would be worse than a solution found on continuous neighborhoods; therefore, the resulting discrete sample tour is locally optimized by a modified version of the Pan et al. [13] *rubber-band algorithm* (RB). The RB algorithm iterates over the path and minimizes the path length by pulling the path closer together. This "pulling together" is done by iterating through each sequential triplet of path points and adjusting the position of the middle one such that the distance to its neighboring points is minimized. The algorithm is limited to positioning the point in its respective neighborhood. This is done to maintain the constraint from TSPN that each neighborhood needs to be visited . Which is the *touring polygon problem* (TPP) [14] of finding the shortest tour which visits a predefined sequence polygons.

The proposed approach was experimentally evaluated and compared to state-of-the-art SOM-based [22] and DS+LKH methods [7]. An improvement of at least 10% for cases with $d > 3$ has been shown, although this came at the cost of

[1]The covering regions generated by the same algorithm are used in our proposed solution. A more detailed explanation follows in Chapt. 2.

computation time needed to find the solution.

As previously mentioned, the disadvantage of the method is the higher computational time required compared to the SOM-based and DS+LKH methods. The significant bottleneck is the shortest-paths computation by the *Dijkstra's algorithm* [23], which can take up to 88% A low visibility radius leads to a high number of neighborhood regions and samples (graph nodes) to consider, which results in poor scalability of the framework.

As an example, on the *jf-pb2* map, for visibility radiuses *3,2,1*, the proposed *best* parametrization took up to 176.9 s, 475.0 s, and 4425.1 s, respectively. Even the *trade-off* parametrization meant to improve the computational time needed 7.0 s, 29.6 s, 561.5 s, which is somewhat comparable to the 4.1 s, 22.3 s, and 889.3 s needed by the SOM [22] based method. The DS+LKH [7] method required only 0.5, 2.1, and 24.6 s, on the same instance, showcasing the possibility for good computation speed despite a high neighborhood count.

Constructive heuristics, improving heuristics and metaheuristics. Constructive and improving heuristics are used as part of our proposed solution (see Chapt. 3). Constructive and improving heuristics are powerful tools employed in optimization problems to navigate solution spaces efficiently while allowing a viable solution to be returned after any step. In path planning problems, constructive heuristics [24] serve to construct an initial path through the solution space. These heuristics leverage problem-specific information to explore feasible solutions efficiently. Improving heuristics [24] enhance solutions by iteratively modifying and evaluating them to improve their quality.

In mathematical terms, the improvement heuristics (operators) are functions which, based on their parameters p from the parameter space $\boldsymbol{\mathcal{P}}$, map from a valid solution space $\boldsymbol{\mathcal{S}}$ (improving operators) to a transformed solution space $\boldsymbol{\mathcal{S}}$:

$$op : \boldsymbol{\mathcal{S}} \times \boldsymbol{\mathcal{P}} \mapsto \boldsymbol{\mathcal{S}}. \tag{1.1}$$

The space of all solutions that can be reached by applying an operator on a given solution s is described as:

$$neigh_{op}(s) = \{s' \in \boldsymbol{\mathcal{S}} \mid s' = op(s, p)\ \forall p \in \boldsymbol{\mathcal{P}}\}. \tag{1.2}$$

A metaheristic is a combines local improvement heuristics, high level strategies and a method for escaping local minima [24] to find an optimal solution by intelligently exploring the solution space while balancing between exploration and exploitation to find the best possible solution within a reasonable computational time budget. The metaheuristic chosen for our approach is the *iterated local*

search (ILS) [15]. The ILS algorithm works by iteratively applying a local search algorithm to optimize a solution while applying a perturbation operator to allow it to escape from local minima.

Chapter 2

Problem Description

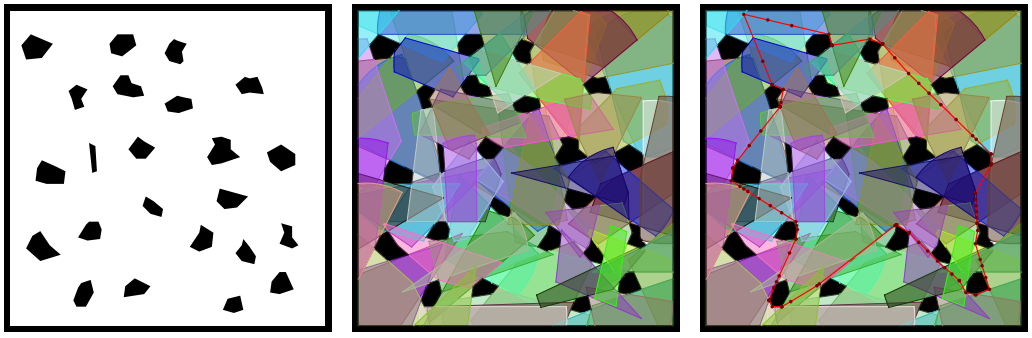
This thesis builds upon Mikula and Kulich [8] where the d-WRP is heuristically solved by a two-stage process wherein a set of regions $\boldsymbol{\mathcal{R}}$ covering the entire environment $\mathcal{W}$ is generated, and then the generated set serves as (input) neighborhoods for a TSPN problem. The neighborhood regions $\mathcal{R}$ are generated with the use of a *dual sampling algorithm* [25], computing the *(d/2)-visibility regions* [26], and then finding the *maximally-covering convex subsets* of these regions. These subsets are inspired by the *maximum area convex subsets of star-shaped polygons* studied in [27]. With this approach, a reasonably small number of regions covering a sufficient (specified by the user) portion of the environment ($\mathcal{W}$) is generated. The generated regions can overlap and have the property that visiting any point from the region guarantees the whole region is seen. For further details, see [8].

In contrast to [8], this thesis exclusively focuses on the second stage of the solution process. As such, we solve a TSPN with the environment $\mathcal{W}$ and a set of neighborhood regions $\boldsymbol{\mathcal{R}}$ as inputs.

This chapter contains a description of the problem instance and introduces notation used in later chapters. Section 2.1 describes what constitutes a TSPN problem instance. Section 2.1.1 further describes the structure of the neighborhood regions $\mathcal{R}$. Section 2.2 describes the structure of the solution and the criterion to be optimized. The last Section 2.3 briefly describes the approach used by [8] upon which we aim to improve.

2.1 Problem Inputs

Each TSPN problem instance is described by a *map* and *visibility radius d*. We consider the map a 2D environment, represented as a polygon with holes $\mathcal{W}$, delimiting the space that can be seen and traveled through. See Fig. 2.1a where $\mathcal{W}$ is white and obstacles are black. The second input is a set of neighborhood regions $\boldsymbol{\mathcal{R}} = \{\mathcal{R}_0, \mathcal{R}_1, ..., \mathcal{R}_{M-1}\}$, created based on the *visibility radius* (see Fig. 2.1b).



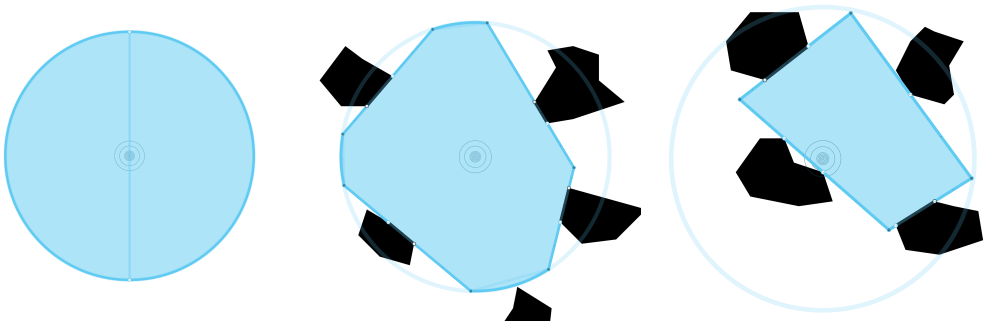
(a) : $\mathcal{W}$ **(b) :** Coverage by $\boldsymbol{\mathcal{R}}$ **(c) :** Output path $\boldsymbol{\mathcal{X}}$

Figure 2.1: Examples of problem inputs and outputs. (Map = potholes, d = 15)

2.1.1 Neighborhood Regions

The union of the neighborhood region set $\boldsymbol{\mathcal{R}}$ covers the whole environment i.e. $\bigcup_{i=0}^{M-1} \boldsymbol{\mathcal{R}_i} = \mathcal{W}$. Individual neighborhood regions are created so that the whole $\mathcal{R}$ is d-visible from any point within itself. Two points are d-visible if a straight line of length$\leq d$ from the point to point can be fully within $\mathcal{W}$. Due to this constraint, the regions take the form of so-called *polygon-circles*. A polygon-circle is a polygon with some of its edges replaced by concentric circular arches, depending on where it was limited by the environment and where by the visibility radius. A pure polygon or a circle can be created in some special cases. The possible shapes can be seen in Fig. 2.2. This is in contrast to previous approaches, where either only polygonal [11], or circular [12] neighborhood regions were considered. This *polygon-circles* shape more closely matches the (convex) visible region that could be expected from an omnidirectional sensor with limited visibility in an environment with obstacles, thus reducing the level of abstraction for practical applications.

A neighborhood region $\mathcal{R}$, as shown in Fig. 2.3, is created from a seed point $s(\mathcal{R})$, which is the center point for circle regions. The radius $r(\mathcal{R})$ gives the maximal distance any point from $\mathcal{R}$ can have from $s(\mathcal{R})$. The radius is half the visibility radius ($r(\mathcal{R}) = d/2$) to satisfy the d-visibility requirement. In the



(a) : Circle **(b) :** Polygon-circle **(c) :** Polygon

Figure 2.2: Possible neighborhood region shapes.

general case, $\mathcal{R}$ is described by a list of N vertices $V(\mathcal{R}) = \langle v_0, v_1, ..., v_{N-1} \rangle$ defining vertex positions and the connected edges $E(\mathcal{R}) = \langle e_0, e_1, ..., e_{N-1} \rangle$ in the clockwise direction. Edges can be of different types:

- $t(e) = o$; obstacle edge: created directly by a neighboring obstacle.
- $t(e) = f$; free edge: line segment which does not neighbor an obstacle.
- $t(e) = a$; arc edge: created due to visibility constraint.

Both obstacle and free edges are line segments defined by two consecutive vertices $e_i(v_i, v_{i+1})$. Arc edges are parts of the visibility circle given by $s(\mathcal{R})$ and $r(\mathcal{R})$. In some later algorithms (PNP), the concept of a *cone* will be used. In the context of $\mathcal{R}$, a *cone* means a conic area whose vertex is $s(\mathcal{R})$ and spans the region between two consecutive neighborhood vertices. In total each $\mathcal{R}$ has N cones denoted as $C(\mathcal{R}) = \langle c_0, c_1, ..., c_{N-1} \rangle$. Cones are indexed with the same i as the clockwise first vertex defining the edge the area contains $e_i \in c_i$. In Fig. 2.4, points are colorized based on the cone they fall within. As such, the vertex point, the edge it defines, and the cone containing the edge are all indexed the same: $v_i \in e_i \subset c_i$.

2.2 Objective

The solution to the TSPN is the shortest collision-free closed tour touching each $\mathcal{R}$ in at least one point. Collision-free, meaning no point of the tour intersects an obstacle ($\mathcal{X} \subset \mathcal{W}$). An example tour is shown in Fig. 4.1f drawn in red.

The tour $\mathcal{X}$ can also be described by a sequence of points, denoted as $\mathcal{X} = \langle x_0, x_1, ..., x_{M-1} \rangle$, with M being the number of points in the tour, which is the same

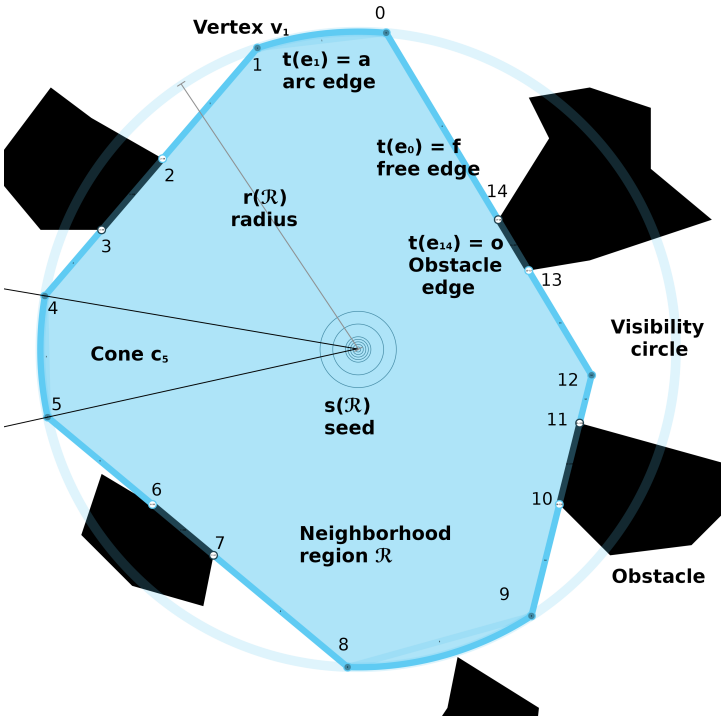


Figure 2.3: Description of neighborhood region notation.

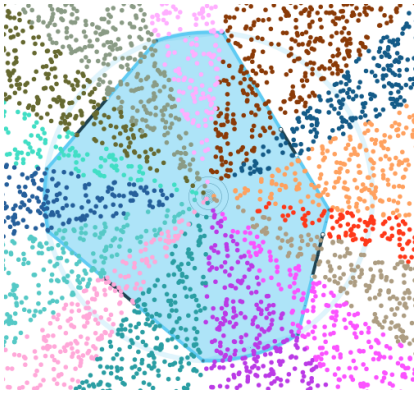


Figure 2.4: Visualization of cones for $\mathcal{R}$.

as the number of neighborhood regions in $\boldsymbol{\mathcal{R}}$. The indexation x_M equals point x_0 to keep the cyclical structure. The description by points means the tour combines the shortest collision-free paths between points x_i and x_{i+1} for $0 \leq i \leq M-1$. Let l_i be the length of the shortest collision-free path between points x_i and x_{i+1}. Then, we optimize the length of the whole tour $L_T = \sum_{i=0}^{M-1} l_i$. The goal is then to find a sequence of points $\mathcal{X}$ such that $\forall \mathcal{R} \in \boldsymbol{\mathcal{R}}, \exists x_i \in \boldsymbol{\mathcal{X}} : x_i \in \mathcal{R}$, and L_X is minimized. This is done in the implementation by creating an initial solution, then iteratively improving the ordering of $\mathcal{R}$ and positioning of x_i until a stopping condition is met.

2.3 Reference Method Description

Our method aims to improve upon the step where the reference method from [8] solves the TSPN problem on the neighborhood regions generated as described in Chpt. 2. Therefore, we will take a moment to explain their approach; for a full explanation see Alg 2. in [8].

The reference method works by sampling the input neighborhood regions and finding a solution to the *generalized traveling salesman problem* GTSP [10], which also solves the TSPN. The sampling is done on all neighborhood edges that do not neighbor an environmental obstacle (see Fig. 2.3), according to the maximal sample distance parameter d_{samp}. Free edges are sampled equidistantly so that the distance (along the edge) between two points is d_{samp} at maximum. Then, the samples (grouped by neighborhood region) serve as an instance of the GTSP problem. The GTSP instance is represented as a distance matrix D storing the shortest path length between all sample pairs. The distance matrix is filled by constructing a *visbility graph* [28] over all samples and reflex environment points and using *Dijkstra's algorithm* [23] to find the shortest path between samples. For solving the GTSP, an effective *large neighborhood search* heuristic GLNS [21] is used. The GLNS algorithm is stochastic; therefore, multiple iterations are done when we evaluate the method in Sec. 4.2 to create a statistic.

The use of Dijkstra's algorithm for constructing the distance matrix constitutes a substantial portion of the computational time required by the reference method. Our approach aims to bypass the entire process of generating a GTSP instance from the TSPN by attempting to solve the TSPN directly, thereby eliminating the need for the time-consuming conversion step.

Chapter 3

Solution Approach

Our solution approach considers several constructive and improving heuristics, which are then combined into metaheuristics. The first chapter concerns the *point neighborhood point* (PNP) algorithm, which is used to find the point on a neighborhood region, which minimizes the length to its neighboring points. While PNP technically falls under the improvement heuristics category, it is explained in the first Sec. 3.1 as it is used to enhance constructive heuristics. The constructive heuristics are described in Sec. 3.2 and are separated into *nearest neighbor* and *insertion* based methods. Improvement heuristics are covered by Sec. 3.3. As part of the improving heuristics, the *variable neighborhood descent*[1] (VND) [29] is also introduced. Sec. 3.4 covers the combination of the constructive and improving heuristics into the *iterated local search* (ILS) [15] metaheuristic. Furthermore, it describes the *double bridge* (DB) perturbation operator used to enhance the exploration of the solution space by escaping local optima.

[1]The neighborhood used in VND differs from the neighborhood region structure used in our problem definition. The neighborhood in VND denotes the solution neighborhood, with regards to the operator $\mathcal{N}_{op}$ (see Chpt. 1.2).

3.1 Point Neighborhood Point (PNP)

3.1.1 PNP Problem

In order to find a high-quality TSPN path, a way to locally optimize path segments is needed. For our particular case, this means finding the optimal position of any point on a tour, with regard to the distance to its neighboring points a,b, and any constraints placed upon it. The constraint is the need of each point of the tour to belong to a neighborhood ($x_i \in \mathcal{R}_i, 0 \leq i \leq M-1$) established in section 2.2. As such, PNP is the problem of finding a point $p_{min} \in \mathcal{R}$ such that the length of the shortest paths $l(a, p_{min}) + l(p_{min}, b)$ (further just $l(a, p_{min}, b)$) to two given points a and b, is minimized (see Fig. 3.1). Then, it is possible to arrive at a high-quality path by iteratively optimizing over path segments.

3.1.2 PNP Algorithm

The PNP algorithm must find point p_{min} given a, b and $\mathcal{R}$. The positions a, b, and $\mathcal{R}$ can be separated into the two cases shown in Fig. 3.1. In the first case (blue), the line segment intersects the neighborhood between two points $\overline{ab}$; therefore, the closest point lies anywhere at the intersection segment $\overline{ab}$ of the line segment and neighborhood. In the second case (red), the region and $\overline{ab}$ do not intersect; therefore, the closest point lies on the boundary of the outer neighborhood region. The respective cases are handled by the SNI and CPE algorithms we designed. The PNP algorithm uses the *segment neighborhood intersection* (SNI) algorithm to try to find an intersection segment; if no intersection is found, it returns the closest point found by the *closest point on edge* (CPE) algorithm. The following subsections are detailed descriptions of the SNI and CPE algorithms.

Line segment neighborhood intersection (SNI)

The proposed algorithm 3.1 returns the intersection between the neighborhood $\mathcal{R}$ and a line segment defined by two points a and b or reports it does not exist. Points a, b can be placed within or outside $\mathcal{R}$. As $\mathcal{R}$ is convex, the intersection, if it exists is a single line segment ($\overline{a'b'}$) defined by two intersection points a', b' whose set is denoted as P=$\{a', b'\}$. If it exists, the intersection segment is described as

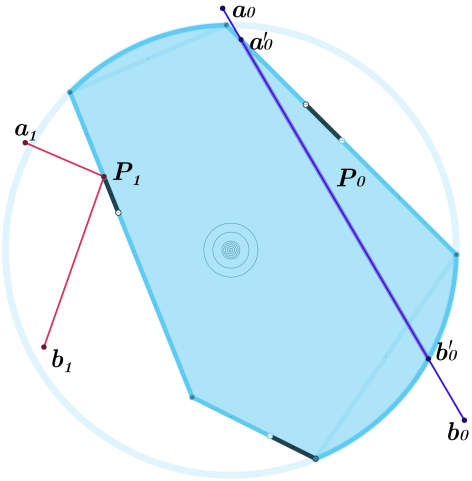


Figure 3.1: Showcase of the PNP problem. (Darker edges are obstructed)

$\overline{a'b'} = \mathcal{R} \cap \overline{ab}$. In special cases, a single-point intersection with an arc edge or vertex is possible.

In simple terms, the algorithm works by iteratively going through the cones of $\mathcal{R}$, checking if an intersection occurred within, and finding the specific points if it did. This approach is made more efficient by using additional rules to dismiss some instances quickly. For example, if the line segment a, b does not intersect the visibility circle, no intersection with $\mathcal{R}$ can occur. Alternatively, if $\mathcal{R}$ is circular, the line circle algorithm from [12] is used. When both points $a, b \in \mathcal{R}$ we know that $P = \{a, b\}$.

The detailed description of Alg. 3.2 is as follows. In the beginning, the algorithm checks if an intersection is possible (lines 1- 2) by checking if the point on the line segment closest to $s(\mathcal{R})$ is within $r(\mathcal{R})$. Otherwise, an intersection is impossible, and $P = \emptyset$. Following this, a *line circle intersection* algorithm is used to find the intersection points P_c between the line segment and neighborhood visibility circle (line 3); these are relevant for the arc edges of the neighborhood. If the neighborhood consists solely of arc edges, the found points are equivalent to the neighborhood intersection points (line 4) and $P = P_c$. The *Orient* (orientation predicate) [30] [2] function returns the position of the third point relative to the line segment defined by its first two arguments. On line 5 points a, b are swapped such that $s(\mathcal{R})$ always lies to the *left* of the oriented line segment $\overline{ab}$; this is a necessary assumption for finding intersections in the main cycle of the algorithm. If the line segment points a, b lie in the neighborhood, they are also intersection endpoints and, as such, are added to P (6). The *line_inside* variable is set based on point a in preparation for the main algorithm loop. This variable is used to

[2] The orientation predicate is often used when numerical robustness is required. In our algorithms, an adaptive robust implementation [30], optimized for speed and robustness, is used.

Algorithm 3.1: Segment-neighborhood intersection (SNI)

Input: $\mathcal{R}$ ... neighborhood, a, b ... points defining line segment
Output: P ... intersection points

```
1  d_abs ← shortest distance of ab to neighborhood seed
2  if d_abs > r(R) then return ∅
3  P_c ← intersection points between line a,b and circle(s(R), r(R))
4  if R is circle then return P_c
5  if Orient(a, b, s(R)) = right then Swap a and b
6  if a or b ∈ R then add them to P
7  line_inside ← bool[a ∈ R]^a
8  Find cone c_i such that a ∈ c_i
9  while b ∉ c_{i-1} do
10     if t(e_i) = a and ∃p_c ∈ P_c : p_c ∈ c_i then add p_c to P
11     intersection ← false
12     switch Orient(a, b, v_i) do
13         case Right do intersection ← not line_inside
14         case Left do intersection ← line_inside
15         case Collinear do add v_i to P; line_inside ← [(v_i + Δb_dir) ∈ R]^b
16     if intersection then
17         line_inside ← not line_inside
18         if t(e_i) ≠ a then
19             add intersection point between ab and e_i to P
20     i ← (i + 1) mod N
21 return P
```

[a] Iverson bracket
[b] See fig. 3.2

track if the line is inside (true) or outside (false) $\mathcal{R}$ in the current cone c_i (line 7); this is then used to find where the line intersects. Next, the current cone is initialized as the one containing point a (line 8).

The lines from 9 to 20 constitute the main loop of the algorithm, which, starting from the cone containing point a iterates through subsequent cones detecting intersections until point b is found. At the start of each cycle (line 10), if the edge in the current cone is an arc ($t(e_i) = a$), any points from P_c lying within are added, as on this edge the neighborhood equals the visibility circle (see points p_{c1}, p_{c2} in Fig. 3.2). Next, intersections on the current segment are detected by checking the relative position of the current vertex to the $\overline{ab}$ segment. With the direction of $\overline{ab}$ having been fixed beforehand, it is given that the line is inside if the vertex is to the right and outside if to the left. This information, combined

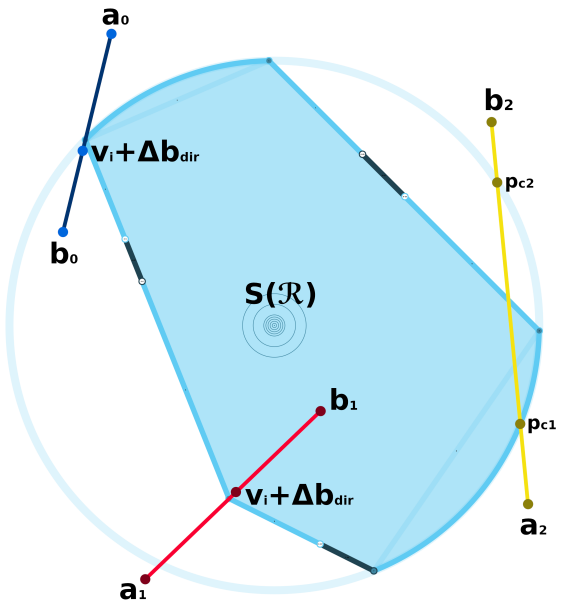


Figure 3.2: Showcase of different intersection cases

with the tracked current position of the line (*line_inside*), is used to detect when an intersection occurred (lines 12-15). In the edge case, $\overline{ab}$ intersects directly on the vertex, the vertex is added to P. As shown in Fig. 3.2 there are differing ways a line segment can intersect in a vertex. In such a case, the *line_inside* variable has to be determined based on a point offset in the direction of point b (line 15). When an intersection has been found on the current segment, the *line_inside* variable is inverted. As arc intersections have already been included, only line edge intersections need to be added (lines 18, 19). A line-line intersection algorithm is then used to find the intersection point.

Closest Point on Edge (CPE)

The second part of the PNP algorithm is Alg. 3.2, which is used when no intersection segment was found. The algorithm takes the neighborhood structure $\mathcal{R}$ and two points a, b and returns the point on the neighborhood edge, minimizing the distance to a and b. The CPE algorithm works by finding the closest edge. If the edge is an arc, the closest point is found using the *point circle point* (PCP) algorithm from Fanta (2021) [12]. As the name suggests, the PCP algorithm serves to find the point on a circular region, which minimizes the distance to two chosen points; the solution is derived from geometry. The code implemented for the thesis [12] is used here, with only minor modifications to fit our framework. The complete solution will not be described here, as it is explained in the original

Algorithm 3.2: Closest point on edge algorithm (CPE)

Input: $\mathcal{R}$ ... neighborhood, a, b ... points
Output: P ... closest point on neighborhood edge

1 **if** $\mathcal{R}$ *is circle* **then return** *closest point on circle*
2 $v_c \leftarrow \arg\min_{v_i}(l(a, v_i, b));\ \forall v_i \in V$
3 $p_{o1} \leftarrow$ point slightly offset from v_c in direction of v_{c-1}
4 $p_{o2} \leftarrow$ point slightly offset from v_c in direction of v_{c+1}
5 $d_c \leftarrow\ l(a, v_c, b),\ d_{o1} \leftarrow\ l(a, v_{o1}, b),\ d_{o2} \leftarrow\ l(a, v_{o2}, b)$
6 **if** $(d_c \leq d_{o1}$ ***and*** $d_c \leq d_{o2})$ **then return** v_c
7 **else**
8 **if** $(d_{o1} \leq d_{o2})$ **then** $e_{closest} \leftarrow e_{c-1}$
9 **else** $e_{closest} \leftarrow e_c$
10 **if** $t(e_{closest}) \neq a$ **then return** *closest point on line*
11 **else return** *closest point on circle*

work. If the edge is a line segment, the *point segment point* (PSP) is used. The PSP algorithm is derived in closed form from geometry and implemented using Shewchuck's adaptive robust geometry predicates [31].

If the neighborhood is entirely circular, the PCP algorithm is used (line 1). Otherwise, the vertex v_c closest to points a and b is found (line 2), and two close points on the neighboring edges are found. The goal is to approximate a derivative and find the trend of each edge, so points slightly offset from v_c are found on the neighboring edges, and their distances to $a,\ b$ are calculated (lines 3 - 5). If both edge points are farther than v_c, the vertex is returned as the closest point (line 6); otherwise, the edge with the closer point is selected as the *closest edge*. Next, depending on the type of edge (arc or line), either the PNP algorithm is used, or the closest point on a line is calculated (lines 10 - 11).

3.1.3 PNP with Obstacles

Until now, we assumed there was a direct line of sight from points a, b to $\mathcal{R}$. However, as shown previously, individual maps contain obstacles that need to be considered when finding the shortest path. Adapting the PNP algorithm to an environment with obstacles (see alg 3.3) is to find a path from the original points to the closest vertex in the neighborhood. Then, the points from which a direct line of sight to $\mathcal{R}$ exists are found, and the PNP algorithm is used with those.

Algorithm 3.3: PNP with obstacles

Input: $\mathcal{R}$ ... neighborhood, a, b ... points
Output: P ... closest point on $\mathcal{R}$

1 $v_c \leftarrow \arg\min_{v_i}(l(a, v_i, b));\ \forall v_i \in V$
2 $P_a \leftarrow$ shortest collision-free path from a to v_c
3 $P_b \leftarrow$ shortest collision-free path from b to v_c
4 $c \leftarrow$ last point before v_c on P_a
5 $d \leftarrow$ last point before v_c on P_b
6 **return** *PNP*$(c, \mathcal{R}, d)$

3.2 Constructive Heuristics

The first step for many optimization algorithms is to generate an initial solution that can be further improved. In our case, this means finding an initial tour and a corresponding sequence of neighborhood regions. The more important part is to have the correct order of neighborhoods, as individual points have less influence on the overall path length than the neighborhood sequence. Furthermore, the positioning of points can easily be optimized using the rubber-band algorithm. The following subsections cover the *nearest neighbor* and *insertion* based algorithms which have been used for generating initial tours.

3.2.1 Nearest Neighbor (NN)-based Methods

Nearest Neighbor (NN). One of the tested algorithms is an implementation of the *nearest neighbor* [32] algorithm (see Alg. 3.4). The algorithm input is a list of neighborhood seeds $s(\mathcal{R})$, which here function as an approximation of $\mathcal{R}$. The quality of the approximation is then dependent on the *visibility radius* d. Starting from a randomly chosen seed, the algorithm repeatedly selects the nearest unvisited seed. It then adds it to the tour, forming a solution by iteratively connecting seeds until all have been visited. This resulting tour is often close to but not guaranteed to be optimal. Another problem is that the space occupied by $\mathcal{R}$ is not considered, leading to the resulting path quality decreasing with a rising value of d.

Nearest Neighbor optimizing the second to last point (NN-SL). This is a variation on the *nearest neighbor* algorithm, which aims to improve upon it using PNP. The idea is to use PNP to find the point in each neighboring region,

Algorithm 3.4: Nearest Neighbor (NN)

Input: Set of M neighborhood regions $\boldsymbol{\mathcal{N}}$. Initial region $\mathcal{R}_1 \in \boldsymbol{\mathcal{N}}$.
Output: Sequence of neighborhood regions $\boldsymbol{\mathcal{R}} = \langle\, \mathcal{R}_i \,|\, 1 \leq i \leq M \,\rangle$.
Tour as a sequence of points $\boldsymbol{\mathcal{X}} = \langle\, x_i \,|\, x_i \in \mathcal{R}_i \,\rangle$.

1 $\boldsymbol{\mathcal{R}} \leftarrow \langle\, \mathcal{R}_1 \,\rangle$; $\boldsymbol{\mathcal{X}} \leftarrow \langle\, s(\mathcal{R}_1) \,\rangle$; $\boldsymbol{\mathcal{N}} \leftarrow \boldsymbol{\mathcal{N}} \setminus \{\mathcal{R}_1\}$
2 **for** $i \leftarrow 2, \ldots, n$ **do**
3 $\quad \mathcal{R}_i \leftarrow \underset{\mathcal{R} \in \boldsymbol{\mathcal{N}}}{\arg\min}\, l(s(\mathcal{R}_{i-1}), s(\mathcal{R}))$
4 $\quad$ Append $\mathcal{R}_i$ to $\boldsymbol{\mathcal{R}}$. Append $s(\mathcal{R}_i)$ to $\boldsymbol{\mathcal{X}}$. $\boldsymbol{\mathcal{N}} \leftarrow \boldsymbol{\mathcal{N}} \setminus \{\mathcal{R}_i\}$
5 **return** $\boldsymbol{\mathcal{R}}$, $\boldsymbol{\mathcal{X}}$

which is closest to the current endpoint. Then, the closest point is added to the tour. This way, the whole shape of $\mathcal{R}$ is considered.

The Alg. 3.5 takes a set of n neighborhood regions $\mathcal{N}$ as an input. The parameter k sets the number of tested regions before adding a point. Ideally, all regions not in the tour would be tested, but applying the PNP algorithm is not computationally negligible. The variable k represents the compromise between solution quality and the algorithm's speed. The algorithm's output is a path represented by a sequence of points $\mathcal{X}$ and a corresponding sequence of $\mathcal{R}$, such that $x_i \in \mathcal{R}_i$. The loop at line 4 loops through the k closest regions based on the distance of their seed $s(\mathcal{N})$ to the last point on the tour. Then, the closest point on that region is found (line 5). Following that (line 7), the second to last point is optimized based on the position of q_i. In the lines 8 and 9, the length between the last points of the tour is measured. If the found length is smaller than the current minimum, the current closest point x_i is updated (lines 10 to 12). In the end, the second to last point is updated (line 13), and the closest found point and corresponding neighborhood are added to the tour (line 14). Throughout the iterations, unvisited neighborhoods are tracked in $\boldsymbol{\mathcal{N}}$ and are moved to $\boldsymbol{\mathcal{R}}$ once visited. Based on its optimization of the second to last point position, it is called the PNP-SL algorithm.

Nearest Neighbor using PNP (NN-PN). A simplified version of the previous algorithm that does not deal with optimizing points already in the tour. In essence, the only difference to the NN algorithm is the use of PNP (on the k closest regions) to find the closest point, which is then added to the tour. The algorithm is the same as 3.5, except without lines 7,12 and 13. Then on line 8 only the length $l(x_{i-1}, q_i)$ is measured.

Algorithm 3.5: NN with PNP

Input: Set of M neighborhood regions $\boldsymbol{\mathcal{N}}$. Initial region $\mathcal{R}_1 \in \boldsymbol{\mathcal{N}}$.
Parameters: How many nearest regions to check k.
Output: Sequence of neighborhood regions $\boldsymbol{\mathcal{R}} = \langle\, \mathcal{R}_i \,|\, 1 \leq i \leq M \,\rangle$.
Tour as a sequence of points $\boldsymbol{\mathcal{X}} = \langle\, x_i \,|\, x_i \in \mathcal{R}_i \,\rangle$.

1 $\boldsymbol{\mathcal{R}} \leftarrow \langle\, \mathcal{R}_1 \,\rangle$; $\boldsymbol{\mathcal{X}} \leftarrow \langle\, s(\mathcal{R}_1) \,\rangle$; $\boldsymbol{\mathcal{N}} \leftarrow \boldsymbol{\mathcal{N}} \setminus \{\mathcal{R}_1\}$
2 **for** $i \leftarrow 2, \ldots, n$ **do**
3 $l_{min} \leftarrow \infty$
4 **for** k nearest $\mathcal{N} \in \boldsymbol{\mathcal{N}}$ based on $l(x_{i-1}, s(\mathcal{N}))$ **do**
5 $q_i \leftarrow \mathrm{PNP}(x_{i-1}, \mathcal{N}, x_{i-1})$
6 **if** $i > 2$ **then**
7 $q_{i-1} \leftarrow \mathrm{PNP}(x_{i-2}, \mathcal{R}_{i-1}, q_i)$
8 $l \leftarrow l(x_{i-2}, q_{i-1}, q_i)$
9 **else** $l \leftarrow l(x_{i-1}, q_i)$
10 **if** $l < l_{min}$ **then**
11 $x_i \leftarrow q_i$; $\mathcal{R}_i \leftarrow \mathcal{N}$; $l_{min} \leftarrow l$
12 **if** $i > 2$ **then** $x^{\star}_{i-1} \leftarrow q_{i-1}$
13 **if** $i > 2$ **then** $x_{i-1} \leftarrow x^{\star}_{i-1}$
14 Append $\mathcal{R}_i$ to $\boldsymbol{\mathcal{R}}$; Append x_i to $\boldsymbol{\mathcal{X}}$; $\boldsymbol{\mathcal{N}} \leftarrow \boldsymbol{\mathcal{N}} \setminus \{\mathcal{R}_i\}$
15 **return** $\boldsymbol{\mathcal{R}}$, $\boldsymbol{\mathcal{X}}$

3.2.2 Insertion-based Methods

Another family of algorithms tested were insertion-based methods [32]. In [32], multiple insertion methods are compared, including *Farthest Insertion*, *Nearest Insertion*, *Arbitrary Insertion*. From these, the *Farthest Insertion* (FI) and *Random (Arbitrary) Insertion* (RI) algorithms were chosen, as the FI algorithm was shown to give generally better results than the nearest insertion algorithm, and the RI algorithm gave results faster, allowing for more time for subsequent optimizations.

Farthest Insertion (FI). The FI algorithm works by maintaining a closed tour, into which points are iteratively added, such that the insertion's cost (increase in length) is minimized. For the FI algorithm, the $s(\mathcal{R})$ are added sequentially, starting from the farthest ones from the $s(\mathcal{R})$ already in the tour. The Alg. 3.6 has inputs and outputs equivalent to the other constructive heuristics. First (lines 1, 2), the tour is initialized as the two neighborhood region seeds farthest from one another. Then, new points and regions are added until all regions are part of the tour ($\boldsymbol{\mathcal{N}} = \emptyset$). This is done by selecting the region farthest from all regions in the tour $\mathcal{R}_f$ (line 4) and finding the insertion position that minimizes the increase in length (line 5).

Algorithm 3.6: Farthest Insertion algorithm

Input: Set of M neighborhood regions $\boldsymbol{\mathcal{N}}$
Output: Sequence of neighborhood regions $\boldsymbol{\mathcal{R}} = \langle\, \mathcal{R}_i \,|\, 1 \leq i \leq M \,\rangle$.
Tour as a sequence of points $\boldsymbol{\mathcal{X}} = \langle\, x_i \,|\, x_i \in \mathcal{R}_i \,\rangle$.

1 $\mathcal{R}_{f1}, \mathcal{R}_{f2} \leftarrow \underset{\mathcal{R}_i, \mathcal{R}_j \in \boldsymbol{\mathcal{N}}}{\arg\max}\, l(s(\mathcal{R}_i), s(\mathcal{R}_j))$
2 $\boldsymbol{\mathcal{R}} \leftarrow \langle \mathcal{R}_{f1}, \mathcal{R}_{f2} \rangle$; $\boldsymbol{\mathcal{X}} \leftarrow \langle s(\mathcal{R}_{f1}), s(\mathcal{R}_{f2}) \rangle$; $\boldsymbol{\mathcal{N}} \leftarrow \boldsymbol{\mathcal{N}} \setminus \{\mathcal{R}_{f1}, \mathcal{R}_{f2}\}$;
3 **while** $\boldsymbol{\mathcal{N}} \neq \emptyset$ **do**
4 $\quad \mathcal{R}_f \leftarrow \underset{\mathcal{R}_i \in \boldsymbol{\mathcal{N}}}{\arg\max}\, l(s(\mathcal{R}_i), s(\mathcal{R}_j))$, for $\forall \mathcal{R}_j \in \boldsymbol{\mathcal{R}}$
5 $\quad x_{insert} \leftarrow \underset{x_i \in \boldsymbol{\mathcal{X}}}{\arg\min}\, l(x_i, s(\mathcal{R}_f), x_{i+1}) - l(x_i, x_{i+1})$
6 $\quad \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{X}} \leftarrow$ insert $\mathcal{R}_f, s(\mathcal{R}_f)$ at position $insert + 1$; $\boldsymbol{\mathcal{N}} \leftarrow \boldsymbol{\mathcal{N}} \setminus \{\mathcal{R}_f\}$;
7 **return** $\boldsymbol{\mathcal{R}}$, $\boldsymbol{\mathcal{X}}$

Random (Arbitrary) Insertion (RI). The RI algorithm differs from the FI algorithm only in how the insertion points are selected. The difference in Alg. 3.6 is only in line 4, where it is chosen randomly instead of selecting the farthest $\mathcal{R}$. As shown in [33], this generally leads to worse results, but the algorithm was included to have a comparison to the FI algorithm.

FI-PNP. An implementation using the PNP algorithm was also tested because the FI and RI algorithms only approximate the neighborhood regions by their seeds, and this approximation declines in quality with the rising visibility radius. The implementation has the same initialization and selection of neighborhoods as FI (Alg. 3.6) but differs in the way insertion points are added. On line 5 instead of comparing the length to $s(\mathcal{R})$ it is compared to $\mathrm{PNP}(x_i, s(\mathcal{R}_f), x_{i+1})$, i.e. the point from $\mathcal{R}_f$ closest to x_i, x_{i+1}. Then, the closest such point is added to $\boldsymbol{\mathcal{X}}$. This way, the whole shape of $\mathcal{R}$ is considered instead of just the approximation in the form of $s(\mathcal{R})$.

While later experiments showed that, on average, the FI-PNP gave the best results from the implemented constructive heuristics, they also showed that the repeated PNP calls ($\sum_{i=1}^{M-3} i$) significantly slowed down the algorithm. This could possibly be alleviated by implementing the k parameter similarly to how it was done in NN-PN. However, no such tests were made, as there was no time for additional experiments in this part of the thesis.

Algorithm 3.7: Rubberband

Input: Sequence of neighborhood regions $\boldsymbol{\mathcal{R}} = \langle \mathcal{R}_i \,|\, 1 \leq i \leq n \rangle$.
Initial tour to be optimized as a sequence of points $\boldsymbol{\mathcal{X}} = \langle x_i \,|\, x_i \in \mathcal{R}_i \rangle$.
Parameters: Region ID schedule $\boldsymbol{\mathcal{S}}$. Max iterations limit k_{max}.
Improvement threshold ϵ_{imp}
Output: Optimized tour $\boldsymbol{\mathcal{X}}$.

```
l_min ← Length(𝒳)
for k_max iterations do
    for i ∈ 𝒮 do x_i ← PNP(x_{i−1}, ℛ_i, x_{i+1})
    if l_min − Length(𝒳) < ε_imp then break
    l_min ← Length(𝒳)
return 𝒳
```

3.3 Improving Heuristics

After an initial solution is obtained from the constructive heuristics, it is then further refined via improving heuristics. This can be imagined as iterating towards a local minimum in the solution space. The implemented algorithms are the *rubber-band algorithm* (RB) [13], which iteratively applies PNP to shorten the tour without changing the sequence of neighborhood regions. However, as the initial neighborhood region sequence is not guaranteed to be optimal, further algorithms, such as 2-OPT [24], OR-opt [24], and Gutin neighborhood [34], were also implemented. Furthermore, the *variable neighborhood descent* (VND) [24] heuristic was used to combine our improving heuristics to locally search the solution space for an optimal tour that solves the TSPN problem.

3.3.1 Rubber-band (RB)

The *rubber-band algorithm* (RB) [13] works by dynamically adjusting a path to navigate around obstacles in a way that mimics a rubber band's flexibility, minimizing its length. Alg. 3.7 takes a fixed sequence of neighborhood regions $\boldsymbol{\mathcal{R}}$ and the tour to be optimized $\boldsymbol{\mathcal{X}}$. The algorithm then uses PNP to iteratively optimize the position of individual points while respecting the condition that $x_i \in \mathcal{R}_i$. The optimization ends when the maximum number of iterations k_{max} has been reached or if the length improvement falls below the improvement threshold ϵ_{imp}. The algorithm brings differing degrees of improvement depending on the schedule $\boldsymbol{\mathcal{S}}$ in which individual points are optimized. Three schedules have been implemented and tested:

- Sequential: $\boldsymbol{\mathcal{S}} = \langle 0, 1, ...,$N-1$\rangle$
- Permutation: e.g. $\boldsymbol{\mathcal{S}} = \langle 21, 3, ..., 12\rangle$
- Odd-even: $\boldsymbol{\mathcal{S}_{odd}} = \langle 1, 3, ...,$N-3,N-1 $\rangle$, $\boldsymbol{\mathcal{S}_{even}} = \langle 2, 4, ...,$N-2,N $\rangle$, first iterate over all odd points and then over all even points

3.3.2 Other Improvement Operators

This section contains various improvement operators, most of which work on the principle of changing the order of points in the tour and seeing if the tour was improved. As most of the algorithms tested required the length of the shortest tour between point pairs, it was beneficial to precompute the length for all point pairs and store it in a table. Given a sequence of neighborhood regions $\boldsymbol{\mathcal{R}} = \langle \mathcal{R}_i \,|\, 1 \leq i \leq M \rangle$, and tour as a sequence of points $\boldsymbol{\mathcal{X}} = \langle x_i \,|\, x_i \in \mathcal{R}_i \rangle$. A *path lengths* table is filled with the shortest lengths: $l(x_i, x_j)$, $1 \leq i, j \leq M$ for all possible point pairs. Where $l(x_i, x_j)$ is the length of the shortest collision-free path from point x_i to x_j.

The operators can be applied with two differing strategies. This being the *first* strategy where the first operation which brings an improvement is applied. And the *best* strategy, where first all possible operations are tested, and then the best is applied.

Neighboring swap. The neighbor swap operator is a cheap operator that iterates over all points in the tour and checks if the length can be reduced by swapping the order of two consecutive points. In mathematical notation, points x_i and x_{i+1} are swapped if:

$$l(x_{i-1}, x_{i+1}) + l(x_{i+2}, x_i) - l(x_{i-1}, x_i) - l(x_{i+1}, x_{i+2}) \leq 0$$

OR-opt. The OR-opt algorithm [35] tries to improve the tour by shifting a segment of k consecutive points to another point in the tour. So if the index i describes the start of the segment, k describes the length of the segment, and the index j the insertion position, it is possible to represent the resulting change in length as:

$$l(x_i, x_{i+k-1}) + l(x_{i+k}, x_{i-1}) + l(x_i, x_{j-1}) - l(x_i, x_{i-1}) - l(x_{i+k-1}, x_{i+k}) - l(x_j, x_{j-1})$$

For $1 \leq i, j \leq M$; $j \notin \langle i, ..., i + k\rangle$.

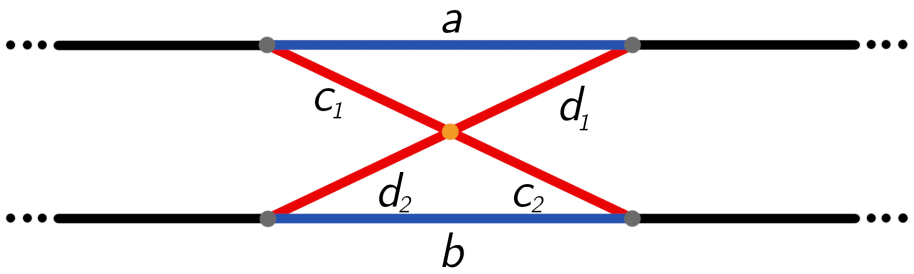


Figure 3.3: Tour shortened by replacing edge c and d by new edges a and b.

2-OPT. The 2-OPT [36] algorithm checks if the tour can be improved (shortened) by swapping pairs of edges. Alternatively, in a sequential tour, this is equivalent to reversing a segment from the tour. So the segment from point x_{i+1} to point x_j is reversed, if it is true, that:

$$l(x_i, x_j) + l(x_{i+1}, x_{j+1}) - l(x_i, x_{i+1}) - l(x_j, x_{j+1}) \leq 0$$

For $1 \leq i, j \leq M$; $i \neq j$.

Cross remove. As an optimal tour cannot cross with itself, the cross remove operator serves to replace any crossing edge pair with its non-crossing equivalent. This is based on triangle inequalities as shown in Fig. 3.3, where the crossing of edges c and d (red) are replaced by new edges a and b (blue). We get the equations $a < c_1 + d_1$ and $b < c_2 + d_2$ from the triangle inequality. The sum of these equations is then $a + b < (c_1 + c_2) + (d_1 + d_2)$, showing that the new edges are shorter than the old crossing ones. In contrast to the 2-OPT operator, which also removes crossings in its process, the *cross remove* algorithm directly targets crossings, removing the need to calculate changes in tour length.

Gutin neighborhood. An adaptation of the *Gutin neighborhood* [34] for GTSP as described in [37]. A subset of neighborhood regions is initially chosen and removed from the sequence in a way that ensures no two adjacent points are selected (see Alg. 3.8). This results in a set of unassigned neighborhood regions and the creation of 'holes' in the tour. The cost (length of the shortest collision-free path) of inserting any unassigned neighborhood region into any 'hole' is then calculated based on the formula $l(x_{i-1}, x_i*, x_{i+1})$. Where x_i* is the optimal point for insertion, found via the PNP algorithm as $x_i* = \text{PNP}(x_{i-1}, \mathcal{R}_i, x_{i+1})$. Then, the problem of assigning neighborhood regions to holes is solved using the Hungarian algorithm [38], minimizing the cost of the overall insertion. This optimization process allows for the simultaneous optimization of the neighborhood region sequence and point position.

Algorithm 3.8: Gutin neighborhood

Input: Sequence of neighborhood regions $\boldsymbol{\mathcal{R}} = \langle \mathcal{R}_i \,|\, 1 \leq i \leq M \rangle$.
Tour as a sequence of points $\boldsymbol{\mathcal{X}} = \langle x_i \,|\, x_i \in \mathcal{R}_i \rangle$.
Selection probability p
Output: Optimized NR sequence and tour $\boldsymbol{\mathcal{R}'}, \boldsymbol{\mathcal{X}'}$

1 $\boldsymbol{\mathcal{U}}, \boldsymbol{\mathcal{H}} \leftarrow \emptyset$
2 **for** $\mathcal{R}_i \in \boldsymbol{\mathcal{R}}$ **do**
3 **if** $R_{i-1} \notin \boldsymbol{\mathcal{U}}$ **then** with probability p add $\mathcal{R}_i$ to $\boldsymbol{\mathcal{U}}$ and i to $\boldsymbol{\mathcal{H}}$
4 **for** all combination of holes $h \in \boldsymbol{\mathcal{H}}$ and regions $\mathcal{U} \in \boldsymbol{\mathcal{U}}$ **do**
5 Find the optimal insertion point $x_{h\mathcal{U}*} \leftarrow \mathrm{PNP}(x_{h-1}, \mathcal{U}, x_{h+1})$
6 Calculate the insertion cost $l_{h\mathcal{U}}(x_{h-1}, x_{h\mathcal{U}*}, x_{h+1})$
7 Solve the assignment problem using the *Hungarian algorithm*
8 Assign optimal points and regions to holes in the tour $\boldsymbol{\mathcal{X}'}$ and sequence $\boldsymbol{\mathcal{R}'}$.
return $\boldsymbol{\mathcal{X}'}$, $\boldsymbol{\mathcal{R}'}$

3.3.3 Variable Neighborhood Descent (VND)

The *Variable neighborhood descent* (VND) [29] algorithm is an iterative optimization method employed for solving combinatorial optimization problems. VND differs from the other improvement operators in this section, as it is not a single operation applied to the tour but instead a framework for combining other improvement operators. While VND combines multiple heuristics, it is not classified as a metaheuristic because it does not have a way to leave local optima [24]. At its core, VND (see Alg. 3.9) explores multiple neighborhood structures (improvement operators) in search of an optimal solution. It iteratively evaluates optimal solutions within different improvement operators, exploiting local improvements until no further enhancements can be made. This is done by iterating over a given sequence of improving operators (see line 2) and returning to the beginning of the sequence if improvements were made. This process is repeated until no operator can improve the tour anymore.

3.4 Metaheuristics

If individual constructive or improving heuristics are not satisfactory, they can be combined into *metaheuristics*. As the improving heuristic only searches the local solution neighborhood, we combined the VND with a perturbation operator in the *Iterated Local Search* (ILS) [15] framework. ILS allows us to search a wider solution space and avoid being stuck in a local minimum.

Algorithm 3.9: Variable neighborhood descent (VND)

Input: NR sequence and tour $\boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{X}}$
Sequence of improvement operators $\boldsymbol{\mathcal{S}}$
Output: Optimized NR sequence and tour $\boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{X}}$

```
while tour_improved do
    for S ∈ S do
        R, X', tour_improved ← S(R, X)
        if tour_improved then break
return X, R
```

3.4.1 Iterative Local Search (ILS)

In order to consistently provide the most effective solution at any given time, we have chosen to implement the *iterated local search* (ILS) as the framework for our metaheuristic, as outlined in Algorithm 3.10 [15]. It operates by iteratively refining candidate solutions through a cyclical process of local improvement and perturbation. Initially, a solution is generated, followed by a local search to enhance its quality. Subsequently, the solution undergoes perturbation to diversify the search, and local search is applied again to explore the altered solution space. This iterative refinement cycle continues until a termination criterion is met. In our implementation the local improvement operator is the VND algorithm 3.9. A new tour is accepted if the length of the collision free path over all tour points ($l(\boldsymbol{\mathcal{X}})$) is decreased. And the process terminates if the last k iteration did not bring any improvement of the tour length.

A variation on the ILS algorithm, where the perturbation is applied multiple times in sequence, has also been implemented. As the perturbation operator is variable, this implementation is more akin to the *generalized variable neighborhood search* (GVNS) algorithm [24] than ILS. However, as ILS and GVNS are quite similar, we keep the name ILS for simplicity.

Perturbation operator / Double Bridge

Initially proposed for the Traveling Salesman Problem, the *double bridge* (DB) [24] move aims to enhance the exploration of the solution space by perturbing solutions through a series of edge replacement operations. The DB perturbation move is a 4-opt [39] algorithm, where new edges are added in a way that preserves their original orientation. This move randomly selects four edges, removes them,

Algorithm 3.10: Iterative local search (ILS)

Input: Set of neighborhood regions $\boldsymbol{\mathcal{N}}$
Parameters: Constructive heuristic $\mathcal{G}$
Sequence of improvement operators $\boldsymbol{\mathcal{S}}$
Number of successive not improving iterations to terminate k
Output: Optimized NR sequence and tour $\boldsymbol{\mathcal{X}}^*$, $\boldsymbol{\mathcal{R}}^*$

$\boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{X}} \leftarrow \mathcal{G}(\boldsymbol{\mathcal{N}})$
$\boldsymbol{\mathcal{R}}^*, \boldsymbol{\mathcal{X}}^* \leftarrow \text{VND}(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{X}})$
while Tour was improved in the last k iterations **do**
    $\boldsymbol{\mathcal{R}'}, \boldsymbol{\mathcal{X}'} \leftarrow \text{Perturbation}(\boldsymbol{\mathcal{R}}^*, \boldsymbol{\mathcal{X}}^*)$
    $\boldsymbol{\mathcal{R}}^{*\prime}, \boldsymbol{\mathcal{X}}^{*\prime} \leftarrow \text{VND}(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{X}})$
    if $l(\boldsymbol{\mathcal{X}}^{*\prime}) < l(\boldsymbol{\mathcal{X}}^*)$ **then** $\boldsymbol{\mathcal{X}}^* \leftarrow \boldsymbol{\mathcal{X}}^{*\prime}$, $\boldsymbol{\mathcal{R}}^* \leftarrow \boldsymbol{\mathcal{R}}^{*\prime}$
return $\boldsymbol{\mathcal{X}}^*$, $\boldsymbol{\mathcal{R}}^*$

and constructs new bridges between previously unconnected points. This way, an escape from local optima is facilitated, promoting the discovery of improved solutions.

The implementation is done through a series of segment reversals. The algorithm 3.11 takes four position indices c_1, d_1, c_3, d_3 which indicate the edges, which will be replaced. This way, the tour is split into four separate segments ABCD. The DB move is achieved by changing the order of the segments to ADCB; this could be done either by removing and creating links or by a series of reversals. In our case, the series of reversals is preferable due to the tour representation we are working with. The move is done via only three moves; a segment having its order reversed is indicated via a lover case letter. First, the segment {BC} is reversed (AcbD), then {cD} is reversed (AcdB), and lastly {cd} leading to the final order ADCB. A visual representation of the moves can be seen in Fig. 3.4.

In the GVNS-like implementation of the perturbation, the DB move was applied multiple times in a sequence. The number of applications starts at one and increases every time the VND algorithm does not improve upon the tour, up to a maximum of three. In the same way, if an improvement was found, the number of iterations decreased to a minimum of one. As previously mentioned, this is technically the GVNS algorithm, but the name ILS was kept within this thesis due to their similarity.

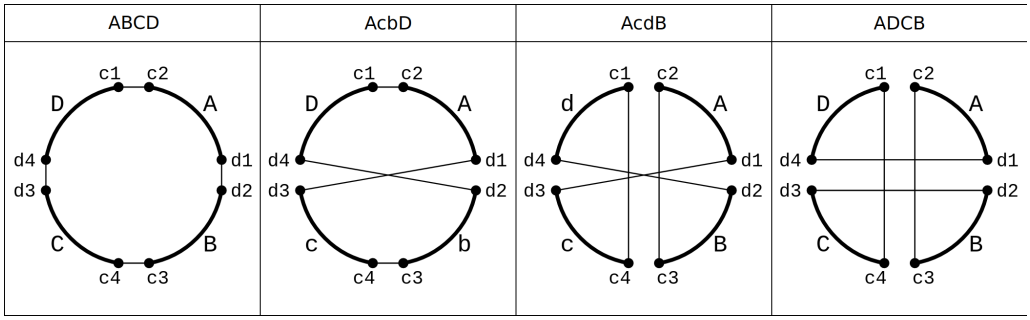


Figure 3.4: Double bridge move implementation through a sequence of segment reversals. [40]

Algorithm 3.11: Double Bridge

Input: Sequence of neighborhood regions $\boldsymbol{\mathcal{R}} = \langle \mathcal{R}_i \,|\, 1 \leq i \leq M \rangle$.
Tour as a sequence of points $\boldsymbol{\mathcal{X}} = \langle x_i \,|\, x_i \in \mathcal{R}_i \rangle$.
Bridge indexes c_1, d_1, c_3, d_3,
Output: Perturbed NR sequence and tour $\boldsymbol{\mathcal{R}'}, \boldsymbol{\mathcal{X}'}$

1 $c_2 \leftarrow c_1 + 1$, $d_2 \leftarrow d_1 + 1$, $c_4 \leftarrow c_3 + 1$, $d_4 \leftarrow d_3 + 1$
2 **for** both $\boldsymbol{\mathcal{R}}$ and $\boldsymbol{\mathcal{X}}$ **do**
3 Reverse segment d_2 to d_3
4 Reverse segment c_2 to c_4
5 Reverse segment d_3 to d_4
6 **return** $\boldsymbol{\mathcal{X}'}$, $\boldsymbol{\mathcal{R}'}$

Chapter 4

Computational Evaluation

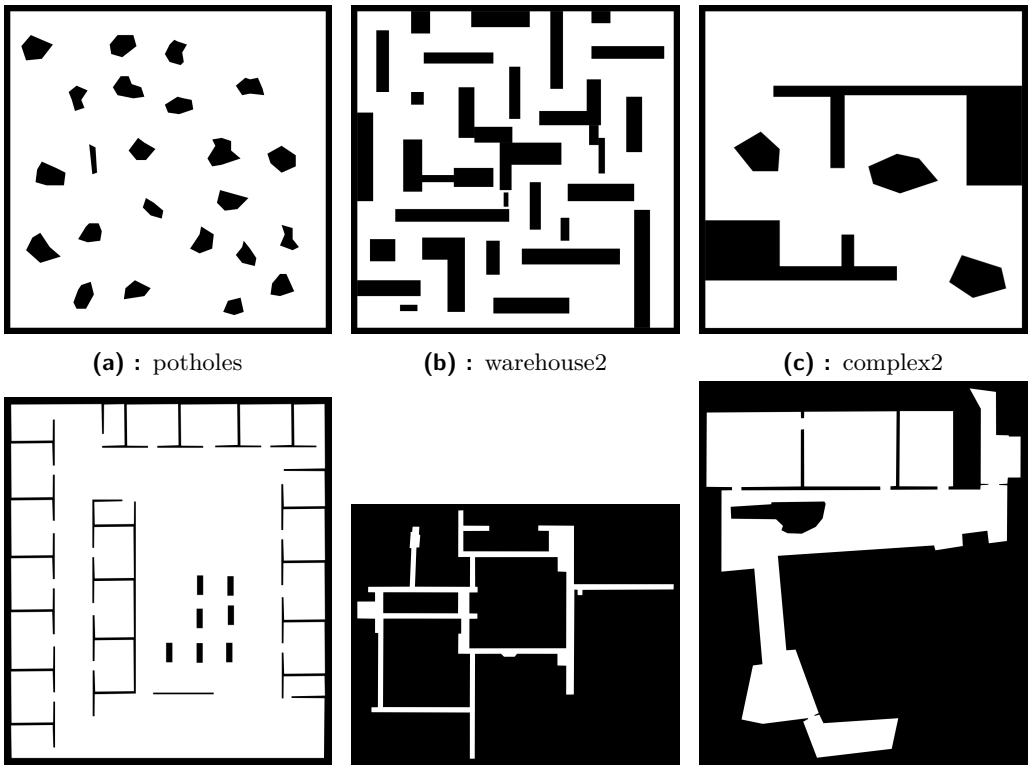
This chapter evaluates the reference and our heuristics (described in Chapt. 3) across varying instances. Furthermore, the best from our implementations are compared to the reference.

4.1 Evaluation Methodology

While we build upon a solution to the d-WRP [4], this thesis exclusively focuses on the TSPN [9] problem, which is solved as part of the d-WRP in the approach used in [8]. The input of the d-WRP is an environment (map) $\mathcal{W}$ and *visibility radius* d. The map is covered by overlapping neighborhood regions $\mathcal{R}$ so that full coverage is achieved; these regions form the TSPN instance solved in this thesis. The neighborhood regions $\mathcal{R}$ are generated with the use of a *dual sampling algorithm* [25], computing the *(d/2)-visibility regions* [26], and then finding the *maximally-covering convex subsets* of these regions. So a problem instance for our TSPN problem is an environment $\mathcal{W}$ and a set of neighborhood regions $\boldsymbol{\mathcal{R}} = \{\mathcal{R}_0, \mathcal{R}_1, ..., \mathcal{R}_{M-1}\}$. For a more in-depth description of the problem instances, see Chapt. 2.

For our experiments, seven maps were available: *complex2, jf-jh, jf-pb2, jf-ta2, potholes, warehouse2*, as shown in Fig. 4.1 and *empty*, which is an environment without obstacles. Furthermore, a set of *neighborhood regions* based on the visibility radii $d \in \{1, 1.5, 2, 3, 5, 10, 15, \infty\}$ were generated using the approach from [8] for each map. An example of the instances for different values of d is shown

in Fig. 4.2; here, the increase in the number of neighborhood regions $M = |\mathcal{R}|$ with decreasing values of d can easily be seen. With M being a significant factor increasing the computation time needed to solve the instance, instances with low visibility can take substantially longer. To make the timespan of our experiments feasible, instances with a visibility radius of one ($d = 1$) were omitted in our experiments. For some time-demanding experiments, other low-visibility instances were omitted; in such a case, it will be mentioned in the experiment description. However, even with these reductions, experiments were always done on a minimum of 30 instances.



(a) : potholes **(b) :** warehouse2 **(c) :** complex2

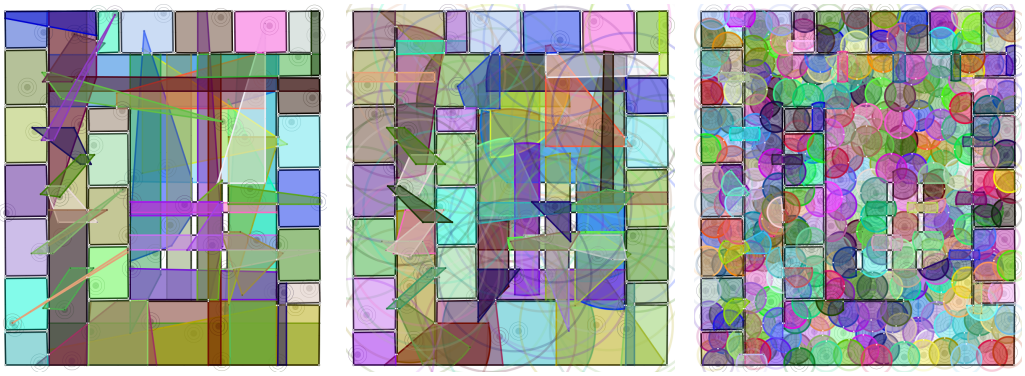
(d) : jf-jh **(e) :** jf-pb2 **(f) :** jf-ta2

Figure 4.1: Maps used for experiments.

The resulting path length L_X and computation time t vary significantly between individual instances, as we require to compare results between all instances, L_X and t are relativized with regard to the instances. For this, they are converted to relative values as: *percentage best-known solution gap* and the *percentage relative runtime* t_{rel} respectively:

$$gap(L) = \frac{L - L_{best}}{L_{best}} \times 100\%, \tag{4.1}$$

$$t_{rel}(t) = \frac{t - t_{min}}{t_{max} - t_{min}} \times 100, \% \tag{4.2}$$



(a) : $d = \text{inf}$, $M = 68$ **(b) :** $d = 10$, $M = 71$ **(c) :** $d = 2$, $M = 367$

Figure 4.2: Example instances of the *jf-jh* map for changing visibility radius d and the resulting quantity M of neighborhood regions.

where L_{best} is the minimal recorded path length for a given instance, and t_{min}, t_{max} are the minimal and maximal recorded times for that instance.

All experiments were run on a desktop computer with an Intel Core™ i7-7700 CPU (3.60GHz), 8 GB of RAM and running Ubuntu 22.04. The thesis code, namely PNP, rubber-band and ILS was implemented in the C++17 standard. We build upon the code base from [8] which implements the segmentation into neighborhood regions and the reference method and uses the libraries *Triangle* [41], *Clipper* [42]. The implementation of CPC (recall Subsec. 3.1.2) from [12] is used as part of our implementation of the PNP algorithm.

In the rest of this chapter, multiple parametrizations of the reference method are evaluated in Sec. 4.2. Then Sec. 4.3 describes how the robustness of the PNP algorithm was tested. In Sec. 4.4, the implemented schedules for the RB method are compared. Our implemented heuristics are evaluated in Sec. 4.5, and the best solutions are compared to the reference solutions in Sec. 4.6.

4.2 Reference Methods

To evaluate the quality of our methods, we first opted to evaluate the solutions of the reference method described in 2.3. As described in section 1.1, the goal is to improve upon the time-consuming step of discretizing the TSPN to GTSP which is part of the solution proposed in [8]. To be able to evaluate any improvement, a graph showing the change in solution quality relative to time is plotted for the reference solution parametrized by maximal sampling distance $d_{samp} = \{0.25, 0.5, 1, 2, 4, 8\}$. Here, d_{samp} is the maximal distance two neighboring

sample points can have from each other when sampling neighborhood region edges in the process of transforming the TSPN to GTSP. The minimal number of samples per $\mathcal{R}$ was capped at three, i.e., the actual sampling distance used is $d'_{samp} = min(d_{samp}, d\pi/3)$, where d is the visibility radius. The parametrizations of the reference method will be referred to as REF-0.5, REF-1, REF-2, REF-4, and REF-8, according to the parameter used.

The evaluation ran for four iterations with different random seeds, over all instances $d \in \{\infty, 15, 10, 5, 3, 2, 1.5\}$, $\mathcal{W} \in \{$*complex2, jf-jh, jf-ta2, potholes, warehouse2*$\}$, and parameters $d_{samp} \in \{8, 4, 2, 1, 0.5, 0.25^1\}$. As it turned out during evaluation, due to the size of map *jf-pb2*, the computation took multiple times longer than for other instances (see Fig. 4.3). Therefore, map *jf-pb2* was cut from the evaluation so as to be able to manage more iteration over other instances.

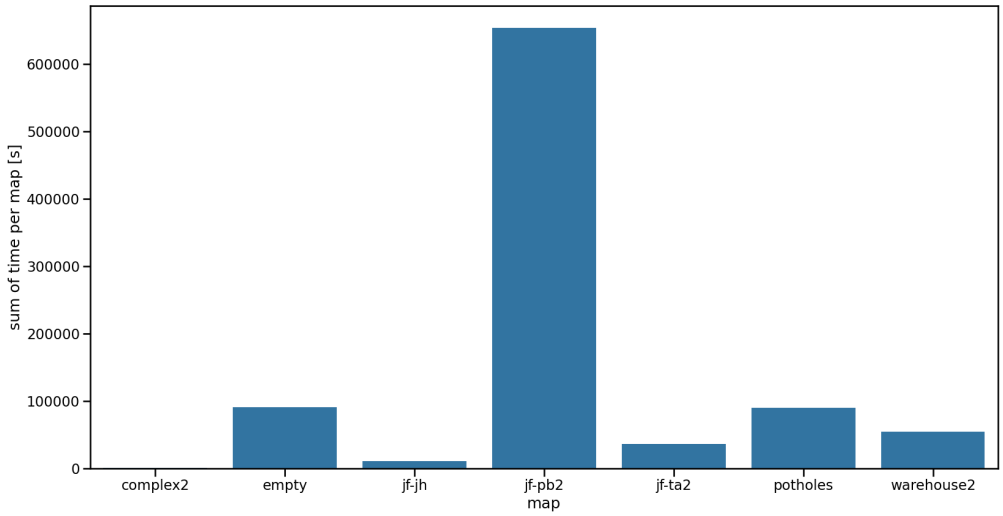
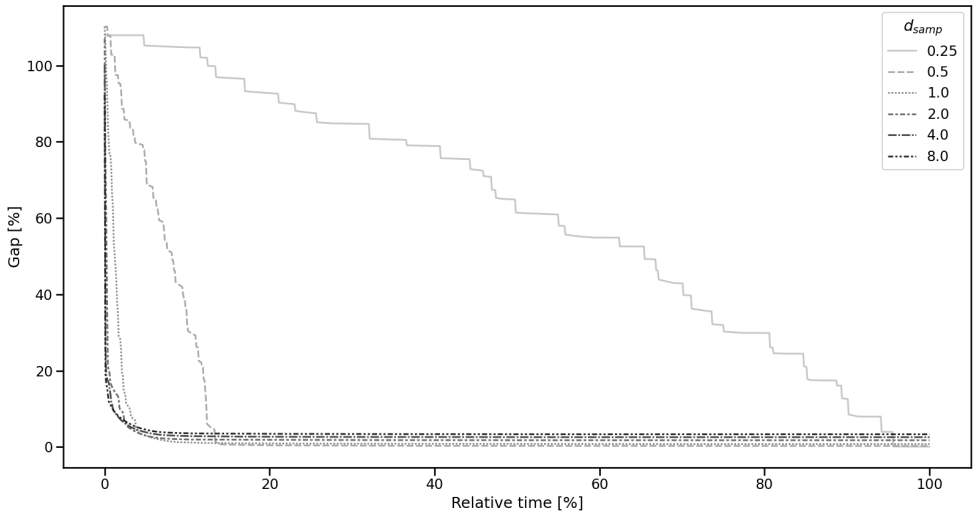


Figure 4.3: Sum initialization time of reference method per map.

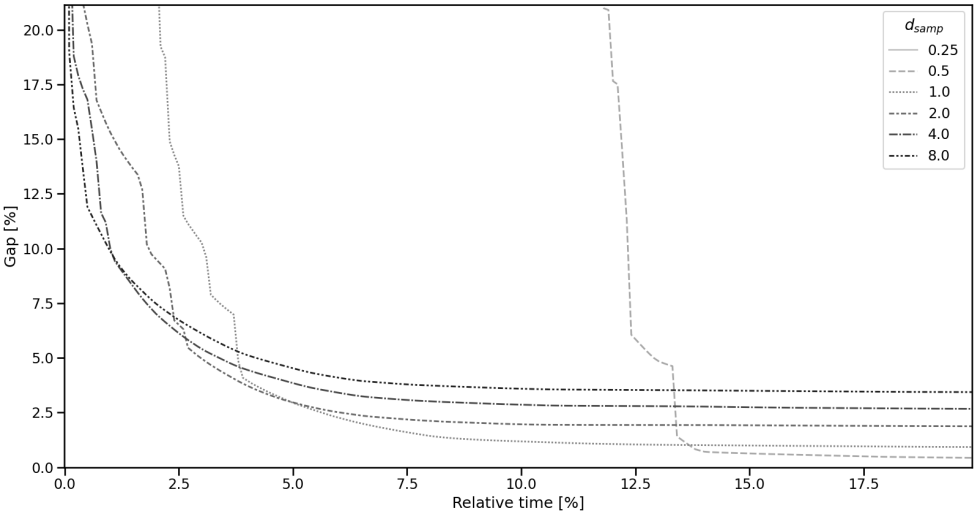
The algorithm was evaluated without any time restriction to track the evolution of the solution quality over time. The resulting average gap to t_{rel} graph is shown in Fig. 4.4a. Here, the increase in time taken with the number of samples can clearly be seen. Particularly, the reference method with $d_{samp} = 0.25$ parameter takes exponentially more time when compared to the other versions. As the difference in quality is minimal ($\approx 0.15\%$ between 0.5 and 0.25) when compared to the time taken, we opted to omit the parameter $d_{samp} = 0.25$ from further evaluation and only one iteration was done, and this parametrization was omitted from further comparisons.

Fig. 4.4b is a zoomed-in version of Fig. 4.4a. It focuses on the time phase

[1] $d_{samp} = 0.25$ ran only for one iteration due to high computational time demands.



(a) : Full



(b) : Zoom

Figure 4.4: Mean gap/t_{rel} graph of reference method parametrized by sample distance d_{samp}

where the parametrizations iterate toward their respective minima. The closer look shows the result that one would generally expect; with decreasing d_{samp} values (thus increasing the number of samples), the solution quality takes longer to settle, but the final quality improves.

4.3 Testing PNP

The implementation of the *point neighborhood point* PNP algorithm was non-trivial due to its complex nature and the requirement for exact geometry while working with floating-point arithmetic, making the implementation rather bug-prone. Therefore, we implemented tests to make sure PNP workes correctly. The correctness of the PNP algorithm implementation is tested by running the algorithm on individual neighborhoods and randomly generated test points in the environment. Recall that the PNP algorithm takes an input of two points a, b and a neighborhood region $\mathcal{R}$, and returns the point $p_{min} \in \mathcal{R}$ that minimizes the path length $l(a, p_{min}, b)$. Recall further that PNP is a combination of two algorithms, SLI, which detects intersections between the line segment $\overline{ab}$ and $\mathcal{R}$, and CPE, which finds the closest point on the neighborhood region edge if no intersection was found, see Sec. 3.1 for an in-depth description of PNP.

The *closest point on edge* (CPE) part of PNP was tested by selecting a neighborhood region and equidistantly sampling ($d_{samp} = 0.1$) all of its edges. Then, for 10,000 iterations, two random points were generated outside of $\mathcal{R}$. Then, the distance of the paths going through the closest sample point p_s and the point p_{CPE} found by CPE is compared. As the PNP algorithm operates on the continuous space, it is expected to always return a path of at least the same length as the path over the sample. With this, cases where the algorithm fails could easily be identified.

The *neighborhod line intersection* (NLI) part was tested by running 10 000 iterations with intersecting points and drawing their connection in black and the detected intersection in blue. This makes cases where an intersection was not found or was wrongfully found easily visible.

In the end, both CPE and NLI were tested for each map over 20 neighborhood regions of varying sizes without any faults being found. The combined PNP algorithm without obstacles also did not show any faults over 20 tested regions and 10,000 iterations each. Though as was previously mentioned in Subsec. 3.1.3, the algorithm 3.3 used for PNP with obstacles does not guarantee optimality; therefore, testing for PNP with obstacles was primarily comprised of visually evaluating the correctness of the found paths.

4.4 Testing Rubber-band

The following test aimed to determine the effect of the schedule $\boldsymbol{\mathcal{S}}$ on the final solution length and computational time of the *rubber-band* (RB) algorithm. As a reminder, the schedule $\boldsymbol{\mathcal{S}}$ is the order in which PNP is applied on the points in the tour, with the options being: points are chosen in *sequence*; points are chosen in *random* order; first all odd points are iterated over, then all even points (*odd even*). See Subsec. 3.3.1 for more information on the *rubber-band* algorithm.

Evaluation is done over the full dataset of instances $d \in \{\infty, 15, 10, 5, 3, 2, 1.5, 1\}$, $\mathcal{W} \in \{$*complex2, jf-jh, jf-ta2, potholes, warehouse2, jf-pb2*$\}$, with the NN as the constructor heuristic. The *random* schedule was run 20 times for each instance to generate a performance statistic. The gap is calculated relative to the shortest path found in each instance. Time is displayed as a percentage of the minimum and maximum value of time taken for the instance.

	sequence		**random**		**odd even**	
d [m]	Gap [%]	t_{rel} [%]	Gap [%]	t_{rel} [%]	Gap [%]	t_{rel} [%]
1.0	0.50 ± 0.19	36 ± 32	0.31 ± 0.28	49 ± 34	0.35 ± 0.26	37 ± 31
1.5	0.75 ± 0.36	72 ± 27	0.48 ± 0.36	79 ± 29	0.55 ± 0.39	80 ± 14
2.0	0.61 ± 0.42	66 ± 34	0.59 ± 0.43	78 ± 28	0.69 ± 0.37	77 ± 28
3.0	1.19 ± 1.05	59 ± 41	0.94 ± 0.94	75 ± 33	0.99 ± 1.28	86 ± 21
5.0	1.03 ± 1.23	65 ± 40	0.87 ± 0.98	72 ± 28	1.11 ± 1.31	65 ± 31
10.0	0.49 ± 0.90	45 ± 36	0.35 ± 0.58	47 ± 30	0.19 ± 0.17	39 ± 37
15.0	2.65 ± 5.35	44 ± 31	2.18 ± 5.24	57 ± 34	0.69 ± 0.92	60 ± 35
inf	0.48 ± 0.49	57 ± 36	0.38 ± 0.50	47 ± 33	0.40 ± 0.50	30 ± 26
Avg	0.97 ± 2.15	55 ± 37	0.77 ± 2.05	63 ± 34	0.62 ± 0.84	59 ± 35

Table 4.1: Comparison of the schedules $\boldsymbol{\mathcal{S}}$ for the rubber-band algorithm

Table 4.1 shows the resulting values and standard deviation of the gap and t_{rel}, averaged over all maps for individual values of the *visibility radius d*. As can be noticed, the standard deviation over individual maps is relatively high, meaning the result of the RB algorithm is highly dependent on the instance. Even though the average results showed only minor differences, the *odd even* schedule was chosen to be used in the RB algorithm, as it provides the best results on average.

4.5 Evaluating the Metaheuristic

In this section, the heuristics that build the ILS metaheuristic are evaluated, and variations of the ILS are tested.

4.5.1 Comparing Constructive Heuristics

To determine how to initialize our metaheuristic, six constructive heuristics NN, NN-PNP, NN-SL, FI, FI-PNP, RI (recall Sec. 3.2) were compared to initial solutions of the parametrized reference method, which is described in Sec 2.3. The initial solutions are the solutions given by our implemented generator heuristics and the first solution provided by GLNS for the reference method, with ten iterations of the RB algorithm being applied to each solution. The evaluation was done over instances $\mathcal{W} \in \{$*complex2, jf-jh, jf-ta2, potholes, warehouse2*$\}$ and $d \in \{1, 1.5, 2, 3, 5, 10, 15, \infty\}$.

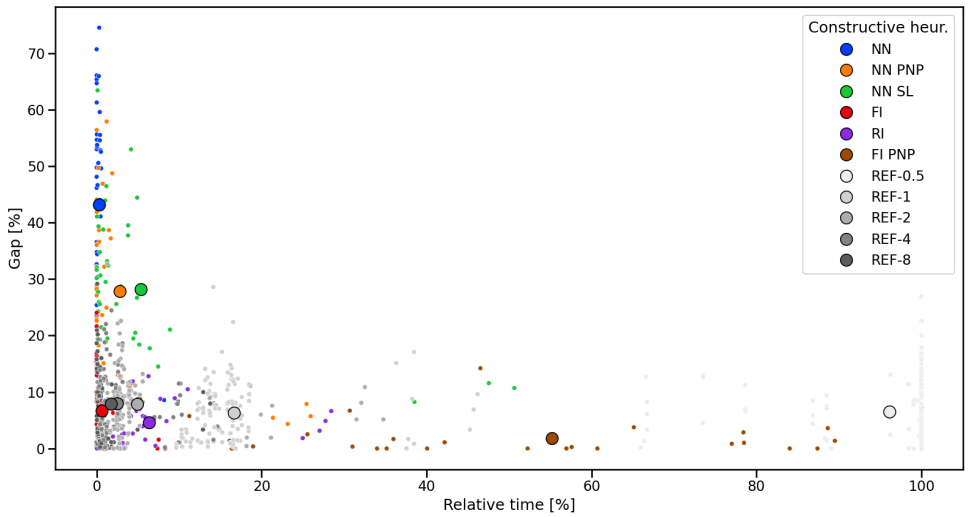


Figure 4.5: Initial solution time and quality comparison.

Figure 4.5 shows the relation between the initial solution gap and computational time, comparing our constructive heuristics to the parametrized reference method. Among our implemented methods, the *nearest neighbor* (NN) based approaches performed the worst, with the simple NN being the fastest yet yielding poor solution quality. The addition of the PNP algorithm to the NN methods (NN-PNP, NN-SL) improved the average gap by approximately 16%. However, this improvement came at the expense of increased time, making it comparable to the

reference methods. This trade-off is undesirable, as the primary objective of this thesis is to bypass the slow initialization step present in the reference method. On the other hand, the insertion-based methods showed superior performance. The farthest insertion (FI) method outperformed all reference methods in speed while achieving an initial solution quality comparable to the reference at both $d_{samp} = 0.5$ and $d_{samp} = 1$. The random insertion (RI) method yielded even better results, albeit with a slightly longer execution time than FI.

The variant of the FI algorithm incorporating the PNP algorithm delivered the best overall initial solution quality. Nonetheless, the repeated application of PNP significantly increased the method's execution time, making it impractical within the desired time constraints. Future work might benefit from considering an FI implementation with a selective application of the PNP algorithm to balance the trade-off between solution quality and execution time.

Based on these results, we selected the *farthest insertion* (FI) and *random insertion* (RI) methods for further evaluation, due to their superior solution quality and efficient execution times. Additionally, the *nearest neighbor* (NN) with PNP method was chosen to assess the performance of an NN-based approach within the complete context of the Iterated Local Search (ILS) implementation, see Subsec. 4.5.2.

4.5.2 Initial ILS Implementation

We evaluated our implementation of the *iterated local search* (ILS) metaheuristic [15], as detailed in Section 3.4.1, and plotted the change in solution quality over time analogous with the reference method. The entire ILS algorithm was assessed without a time constraint for the three selected constructive heuristics: FI, RI, and NN-PNP. The evaluation was conducted over four iterations of instances $\mathcal{W} \in$ *complex2, jf-jh, jf-ta2, potholes, warehouse2*$\}$ and $d \in 1.5, 2, 3, 5, 10, 15, \infty\}$. The sequence of improvement operators $\boldsymbol{\mathcal{S}}$ was based on the expected time demand of the operators as follows (from low to high): cross remove, neighboring swap, 2-OPT, OR-opt 1, OR-opt 2, OR-opt 3.

The average results are depicted in Figure 4.6. The initial algorithm evaluation results are confirmed here, with FI being the fastest, NN-PNP intermediate, and RI the slowest to provide initial results. In terms of post-initial solution, FI is also the quickest to improve, although at around 7% of the relative time, RI begins to provide better solutions. The NN-PNP method is the slowest overall in terms of improvement. Once RI overtakes FI, the relative solution quality ordering remains consistent, with final steady states of NN-PNP = 2.5%, FI = 2.2%, and

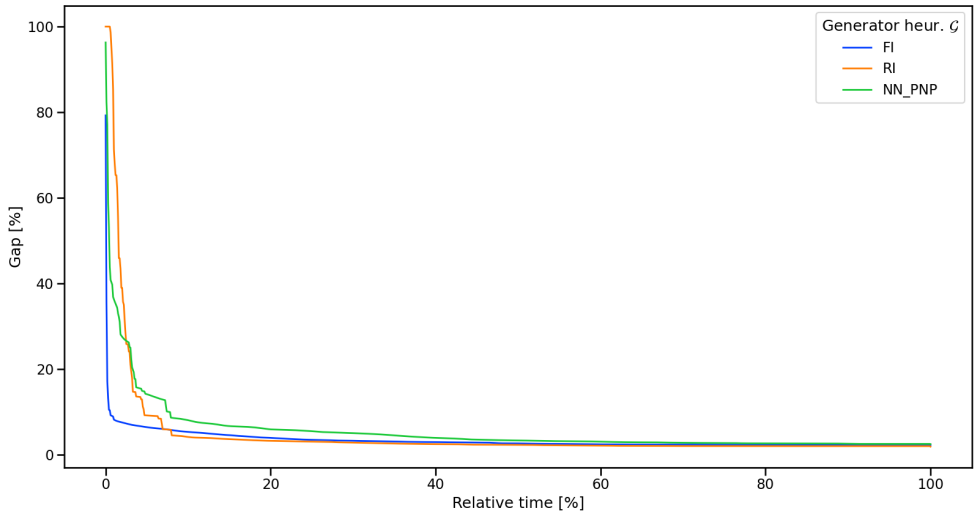


Figure 4.6: Mean gap / t_{rel} graph of the default ILS method

RI = 2.0%.

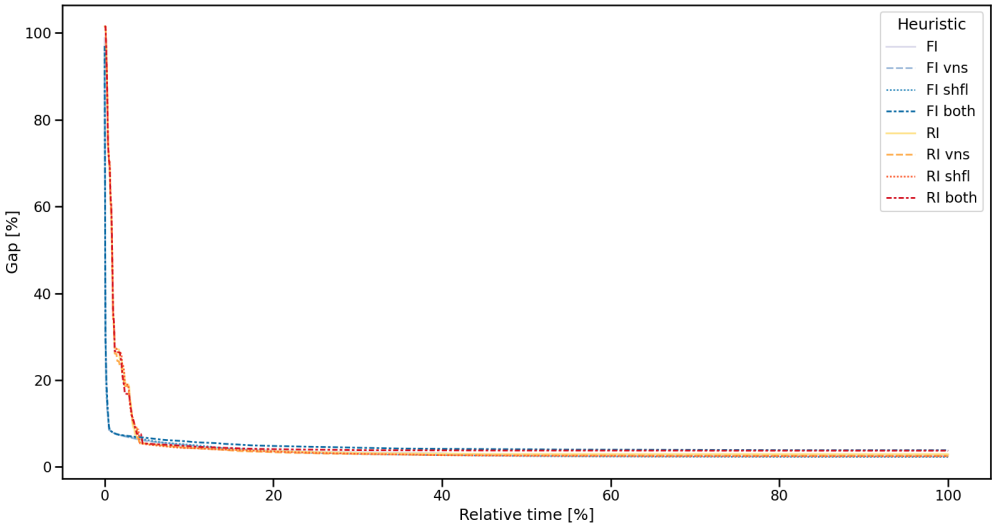
While the RI method yields the best average result, the improvement over FI is relatively small (approximately 1% at its peak). Thus, FI remains preferable due to its rapid improvements. However, as optimization time increases, the choice of constructive heuristic becomes less critical, as evidenced by all methods converging within a 0.5% gap of each other.

4.5.3 Expanding upon ILS

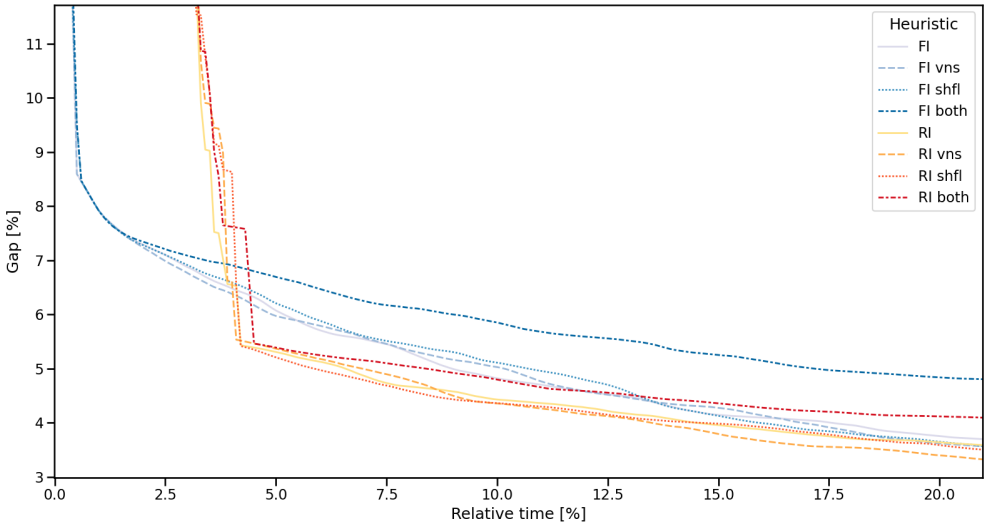
The results from the evaluation of the default Iterated Local Search (ILS) demonstrated a rapid initial improvement, particularly in the case of FI, but limited enhancement was achieved after that. Therefore, variations on the ILS implementation were introduced and tested to achieve further solution improvement. Given that the algorithm relies on beneficial perturbations to progress beyond the first local optimum, the initial modification involved adopting a more aggressive perturbation strategy. This concept, discussed in Subsection 3.4.1, involves applying the *double bridge* (DB) perturbation multiple times in sequence. The number of iterative applications ranged up to three, increasing if no improvement in path length was found and decreasing otherwise. As this approach varies the perturbation operation and, therefore, is more similar to the *variable neighborhood search* (VNS) [24] algorithm, it is referred to as the *vns* variation.

The second modification entailed altering the order of improvement heuristics

if the current sequence failed to enhance the path. As the reordering was done by randomly reordering the improvement heuristic sequence, this version was termed *shuffle*. Finally, a combination of both previous variations was evaluated, simply called *both*. Evaluation was done over four iterations of instances $\mathcal{W} \in \{$*complex2, jf-jh, jf-ta2, potholes, warehouse2*$\}$ and $d \in \{2, 3, 5, 10, 15, \infty\}$.



(a) : Full



(b) : Zoomed

Figure 4.7: Mean gap / t_{rel} graph of the expanded ILS methods

Fig. 4.7a presents a comparison of the ILS variations with the default methods. Due to the subtlety of the changes, Fig. 4.7b focuses on the early time results, which are of primary interest.

The first observation is that after some initial time, the variation with *both*

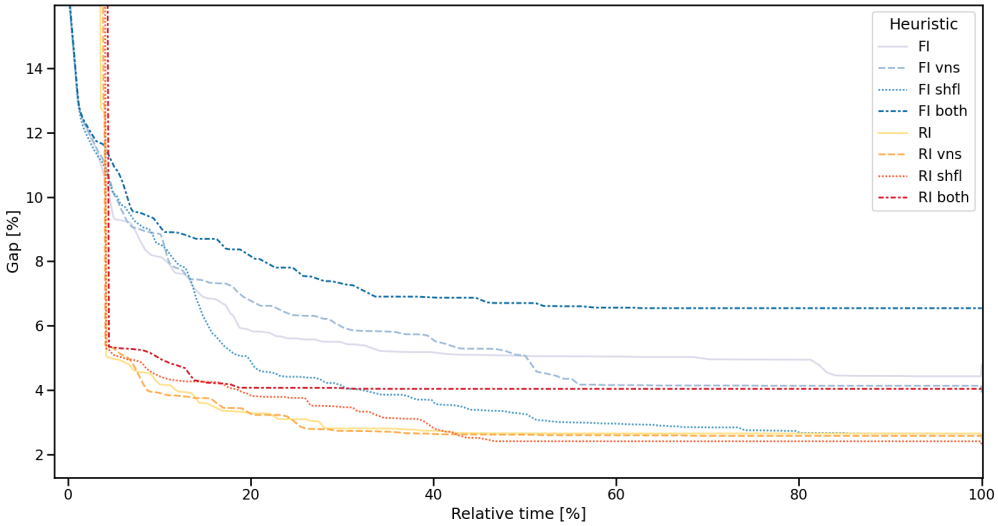
algorithms provides consistently worse results. With both *vns* and *shuffle* variations providing similar results to their respective unmodified methods, until about $t_{rel} \approx 16\%$ from where on both variations outperform the unmodified methods. For the FI-initiated methods, one approach does not outperform the other until 21 % t_{rel}, at which point the *shuffle* variation begins to give the best results. In the case of the RI method, the best method varies between the *vns* and *shuffle* variations until 39% t_{rel} where the *shuffle* variation starts giving better results. However, since the variations to the default method generally do not surpass 0.5% of the gap, the result does not appear overly significant.

Overall, the improvements introduced by the variations are relatively minor and are more evident over a more extended period. At the final convergence point, the overall best solution is provided by FI *shuffle* with a 2.3% gap, followed by RI *shuffle* and FI *vns*, both at 2.4% gap. This can be compared to the default RI method at 2.9% and FI at 2.7%. The combined (*both*) method under-performs compared to the default methods, with both variations settling at approximately a 3.8% gap.

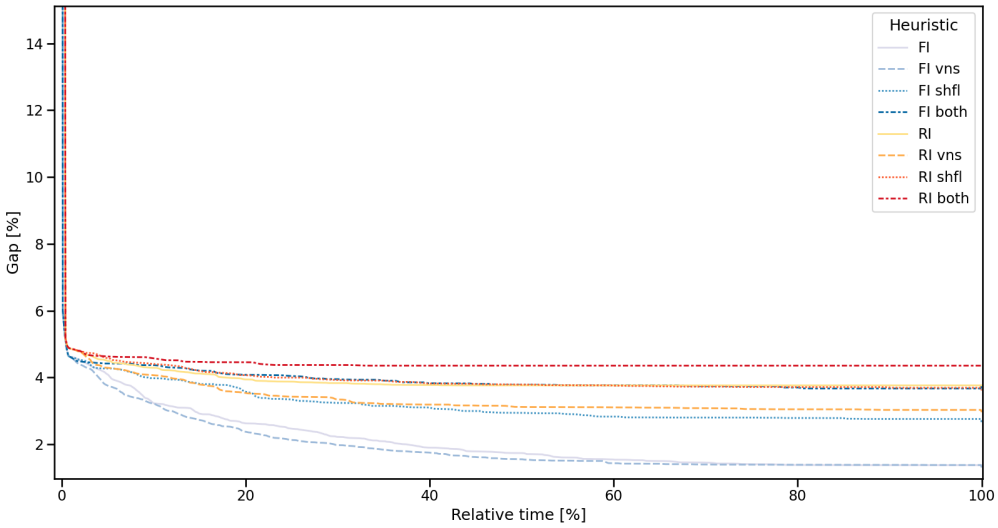
Given that the combined methods (*both*) consistently underperformed compared to both default methods, they can be disregarded as a viable avenue for further improvement. Aside from this, the variations only yielded marginal improvements over the long term. Therefore, we cannot definitively identify one method as an overall improvement over the others. However, the effectiveness of a given method also depends on the instance, as shown in Fig. 4.8, where the change in solution quality is shown for instances with a high (Fig. 4.8a) or low (Fig. 4.8b) visibility radius d. The figures plainly show how the variation FI *vns* is preferable in instances with low visibility. In contrast, instances with high visibility are better served with a method such as RI *vns*. However, again, only minor improvements were archived when compared to the default methods. Generally, it is worth considering one of the variations to achieve a longer, gradual improvement, potentially leading to a slightly enhanced final solution.

4.5.4 Gutin Neighborhood

Inspired by the method from [37], we tried adding the *gutin neighborhood* (GN) [34] to the sequence of improvement operators $\boldsymbol{\mathcal{S}}$ in the VND algorithm. As a reminder, the GN removes a random subset of non-adjacent points in a tour and then uses PNP to return the points into the sequence in a way that minimizes the tour length (see Subsec. 3.3.2). The sequence of operators $\boldsymbol{\mathcal{S}}$ in VND thus being: cross remove, neighboring swap, 2-OPT, OR-opt 1, OR-opt 2, OR-opt 3, gutin neighborhood. Evaluation was done over one iterations of instances $\mathcal{W} \in \{$*complex2, jf-jh, jf-ta2, potholes, warehouse2*$\}$ and $d \in \{2, 3, 5, 10, 15, \infty\}$.



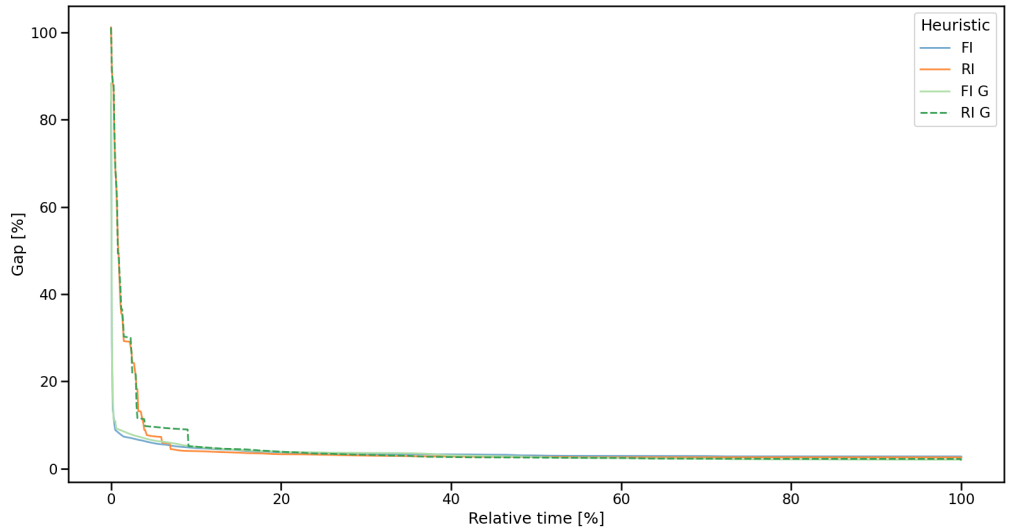
(a) : $d = \infty$



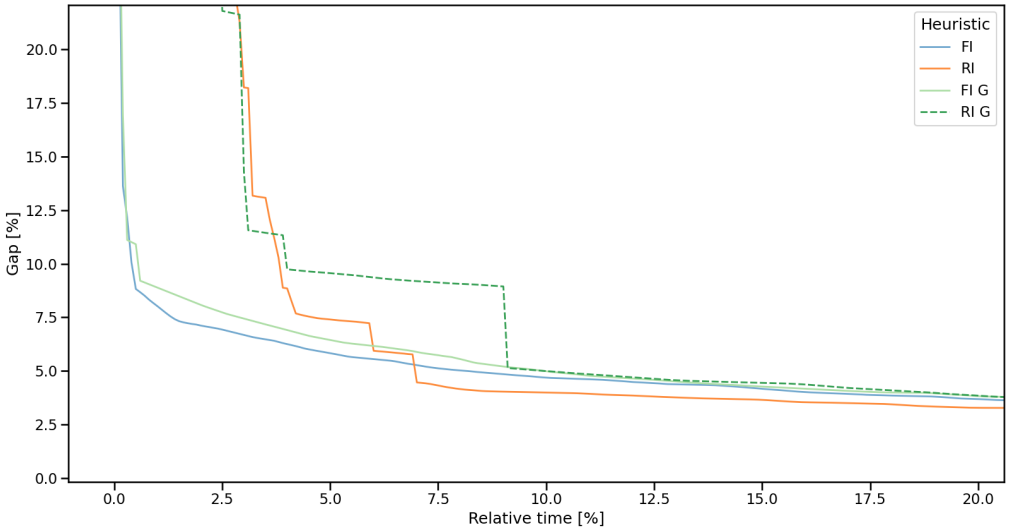
(b) : $d = 2$

Figure 4.8: Varying effectiveness of the expanded ILS methods depending on visibility radius d.

Fig. 4.9 shows the results compared to the default ILS implementation evaluated in Subsec. 4.5.2. As shown in the figures, compared to the default implementation, adding the GN does not bring faster improvement in our results. Quite the opposite, the version with GN improves slower, as shown in Fig.4.9b on the RI-initialized methods. The results are similar to the ILS variations evaluated in Subsec. 4.5.3, in that its improvement over the default method is more evident over a more extended period. Compared to the variations from Subsec. 4.5.3 the improvement brought is minimal (within 1% gap of the default ILS); therefore, the addition of GN will not be considered in the final comparison.



(a) : Full



(b) : Zoomed

Figure 4.9: Mean gap / t_{rel} graph comparing the default and gutin neighborhood results.

4.6 Final Comparison

This section compares the parametrized reference method presented in 4.2 to the algorithms selected in 4.5. The evaluation was done over the set of overlapping instances from all included evaluations. That being four iterations with changing random seeds of instances $\mathcal{W} \in \{$*complex2, jf-jh, jf-ta2, potholes, warehouse2*$\}$ and $d \in \{2, 3, 5, 10, 15, \infty\}$. The reference methods parametrized by sampling

distances $d_{smapl} \in \{0.5, 1, 2, 4, 8\}$ are compared. Our proposed methods included the default ILS version for both FI and RI and variations *vns* and *shuffle* for both.

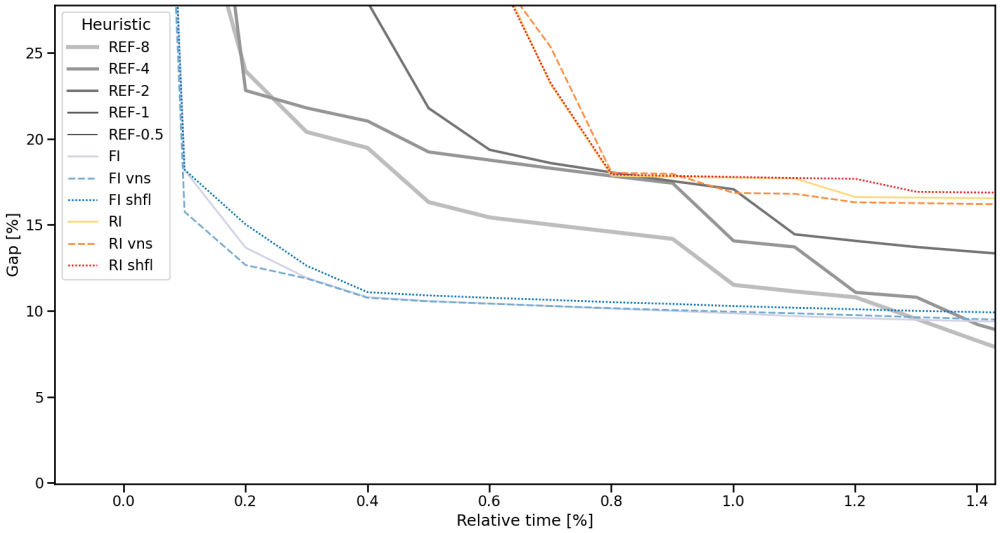
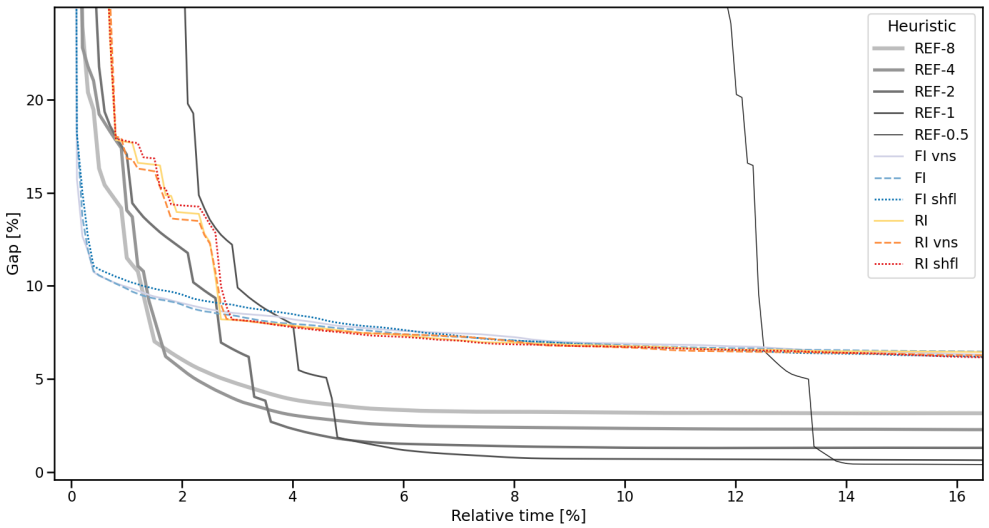


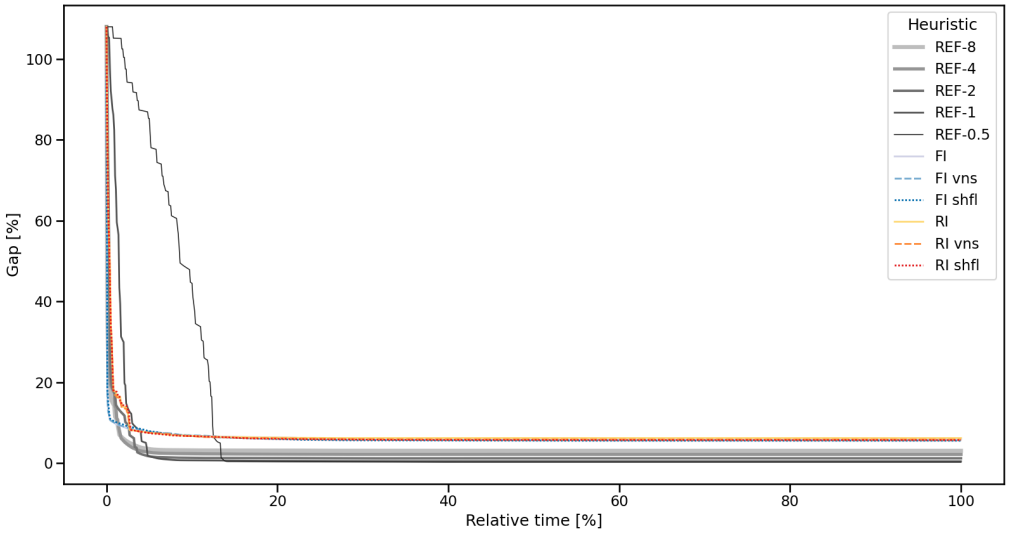
Figure 4.10: Comparison between our implemented metaheuristic and the reference for fast solutions.

Fig. 4.10 shows the mean gap values for the reference and ILS methods in the relative time for very early solutions. We can see that the metaheuristics initialized with RI are outperformed by the REF-8 REF-4 and relatively soon even by the REF-2 methods, even for such early solutions. On the other hand, the FI initialized metaheuristics outperformed all reference methods at this time interval. For example, at $t_{rel} = 0.2\%$, the FI metaheuristic is at 12% gap, compared to the 24% gap of REF-8, 22% gap of REF-4, and 73% gap of REF-2; the methods REF-1 and REF-0.5 did not yet start generating solutions at this time point. Regrettably, on a broader time scale, as illustrated in Figures 4.11a and 4.11b, the reference methods quickly catch up to and surpass our methods after the initial delay, continuing to improve further upon the solution. Although the initial improvement speed of both our methods and the reference methods is similar, once the reference methods are initialized, they achieve a higher degree of solution improvement. This indicates that while our current improvement heuristics are adequate, there is a need for a more effective strategy to escape local minima.

The improvement of our ILS methods begins to decelerate around a 10.7% gap for FI, compared to 7%, 6.2%, 2.6%, 1.8%, and 0.5% gaps for the reference methods in descending order of d_{samp}. Our proposed heuristics cannot currently leverage the lead gained by eliminating the conversion step from TSPN to GTSP present in the reference methods. At the final time, the solutions stabilized at a 0.25% gap for REF-0.5, 0.5% for REF-1, 1.3% for REF-2, 2.2% for REF-4, 3.1% for REF-8, with the best result from our heuristics being FI *shuffle* at a 5.6% gap.



(a) : Zoomed



(b) : Full

Figure 4.11: Comparison between our implemented metaheuristic and the reference for a wider time scale.

Though it can be seen from Fig. 4.12 if compared on instances with *visibility radius* $d = 2$ the FI method provides the best result until 2.2% t_{rel}. Showing that computations speed of ILS is impacted less by the increase in neighborhood region quantity, than the reference method.

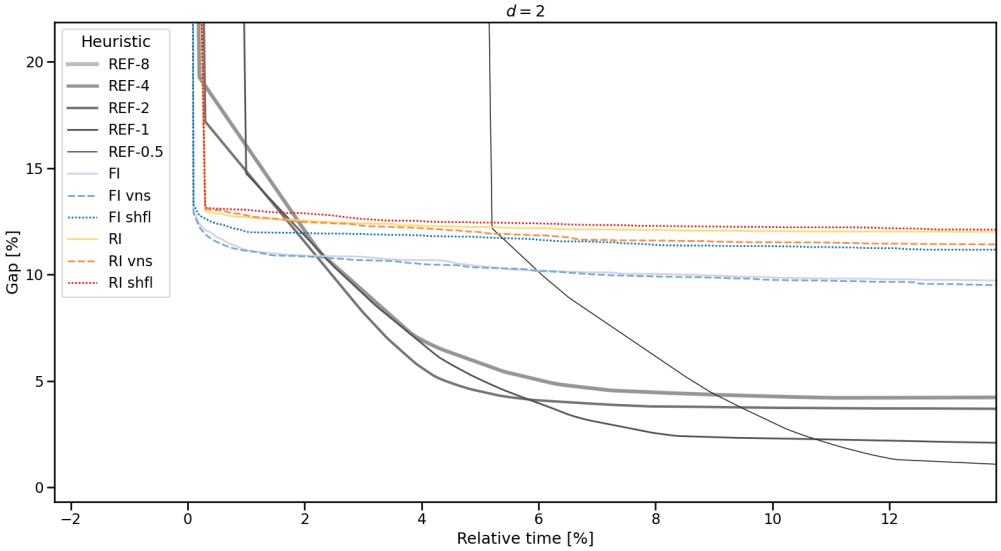


Figure 4.12: Comparison between our implemented metaheuristic and the reference for $d = 2$.

Chapter 5

Final remarks

5.1 Conclusions

This thesis proposed a new algorithm *point neighborhood point* (PNP) for finding the shortest path between a point-neighborhood-point triple for polygon-circle neighborhood regions $\mathcal{R}$. Before this thesis, such algorithms were only for purely circular [12] or polygonal [11] neighborhood regions. Tests showed that in an environment without obstacles, the results provided by PNP were always of the same or better quality than the results of a sample-based method, finely discretizing the region's boundary. The PNP implementation was extended to work on polygonal environments with obstacles, and the correctness was affirmed.

The PNP algorithm was used as part of a *rubber-band* (RB) [13] algorithm for the local improvement of tours over $\mathcal{R}$, which is used both as part of our proposed metaheuristic while also being used to improve the reference solution from [8]. In Subsec. 4.4 three parametrizations (see Subsec. 3.3.1) of the RB algorithm were compared across varying TSPN instances. The *odd even* parametrization provided the best results and was therefore used to improve the solutions of our proposed metaheuristic and the reference from [8], which previously used an algorithm that approximates the neighborhood regions with polygons. Our implementation of the RB algorithm works on the polygon circle directly, having reduced memory and computational effort requirements compared to the approximation.

The original idea behind this thesis was to avoid the discretization scheme involving the construction and solution of a GTSP instance and leverage the time

gained to optimize the solutions. However, as we did a comprehensive evaluation over five parametrizations of the reference (compared to two parametrizations in [8]), the reference method was discovered to perform better than expected. The results from Sec. 4.2 showing that solutions for higher sampling distances $d_{samp} \in \{8, 4\}$ are comparable to results from methods with denser sampling without such large initial delay. This meant our proposed methods had less time to optimize the solution than initially thought.

In Subsec. 4.5.1 the results from six proposed constructive heuristics were compared to the initial solutions (first solution from GLSN + RB) of the parametrized reference methods. The *nearest neighbor* (NN)-based methods showed the worst results, but we showed they could be improved via the PNP algorithm at the cost of increased computational time. The *farthest insertion* (FI) and *random insertion* (RI) methods provided results of a similar or better quality than the reference method, with FI also being faster than the reference. Also, the possibility of improving the FI method using the PNP algorithm was explored. However, while it provided the best overall results, the increase in required computational time made the method undesirable for our use.

The RB and other improving and constructive heuristics, which are described in Chapt. 3, were combined in a *iterative local search* (ILS) [15] metaheuristic. Several ILS parametrizations were experimentally evaluated in Sec. 4.5.

In Subsec. 4.5.2 the ILS metaheuristic with differing constructive heuristics was evaluated over a longer time scale. All versions of the ILS showed a fast rate of initial improvement but a limited capability to improve over a longer time period, indicating a lacking ability to escape from local minima. Overall, the ILS with the RI constructive heuristic provided the best results. However, the choice of the constructive heuristic became less critical with increasing computational time as all versions converged within 0.5% gap[1] of each other.

In Subsec. 4.5.3 we evaluate three variations to the ILS, which aim to make the method better able to escape from local minima. These variations were *vns*, which varied the strength of the perturbation operator, *shuffle*, which varied the order of improvement operators, and *both*, which employed both of the previous approaches simultaneously. The *both* variation provided consistently worse results than the other methods. With both *vns* and *shuffle* variations providing similar results to their respective unmodified methods, until about $t_{rel} \approx 16\%$ from where on both variations outperform the unmodified methods. Overall, the improvements introduced by the variations are relatively minor and are more evident over a more extended period. Therefore, we cannot definitively identify one method as an overall improvement over the other.

[1] Best-known solution gap. See Sec. 4.1.

In Subsec. 4.5.4 the addition of the *gutin neighborhood* (GN) [34] to the improvement operators was evaluated. The results were similar to the ILS variations previously evaluated in that its improvement over the default method was only visible over a more extended period. Compared to the variations from Subsec. 4.5.3 the improvement made was even less noticeable.

Our method and the reference were compared in Sec. 4.6. We showed that our FI-based ILS implementations can provide higher-quality solutions on a minimal computational budget. Regrettably, on a broader time scale, the reference methods quickly catch up to and surpass our methods after the initial delay, continuing to improve further upon the solution. While the initial improvement speed of both our methods and the reference methods is similar, once the reference methods are initialized, they achieve a higher degree of solution improvement. This indicates that while our current improvement heuristics are adequate, there is a need for a more effective strategy to escape local minima. Unfortunately, in its current state, our method cannot keep up with the reference over a longer time period. However, it was shown that the increasing number of neighborhoods impacts the speed of our methods slightly less than the reference.

While the final result did not prove as significant as initially hoped, various advances were made throughout the course of the thesis. The point PNP algorithm for polygon-circle neighborhoods was implemented and expanded to work in environments with polygonal obstacles. The PNP was used as part of a RB algorithm to improve tours with fixed neighborhood order. The RB was also used to improve the reference solution from [8]. An evaluation of parametrizations for both RB and the reference has been created. Moreover, a possible avenue for generating solutions to the TSPN problem was explored.

5.2 Suggestions for Possible Improvements

Although the initial solutions constructed by the FI heuristics are satisfactory, accelerating the FI-PNP heuristic warrants consideration, as it provided the best initial solution overall, even when compared to the references with dense sampling. It remains to be determined whether the superior initial solution would translate into an improved final solution after the ILS process or if it would similarly become trapped in a local minimum. Given that our initial constructive heuristics demonstrated improved performance compared to the reference initialization, evaluating the reference improvement algorithm on a tour constructed by our constructive heuristics might also be worthwhile.

The proposed ILS framework could benefit from further enhancements to escape local minima more efficiently. To this end, it might be advisable to evaluate the trade-off between the improvement achieved by the improvement heuristics, which are part of the VND, and the computational time they require. Additionally, exploring alternative perturbation heuristics may bring a better way to escape from local minima, thus offering another potential avenue for improvement.

Appendix A

Bibliography

[1] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," in *2010 IEEE International Conference on Robotics and Automation*, pp. 300–307, 2010.

[2] S. Macenski, F. Martín, R. White, and J. G. Clavero, "The marathon 2: A navigation system," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2718–2725, 2020.

[3] W. J. Cook, *In Pursuit of the Traveling Salesman*. Princeton: Princeton University Press, 2012.

[4] S. Ntafos, "Watchman routes under limited visibility," *Computational Geometry*, vol. 1, no. 3, pp. 149–170, 1992.

[5] A. Dumitrescu and C. D. Tóth, "Watchman tours for polygons with holes," *Computational Geometry*, vol. 45, no. 7, pp. 326–333, 2012.

[6] F. Li and R. Klette, "An approximate algorithm for solving the watchman route problem," in *Robot Vision* (G. Sommer and R. Klette, eds.), (Berlin, Heidelberg), pp. 189–206, Springer Berlin Heidelberg, 2008.

[7] T. Danner and L. Kavraki, "Randomized planning for short inspection paths," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, pp. 971–976 vol.2, 2000.

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

[9] E. M. Arkin and R. Hassin, "Approximation algorithms for the geometric covering salesman problem," *Discrete Applied Mathematics*, vol. 55, no. 3, pp. 197–218, 1994.

[10] S. Srivastava, S. Kumar, R. Garg, and P. Sen, "Generalized traveling salesman problem through n sets of nodes," *CORS journal*, vol. 7, no. 2, p. 97, 1969.

[11] M. Kulich, J. Vidašič, and J. Mikula, "On the travelling salesman problem with neighborhoods in a polygonal world," in *Climbing and Walking Robots Conference*, pp. 334–345, Springer, 2022.

[12] L. Fanta, "The Close Enough Travelling Salesman Problem in the polygonal domain," master's thesis, Czech Technical University, 2021.

[13] X. Pan, F. Li, and R. Klette, "Approximate shortest path algorithms for sequences of pairwise disjoint simple polygons," in *Proceedings of the 22nd Annual Canadian Conference on Computational Geometry, CCCG 2010*, pp. 175–178, 2010.

[14] M. Dror, A. Efrat, A. Lubiw, and J. S. Mitchell, "Touring a sequence of polygons," in *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pp. 473–482, 2003.

[15] J. Baxter, "Local optima avoidance in depot location," *Journal of the Operational Research Society*, vol. 32, no. 9, pp. 815–819, 1981.

[16] D. J. Gulczynski, J. W. Heath, and C. C. Price, "The close enough traveling salesman problem: A discussion of several heuristics," *Perspectives in Operations Research: Papers in Honor of Saul Gass' 80 th Birthday*, pp. 271–283, 2006.

[17] J. Faigl, "Gsoa: growing self-organizing array-unsupervised learning for the close-enough traveling salesman problem and other routing problems," *Neurocomputing*, vol. 312, pp. 120–134, 2018.

[18] W. pang Chin and S. Ntafos, "Optimum watchman routes," *Information Processing Letters*, vol. 28, no. 1, pp. 39–44, 1988.

[19] E. Packer, "Computing multiple watchman routes," in *Experimental Algorithms: 7th International Workshop, WEA 2008 Provincetown, MA, USA, May 30-June 1, 2008 Proceedings 7*, pp. 114–128, Springer, 2008.

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

[21] S. L. Smith and F. Imeson, "Glns: An effective large neighborhood search heuristic for the generalized traveling salesman problem," *Computers & Operations Research*, vol. 87, pp. 1–19, 2017.

[22] J. Faigl, "Approximate solution of the multiple watchman routes problem with restricted visibility range," *IEEE Transactions on Neural Networks*, vol. 21, no. 10, pp. 1668–1679, 2010.

[23] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[24] M. Gendreau, J.-Y. Potvin, *et al.*, *Handbook of metaheuristics*, vol. 2. Springer, 2010.

[25] H. González-Banos and J.-C. Latombe, "Planning robot motions for range-image acquisition and automatic 3d model construction," in *AAAI Fall symposium*, 1998.

[26] J. Mikula and M. Kulich, "Triangular expansion revisited: Which triangulation is the best?," in *ICINCO*, pp. 313–319, 2022.

[27] D. Coeurjolly and J. Chassery, "Fast approximation of the maximum area convex subset for star-shaped polygons," *CNRS*, vol. 1, pp. 1–18, 2004.

[28] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Commun. ACM*, vol. 22, p. 560–570, oct 1979.

[29] A. Duarte, N. Mladenovic, J. Sánchez-Oro, and R. Todosijević, "Variable neighborhood descent," *Handbook of heuristics*, pp. 341–367, 2018.

[30] J. Richard Shewchuk, "Adaptive precision floating-point arithmetic and fast robust geometric predicates," *Discrete & Computational Geometry*, vol. 18, pp. 305–363, 1997.

[31] J. R. Shewchuk, "Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates," *Discrete & Computational Geometry*, vol. 18, pp. 305–363, Oct. 1997.

[32] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, "Approximate algorithms for the traveling salesperson problem," in *15th Annual Symposium on Switching and Automata Theory (swat 1974)*, pp. 33–42, IEEE, 1974.

[33] B. Golden, L. Bodin, T. Doyle, and W. Stewart Jr, "Approximate traveling salesman algorithms," *Operations research*, vol. 28, no. 3-part-ii, pp. 694–711, 1980.

[34] G. Gutin, A. Yeo, and A. Zverovitch, "Exponential neighborhoods and domination analysis for the tsp," in *The traveling salesman problem and its variations*, pp. 223–256, Springer, 2002.

[35] I. Or, *Traveling salesman type combinatorial problems and their relation to the logistics of regional blood banking.* Northwestern University, 1976.

[36] G. A. Croes, "A method for solving traveling-salesman problems," *Operations research*, vol. 6, no. 6, pp. 791–812, 1958.

[37] J. Schmidt and S. Irnich, "New neighborhoods and an iterated local search algorithm for the generalized traveling salesman problem," *EURO Journal on Computational Optimization*, vol. 10, p. 100029, 2022.

[38] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the society for industrial and applied mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[39] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations research*, vol. 21, no. 2, pp. 498–516, 1973.

[40] "TSP Basics non-sequential 4-opt moves - part 1 of 4." `http://tsp-basics.blogspot.com/2017/09/non-sequential-4-opt-moves-part-1.html`. Accessed: 2024-05-07.

[41] J. R. Shewchuk, "Triangle: Engineering a 2d quality mesh generator and delaunay triangulator," in *Workshop on applied computational geometry*, pp. 203–222, Springer, 1996.

[42] A. Johnson, "Clipper-an open source freeware library for clipping and offsetting lines and polygons," *Retrieved September*, 2014.

Appendix B

ZIP content

- Thesis ... The tex files of this thesis.
- Thesis code ... Code base of the thesis.
- Experiments code ... Code for running the evaluations and creating graphs.