Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

The Close Enough Travelling Salesman Problem in polygonal domain

Bc. Lukáš Fanta

Supervisor: RNDr. Miroslav Kulich, Ph.D.
Field of study: Cybernetics and Robotics
Subfield: Cybernetics and Robotics
August 2021

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Fanta Lukáš**	Personal ID number:	**457018**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Control Engineering**		
Study program:	**Cybernetics and Robotics**		
Branch of study:	**Cybernetics and Robotics**		

II. Master’s thesis details

Master’s thesis title in English:

The Close Enough Travelling Salesman Problem in the polygonal domain

Master’s thesis title in Czech:

Close Enough Travelling Salesman Problem v polygonální doméně

Guidelines:

1. Get acquainted with the Close Enough Travelling Salesman Problem (CETSP) and state-of-the-art metaheuristics for routing problems.
2. Design and realize a solver for finding the shortest path between two points and a circle (point-circle-point, PCP).
3. Design and realize a method for the CETSP based on a combination of a PCP solver and the GLNS solver [2].
4. Compare the method with the state-of-the-art.
5. Extend the realized method to the environment with polygonal obstacles.
6. Evaluate experimentally properties of the extended algorithm. Describe and discuss the obtained results.

Bibliography / sources:

[1] Yuepeng Ding, Xiong Xie, Bo Jiang. An Efficient Algorithm for Touring n Circles, MATEC Web Conf. doi: 10.1051/matecconf/201823203027, 2018
[2] Stephen L. Smith, Frank Imeson,GLNS: An effective large neighborhood search heuristic for the Generalized Traveling Salesman Problem, Computers & Operations Research, Volume 87, Pages 1-19, ISSN 0305-0548, doi:10.1016/j.cor.2017.05.010, 2017.
[3] Gendreau M., Potvin JY. (eds) Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol 272. Springer, Cham, 2019
[4] Michael L. Cui, Daniel D. Harabor, and Alban Grastien. Compromise-free pathfinding on a navigation mesh. In Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17). AAAI Press, 496–502, 2017.

Name and workplace of master’s thesis supervisor:

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

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

Date of master’s thesis assignment: **29.01.2021** Deadline for master's thesis submission: **13.08.2021**

Assignment valid until:
by the end of winter semester 2022/2023

RNDr. Miroslav Kulich, Ph.D.	prof. Ing. Michael Šebek, DrSc.	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 would like to thank my supervisor RNDr. Miroslav Kulich, Ph.D. for his guidance, advice, patience and warm approach. I also thank my family for their support Ing. Eliška Sirůčková for her helpful advice and very loyal support.

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, August 13th, 2021

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 13. 8. 2021

. . . .

Abstract

Close-Enough Traveling Salesman Problem (CETSP), a variant of well-known Traveling salesman problem (TSP), is one of the important problems in routing and circular tour planning applications, such as automatic meter reading, collecting data in circular regions and searching for sources of gamma radiation. The aim of this thesis was to combine existing algorithms in pursuit of higher quality results obtained in shorter time than previously attained. A novel heuristic method called *GLNS-CETSP* was proposed. It combines a solver for finding the shortest path between two points and a circle called point-circle-point (PCP) with GLNS, and touring circle problem (TCP) algorithms. Experiments were carried out to test various configurations of *GLNS-CETSP*. These include mode (fast, medium, slow), initialization heuristics (random_insertion, random, GSOA, and LKH) and two versions of *PCP* (simplified and precomputed). Additionally, the novel approach was rigorously tested against the state-of-the-art metaheuristics. The obtained results showed the *GLNS-CETSP* in the majority of the cases obtains higher quality results than the state-of-the-art metaheuristics. It was also demonstrated that the *GLNS-CETSP* is the second fastest algorithm in the majority of cases. However, the computation time significantly increases in instances containing more than 200 circles. In response, a second new algorithm *GSOA+TCP* was proposed. As the name suggests, it combines growing self-organizing array (GSOA) and TCP algorithms. Although, *GSOA+TCP* was not able to reach as good results as the *GLNS-CETSP* the combination of two very fast algorithms resulted in very fast and effective method. Adding the TCP to GSOA improved significantly the quality of results at minimal time cost. Based on the properties of the two new methods, a conclusion has been reached that each of the algorithms should be used in different situation. The *GSOA+TCP* should be used especially in applications where the computational time is limited and where the instances contain more than 500 circles. The CETSP problem has 4 different variants of which only two have been previously solved by approximation algorithms. *GLNS-CETSP* was extended to solve the other two types, which contains polygonal obstacles. Therefore, new *CETSP* instances were generated on four maps (jari-huge, large, potholes and warehouse) with and without polygonal obstacles by proposed generator of *CETSP* instances.

Keywords: CETSP, TCP, GLNS-CESTP, GLNS, GSOA, point-circle-point

Supervisor: RNDr. Miroslav Kulich, Ph.D.
IMR - Intelligent Mobile Robotics,
CIIRC, CTU in Prague,
Jugoslávských partyzánů 1580/3,
160 00 Praha 6, Dejvice,
Czech Republic

Abstrakt

Close-Enough Traveling Salesman Problem (CETSP), varianta dobře známého problému obchodního cestujícího (TSP), je jedním z důležitých problémů při aplikacích zabývajících se trasováním a plánováním cest na oblastech tvaru kružnic. Příklady těchto aplikací jsou následující: automatické odečty elektroměrů, sběr dat v kruhových oblastech a hledání zdrojů gamma záření. Cílem této práce bylo zkombinovat stávající algoritmy za účelem dosažení kvalitnějších výsledků získaných v kratším čase, než bylo dříve dosaženo. Byla navržena nová metoda s názvem *GLNS-CETSP*, která kombinuje metodu pro nalezení nejkratší cesty mezi dvěma body a kružnicí zvanou bod-kružnice-bod (PCP) s algoritmy GLNS a TCP. Byly provedeny experimenty k otestování různých konfigurací *GLNS-CETSP*. Mezi testované konfigurace patří mód (rychlý, střední, pomalý), inicializační heuristiky (random_insertion, random, GSOA, and LKH) a dvě verze *PCP*. Kromě toho byl nový přístup srovnán s nejmodernějšími metaheuristikami. Naměřené výsledky ukázaly, že *GLNS-CETSP* ve většině případů dosahuje kvalitnějších výsledků než nejmodernější metaheuristiky. Ukázalo se také, že *GLNS-CETSP* je ve většině případů druhým nejrychlejším algoritmem. V případech obsahujících více než 200 kruhů se však doba výpočtu výrazně prodlužuje. V reakci na to byl navržen druhý nový algoritmus *GSOA+TCP*. Jak název napovídá, kombinuje algoritmy GSOA a TCP. Ačkoli *GSOA+TCP* nebyl schopen dosáhnout tak dobrých výsledků jako *GLNS-CETSP*, kombinace dvou velmi rychlých algoritmů vedla k velmi rychlé a efektivní metodě. Přidání TCP k GSOA výrazně zlepšilo kvalitu výsledků za minimální časové ztráty. Na základě vlastností těchto dvou nových metod bylo rozhodnuto, že každý z algoritmů by měl být použit v jiné situaci. *GSOA+TCP* by měl být používán zejména v aplikacích, kde je výpočetní čas omezený a kde instance obsahují více než 500 kružnic. Problém CETSP má 4 různé varianty, z nichž pouze dvě byly dříve vyřešeny zmíněnými aproximačními algoritmy. *GLNS -CETSP* byl rozšířen o řešení dalších dvou typů, které doplňují stávající varianty o polygonální překážky. Proto byly vegenerovány nové *CETSP* instance na mapách (jari-huge, large, potholes and warehouse) s polygonálními překážkami a bez nich pomocí námi vytvořeného generátoru *CETSP* instancí.

Klíčová slova: CETSP, TCP, GLNS-CESTP, GLNS, GSOA, bod-kružnice-bod

Překlad názvu: Close Enough Travelling Salesman Problem v polygonální doméně

Contents

Figures

Tables

Chapter 1

Introduction

Travelling Salesman Problem (*TSP*) is a well-known NP-hard problem with an aim to visit a given set of cities by a salesman while minimizing the length of a traversed tour. The salesman starts and ends at the same location and visits each city exactly once. Close Enough Traveling Salesman Problem (*CETSP*) is an extension of *TSP*, where the salesman visits an arbitrary point in the circular area around each city. Both TSP and CETSP are combinatorial optimization problems that play a crucial role in robotic applications. In mobile robotics, examples of usage include monitoring and collecting data in regions [1], [2]. Examples of usage in industrial robotics include optimization of the sequence of robotic tasks [3] and planning time-optimal motions of manipulators [4].

Several algorithms were proposed to solve *CETSP*, such as Steiner zone heuristic [5], branch-and-bound [6], mixed-integer nonlinear program [7], Growing Self-Organizing Array [8], discrete gravitational search algorithm [9], and regression model to estimate the solution [10]. Some of the algorithms are able to compute the shortest tour possible but only for a small number of cities. For a larger number of cities, these algorithms are computationally demanding making them slow. On the other hand, there are also very fast algorithms, but these generally find worse solutions than the slower algorithms. As a result, this thesis is motivated by the need to solve *CETSP* even for a larger number of cities in the shortest possible computation time. At the same time, the solution proposed by the thesis will attempt to find better solutions than the already best-found solutions.

The aim of the thesis is to devise a new heuristic method to solve *CETSP* in the polygonal domain and evaluate it on existing *CETSP* datasets. Also, new *CETSP* instances in the environment with polygonal obstacles are generated and solved. The proposed heuristic method combines three algorithms: point-circle-point (*PCP*), *GLNS*, and touring circle problem (*TCP*). First, *PCP* computes a point on the circle that minimizes the sum of distances between the the point and the other two points. *GLNS* determines a sequence of circles to be visited and a corresponding sequence of points, where each point lies on a different circle from the sequence. The order of circles in the sequence minimizes the length of the tour traversing the sequence of points. Finally, for each circle, *TCP* computes a point, which minimizes the sum of distances between the consecutive points in the sequence given by *GLNS*.

The following Section 1.1 provides a literature review of *CETSP* problem. The rest of the thesis is organized as follows. Chapter 2 defines all terms used in the thesis and specifies *TSP* and *CETSP* problems. Also, the variants of *CETSP* problem in the polygonal domain are introduced. Chapter 3 describes in detail the algorithms *PCP*, *GLNS* and *TCP* including their implementation. The process of generation of *CETSP* instances in the environment with polygonal obstacles is also described. Finally, the experimental setup and results are outlined in Chapter 4.

1.1 State of the art

The CETSP is a combinatorial optimization problem extensively studied in researches with practical applications, such as automatic meter reading using radio frequency identification (RFID), monitoring geographical regions by drones, routing in wireless sensor networks, and optimizing a sequence of robotic tasks in industrial robotics [5], [11]. Since the exact solution for this problem can not be found in polynomial time, the approximation algorithms running in a reasonable time are being proposed.

1.1.1 First specification and heuristics to CETSP problem

The CETSP problem was first mentioned by Heath et al. (2006) [11] in an automatic meter reading (*AMR*) application and was defined as follows. A service team has to visits customers (labeled as *nodes*) that have RFID transmitter transmitting a signal in a circular area of a radius r. The service team starts and ends in a point called *depot* and each node can be visited anywhere in its surrounding circular area. The authors proposed a method to find the shortest tour that visits all nodes and the depot. The method is divided into three consecutive parts:

1. Generate a feasible set of supernodes S (defined in next paragraph) that covers the whole plane with nodes, where each node is distant no more than r from the closest supernode.
2. Find near-optimal TSP tour T on S.
3. Improve tour T, i.e. reduce the distance traveled in T.

A feasible set of supernodes S is determined by six heuristics as follows:

- First three heuristics use variants of the *tilling method* that initially cover the whole plane with a set of equal-sized regular hexagons, where each hexagon can be inscribed in a circle of a radius r. Regular hexagons were chosen from all types of polygons due to their minimal overlap area. The hexagon centers form a set of supernodes S_r, size of which number must be reduced.

  The first variant called *shifting* reduces the number of supernodes by performing a series of small random shifts (vertically or horizontally) of all hexagons at once. The output of each shifting process is a new set of supernodes, where each supernode corresponds to the center of a shifted hexagon that covers at least one node. The set of supernodes with the smallest cardinality is considered to be a feasible set S.

  The second variant called *merging* attempts to merge every two adjacent hexagons on the plane. Assume two adjacent hexagons H_1, H_2 with centers h_1, $h_2 \in S_r$ and a point h created as a midpoint on a line segment $\overline{h_1h_2}$. A new hexagon H with equal size to H_1 and with the center in h is created. If H covers all nodes that lie in H_1 or H_2, the hexagons H_1, H_2 are merged and only h is considered to be supernode in set S. Otherwise, the centers h_1, h_2 are considered to be supernodes in S.

  The third variant, called *circular extension* replaces the hexagons with circles of radii $w$ith the same centers and tries to reduce S_r by considering the intersection of the circles. Assume three intersecting circles C_1, C_2 and C_3, where C_2 intersects C_1 and C_3, and C_1 and C_3 do not intersect with each other. If all nodes that are covered by C_2 are located only in the intersect areas with C_1 or C_3, C_2 is labeled as superfluous and is eliminated, i.e. the center of C_2 is removed from S_r. S_r is considered to be feasible S after eliminating all superfluous circles.

- For the second heuristic, assume n intersecting circles. An area where the circles intersect is called *Steiner Zone* of degree n. *Steiner Zone* heuristic initially creates equal-sized circles with centers in all nodes. To quickly obtain S, Steiner zones with degrees no more than three are picked as an area suitable for possible supernodes. An arbitrary point in each Steiner zone is considered to be a supernode, and it is saved to S. The process of creating and saving supernodes starts from the Steiner zone with the highest degree and repeats until all nodes are covered with at least one supernode.
- *Sweeping Circle* heuristic covers the whole plane with overlapping equal-sized circles with centers shifted by a given value. The process of choosing supernodes is iterative. In each iteration, the circle with the highest number of nodes, is picked and its center is considered to be a new supernode of S. The process is repeated until all nodes are covered.
- In the last heuristic, node n_1 is said to be adjacent to node n_2 if the distance between n_1 and n_2 is more than 0 and no more than r. The degree of n_1 is the number of nodes adjacent to n_1. *Radial Adjacency* heuristic creates S using an iterative process. In each iteration, node n with the highest degree is picked from a set of all available nodes, and the geometric mean of n and all of its adjacent nodes is computed. If the resulting point is adjacent to fewer nodes than n, the node n is considered to be a supernode and is added to S. Otherwise, the point resulting from the computation of the geometric mean is a supernode and node n along with its neighbors is removed from the set of all available nodes.

Near-optimal T on S is found by a non-specified heuristic. The process of improving T is based on minimization of the marginal cost of visiting supernodes in T.

The authors tested the heuristics on seven instances consisting of 100 to 1000 nodes. The results show that the *merging tilling* and *Steiner Zone* heuristics are the most effective in the way of shortening the length of tour T. The computation time was not measured.

1.1.2 Mixed-integer nonlinear program (*MINLP*)

Dong et al. (2007) [12] formulated a mixed-integer nonlinear program (*MINLP*) for CETSP, but they did not solve it. Furthermore, they proposed two approximation heuristics to find a set of supernodes S on the plane in the *AMR* application. The first heuristic, called *clustering-based* is the same as the *merging tilling* heuristic. The second heuristic, called *convex hull-based* finds a set S that covers the whole plane with nodes using an iterative process. The process can be described by the following steps:

1. Depot is added to S.
2. Nodes that lie within the distance r from closest supernode in S are called *covered nodes*.
3. A convex hull over all noncovered nodes is created by the Quickhull algorithm [13].
4. A centroid O of the convex hull is computed.
5. A set of all vertices of the convex hull is iterated. In each iteration a current vertex is labeled as V and a distance between V and O is measured. If the distance is less than or equal to r, O is a new supernode and added to S. Otherwise, a new point is created on $\overline{VO}$ with distance r from V and considered to be a new supernode.
6. Go to step 2. until all nodes are covered.

The TSP tour T on S is found by the convex hull insertion algorithm, which the authors consider to be an algorithm with a remarkable speed and a surprising accuracy. The process of improving T is done by the simulated annealing algorithm.

The authors constructed and tested 190 cases consisting of 100 to 1000 nodes and their radii from 2 to 20 by 2, i.e., ten different radii. Both heuristics produce good results in a computation time not exceeding 800 milliseconds, which is very fast.

1.1.3 *Steiner Zone* heuristic *SZH* and *CETSP-lib*

Mennell (2009) [5] formulated the *MINLP* and introduced a new lower bounds (*LB*) techniques derived from *MINLP*. Next, the author developed several solvers of the CETSP tour based on the *Steiner Zone* heuristic (*SZH*) and generated a set of 62 CETSP instances, which consists of 17 to 1001 circles. We labeled the set of instances as *CETSP-lib*. *CETSP-lib* contains a set of seven instances with three different overlap ratios (low, moderate, and high), a set of fourteen instances with circles with random values of radii, and a set of twenty-seven instances with circles with equal-sized radii. Overlap ratio defines the ratio between the areas where circles overlap and where not.

The author shows that the results of *LB* techniques for the CETSP problem are very weak, i.e., computed *LB* values of most instances with moderate and high overlap ratios are equal to zero. The computation process of general *SZH* is divided into three phases:

1. The phase called *Steiner zone generation* generates a set of several suitable disjoint or intersecting Steiner zones on a given instance. For further information on this phase the reader is advised to read chapter *2.4.1* on page *22* of the original article [5].
2. The phase called *Tour finding* finds a representative point in each Steiner zone and computes TSP tour *T* on these points. *T* is computed by Lin-Kernighan heuristic *LKH* [14], which is considered to be a fast heuristic.
3. The third phase called *Tour improvement* tries to shorten *T* by changing positions of points in *T*. The order of points in *T*, where each point corresponds to one Steiner zone, is fixed and the position of points in each Steiner zone minimizing the length of *T* is solved by an algorithm called the Touring polygon problem (*TPP*) [15]. *TPP* originally optimizes tour on a sequence of polygons, but the convex regions (Steiner zones) are used in SZH. The author modified *TPP* to Touring Steiner Zones Problem (*TSZP*), and proposed two types of solution of *TSZP*: optimal and near-optimal. The optimal solution is formulated by the second-order cone program (*SOCP*) and solved by *CPLEX* optimizer [16]. The near-optimal solution is found by a heuristic called IP_{PhIII}. Detailed definition can be found on page *261* in the original article [5]. IP_{PhIII} solver finds the solution on up to 1001 circles in under 1 second and no more than 1% from the currently shortest length. On the other hand, *CPLEX* finds the optimal solution on 1000 circles in roughly 6 seconds, which is more than six times slower than IP_{PhIII}.

Moreover, it proposes several variants of *SZH*.

The variants of *SZH* are called *SZ1* and *SZ2*. *SZ1* prioritizes the computational speed over the tour length and therefore, uses IP_{PhIII} solver in the third phase. On the other hand, *SZ2* prioritizes the tour length and hence uses the *CPLEX* solver in the third phase and repeats phases 2. and 3. until no improvement of tour length is met. *SZ1* and *SZ2* were compared with Generalized Traveling Salesman Problem (*GTSP*) based algorithms and another 11 heuristics on *CETSP-lib*. The GTSP-based algorithms use a genetic algorithm (GA) to solve the exactly-one-in-set GTSP problem on an instance with approximated circles. The author tried to approximate the circles by 3, 6, 12, and 24 points and determined that only a 24-points approximation is suitable.

The results show that *SZ1* and *SZ2* are very fast and provide the best combination of computation time and length of the tour found. Although, GTSP based algorithms find shorter tours than *SZH* based heuristics on instances with a low and moderate overlap ratio, the computation time is very slow, taking days or even weeks for larger instances.

1.1.4 Behdani & Cole (2014) and *CETSP-lib-small*

Behdani and Smith (2014) [7] approached CETSP problem as problem, where the area around each point is not necessarily a circle and introduced two discretization methods: *grid-based* and *arc-based* to approximate the areas. *Grid-based* discretization method approximates close-enough areas of each point by rectangular cells. *Arc-based* discretization is intended only for circular areas. Assume a sequence of circles and a convex hull created from the centers of circles in the sequence. Boundaries of circles lying on or inside the convex hull are approximated by points called arc cells.

In addition, the authors generated a new set of 720 CETSP instances consisting of 6 to 30 circles. We labeled the set of instances as *CETSP-lib-small* due to a small number of circles in each instance. *CETSP-lib-small* contains instances with three different overlap ratios: low, moderate, and high. Last but not least, three mixed-integer program (*MIP*) approaches (*LB1*, *LB2*, and *LB3*) were formulated on discretization schemes to find upper and lower bounds on *CETSP-lib-small* and two alternative formulations (Bender Decomposition (*BD*) and Iterative algorithm (*IA*)) were proposed. *IA* improves the lower bound found by *LB1*, *LB2* or *LB3*. *BD* significantly improves the solvability of the general formulation of *MIP* by reformulating the problem as a two-stage problem that is suitable for decomposition. The formulations of *LB1*, *LB2*, and *LB3* are solved by *CPLEX*.

LB1, *LB2*, and *LB3* were tested on instances with 6 circles. In comparison with the other two algorithms, the *LB2* has proven to be very slow and has been excluded from further testing. *LB1* and *LB3* were then tested on instances of up to 10 circles. The tightest lower and upper bounds were found by *LB3* in shorter computation time than *LB1*. *BD* has been shown to be effective for larger instances of 14 to 20 circles. The improving algorithm *IA* was tested on instances of 12 circles with two different overlap ratios: low and moderate. The upper and lower bounds were found in up to 210 seconds on instances with a low overlap ratio. On the contrary, the 1000 seconds limit was reached in most instances with a moderate overlap ratio. Thus, *IA* is very time-consuming on instances with moderate or high overlap ratios.

1.1.5 Branch-and-Bound (*B&B*)

Coutinho et al. (2016) [6] proposed an effective branch-and-bound (*B&B*) algorithm for the CETSP and compared its results on *CETSP-lib* and *CETSP-lib-small*.

Assume that the circular tour T starts and ends in the same position called the depot and visits each circle in the CETSP instance exactly once. The tour T=(T^C, T^P), where T^C is a sequence of all circles in the CETSP instance and T^P is a sequence of points. Partial tour T_{part}=(T_p^C, T_p^P), where T_p^C is sub-sequence of T^C and T_p^P is sub-sequence of T^P. A set of circles that are not visited by T_{part} is called *uncovered circles*.

B&B algorithm finds the circular tour T with length that represents *LB* estimate for a given instance and which can be considered to be optimal length of T, if T contains all circles in the instance. The algorithm starts with the initialization of a feasible T_{part} and its T_p^C consists of three circles that are picked using a method called *root relaxation*, which will be described later. T_p^P is found using *CPLEX* on formulated second order cone program (*SOCP*) also proposed by Mennell (2009) [5]. Next, the branching process is applied and described by the following steps:

1. A circle from *uncovered circles* is selected based on one of the two branching rules and placed at each position of T_p^C in the feasible T_{part}. The first branching rule selects from *uncovered circles* the circle that is the most distant from the closest line segment created by two consecutive points in T_{part}. The second branching rule inserts each uncovered circle from *uncovered circles* to the feasible T_{part} between two closest circles in T_p^C. A new T_p^P is computed, the length of T_{part} is determined and the uncovered circle is removed from T_{part}. When the lengths of all T_{part} are determined, the circle from *uncovered circles* that maximizes length of T_{part} is selected. In cases where all circles have the same radii, the first branching rule is applied. Otherwise, the second rule stands.

2. Three new T_{part} containing T_p^C are created and their T_p^P are generated using *CPLEX*. T_{part} with the minimal length is picked as new feasible T_{part}.
3. All consecutive points in feasible T_{part} are connected using a line segment and an area created from these connections is called *convex bounding region*.
4. If a circle from *uncovered circles* intersects some line segment in *convex bounding region*, it is removed from *uncovered circles*.
5. If *uncovered circles* is not empty, go to step 1. Otherwise, the branching process is finished and T_{part} is considered to be T.

Root relaxation method picks the initial circle with radius equal to zero (*depot*) as a first circle. The circle that is the most distant from *depot* (based on Euclidian distance measured between the coenters of the circles) is picked as a second circle. The remaining circle with the maximal insertion cost is picked as the third circle. The insertion cost of circle C is the length of T_{part} consisting of the two previously picked circles and C.

The results of the *B&B* algorithm were compared with the best results found on *CETSP-lib* and *CETSP-lib-small*. *B&B* algorithm found a new optimal solution in 22 out of 62 instances of *CETSP-lib* and improved the *LB* estimates in the rest of the instances, which were initially computed by Mennell (2009) using *MINLP* [5]. On the other hand, there was a vast difference in the computation time between instances. The computation time of instances with a high overlap ratio was less than one second. However, the computation time of instances with a low or moderate overlap ratio almost always reached the time limit of four hours. This means, the *B&B* algorithm is effective only for instances with a high overlap ratio. Next, the *B&B* algorithm is much more effective on *CETSP-lib-small* because it solves all instances to optimality in no more than two seconds.

1.1.6 Carrabs et al. (2017)

Francesco Carrabs et al. (2017) [17] proposed two new discretization schemes: *perimetral* and *internal* that effectively discretize the solution space (boundary of circles). Moreover, they introduced a graph reduction algorithm that significantly reduces the problem size on already discrete solution space and speeds up the computation process. The optimal circular tour T is then computed by solving the GTSP problem on the reduced discrete solution space. GTSP problem is formulated by *MIP* and solved by *CPLEX*. The process of creating a discrete solution space, reduction of discrete space, and computing optimal T is called *ULB*.

ULB was compared with *LB3*, *BD* and *IA* proposed by Behdani and Smith et al. (2014) [7] in Section 1.1.4. The results show that *ULB* find tighter lower and upper bounds and much faster than the other algorithms in most instances. Overall, *ULB* outperforms the other tested algorithms.

1.1.7 F. Carrabs et al. (2017)

F. Carrabs et al. (2017) [18] proposed an improved version of *internal discretization scheme*, which was initially proposed by the same authors and in the same year. In addition, they proposed a heuristic *IULB* combining the scheme with *SOCP* to solve CETSP.

IULB was compared with *ULB* and was shown to be an improved version of *ULB* as *IULB* solves 17 instances in less than 100 seconds while during the same time *ULB* solves only 12 instances.

1.1.8 Growing Self-Organizing Array *GSOA*

Jan Faigl (2018) [8] introduced unsupervised learning procedure Growing Self-Organizing Array *GSOA* inspired by Self-Organizing maps *SOM* [19] to solve TSP and CETSP problems.

Assume a set of circles S that need to be visited and a growing array structure $\mathcal{N}$ initialized with one point (centroid to S). *GSOA* computes T by an iterative process, where each iteration is called a *learning epoch*. Each *learning epoch* can be described by the following steps:

1. A temporal circular tour T_{tmp} encompasses all points from $\mathcal{N}$.
2. *Winner selection and adaptation* step randomly iterates through all circles in S and for each determines a winner point w. w is the closest point on T_{tmp} to a given circle. Each w is then adapted towards the corresponding circle and inserted into $\mathcal{N}$.

3. *Solution extraction* step removes all not winning points in $\mathcal{N}$.

T_{tmp} is the new T after completion of all epochs.

GSOA was compared with *SZ2* and *GTSP-GLNS* on *CETSP-lib*. *TSPlib* [20] was used for comparison with *LKH*, *Co adaptive Net* and *ORC-SOM*. *GTSP-GLNS* is exactly-one-in-set GTSP solver and is applied to circles in CETSP instances that are approximated by 24 points. Algorithms *ORC-SOM* on *TSPlib* are considered to be the most representative *SOM*-based algorithms for the *TSP* by the author. The results show that *GSOA* is very fast because the computation time of the biggest instance consisting of 1001 circles is approximately 0.6 seconds, and the total average of all *CETSP-lib* instances is approximately 0.1 seconds, which is several orders of magnitude faster than other algorithms. On the contrary, solutions found by *GSOA* are worse than the best-found solutions in most cases. Therefore, *GSOA* is recommended to be used only as an initialization tour heuristic for other algorithms.

1.1.9 Steiner Zone Variable Neighborhood heuristic (*SZVNS*)

Wang et al. (2019) [21] proposed Steiner Zone Variable Neighborhood heuristic (*SZVNS*) to find circular tour T on CETSP instance *CETSP-lib* and *CETSP-lib-small*.

SZVNS works in three phases:

1. *Data cleaning* phase eliminates redundant circles to reduce the number of circles and improve computation time. When a Steiner zone created by set of circles is inscribed in another circle, the circle is eliminated, because any point from the Steiner zone always lies in this circle.
2. *Construction* initially creates a set of all Steiner zones from a reduced set of circles using *Sweep line* algorithm, which effectively finds each subset of circles producing a Steiner zone. Then, the set containing the fewest Steiner zones that cover all circles is selected using Set Covering Problem (*SCP*). *SCP* is formulated by Binary Integer Program (*BIP*) and solved by optimization software *Gurobi*. Finally, an initial tour T is generated from the optimal set of Steiner zones.

3. Let us define a fixed point in a Steiner zone as *Steiner point*. *Tour improving* phase improves T, i.e., shorten length of T if it is possible using *TSP solver* and *Steiner point selection*. *TSP solver* finds a sequence of Steiner zones minimizing T from their Steiner points using the *LKH* heuristic. Once the sequence is found, *Steiner point selection* method computes a Steiner point on each Steiner zone, which minimizes the distance between the previous and next points in sequence. The Steiner points can be computed by *SOCP*, but due to its time consumption, the authors applied a greedy algorithm.

SZVNS is compared with the best results computed by *B&B* and ULB algorithms on *CETSP-lib-small* and the results of all heuristics introduced by Mennell (2009) [5] including *SZ1*, *SZ2*, and *GTSP based* on *CETSP-lib*. *SZVNS* found optimal solutions on *94.4%* instances of *CETSP-lib-small* in less than one second in all cases, which is much faster than *B&B* and ULB algorithms. In *CETSP-lib*, *SZVNS* found the new optimal solutions in 29 out of 62 instances. Total average computation times of *SZVNS*, *SZ2*, and *GTSP based* algorithm are 72, 7, and 1,053,012 seconds, respectively. The fastest is *SZ2*, but it has not found as many optimal solutions as *SZVNS*. Overall, *SZVNS* seems to be a very effective solver.

1.1.10 M. Antonescu and C. Bîră (2019)

M. Antonescu and C. Bîră (2019) [9] introduce new TSP and CETSP solvers (DGSA-TSP and DGSA-CETSP) based on discrete gravitational search algorithm (*DGSA*). In addition, two types of rubber band algorithm (*RBA*) are proposed: Bisector RBA and Segment Middle Point RBA. These algorithms are used as a part of *DGSA* in *DGSA-CETSP* solver. The main purpose of the RBA algorithm is to improve the circular tour T, i.e., find a point on each circle of the tour that shortens the length of the tour.

The results of DGSA-TSP and DGSA-CETSP were compared with *GSOA* algorithm introduced by Faigl (2018) of auger engineering radio array (*AERA*) instances. The authors generated *AERA* instances consisting of circles with equal-sized radii. In the case of the TSP problem, the radii are equal to zero, and in the case of the CETSP problem, the radii are equal to 21 and 30 meters. The results show that DGSA-TSP finds the tour slightly shorter than *GSOA*. On the other hand, DGSA-CETSP generates poor results.

1.1.11 Regression model (RM)

Roy et al. (2021) [10] built a regression model *RM* to estimate circular tour lengths without generating the actual tour and tested the lengths on *CETSP-lib-small*. In addition, they generated a new set of 234 instances similar to the already proposed CETSP instances and a new set of 72 larger instances.

RM uses tour lengths computed by *SZVNS* heuristic proposed by Wang et al.(2019) [21] to estimate the tour lengths. The results show that the tour lengths computed by *SZVNS* can be estimated by *RM* with eight independent variables with an average error of approximately 4%.

Chapter 2

Problem specification

This chapter opens with an introduction to GLNS and TSP problems that are the basis of this thesis. The basic terms that are necessary to describe these problems are defined. At the end of this chapter, the CETSP problem is defined along with variants of CETSP that this thesis aims to solve.

2.1 Coordinate system

We work in a two-dimensional Cartesian coordinate system to represent the points on the plane. Each point p is uniquely described as $p = (p_x, p_y)$, where p_x and p_y are numerical coordinates.

The distance d between points p and q is computed as the Euclidean distance:

$$d(p, q) = \sqrt{(q_x - p_x)^2 + (q_y - p_y)^2}$$

2.2 Tour on graph

Assume a complete weighted graph G = $(V,\ E,\ w)$, where

- V is a set of (m) vertices defined as $V = \{V_1, V_2, \ldots, V_m\}$.
- E is a set of paired vertices called edges, i.e. $E \subseteq \{(x, y) \mid (x, y) \in V^2 \text{ and } x \neq y\}$.
- w is a function assigning weight to each edge: $E \to \mathbb{R}$.

Finite path P on G is defined as $P = (V_T, E_T)$, where V_T is a set of n vertices $V_T = \{V_1, V_2, \ldots, V_n\}$ and E_T is a set of $n-1$ edges $E_T = \{e_1, e_2, \ldots, e_{n-1}\} \subseteq E$ that joins consecutive vertices from V_T, i.e. $e_i = (V_i, V_{i+1})$. The path P does not have to visit all vertices on G, i.e., $V_T \subseteq V$.

Cycle on G is a finite path P that starts and ends in the same vertex. The *Hamiltonian path* is a finite path P that visits each vertex on G exactly once. The *Hamiltonian cycle* is a *Hamiltonian path* that starts and ends in the same vertex [22]. The *optimal tour* $T = (V_T, E_T)$ is a Hamiltonian cycle on G and has a minimal cost defined as:

$$cost(T) = \sum_{e \in E_T} w(e).$$

2.3 Circular tour

Assume a set of m circles $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$, where C_i is described by its center and radius, i.e., $C_i = (c_i, r_i)$. The circular tour T of $\mathcal{C}$ describes a cyclic path on a sequence of consecutive circles.

The tour $T = (T^C, T^P)$, where

1. $T^C = (T_1^C, T_2^C \ldots, T_m^C)$, where $T_i^C \in \mathcal{C}$.

2. $T^P = (T_1^P, T_2^P \ldots, T_m^P)$, where p_i is a point and lies on the circle $C_i \in T^C$.

A partial circular tour $T_{part} = (T_p^C, T_p^P)$, where $T_p^C \subseteq T^C$ and $T_p^P \subseteq T^P$.

The distance *dist* between two circles $C_i \in \mathcal{C}$ and $C_j \in \mathcal{C}$ is defined as

$$dist(C_i, C_j) = d(c_i, c_j) - r_i - r_j$$

The main metric for measuring the quality of the tour is the tour's length (len). It is computed as a sum of the distances between each two consecutive points in the tour's path:

$$len(T) = \sum_{i=1}^{|T|-1} d(T_i^P, T_{i+1}^P) + d(T_1^P, T_{|T|}^P),$$

where $|T|$ is the number of circles in the tour.

2.4 Traveling Salesman Problem (TSP)

Traveling Salesman Problem (TSP) is an NP-hard problem in combinatorial optimization. Let us assume a salesman wants to visit several cities and then come back home. The problem is to arrange the order of cities such that the cost of the tour is minimized [3]. This problem can be modeled as the TSP problem on the graph $G = (V, E, w)$, where V represents cities, E represents the roads between cities and w is a distance function of E.

The optimal TSP tour visiting all cities is the optimal tour T on G.

2.5 Generalized Traveling Salesman Problem (GTSP)

Generalized Traveling Salesman Problem ($GTSP$) is an extension of the TSP. The variant of the GTSP that we work with is called an exactly-one-in-set GTSP problem. Assume the graph $G = (V, E, w)$ mentioned above and a set of m clusters $\mathcal{S} = \{S_1, ..., S_m\}$. Each cluster $S_i \in \mathcal{S}$ contains a set of n_i vertices $S_i = \{V_1, ..., V_n i\}$, where $V_n i \in V$.

The GTSP optimal tour T is a cycle on G that visits exactly one node from each cluster and its cost is minimal among all such cycles.

2.6 Close-Enough Traveling Salesman Problem (CETSP)

CETSP is a variant of TSP, where each city is symbolized by a circle area with radius r. The city is considered to have been visited if the salesman is within the city's specified radius (in other words, "the salesman is close enough") [5].

To clarify this problem, we extend the TSP problem described on graph G in a discrete space to a continuous space $\mathbb{R}^2$. More specifically, we search for the optimal circular tour T in a continuous space instead of searching for the tour in a graph G.

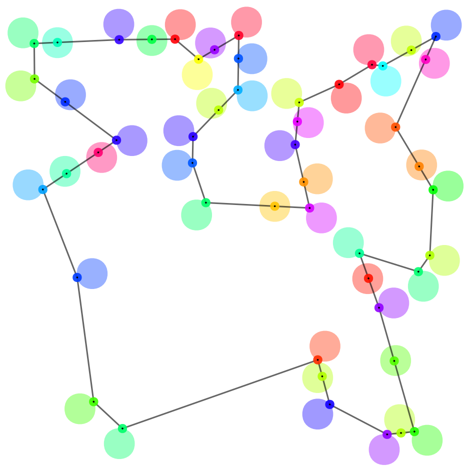
This thesis aims to solve the different variants of CETSP problem. These variants are sorted based on their complexity. Each variant inherits constraints from its predecessor and adds other constraints. Specifically:

1. $CETSP_{dis}$ is an elementary variant of the CETSP problem containing only disjoint circles with arbitrary-sized or equal-sized radii. The main goal of the search for the optimal circular tour T is to find the correct order of circles in T^C and their points in T^P.
2. $CETSP_{int}$ is similar to $CETSP_{dis}$ with the exception that the circles are allowed to intersect with each other.

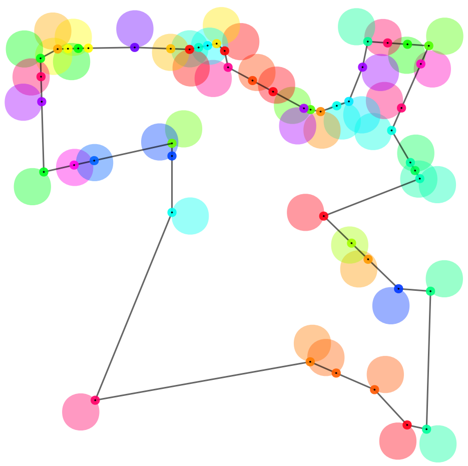
3. $CETSP_{obst\_dis}$ extends the $CETSP_{dis}$ by adding an environment with a set of polygonal obstacles $\mathcal{P}$. The obstacles must be simple polygons and have to be disjoint. Moreover, the obstacles are not allowed to intersect with the circles that are to be visited.

4. $CETSP_{obst\_int}$ is an extension of $CETSP_{obst\_dis}$. The circles are allowed to intersect with each other.

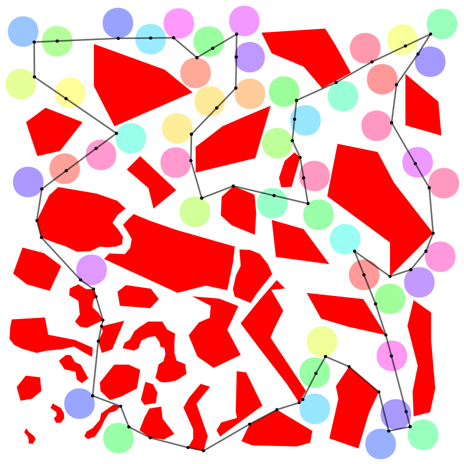
All variants of *CETSP* problem are solved on the *map* that specifies the environment. Each *map* may contain obstacles. *CETSP instance* is a set of m intersecting or disjoint circles on a *map* defined as $\mathcal{C} = \{C_1, \ldots, C_m\}$, where $C_i = (c_i, r_i)$ and $C_i \in \mathcal{C}$. An example of CETSP problem and of all considered variants shown on Fig.2.1 on two different CETSP instances placed on the map with and without obstacles. Subfigures *(a)* and *(c)* (similarly Subfigures *(b)* and *(d)*) produce similar T because the position of circles is the same but the environment differs in the presence of polygonal obstacles highlighted by red color.



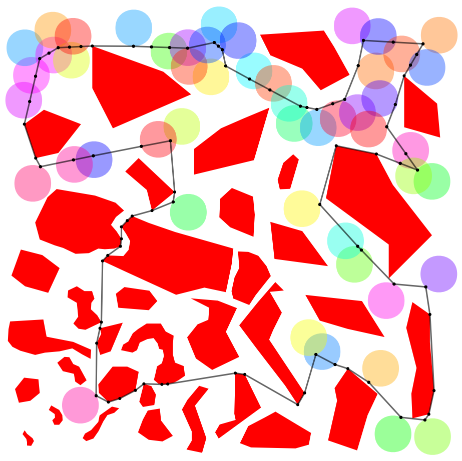
(a) : Solution to $CETSP_{dis}$ problem on instance called *disjoint_50*. Total number of circles is 50 and the circles are disjointed and with all with the same radius.



(b) : Solution to $CETSP_{int}$ problem on instance called *intersect_50*. Total number of circles is 50 and the circles are intersected and with all with the same radius.



(c) : Solution to $CETSP_{obst\_dis}$ problem on instance called *disjoint_50* on the map with obstacles.



(d) : Solution to $CETSP_{obst\_int}$ problem on instance called *intersect_50* on the map with obstacles.

Figure 2.1: Examples of solved CETSP problem and all variants on the map *large*.

Chapter 3

Solution Approach

This thesis aims to solve the CETSP problem and its extension to an environment with polygonal obstacles (i.e., to find circular tour T).

Firstly, let us describe the proposed solvers to the CETSP problem without obstacles. The following algorithms, sorted by increasing complexity, were used to resolve parts of this problem: point-circle-point (PCP), touring polygon problem (TPP), and GLNS. PCP solver finds a point p on the circle C such that the path between p and two points on the map is the shortest. PCP is the basis for the other two more complex algorithms and it is described in Section 3.1.

The general TPP algorithm (Section 3.2) determines a path touring a sequence of polygons. This general algorithm is not sufficient for our problem because we work with circles instead of polygons. Therefore, we modified TPP to find the shortest path on the circular tour T as defined in Section 2.3. Let us name the modified TPP solver TCP (Section 3.3). TCP is able to find an optimal solution to the circular tour T only for a fixed order of circles. Therefore, it is not suitable for solving the general CETSP problem, where the order of circles is not specified.

The optimal order of circles and the optimal path can be found by a modified GLNS algorithm. The original version of the GLNS algorithm (Section 3.4) can solve the GTSP problem. Our modified version called GLNS-CETSP (Section 3.5) determines the optimal path and order of circles in circular tour T. The TCP is used as a part of the optimization process of GLNS for the CETSP problem.

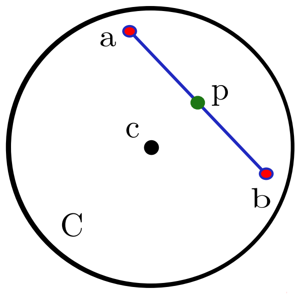
Secondly, we address the CETSP problem with obstacles. This problem extends the CETSP problem by adding polygonal obstacles that need to be avoided into the environment. The previously described algorithms were used to solve this problem. However, with a slight modification, the shortest collision-free path T^P needs to be found, i.e., the Euclidean distance between two points on the map can not be used.

3.1 Point-circle-point (PCP)

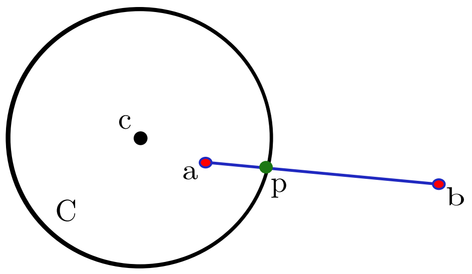
This section proposes a technique of finding the shortest path between two points a and b through an optimal point p located on a circle C (point-circle-point PCP). The point p is called optimal if it minimizes the sum of the Euclidean distances between p and a and between p and b. There are four different versions of the scenario with respect to the mutual positions of these points.:

1. Both points a and b are located on C. The optimal point p is located always inside this circle. Fig. 4.2a.
2. One of the points a or b is located inside C. The optimal point p is located on C or inside this circle. Fig. 4.2d and 4.2c.
3. The position of points a and b is outside C and line segment $\overline{ab}$ intersects C. The optimal point p is located on C or inside this circle. Fig. 3.1d and 3.1e.
4. The position of points a and b is outside C and line segment $\overline{ab}$ does not intersect C. The optimal point p is always located on C. Fig. 3.1f.

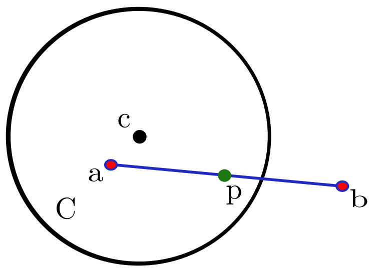
The last scenario is nontrivial to solve, while the others are trivial. The following subsections introduce the *PCP solver*, which provides solutions to the particular cases and an additional technique that improves the computation speed of the solver. Subsection 3.1.1 describes the framework of the solver. Subsection 3.1.2 introduces the computation of points located both on C and $\overline{ab}$, which is used in scenarios *2* and *3*. Subsection 3.1.3 describes the solution of the nontrivial scenario.



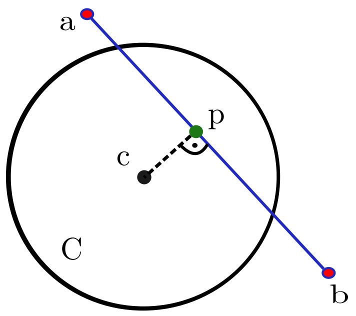
(a) : Scenario 1



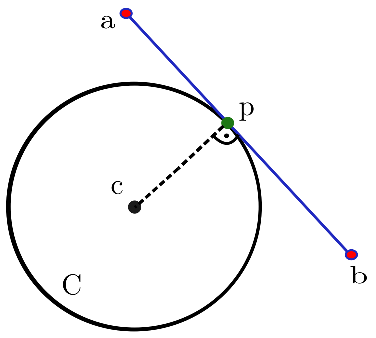
(b) : Scenario 2a



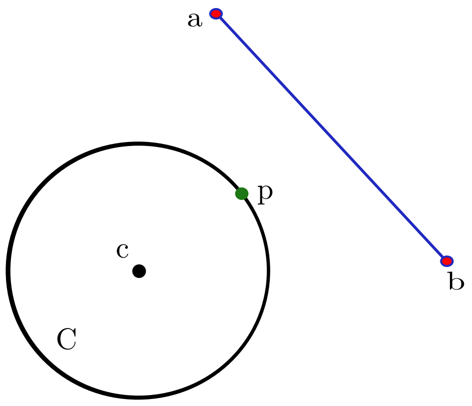
(c) : Scenario 2b



(d) : Scenario 3a



(e) : Scenario 3b



(f) : Scenario 4

Figure 3.1: Scenarios of mutual positions of points a and b and circle C

3.1.1 Solver framework

Algorithm 1 describes the *PCP solver*. The solver's inputs are points a and b and circle C defined by its center point c and radius r. It returns an optimal point p. Using a switch statement (lines 1 - 18), the solver is divided into four separate parts, each representing one of the scenarios.

The first part (lines 2 - 3) corresponds to the first scenario and occurs

Algorithm 1: PCP solver

```
   Input: Points $a$ and $b$ and circle $C$
   Output: Point $p$ on circle $C$
1  switch scenario do
2      case 1 do
3          p ← compute_mid_point(a, b);
4      case 2 do
5          p ← compute_mid_point(a, b);
6          if is_not_inside(p, C) then
7              $a^z$, $b^z$ ← transform_to_zero($a$, $b$, $C$)
8              $p^z$ ← point_on_circle_and_line($a^z$, $b^z$, $C$);
9              p ← transform_from_zero($p^z$, $C$);
10     case 3 do
11         $a^z$, $b^z$ ← transform_to_zero($a$, $b$, $C$)
12         $p_1$, $p_2$ ← points_on_circle_and_line($a^z$, $b^z$, $C$);
13         $p^z$ ← compute_mid_point($p_1$, $p_2$);
14         p ← transform_from_zero($p^z$, $C$);
15     case 4 do
16         $a^t_y$, $b^t_x$, $b^t_y$, $\alpha$ ← transform($a$, $b$, $C$);
17         $p^t$ ← solve($a^t_y$, $b^t_x$, $b^t_y$) ; // solving of nontrivial scenario
18         p ← inverse_transform($\alpha$, $p^t$, $C$);
19 return $p$
```

when both points a and b are inside C. The point p can be located on any position of the line segment $\overline{ab}$. In case of simplification of the computation process, we always compute p in the middle of the line segment $\overline{ab}$ as follows:

$$p = (\frac{a_x + b_x}{2}, \frac{a_y + b_y}{2}) \tag{3.1}$$

The second part (lines 4 - 9) occurs, when the either a or b is inside C while the other point is outside C. If so, the point p is placed in the middle of $\overline{ab}$ and its position is checked. If p is inside C, the result is saved and returned. Otherwise, a new p equal to the position, where $\overline{ab}$ intersects C need to be computed. In the first step, points a, b and C are shifted so that c is in the origin. The new shifted points are labeled as a^z and b^z (line 7). Next, the p^z is computed as described in detail in Section 3.1.2. The computed p^z is finally shifted back using the original c and a resulting point p is assigned to the solver's output.

The third part (lines 10 - 14) occurs, when $\overline{ab}$ intersects C. Let us define points p_1 and p_2 as points located on both C and $\overline{ab}$. The point p can be located anywhere on line segment $\overline{p_1p_2}$, i.e, p can be located on or inside C. To simplify the computation process, we consider p as a middle point on $\overline{p_1p_2}$, which is closest to the center of C. The computation process is described by the following steps:

1. Points a, b and C are shifted to center c to the origin (line 11).
2. Points p_1 and p_2 are computed using shifted points a^z, b^z and C (line 12). Section 3.1.2 describes the computation process in detail.
3. Point p^z is computed as a middle point on line segment $\overline{p_1p_2}$ (line 13).
4. p^z is shifted back using the original c and resulting p is assigned to the solver's output (line 14).

The fourth part (lines 15 - 19) corresponds to the last scenario. General position of points a and b and circle C in the space is described by seven parameters (a_x, a_y, b_x, b_y, c_x, c_y, r), where c_x and c_y are the coordinates of center c of C and r is its radius. Example of the general position of points a, b and the circle C is shown on Fig. 3.2.

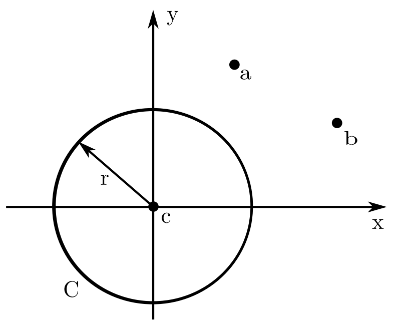


Figure 3.2: Position of points a, b and circle C with radius r and center c in the space described by seven parameters

To simplify the computation, we can transform this space to a space that is described by three parameters (a_y, b_x, b_y), by moving the circle center to the origin O, scaling to unit radius and rotating so that a lies on the *y-axis*. The transformation process (line 16) is shown on Fig. 3.3 and described as follows:

1. Points a and b are shifted by c (Fig. 3.3a):

$$a^f = (a^f_x, a^f_y) = (a_x - c_x, a_y - c_y);$$

$$b^f = (b^f_x, b^f_y) = (b_x - c_x, b_y - c_y);$$

2. Points a^f and b^f are scaled by *1/r* (Fig. 3.3b):

$$a^s = (a^s_x, a^s_y) = (\frac{a^f_x}{r}, \frac{a^f_y}{r})$$

$$b^s = (b^s_x, b^s_y) = (\frac{b^f_x}{r}, \frac{b^f_y}{r})$$

3. Angle α between half-line $\overrightarrow{\mathrm{Oa^s}}$ and *y-axis* is computed using function *arctan* (Fig. 3.3c):

$$\alpha = arctan(\overrightarrow{\mathrm{Oa^s}}, y - axis)$$

4. The resulting points a^t and b^t are computed by rotating points a^s and b^s about α (Fig. 3.3d):

$$a^t = (a^t_x, a^t_y) = (0,\ sin(\alpha) \cdot a^s_x + cos(\alpha) \cdot a^s_y)$$

$$b^t = (b^t_x, b^t_y) = (cos(\alpha) \cdot b^s_x - sin(\alpha) \cdot b^s_y,\ sin(\alpha) \cdot b^s_x + cos(\alpha) \cdot b^s_y)$$

Point p^t is then computed by the solver described in Section 3.1.3 and p is determined using inverse transformation function (line 18) from p^t, C_p and α as follows:

1. Point p^r is computed by rotating point p^t about -α:

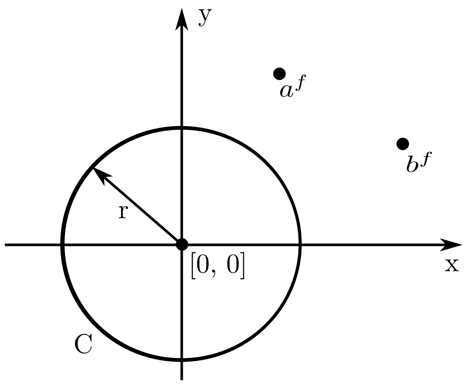
$$p^r = (p^r_x, p^r_y) = (cos(-\alpha) \cdot p^t_x - sin(-\alpha) \cdot p^t_y,\ sin(-\alpha) \cdot p^t_x + cos(-\alpha) \cdot p^t_y)$$

2. p^r is multiplied by r:

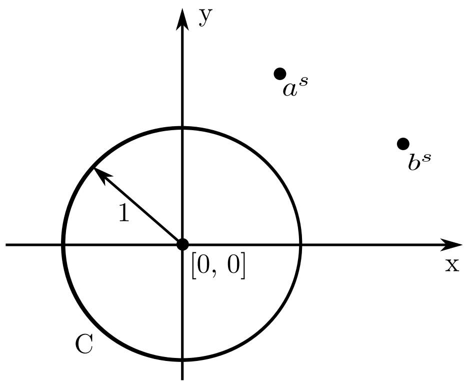
$$p^s = (p^s_x, p^s_y) = (p^r_x \cdot r,\ p^r_y \cdot r)$$

3. The resulting point p is computed by shifting p^s about c:

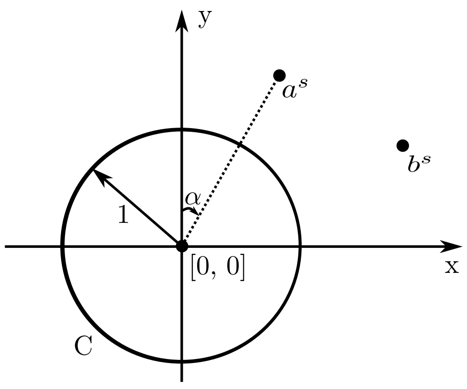
$$p = (p_x, p_y) = (p^s_x + c_x, p^s_y + c_y)$$



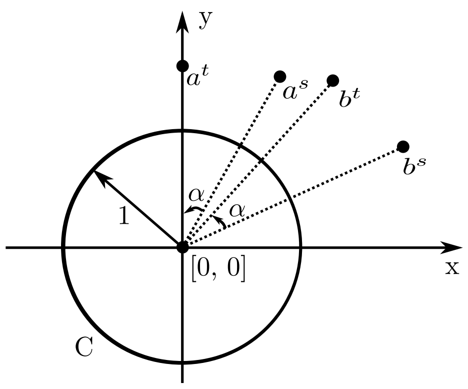
(a) : General points and the circle C shifted by center of circle c. The center of C is at the origin.



(b) : Shifted points and C scaled by *1/r*. The radius of circle C is equal to *1*.



(c) : Computed angle α between half-line $\overrightarrow{\mathrm{Oa}^s}$ and *y-axis*.



(d) : Points a^s and b^s rotated by angle α to result in transformed points a^t and b^t.

Figure 3.3: Transformation of points and a circle in space described by seven parameters to space described by three parameters.

3.1.2 Points on the line and circle

This section describes the computation of points on the line segment $\overline{ab}$ that intersects circle $C = (c, r)$, where c lies in the origin of the coordinate system $(0, 0)$. Let us formulate a point p by the parametric line equation of the direction vector $\vec{ab}$ and point a:

$$p = (p_x, p_y) = (a_x + t \cdot (b_x - a_x), a_y + t \cdot (b_y - a_y)), \tag{3.2}$$

where t is scalar value. To compute t, we use the equation of the circle described by point p:

$$p_x^2 + p_y^2 = r^2 \tag{3.3}$$

By substituting Eq. 3.2 into equation of a circle, we get

$$(a_x + t \cdot b_x - t \cdot a_x)^2 + (a_y + t \cdot b_y - t \cdot a_y)^2 = r^2 \tag{3.4}$$

By separating the variable t and solving Eq. 3.4, we obtain two solutions t_1 and t_2 of t as follows:

$$t_1 = \frac{n \cdot m - a_x\, b_x - a_y\, b_y + {a_x}^2 + {a_y}^2}{q} \tag{3.5}$$

$$t_2 = -\frac{n \cdot m + a_x\, b_x + a_y\, b_y - {a_x}^2 - {a_y}^2}{q} \tag{3.6}$$

Where n, m and q are:

$$n = \sqrt{-{a_x}^2\, {b_y}^2 + {a_x}^2\, r^2 + 2\, a_x\, a_y\, b_x\, b_y - 2\, a_x\, b_x\, r^2} \tag{3.7}$$

$$m = \sqrt{-{a_y}^2\, {b_x}^2 + {a_y}^2\, r^2 - 2\, a_y\, b_y\, r^2 + {b_x}^2\, r^2 + {b_y}^2\, r^2} \tag{3.8}$$

$$q = {a_x}^2 - 2\, a_x\, b_x + {a_y}^2 - 2\, a_y\, b_y + {b_x}^2 + {b_y}^2 \tag{3.9}$$

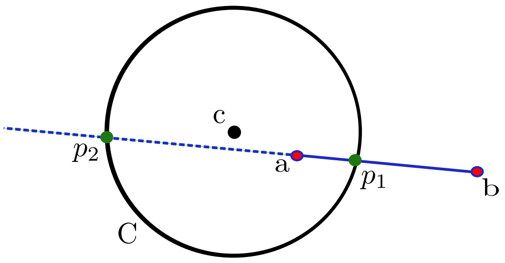
By substituting t_1 and t_2 into equation of p (Eq. 3.2), we get two points p_1 and p_2 that lie on C and line $\overleftrightarrow{ab}$.

$$p_1 = (p_x, p_y) = (a_x + t_1 \cdot (b_x - a_x), a_y + t_1 \cdot (b_y - a_y)) \tag{3.10}$$

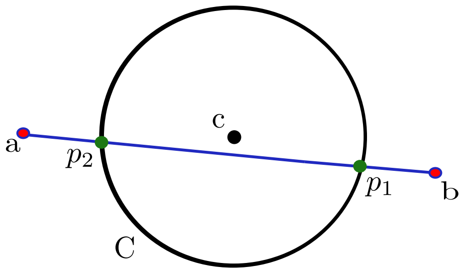
$$p_2 = (p_x, p_y) = (a_x + t_2 \cdot (b_x - a_x), a_y + t_2 \cdot (b_y - a_y)) \tag{3.11}$$

There are two different mutual positions of line segment $\overline{ab}$ and C that affect the number of points located both on C and $\overline{ab}$:

1. One of the points a, b is located inside C. Exactly one point p (p_1 or p_2) is located both on C and $\overline{ab}$. The value of parameter t decides which p will be located on $\overline{ab}$ as only one of the parameters t_1 or t_2 is equal to a value in range [0,1]. Fig. 3.4a shows an example, where point a is inside C and the point p_1 is the only point lying on both C and $\overline{ab}$.
2. Both points a and b are located outside C. Both points p_1 and p_2 are located on both C and $\overline{ab}$. An example is shown on Fig. 3.4b.



(a) : Point a is inside and point b is outside C. Only one point p_1 lies on C.



(b) : Points a and b are outside C. Two points p_1 and p_2 lie on C.

Figure 3.4: Two different mutual positions of line segment $\overline{ab}$ intersecting circle C and found points on $\overline{ab}$ and C.

3.1.3 Solving nontrivial case

The PCP solver of the nontrivial case aims to compute an optimal point $p = (p_x, p_y)$ located on the circle $C = ((0,0),1)$, while $a = (0, a_y)$ and $b = (b_x, b_y)$ are located outside C, i.e., the space described by three parameters is used. The solver minimizes the distances between p and a and between b and p.

The main idea is based on the light reflection theory, specifically on Snell's law that defines the reflection of light with the following equation [23]:

$$n_1 sin(\beta_1) = n_2 sin(\beta_2) \tag{3.12}$$

Where n_1 is the incident index, n_2 is the refracted index, β_1 is incident angle and β_2 is the refracted angle. Let us suppose the situation depicted in Fig. 3.5, where the incident and refracted light ray is on the same side of the medium M, i.e., $n_1 = n_2$. We get $\beta_1 = \beta_2$.

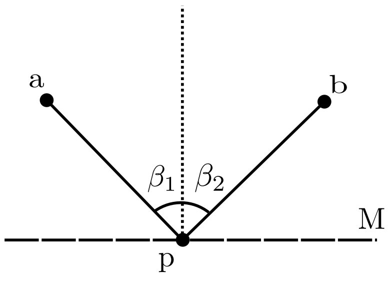


Figure 3.5: Light reflection

According to the following lemma, which is proved in Chou et al. (2008) [23], we are able to find the shortest path using light reflection.

Lemma: *With the incident angle β_1 equal to the reflection angle β_2, the length of traversal path along the line segments $\overline{ap}$ and $\overline{pb}$ is the shortest path.*

Assume a new point $a^* = (a^*_x, a^*_y)$, which is the mirroring of a in line M. Line segment $\overline{ap}$ has the same length as line segment $\overline{a^*p}$, which can be described by the following equation:

$$\sqrt{(a_x - p_x)^2 + (a_y - p_y)^2} = \sqrt{(a^*_x - p_x)^2 + \left(a^*_y - p_y\right)^2} \tag{3.13}$$

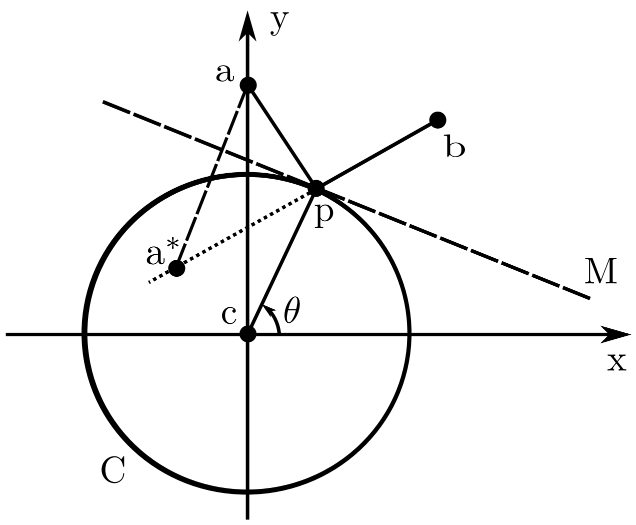


Figure 3.6: Principle of light reflection applied to the PCP problem

By applying the principle of light reflection to the PCP problem, we get into the situation depicted in Fig. 3.6, where M is the tangent line to C and θ is an angle that determines p using polar coordinates as follows:

$$p = (p_x, p_y) = (cos(\theta), sin(\theta)). \tag{3.14}$$

Point a^* can be expressed by a parametric equation as a point lying on a line segment passing through a with slope corresponding to point p. The line segment $\overline{a^*a}$ is parallel to $\overline{cp}$.

$$a^*_x = a_x + t \cdot p_x, \tag{3.15}$$

and

$$a^*_y = a_y + t \cdot p_y, \tag{3.16}$$

where t is a scalar value parameter of the equation. Parameter t is computed by substituting a^* into Eq. 3.13:

$$(a_x - p_x + p_x\, t)^2 + (a_y - p_y + p_y\, t)^2 - (a_x - p_x)^2 - (a_y - p_y)^2 = 0. \tag{3.17}$$

By rearranging the formula, we get:

$$t = 2 - 2 \cdot \frac{a_x p_x + a_y p_y}{{p_x}^2 + {p_y}^2} = \frac{2\, {p_x}^2 + 2\, {p_y}^2 - 2\, a_x\, p_x - 2\, a_y\, p_y}{{p_x}^2 + {p_y}^2}. \tag{3.18}$$

By assuming only three parameters, we can simplify Eq. 3.15 and 3.18 as

$$a^*_x = t \cdot p_x, \tag{3.19}$$

$$t = \frac{2\,{p_x}^2 + 2\,{p_y}^2 - 2\,a_y\,p_y}{{p_x}^2 + {p_y}^2}. \tag{3.20}$$

Eq. 3.16, 3.19 and 3.20 are derived equations of a^*. Since a^*, p and b are collinear, the line segments $\overline{a^*p}$ and $\overline{pb}$ are collinear as well and we get

$$\frac{p_y - a^*_y}{p_x - a^*_x} = \frac{p_y - b_y}{p_x - b_x}. \tag{3.21}$$

Eq. 3.21 can be substituted by a^* with parameter t as:

$$\frac{p_y - a_y - p_y \ \cdot t}{p_x - p_x \ \cdot t} = \frac{b_y - p_y}{b_x - p_x} \equiv t \cdot (b_x p_y - b_y p_x) = b_x p_y - b_x a_y + p_x a_y - p_x b_y \tag{3.22}$$

We can substitute p in the form of polar coordinates Eq. 3.14 into Eq. 3.22:

$$t \cdot (b_x sin(\theta) - b_y cos(\theta)) = b_x sin(\theta) - b_x a_y + a_y cos(\theta) - b_y cos(\theta) \tag{3.23}$$

To improve the readability of the equations, we create expressions from *sin* and sin^2 functions as follows:

$$z = sin^2(\theta) = 1 - cos^2(\theta), \tag{3.24}$$

$$\sqrt{z} = sin(\theta) = \sqrt{1 - cos^2(\theta)}. \tag{3.25}$$

Eq. 3.23 and parameter t (Eq. 3.20) can be simplified using parameter $\sqrt{z}$ as

$$t \cdot (b_x\sqrt{z} - b_y cos(\theta)) = b_x\sqrt{z} - b_x a_y + a_y cos(\theta) - b_y cos(\theta), \tag{3.26}$$

$$t = \frac{2\,cos^2(\theta) + 2 - 2cos^2(\theta) - 2a_y\sqrt{z}}{cos^2(\theta) + 1 - cos^2(\theta)} = 2 - 2a_y\sqrt{z}. \tag{3.27}$$

Since t is simplified Eq.3.27, we can substitute it into Eq. 3.26 and we obtain

$$b_x\sqrt{z} + 2a_y b_y cos(\theta)\sqrt{z} = b_x a_y + a_y cos(\theta) + b_y cos(\theta) - 2b_x a_y cos^2(\theta). \tag{3.28}$$

By substituting parameter $\sqrt{z}$ Eq. 3.25 into Eq. 3.28 as

$$\begin{aligned} &b_x\sqrt{1 - cos^2(\theta)} + 2a_y b_y cos(\theta)\sqrt{1 - cos^2(\theta)} = \\ &b_x a_y + a_y cos(\theta) + b_y cos(\theta) - 2b_x a_y cos^2(\theta), \end{aligned} \tag{3.29}$$

we obtained the equation with three requested parameters (a_y, b_x, b_y) and parameter θ to compute. To remove the square root, we square the equation and obtain

$$\begin{aligned} &-4\,{a_y}^2\,{b_y}^2\cos^4(\theta) + 4\,{a_y}^2\,{b_y}^2\cos^2(\theta) - 4\,a_y\,b_x\,b_y\cos^3(\theta) \\ &+ 4\,a_y\,b_x\,b_y\,\cos(\theta) - {b_x}^2\cos^2(\theta) + {b_x}^2 = \\ &+ 4\,{a_y}^2\,{b_x}^2\cos^4(\theta) - 4\,{a_y}^2\,{b_x}^2\cos^2(\theta) + {a_y}^2\,{b_x}^2 - 4\,{a_y}^2\,b_x\cos^3(\theta) \\ &+ 2\,{a_y}^2\,b_x\,\cos(\theta) + {a_y}^2\cos^2(\theta) - 4\,a_y\,b_x\,b_y\cos^3(\theta) + 2\,a_y\,b_x\,b_y\,\cos(\theta) \\ &+ 2\,a_y\,b_y\cos^2(\theta) + {b_y}^2\cos^2(\theta) \end{aligned} \tag{3.30}$$

By rearranging the equation, we get the final polynomial of degree four of variable *cos(θ)*.

$$\begin{aligned}
&4\left({a_y}^2\,{b_x}^2+{a_y}^2\,{b_y}^2\right)cos^4(\theta)\\
&-4\left({a_y}^2\,b_x\right)cos^3(\theta)\\
&+\left(-4\,{a_y}^2\,{b_x}^2-4\,{a_y}^2\,{b_y}^2+{a_y}^2+2\,a_y\,b_y+{b_x}^2+{b_y}^2\right)cos^2(\theta)\\
&+2\left({a_y}^2\,b_x-a_y\,b_x\,b_y\right)cos(\theta)\\
&+\left({a_y}^2\,{b_x}^2-{b_x}^2\right)=0
\end{aligned} \tag{3.31}$$

The following are the four solutions of the polynomial:

$$cos(\theta)=\begin{cases}\frac{1}{4}z+n-k\sqrt{q+t}\\ \frac{1}{4}z+n+k\sqrt{q+t}\\ \frac{1}{4}z-n-k\sqrt{q-t}\\ \frac{1}{4}z-n+k\sqrt{q-t}\end{cases} \tag{3.32}$$

where z, n, t, q and k are

$$z=\frac{4\,{a_y}^2\,b_x}{\mathrm{ab}}, \tag{3.33}$$

$$n=\frac{\sqrt{m}}{6\,l^{1/6}}, \tag{3.34}$$

$$t=3\,\sqrt{6}\,h\,\sqrt{2\,g^3-72\,f\,g+27\,h^2+3\,\sqrt{3}\,\mathrm{ch}}, \tag{3.35}$$

$$q=-12\,f\,\sqrt{m}-g^2\,\sqrt{m}-9\,l^{2/3}\,\sqrt{m}-12\,g\,l^{1/3}\,\sqrt{m}, \tag{3.36}$$

$$k=\frac{1}{6\,l^{1/6}\,m^{1/4}}. \tag{3.37}$$

Parameters ab, c, d, e, f, g, h, ch, l and m are computed as:

$$ab = 4\,{a_y}^2\,{b_x}^2 + 4\,{a_y}^2\,{b_y}^2 \tag{3.38}$$

$$c = -4\,{a_y}^2\,{b_x}^2 - 4\,{a_y}^2\,{b_y}^2 + {a_y}^2 + 2\,a_y\,b_y + {b_x}^2 + {b_y}^2 \tag{3.39}$$

$$d = 2\,{a_y}^2\,b_x - 2\,a_y\,b_x\,b_y \tag{3.40}$$

$$e = {a_y}^2\,{b_x}^2 - {b_x}^2 \tag{3.41}$$

$$f = \frac{e}{\mathrm{ab}} - \frac{3\,z^4}{256} + \frac{c\,z^2}{16\,\mathrm{ab}} + \frac{d\,z}{4\,\mathrm{ab}} \tag{3.42}$$

$$g = \frac{c}{\mathrm{ab}} - \frac{3\,z^2}{8} \tag{3.43}$$

$$h = \frac{d}{\mathrm{ab}} - \frac{z^3}{8} + \frac{c\,z}{2\,\mathrm{ab}} \tag{3.44}$$

$$ch = \sqrt{-256\,f^3 + 128\,f^2\,g^2 - 16\,f\,g^4 - 144\,f\,g\,h^2 + 4\,g^3\,h^2 + 27\,h^4} \tag{3.45}$$

$$l = \frac{g^3}{27} - \frac{4\,f\,g}{3} + \frac{h^2}{2} + \frac{\sqrt{3}\,\mathrm{ch}}{18} \tag{3.46}$$

$$m = \frac{12\,e}{\mathrm{ab}} - 6\,g\,l^{1/3} + g^2 + 9\,l^{2/3} - \frac{9\,z^4}{64} + \frac{3\,c\,z^2}{4\,\mathrm{ab}} + \frac{3\,d\,z}{\mathrm{ab}} \tag{3.47}$$

Since $cos(\theta) = cos(-\theta)$, we have

$$cos(-\theta) = \begin{cases} \frac{1}{4}z + n - k\sqrt{q+t} \\ \frac{1}{4}z + n + k\sqrt{q+t} \\ \frac{1}{4}z - n - k\sqrt{q-t} \\ \frac{1}{4}z - n + k\sqrt{q-t} \end{cases} \tag{3.48}$$

By deriving θ from Eq. 3.32 and 3.48, we obtain the following eight solutions of θ:

$$\theta = \begin{cases} \theta_1 = cos^{-1}(\frac{1}{4}z + n - k\sqrt{q+t}) \\ \theta_2 = -cos^{-1}(\frac{1}{4}z + n - k\sqrt{q+t}) \\ \theta_3 = cos^{-1}(\frac{1}{4}z + n + k\sqrt{q+t}) \\ \theta_4 = -cos^{-1}(\frac{1}{4}z + n + k\sqrt{q+t}) \\ \theta_5 = cos^{-1}(\frac{1}{4}z - n - k\sqrt{q-t}) \\ \theta_6 = -cos^{-1}(\frac{1}{4}z - n - k\sqrt{q-t}) \\ \theta_7 = cos^{-1}(\frac{1}{4}z - n + k\sqrt{q-t}) \\ \theta_8 = -cos^{-1}(\frac{1}{4}z - n + k\sqrt{q-t}) \end{cases} \tag{3.49}$$

By substitution of θ into polar form of p (Eq. 3.14), we obtain eight final solutions of p:

$$p_1 = \begin{cases} p_x = \frac{1}{4}z + n - k\sqrt{q+t} \\ p_y = \sqrt{1 - \left(\frac{1}{4}z + n - k\sqrt{q+t}\right)^2} \end{cases} \tag{3.50}$$

$$p_2 = \begin{cases} p_x = \frac{1}{4}z + n - k\sqrt{q+t} \\ p_y = -\sqrt{1 - \left(\frac{1}{4}z + n - k\sqrt{q+t}\right)^2} \end{cases} \tag{3.51}$$

$$p_3 = \begin{cases} p_x = \frac{1}{4}z + n + k\sqrt{q+t} \\ p_y = \sqrt{1 - \left(\frac{1}{4}z + n + k\sqrt{q+t}\right)^2} \end{cases} \tag{3.52}$$

$$p_4 = \begin{cases} p_x = \frac{1}{4}z + n + k\sqrt{q+t} \\ p_y = -\sqrt{1 - \left(\frac{1}{4}z + n + k\sqrt{q+t}\right)^2} \end{cases} \tag{3.53}$$

$$p_5 = \begin{cases} p_x = \frac{1}{4}z - n - k\sqrt{q-t} \\ p_y = \sqrt{1 - \left(\frac{1}{4}z - n - k\sqrt{q-t}\right)^2} \end{cases} \tag{3.54}$$

$$p_6 = \begin{cases} p_x = \frac{1}{4}z - n - k\sqrt{q-t} \\ p_y = -\sqrt{1 - \left(\frac{1}{4}z - n - k\sqrt{q-t}\right)^2} \end{cases} \tag{3.55}$$

$$p_7 = \begin{cases} p_x = \frac{1}{4}z - n + k\sqrt{q-t} \\ p_y = \sqrt{1 - \left(\frac{1}{4}z - n + k\sqrt{q-t}\right)^2} \end{cases} \tag{3.56}$$

$$p_8 = \begin{cases} p_x = \frac{1}{4}z - n + k\sqrt{q-t} \\ p_y = -\sqrt{1 - \left(\frac{1}{4}z - n + k\sqrt{q-t}\right)^2} \end{cases} \tag{3.57}$$

The computed points p are visualized in Fig. 3.7, where the red line highlights the distance from a and b to the closest p (p_1 in this case). This p_1 is considered to be the optimal point p.

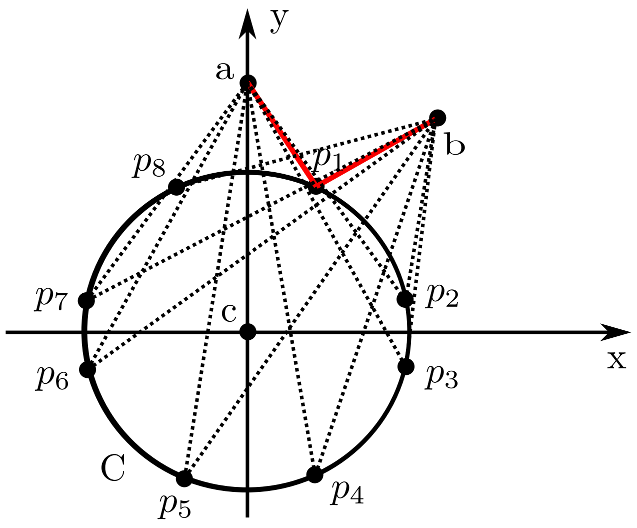


Figure 3.7: Eight computed solutions of point p by PCP solver

The solutions (Eq. 3.50 - 3.57) are inspired by the solutions proposed by Chou (2008) [23]. Initially, the equations (12 - 24) in the original paper were used to solve the nontrivial case. However, these equations contain several transcription errors that prevent the PCP from working correctly. Moreover, five parameters (a_x, a_y, b_x, b_y, r) are used in the original solution instead of three parameters (a_y, b_x, b_y).

Precomputed solution

The previous equations allow us to compute approximately 22900 PCP problems per second as the experimental results show. Although the number of PCP problems is high, we want to make the computation process even faster due to its frequent usage in the complex algorithms (GLNS-CETSP and TCP). One of the possible solutions is to precompute a three-dimensional table of solutions of p for given ranges of three parameters (a_y, b_x, b_y). The table can be precomputed only once at the beginning of the program or stored in the file and loaded to program memory on demand.

The process of precomputation can be divided into two separate phases: storing values and loading values. The first phase can be described by the following steps:

1. The three-dimensional table $\mathbb{R}^{n \times m \times k}$ is created. Each dimension corresponds to parameter's range of values.
2. Point b is converted from Cartesian representation of points (b_x^t, b_y^t) to polar representation of points (s, ϕ), i.e., the parameters (a_y, b_x, b_y) are converted into (a_y, s, ϕ).

$$s = \sqrt{b_x^2 + b_y^2} \tag{3.58}$$

$$\phi = arctan(b_y, b_x) \tag{3.59}$$

3. The three-dimensional space $\mathbb{R}^{n \times m \times k}$ of the parameters is sampled, where each parameter is from a discrete interval defined by lower bound l, upper bound u and step (increment) i.
4. Values of angle θ representing the point p computed and stored for each combination of values of the parameters. To minimize the size of the table in memory, values of θ are precomputed only in range [-$\frac{\pi}{2}$, $\frac{\pi}{2}$] corresponding an area of $I.$ and $IV.$ quadrants. Fig. 3.8 shows example of position of points a and $\mathbf{b}$ and these quadrants as a light red area. The rest of these values are mirrored by angle $\frac{\pi}{2}$ around y-axis in loading process.

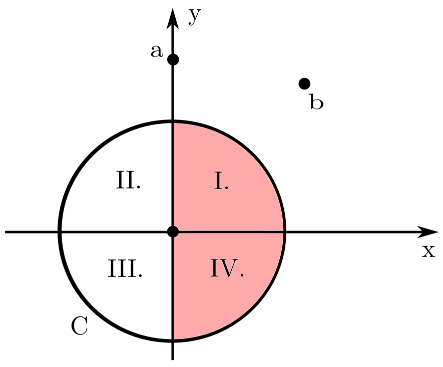


Figure 3.8: Quadrants of circle C, where the angle is computed (light red area).

The second phase describes the loading of precomputed p values. It can be described by the following steps, where the inputs of this phase are the transformed values a_y^t, b_x^t and b_y^t (Algorithm 1 line 16):

1. Point b^t is mirrored to $I.$ and $IV.$ quadrant of circle around y-axis shown on Fig. 3.8.

2. Mirrored point b^t is converted from Cartesian representation of points (b_x^t, b_y^t) to polar representation of points (s, ϕ).

3. a_y^t, s and ϕ are converted to the corresponding indexes in array based on their values:

$$idx_a = \frac{a_y^t - l_a}{i_a};\ idx_s = \frac{s - l_s}{i_s}; idx_\phi = \frac{\phi - l_\phi}{i_\phi}; \tag{3.60}$$

   Where idx is an index in the array of the corresponding parameter, l is a lower bound defined in storing phase and i is the increment defined during storing data.

4. Instead of computing p^t (Algorithm 1 line 17), the value of angle θ is loaded from the table based on the indexes and mirrored back to original quadrant of circle. p^t is then computed as follows:

$$p_x^t = cos(\theta) \tag{3.61}$$

$$p_y^t = sin(\theta) \tag{3.62}$$

   If one of the indexes is out of table's range, this step is skipped and p^t is computed the way without precomputed values.

3.2 Touring polygon problem (TPP)

The touring polygon problem (TPP) [15] is an algorithm that finds the optimal path on a sequence of polygons. More specifically, the general TPP finds the shortest path on n convex polygons consisting of points, where each convex polygon is visited exactly once. The order of convex polygons is fixed and can not be changed.

However, in the thesis, we find the shortest path on circular tour T instead of a tour on the sequence of polygons. We modified the general TPP for this purpose and named it "Touring circle problem" (TCP). The general TPP algorithm was picked and modified because of the easy implementation and quick solving speed [15].

3.3 Touring circle problem (TCP)

The touring circle problem (TCP) is a modified version of the TPP, which finds the optimal circular tour $T = (T^C, T^P)$ for a fixed order of circles, i.e., T^C does not change.

The following sections describe the TCP algorithm and compare the computations on the map with and without obstacles. Section 3.3.1 describes the algorithm using a pseudocode.

3.3.1 Algorithm description

The TCP algorithm is described by Algorithm 2. The input of the algorithm is the circular tour T to optimize. The algorithm's output is already optimized circular tour T_{out}.

The first step of the algorithm is to randomly initialize the path T^P of the circular tour T (line 1). The central part of the algorithm consists of two nested cycles. The first one counts the iterations and exits the program loop when the stopping criteria (detailed in Section 9) are met. The second cycle iterates through the path of the circular tour T and finds the optimal points on the path (line 7). More specifically, for the *jth* iteration, the point T_j^P on C_j that optimizes the distance between its preceding and succeeding point on T^P is computed using the point-circle-point solver. The preceding point (labeled as a) is generated by the *predecessor(j)* function (line 4). This function initially computes an index i of the preceding point:

$$i = j \ominus 1,$$

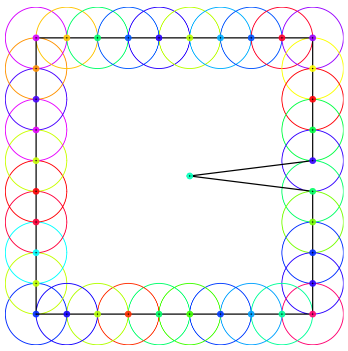
where the symbol $\ominus$ is the subtraction modulo m, where m is the number of circles in the tour T. The point a is then selected from *ith* position of the path: $a = T_i^P$. The succeeding point (labeled as b) is generated by the *successor(j)* function (line 5). This function initially computes an index k of the succeeding point:

$$k = j \oplus 1,$$

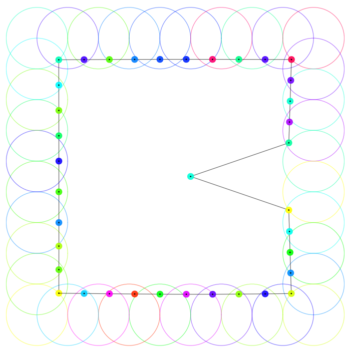
where the symbol $\oplus$ is the addition modulo m. The point b is then selected from *kth* position of the path: $b = T_k^P$. Fig. 3.9 shows the optimization process of TCP for two different CETSP instances *Bubbles1* and *ConcentricCircle1*. Subfig. 3.9a and 3.9c represent the instances with the tour corresponding to the initial order of the centers of circles. Subfig. 3.9b and 3.9d show found T using TCP after six iterations in case of *ConcentricCircle1* and twenty iterations in case of *Bubbles1*.

Algorithm 2: TCP

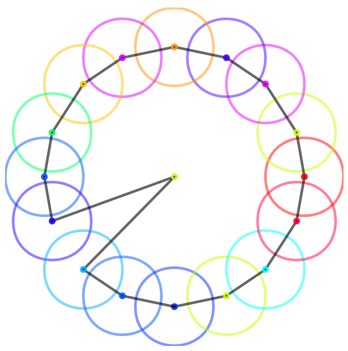
```
  Input: Tour to optimize $T$
  Output: Optimal tour $T_{out}$
1 $T^P \leftarrow$ random_init_path($T^P$) ; // randomly initialize path $T^P$
2 repeat
3     for $j = 1$ to $|T|$ do
4         a ← predecessor($j$);
5         b ← successor($j$);
6         circle ← $T_j^C$;
7         $T_j^P \leftarrow$ find_optimal_point(a, b, circle);
8 until stopping criteria are met ;
9 $T_{out} \leftarrow T$;
```



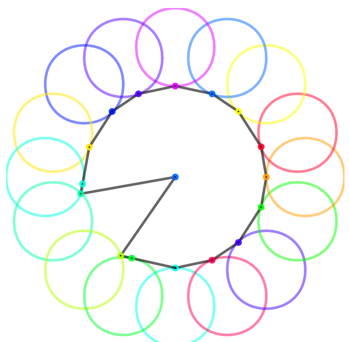
(a) : *Bubbles1* with initial order or circles, where the T touring through centers of circles.



(b) : *Bubbles1* with computed T using by TCP algorithm.



(c) : *ConcetricCircles1* with initial order or circles, where the T touring through centers of circles.



(d) : *ConcetricCircles1* with computed T using by TCP algorithm.

Figure 3.9: Computation of T using TCP algorithm for two different CETSP instances (*Bubbles1* and *ConcentricCircles1*).

Stopping criteria

The stopping criteria (Algorithm 2 line 8) represent several conditions used to stop the main cycle of the algorithm. The choice of these conditions has a key role in the efficiency of the whole algorithm. The conditions are chosen to prevent the long run of the algorithm and oscillate the length of the computed circular tour T.

Two types of conditions exit the main cycle of the algorithm: *regular* and *early*. The regular type occurs when the difference between the length of the previous circular tour (circular tour in the previous iteration) and the current circular tour is lower than some reference value e_{ref}.

The early type occurs when the count of iterations reaches $limit_{iter}$ value. Therefore, the finishing of the algorithm is guaranteed. If $limit_{iter}$ value is correctly chosen, the early condition happens occasionally and only for big circular tours (the tour consisting of one hundred circles or more) or in case of a wrongly chosen e_{ref} value.

3.4 GLNS algorithm

The GLNS algorithm solves the exactly-one-in-set GTSP problem defined in Section 2.5. This algorithm is originally presented in the article [24] together with experimental results on several well-known GTSP instance libraries (GTSP-Lib, BAF-Lib, $GTSP^+$-Lib, and SAT-Lib). The results are compared with the best heuristic algorithms (GK [25] and GLKH [14]), solving this type of problem available at that time. The GLNS algorithm is competitive with the algorithms mentioned above on GTSP-Lib and finds higher quality solutions on other libraries.

At the core of the GLNS solver is the adaptive large neighborhood search (ALNS) algorithm, which was initially proposed for pick-up and delivery problems [26]. The main idea is as follows. In the beginning, the initial solution is created and then repeatedly destroyed and repaired. If a better solution is found, then it is accepted. The destroy and repair procedures are repeated until the stopping criterion is met. The insertion and removal mechanisms, such as nearest, farthest, and random, were used to destroy and repair solutions [24].

The following subsections overview the general GLNS algorithm for exactly-one-in-set GTSP problem. A detailed description of the GLNS algorithm can be found in [24]. Section 3.4.1 explains the solver framework together with the given pseudocode. Section 3.4.2 proposes three different modes: fast, medium, and slow. Section 3.4.3 presents different insertion heuristics and their unification in one unified insertion heuristic. Section 3.4.4 describes three removal heuristics. Section 3.4.5 introduces two types of optimization: Re-opt and Move-opt.

3.4.1 Solver description

The GLNS algorithm is described by pseudocode (Algorithm 3). The algorithm's input is the requested mode, and the output is a found optimal tour T_{out}. The whole algorithm works with three nested loops. The first one (lines 5 - 25) initializes the tour and updates the insertion and removal heuristics and their weights based on the algorithm's phase (early, mid, late). Each iteration can be termed as the cold restart of the algorithm. Number of cold restarts is determined by N_{cold} value. The second one (lines 12 - 23) is the algorithm's warm restart. This cycle makes a copy of the currently best tour (line 13). Number of warm restarts is determined by N_{warm} value. The third one is called *remove insert loop* and performs the removal and insertion of the N_r vertices from the tour (tour reconstruction). The process of tour reconstruction is done by *remove_insert* function using removal and insertion heuristics (line 16). The number N_r is selected uniformly randomly from {1, ..., N_{max}} in each iteration. The new tour T_{new} can be accepted or not, which is determined by applying the standard simulated annealing criterion. The new tour is locally optimized and replaces the currently best tour if the new tour is accepted and the current tour's length is better than the length of the currently best tour. *Remove insert loop* is repeated until the stopping criterion is met. Finally, the lowest tour is locally optimized with higher precision after finishing all iterations.

Stopping criteria

The stopping criterion is used in the solver framework (Algorithm 3 line 23) in the *remove insert loop*. Its main purpose is to optimize the number of calling of removal and insertion heuristics. There are two types of stopping criteria. The first one is met when the specific number of non-improving tour cycles in a row is reached. The non-improving tour cycle is a cycle of *remove insert loop* when the current tour $T_{current}$ is not accepted as the best tour (T_{best}). The specific number of non-improving tour cycles is given by the number called $count_{first}$ in the first turn of the algorithm and the number $count_{latest}$ in the second and other turns. One turn of the algorithm is cycles of *remove insert loop* continuing until $T_{current}$ is accepted as the new best tour T_{best}. If the number of non-improving tour cycles is equal to $count_{first}$ in the first turn or $count_{latest}$ in the second and other turns the *remove insert loop* is finished. The number of turns and the number of non-improving tour cycles are reset after starting each new *warm restart loop*. The numbers $count_{first}$ and $count_{latest}$ have different values in each mode of the algorithm.

Algorithm 3: GLNS(G, $\mathcal{S}$)

```
   Input: Mode of algorithm $m$ ;        // m = {fast, medium, slow}
   Output: Optimal tour $T_{out}$ on $G$
   Data: GTSP instances(G, $\mathcal{S}$)
1  Precompute_distances(G, $\mathcal{S}$) ;  // precompute distances between
     each pair of vertices in $\mathcal{S}$
2  $T_{init} \leftarrow$ init(G, $\mathcal{S}$) ;                       // initialize tour
3  $T_{lowest} \leftarrow T_{init}$ ;                        // intialize lowest tour
4  $N_{cold}$, $N_{warm} \leftarrow$ init_params($m$) ; // intialize parameters based
     on the mode of algorithm
5  for $j$ = 1 to $N_{cold}$ do
6      $T_{best} \leftarrow T_{init}$;
7      if $j$ == 1 then
8          Init insertion (I) and removal (R) heuristics
9      else
10         Update selection weights
11         Update I and R based on the selection weights
12     for 1 to $N_{warm}$ do
13         $T_{current} \leftarrow T_{best}$;
14         repeat
15             Select uniformly randomly a number of vertices to remove
                 $N_r$ from {1, ..., $N_{max}$}
16             $T_{new} \leftarrow$ remove_insert($T_{current}$, I, R, $N_r$);
17             if accept_tour($T_{new}$,$T_{current}$) then
18                 $T_{current} \leftarrow T_{new}$;
19                 Record the improvement made by R and I;
20             if len($T_{best}$) > len($T_{current}$) then
21                 Optimize($T_{current}$);
22                 $T_{best} \leftarrow T_{current}$;
23         until stopping criterion is met ;
24     if len($T_{lowest}$) > len($T_{best}$) then
25         $T_{lowest} \leftarrow T_{best}$
26 Optimize ($T_{lowest}$);
27 $T_{out} \leftarrow T_{lowest}$;
```

Assume that the total iteration of the whole algorithm is equal to one cycle of *remove insert loop* and is not reset after each *cold restart loop* and *warm restart loop*. The second stopping criterion is met when the order of circles in T_{best} and in $T_{current}$ is the same in the specific number of cycles in a row. The order of circles in tours T_{best} and $T_{current}$ is considered to be the same if one of the following states is fulfilled:

1. Positions of all circles in T^C_{best} are the same in $T^C_{current}$.
2. Circles in $T^C_{current}$ are shifted in a ring order with comparison to T^C_{best}.
3. Circles in $T^C_{current}$ are in a reverse order with comparison to T^C_{best}.
4. Circles in $T^C_{current}$ are shifted and in a reverse ring order with comparison to T^C_{best}.

The specific number is given by N_{same} value, which can differ in each mode of the algorithm. This stopping criterion is valid after reaching the specific number of total iterations to prevent early termination of the algorithm, which can lead to an incorrect solution. N_{start} value determines the specific number of total iterations and can differ in each mode of algorithm.

3.4.2 Modes of the algorithm

The main purpose of different modes is to regulate the ratio between the length of the found tour and the execution time for a given data instance. The description of all modes is as follows:

- **Fast mode** minimizes the execution time of the algorithm at the cost of possibly lengthening the tour. The number of cold and warm restarts is sufficiently small to prevent a long run of the algorithm.
- **Medium mode** tries to shorten length of tour by increasing the number of cold and warm restarts with compare to *fast mode*.
- **Slow mode** minimizes the length of the found tour at the cost of increasing execution time.

3.4.3 Insertion heuristics

The main purpose of insertion heuristics is to insert vertices to the tour and create a complete tour T from a partial tour T_s. The partial tour $T_s = (V_s, E_s)$ visits a subset of clusters S_s denoted as $S_s \subseteq \mathcal{S}$. V_s is a subset of vertices $V_s \subseteq V$ and E_s is a subset of edges $E_s \subseteq E$ in the graph G. Firstly, we define a distance *dist* between a cluster $(S_i \in \mathcal{S} \setminus S_s)$ and vertex $u \in V_s$ on T_s:

$$dist(S_i, u) = \min_{v \in S_i} \{\min\{w(v,u), w(u,v)\}\} \tag{3.63}$$

Secondly, we describe the insertion methods initially proposed in [26], which choose the optimal cluster ($S_{opt} \in \mathcal{S} \backslash S_s$):

1. **Nearest insertion** selects a cluster S_{opt} with minimal distance to any vertex u on the T_s:

$$S_{opt} = \arg\min_{S_i \in S \backslash S_s} \min_{u \in V_s} dist(S_i, u) \tag{3.64}$$

2. **Farthest insertion** selects a cluster V_s, whose closest vertex to any vertex on the T_s is maximal:

$$S_{opt} = \arg\max_{S_i \in S \backslash S_s} \min_{u \in V_s} dist(S_i, u) \tag{3.65}$$

3. **Random insertion** selects a cluster S_{opt} uniformly randomly from the set $\mathcal{S} \backslash S_s$.

4. **Cheapest insertion** selects a cluster S_{opt} containing the vertex $v \in S_{opt}$ that minimizes the insertion cost:

$$S_{opt} = \arg\min_{S_i \in S \backslash S_s} \min_{(x,y) \in E_s} \{w(x,v) + w(v,y) - w(x,y)\} \tag{3.66}$$

Furthermore, a vertex v must be chosen from S_{opt}, which was picked by one of the insertion methods. The process of picking v is described by the following steps:

1. Find an edge $(x, y) \in E_s$ that minimizes $w(x,v) + w(v,y) - w(x,y)$.
2. Delete the edge (x, y) from E_s.
3. Add the edges (x, v) and (v, y) to E_s.
4. Add the vertex v to V_s.

The process of selecting S_{opt} and picking v is repeated until T_s contains vertices from all clusters from $\mathcal{S}$ and the tour becomes a complete tour T .

Unified insertion

Because of the four different insertion heuristics, it is essential to find an effective way to switch between multiple insertion heuristics. The solution is the unified insertion heuristic, which unifies selecting a cluster $S_{opt} \in \mathcal{S} \backslash S_s$ for the nearest, farthest, and random insertion methods. The cheapest insertion heuristic is based on different techniques and can not be unified. Firstly, we define the minimum distance (d_{min}) between cluster $S_i \in \mathcal{S} \backslash S_s$ and partial tour T_s.

$$d_{min} = dist(S_i, T_s) = \min_{u \in V_s} dist(S_i, u)$$

Secondly, the heuristic framework for picking the cluster S_{opt} is described by Algorithm 4. Line 1 describes the precomputation of the distances between each pair of vertices in unused clusters $\mathcal{S} \backslash S_s$ using d_{min} formula. Line 2 describes the selection of the parameter k using the probability mass function, which is defined by a set of λ values and its powers from 0 to l. The value l is the number of unused clusters $|\mathcal{S} \backslash S_s|$. The next step is to pick S_{opt} based on the parameter k (lines 3 - 4).

The insertion heuristic type is specified by the parameter $\lambda \in [0, \infty)$. Only three values decide a specific insertion heuristic. The remaining values of λ define combinations of these insertion heuristics. Values (0,1) combine the random and the nearest insertion, and values (1, ∞) combine the nearest and the farthest insertion. Three specific values of λ are described as follows:

- Random insertion is obtained by setting $\lambda = 1$.
- Nearest insertion is achieved by $\lambda = 0$.
- Farthest insertion is received by $\lambda = \infty$ (or some large enough value).

The undefined state 0^0 is equal to 1 in our case. The original paper [24] shows the difference between performances using different λ parameters.

Algorithm 4: Unified insertion heuristic framework

```
  Input: Partial tour $T_s$ on $G$
  Output: The selected cluster $S_{opt}$
  Data: GTSP instances(G, $\mathcal{S}$)
1 Pre-compute_distances(G, $\mathcal{S} \backslash S_s$)
2 Select uniformly randomly $k$ from set $\{1, ..., l\}$ according to
    unnormalized probability mass function $[\lambda^0, \lambda^1, ..., \lambda^{l-1}]$
3 Sort an array of all unused clusters $(\mathcal{S} \backslash S_s)$ according to $d_{min}$ from
    lowest to highest
4 Pick the cluster $S_{opt}$ with the $k$th index from the array of all unused
    clusters
5 return $S_{opt}$
```

3.4.4 Removal heuristics

The main purpose of removal heuristics is to remove N_r vertices from a tour T. The removal heuristics initially proposed in [26], are briefly described below.

Worst removal

The worst removal heuristic removes the vertex v_j from a tour T that maximizes the removal cost r_j, which says how the vertices reduce the tour's length:

$$r_j = w(v_{j\ominus 1}, v_j) + w(v_j, v_{j\oplus 1}) - w(v_{j\ominus 1}, v_{j\oplus 1}), \tag{3.67}$$

While this operation removes only one vertex from the required N_r vertices, it is repeated N_r times.

Distance removal

The main idea of this heuristic is to remove vertices from T that are "close" to each other. At first, the uniformly randomly chosen vertex v_{seed} is removed from T. For each vertex v_i from the tour T without v_{seed} is computed distance d_i as follows:

$$d_i = min\{w(v_{seed}, v_i), w(v_i, v_{seed})\} \tag{3.68}$$

Vertex v_i with minimal distance d_i to v_{seed} is removed. The process is repeated until $N_r - 1$ vertices are removed from T. The last removed vertex corresponds to v_{seed}.

Segment removal

The main purpose of the segment removal heuristic is to resolve the possible local optima by destroying large and contiguous segments of N_r vertices of T. Firstly, the index j is uniformly randomly selected from the tour T. Secondly, vertices $v_j, v_{j\oplus 1}, ..., v_{j\oplus N_r-1}$ are removed from the tour T.

3.4.5 Tour optimization

The main reason for optimizing the tour is possibly shortening its length in a short amount of computation time by fixing some parts of a current solution. Tour optimization is applied several times in the algorithm (Algorithm 3 lines 21 and 26). Two types of optimization methods that can run consecutively or separately based on the mode of the algorithm are applied as follows:

1. **Re-Opt** attempts to optimize the tour T with fixed ordering of a set of clusters $\mathcal{S}$. The main idea is as follows:
   a. A direct acyclic graph (DAG) containing all vertices V from graph G is created.
   b. Each vertex from cluster V_i is connected to all vertices in the next cluster of the ordering.
   c. The vertices in the last cluster are connected to the first cluster of the ordering vertices.
   d. The optimized tour is the shortest path in the DAG graph, which starts and ends in the same vertex in the first cluster. The shortest path is found by the breadth-first search *BFS* algorithm.

2. **Move-Opt** attempts to optimize the ordering of vertices in T. Firstly, a vertex v from cluster $S_i \in \mathcal{S}$ is uniformly randomly selected and removed from the tour $T = (V_T, E_T)$.

   Secondly, a new vertex u from the same cluster S_i is reinserted to the position in T, where an insertion cost is minimal. Thus the order of vertices in T is changed. The insertion cost c_i is defined as:

   $$c_i = w(x,u) + w(u,y) - w(x,y);\ \forall u \in S_i;\ \forall (x,y) \in E_T;$$

   The reinsertion is repeated until N_{move} vertices is reinserted.

3.5 GLNS-CETSP

The original GLNS algorithm has valuable properties such as relatively easy adjustment to solve a slightly different problem (CETSP problem in our case) and a variety of parameters that enable the improvement of the algorithm's performance. Because of that, we used and modified this algorithm to solve the CETSP problem. More specifically, we try to find the circular tour T with a minimal length by this algorithm. Several modifications of the original GLNS were made to resolve the CETSP problem:

1. **Tour modification** - Instead of the general tour described by a graph G, it is used a circular tour T described by circles and points.
2. **Insertion heuristic modification** - The idea of all heuristics remains the same, but we work with circles instead of clusters. In addition, an emplace point heuristic is applied to optimize the inserting point from the picked circle to the circular tour T.
3. **Removal heuristic modification** - The modifications are minimal, and all methods remain the same except the usage of a different type of tour.
4. **Tour optimization modification** - Firstly, a completely different *Re-Opt* optimization solver is proposed. Instead of generating a direct acyclic graph (DAG), the TCP algorithm was used. Secondly, point reinsertion in the *Move-Opt* optimization process is different.
5. **Initial tour generation modification** - We propose two different methods to initialize circular tour T: random and TSP-based. The random method initializes the tour uniformly randomly, which is the same as in the original GLNS. The TSP-based method applies a TSP solver to initialize circular tour T.

The following subsections detail the modifications mentioned above. Section 3.5.1 describes the insertion heuristics and their modifications. Section 3.5.3 describes and compares the removal heuristics with the original GLNS. Section 3.5.2 describes the new emplace point heuristic used to select the optimal point on the circle and insert it to the optimal position in the circular tour T. Section 3.5.4 aims to describe the new *Re-Opt* optimization process and all differences in *Move-Opt* optimization process. Section 3.5.5 defines two methods for initializing the circular tour T.

3.5.1 Insertion heuristics

Assume a partial circular tour $T_s = (T_s^P, T_s^C)$, where $T_s^P \subseteq T^P$ and $T_s^C \subseteq T^C$. T_s visits only subset of circles $C_s \subseteq \mathcal{C}$.

Insertion heuristics find the optimal circle to insert $C_{opt} \in \mathcal{C} \setminus C_s$, compute point p_{opt} on C_{opt} and then insert p_{opt} to T_s^P and C_{opt} to T_s^C on the position in the T_s called emplace index.

The emplace index $i_{emplace}$ is the position in T_s that minimize insertion cost of T. Insertion cost (c_{insert}) of point $q \in T_s^P$ that is inserted between two points p_1, $p_2 \in T_s^P$ on T_s^P is defined as:

$$c_{insert}(q, p_1, p_2) = d(q, p_1) + d(p_2, q) - d(p_1, p_2) \tag{3.69}$$

The insertion methods, which find C_{opt} are described as follows:

1. **Nearest insertion** picks C_{opt} with a minimal distance to any circle $C_u \in T_s^C$:

$$C_{opt} = \arg\min_{C_i \in C \setminus C_s} \min_{C_u \in T_s^C} dist(C_i, C_u) \tag{3.70}$$

2. **Farthest insertion** picks C_{opt}, whose closest distance to T_s is maximal:

$$C_{opt} = \arg\max_{C_i \in C \setminus C_s} \min_{C_u \in T_s^C} dist(C_i, C_u) \tag{3.71}$$

3. **Random insertion** picks a circle C_{opt} uniformly randomly from the set $\mathcal{C}\backslash C_s$.

The process of computing p_{opt} on selected C_{opt} and finding emplace index $i_{emplace}$ is determined by "Emplace point heuristic".

The other heuristic called **cheapest insertion** selects C_{opt} and computes p_{opt} together with $i_{emplace}$ at the same time. It repeatedly calls "Emplace point heuristic" for all $C_i \in \mathcal{C} \backslash C_s$ and picks one together with its point p_i that minimizes c_{insert}:

$$C_{opt}, p_{opt} = \underset{C_i \in C \backslash C_s}{\arg\min} \min_{(p_1,p_2) \in T_s^P} \{c_{insert}(p_i, p_1, p_2)\} \tag{3.72}$$

The process of selecting C_{opt} and picking point p_{opt} is repeated until T_s^P contains points from all circles in T^C from the set of circles S_v and the tour becomes complete.

Unified insertion

The main idea of unified insertion is the same as in Section 3.4.3. The modifications used in our version are proposed as follows:

1. Instead of picking cluster $V_{opt} \in \mathcal{S} \backslash S_s$, the circle $C_{opt} \in \mathcal{C} \backslash C_s$ is picked to partial circular tour T_s.
2. The minimum distance d_{min} function is changed. d_{min} function is defined between $C_i \in \mathcal{C} \backslash C_s$ and T_s as:

$$d_{min} = dist(C_i, T_s) = \min_{C_u \in C_s} dist(C_i, C_u)$$

3.5.2 Emplace point heuristic

Emplace point heuristic is used as a part of the insertion heuristic and optimization process. It aims to find the emplace index $i_{emplace}$ effectively in the circular partial tour T and optimal point p_i on the circle C_i, which must be inserted into the tour.

The heuristic is described by a pseudocode (Algorithm 5). Its input is T and the circle C_i to insert to T. Output is the emplace index $i_{emplace}$ and the computed optimal point p_i on C_i. At the start, it is required to initialize the parameters as follows:

- *Indexes* is an array of indexes of sorted circles in T by distances to C_i from nearest to farthest (line 1).
- *Cost* represents the length of T (line 2) as defined in Section 2.3.
- *Min* represents the length of a shortest tour found so far (line 3).

The main cycle (lines 4 - 16) iterates through each consecutive pair of points in T and checks the optimal place to insert C_i. Firstly, the first index m is obtained from the sorted array *indexes* and k is the index of the next circle in T. The optimal point p_{opt} on the circle C_i is determined by the point-circle-point solver represented by $find\_optimal\_point$ function (line 11). As the solver is time-consuming, we want to prevent unnecessary calls of it. For this purpose, we applied lower-bound checking (line 10) comparing *cost* and lower-bound cost ($cost_{lb}$), where *cost* is initially equal to the tour's length. Lower-bound cost (line 9) is defined by summation of *cost*, Euclidean distance between points a and b, and lower-bound formula.

Lower-bound formula *lb* is defined as the shortest distance between two circles C_a and C_b on T and C_i, which is inserted between these two circles:

$$lb(C_a, C_b, C_i) = dist(C_a, C_i) + dist(C_i, C_b) = d(c_a, c_i) + d(c_i, c_b) - r_a - r_b - 2 \cdot r_i$$

3.5.3 Removal heuristics

Removal heuristics remove N_r points from the path T^P in circular tour T and their corresponding circles T^C. The modified heuristics are proposed as follows:

Algorithm 5: Emplace point heuristic

```
Input: Partial tour $T$ and circle $C_i$ to insert
Output: emplace index $i_{emplace}$ and optimal point $p_i$
1  indexes ← sort($T^C$, $C_i$) ;                  // Sort circles in tour
2  cost ← len(T);
3  min ← $\infty$;
4  for $j = 1$ to $|T|$ do
5      m ← indexes(j);
6      k ← m $\oplus$1;
7      a ← $T^P_m$ ;
8      b ← $T^P_k$ ;
9      $cost_{lb}$ ← cost - d(a, b) + $lb(T^C_m, T^C_k, C_i)$;
10     if $cost_{lb} < cost$ then
11         $p_{opt}$ ← find_optimal_point(a, b, $C_i$);
12         $cost_{min}$ ← cost + d(a, $p_{opt}$) + d($p_{opt}$, b) - d(a, b) ;
13         if $cost_{new} < min$ then
14             min ← $cost_{new}$;
15             $p_i$ ← $p_{opt}$;
16             $i_{emplace}$ ← k;
17 return $i_{emplace}$, $p_i$
```

1. **Worst removal** removes point $p_i \in T^P$ and its corresponding circle $C_i \in T^C$ that maximizes the removal cost r_i:

$$r_i = d(p_{i\ominus 1}, p_i) + d(p_i, p_{i\oplus 1}) - d(p_{i\ominus 1}, p_{i\oplus 1}), \tag{3.73}$$

   While this operation removes only one point and circle from the required N_r points and circles, it is repeated N_r times.

2. **Distance removal** uniformly randomly selects and removes point p_{seed} from T and its circle C_{seed}. For each point $p_j \in T^P$ except p_{seed} is computed distance d_j as follows:

$$d_j = min\{d(p_{seed}, p_j), d(p_j, p_{seed})\} \tag{3.74}$$

   Point p_j with minimal d_j to p_{seed} is removed from T^P together with circle C_j from T^C. The process is repeated until $N_r - 1$ points and circles are removed from T.

3. **Segment removal** removes a segment of points and their circles from T. Firstly, the index j is uniformly randomly selected from T. Secondly, points $p_j, p_{j\oplus 1}, ..., p_{j\oplus N_r - 1}$ are removed from T.

3.5.4 Tour optimization

In the GLNS-CETSP algorithm, the tour optimization has the same purpose as in the original GLNS algorithm (Section 3.4.5). The modifications are described as follows:

1. **Re-Opt** optimizes only points in path T^P with fixed order of circles in T^C. The optimization process is performed by the TCP algorithm. Because of the fixed order of points, this method can be fast assuming the correctly chosen parameters of the TCP algorithm.

2. **Move-Opt** optimizes the ordering of circles in T. Circle C_i is uniformly randomly selected and removed from T^C together with its point on the path T^P. The new optimal point p_i on C_i is computed by the "Emplace point heuristic" together with emplace index $i_{emplace}$. C_i is inserted to T^C on the position of $i_{emplace}$. The optimal point p_i is inserted to T^P on the position of $i_{emplace}$. This process is repeated until N_{move} points, and circles are reinserted to T. Limitations of this method are its time consummation for big N_{move} number and insufficient improvement of the tour length in case of small N_{move} number. Because of these limitations, this method is not used in the *fast* mode of the algorithm.

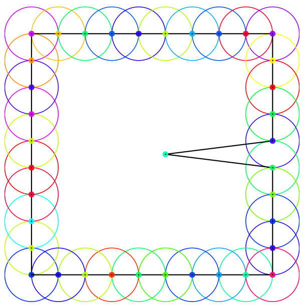
3.5.5 Initial tour generation

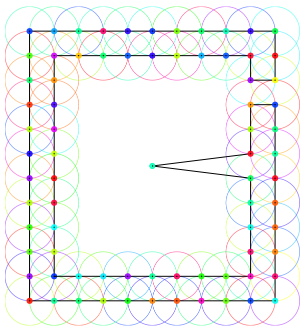
Initial tour of the GLNS-CETSP algorithm T_{init} (Algorithm 3 line 2) defines the initial order of circles in each cold restart loop (Algorithm 3 lines 5 - 25). Quality of T_{init} significantly affects the execution time and the solution of the GLNS-CETSP algorithm. Assume that the order of circles in T_{init} is initialized similar to the order of circles in the optimal solution. Therefore, it is more likely that the computational process of GLNS-CETSP will be faster and the algorithm's output tour T_{out} (Algorithm 3 line 27) will be shorter than in case of inappropriately chosen T_{init}, such as tour where the order of circles is mixed up undesirably. T_{init} is generated by an initialization method.

The GLNS-CETSP algorithm uses four different initialization methods: *random*, *random-insertion*, *unsupervised-learning-based* and *TSP-based* . The *random* method sets the order of circles of the tour T_{init} uniformly randomly. The main advantage is its speed of the initialization process and the disadvantage is a poor quality of a generated solution.

The *random-insertion* method adds circles to T_{init} using the unified insertion heuristic framework with λ=*1*.

The *TSP-based* method initializes the order of circles in T_{init} by solving the *TSP* problem on centers of circles. It was picked Lin–Kernighan–Helsgaun *LKH* heuristic [14] to solve the TSP problem because it is generally considered to be one of the most effective. The heuristic finds an optimal or near-optimal TSP tour T_{tsp}. *LKH* found in many cases the same order of circles as it is in the optimal solution of CETSP problem. On the other hand, *LKH* found completely different order of circles on more complex CETSP instances, where the circles overlap a lot. An example of the same order and completely different order of circles found by *LKH* on two different instances is shown on Fig.3.10. Nonetheless, despite the limitations, the LKH is still a sufficient heuristic used as a tour initializer.


(a) : Instance *Bubbles1* with found order of circles by LKH, which is the same as the order of circles in the optimal solution of CETSP problem for this instance.


(b) : Instance *Bubbles2* with order of circles found by LKH, which is completely different from order of circle in the optimal solution.

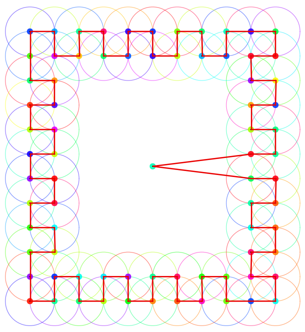

(c) : Instance *Bubbles2* with visualization of order of circles equal to order of circles in the optimal solution.

Figure 3.10: An example of found order of circle by *LKH* algorithm on two CETSP instances *Bubbles1* and *Bubbles2*.

The *unsupervised-learning-based* method initializes T_{init} by solving CETSP problem using an unsupervised learning procedure calling Growing Self-Organizing Array *GSOA* [8]. This procedure is very fast, i.e., GSOA finds the solution on a dataset with 1000 circles in less than one second but slower than *random* and *random-insertion* methods. Moreover, the order of circles is generally very close to the optimal solution and even the same for many instances.

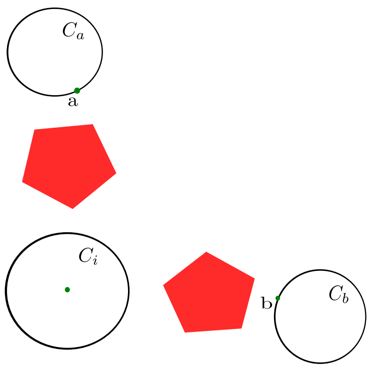
3.6 Environment with obstacles

Assume the variants $CETSP_{obst\_dis}$ and $CETSP_{obst\_int}$ of CETSP problem and *map* that specify the environment for these variations. The shortest collision-free path $path_{free}$ between two points on *map* is defined as set of points $path_{free} = \{p_1, p_2, \ldots, p_l\}$, where p_1 is a start point, p_l is an end point, and the rest of the points are collision-free points. The Euclidean distance metric can not be used to compute the distance between p_1 and p_l because the environment with polygonal obstacles is used. Nevertheless, the Euclidean distance metric is used to compute the distance between each pair of consecutive points in $path_{free}$.

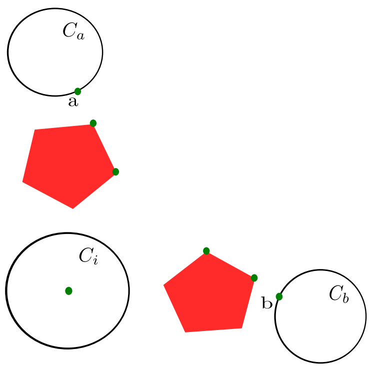
Two modifications of our algorithm are proposed to solve these variations of the CETSP problem:

1. Points a and b in TCP algorithm (Algorithm 2 lines 4 - 5) and emplace point heuristic (Algorithm 5 lines 7 - 8) are picked based on their $path_{free}$. More specifically, a new previous point a_{new} in T^P corresponds to the one before the last point in $path_{free}$ between $a \in T^P$ on circle $C_a \in T^C$ and the center of the current circle $C_i \in T^C$. If there is no obstacle between C_a and C_i, a_{new} corresponds to $p_1 \in path_{free}$, i.e., a_{new} is equal to a. A new next point b_{new} is computed the same way as a_{new} except that $path_{free}$ now starts from $b \in T^P$ on circle $C_b \in T^C$ and ends in the center of $C_i \in T^C$. Points a_{new} and b_{new} are the input of the "find optimal point function" in Algorithm 2 line 7 and Algorithm 5 line 11. The illustrated example of computing a_{new} and b_{new} is shown in Fig. 3.11.

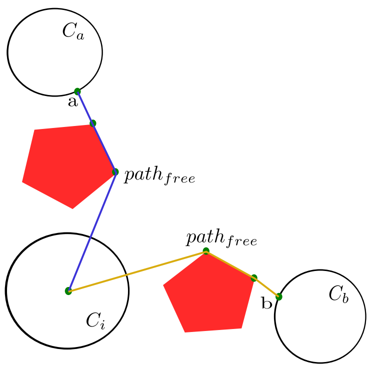
2. The found tour T_{out} in GLNS algorithm (Algorithm 3 line 27) is reconstructed, i.e., $path_{free}$ is computed between each pair of consecutive points in T^P_{out}. Fig. 3.12 shows the difference between final T_{out} and reconstructed final T_{out}



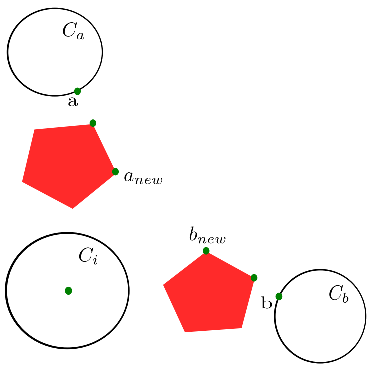
(a) : Points a and b on their circles and C_i with highlight center.



(b) : Created collision-free points on two $path_{free}$.

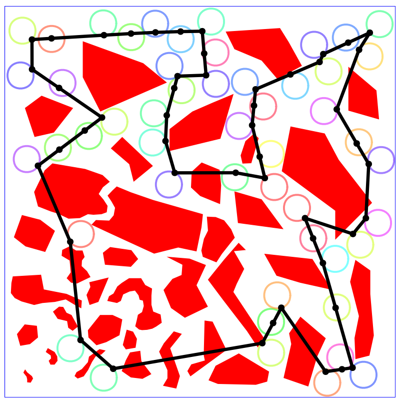


(c) : Created $path_{free}$ from a (blue) and from b (orange) to center of circle C_i.

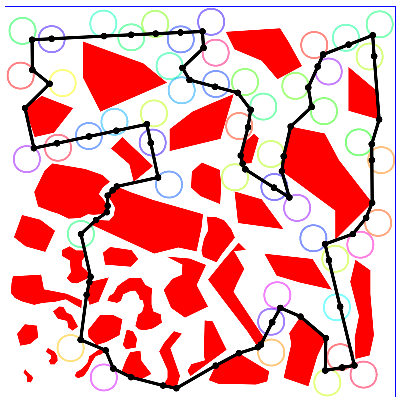


(d) : Determined new points a_{new} and b_{new} from created $path_{free}$.

Figure 3.11: Process of generating two collision free paths $path_{free}$ and computing points a_{new} and b_{new} on corresponding $path_{free}$.



(a) : Found T_{out}, which ignoring obstacles.



(b) : Reconstructed T_{out}, where the obstacles are not ignored.

Figure 3.12: Comparison of found two different T_{out}. The first ignore obstacles and the second tour is reconstructed and avoids the obstacles on the map.

3.6.1 Polyanya solver

The polyanya solver that is fast and effective was chosen to compute a $path_{free}$. This solver is initially presented by Michael Cui and Daniel D. Harabor, and Alban Grastien [27]. The approach of this solver can be divided into three steps as follows:

1. Convert *map* into navigation merged mesh called Constrained Delaunay Triangulation *M-CDT*.
2. Compute shortest Euclidean distances between consecutive points in the $path_{free}$ on *M-CDT* mesh.
3. Generate $path_{free}$.

Example of finding collision-free points and generating $path_{free}$ by polyanya solver is shown on Fig. 3.11. The subfigure *(a)* shows the three initial points (a, b and center of circle C_i). Between these points are found two collision-free paths $path_{free}$. The subfigure *(b)* shows generated all relevant collision-free points. The subfigure *(c)* shows two generated $path_{free}$.

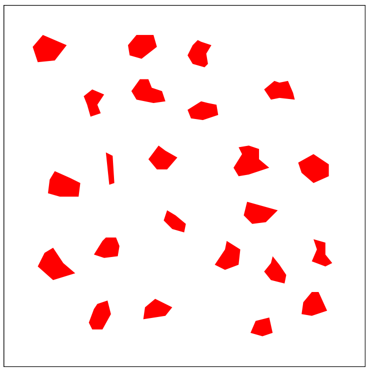
Conversion of the map to mesh

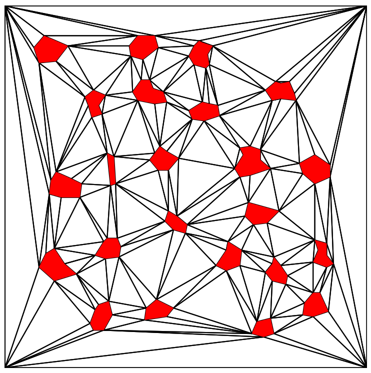
Assume set of s traversable simple polygons is defined $\mathcal{M} = \{M_1, \ldots, M_s\}$, where $M_i = (V_K, E_K)$. V_K is a sequence of m distinct vertices $V_K = \{V_1, V_2, \ldots, V_m\}$ and E_K is a sequence of $m-1$ edges $E_K = \{e_1, e_2, \ldots, e_{m-1}\}$ that joins vertices V_K. V_K and E_K are defined on graph G Section 2.2 as $V_K \subseteq V$ and $E_K \subseteq E$. The navigation mesh $\mathcal{N} = (\mathcal{M}, \mathcal{P})$, where $\mathcal{P}$ is set of n obstacles on the *map* defined in Section 2.6.

The *CDT* is a constrained Delaunay triangulation of traversable area on $\mathcal{N}$. Constrained Delaunay triangulation is described in [28]. The merged *M-CDT* is a *CDT*, which merges all polygons with sharing edges. The main purpose of *M-CDT* is to retain solving complexity and maximize the area of $\mathcal{M}$.

An example of the process of conversion on the input *map* called *potholes* is shown on Fig. 3.13 described as follows:

1. The input *map* is converted into *CDT* mesh using FADE2D library [29].
2. The *CDT* mesh is merged to *M-CDT* mesh .


(a) : Input *potholes* map with obstacles highlighted by red color.


(b) : From *potholes* is generated *CDT* mesh.

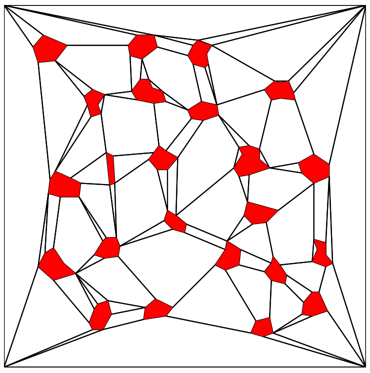

(c) : From *CDT* is generated *M-CDT* mesh.

Figure 3.13: An example of process of conversion *potholes* map.

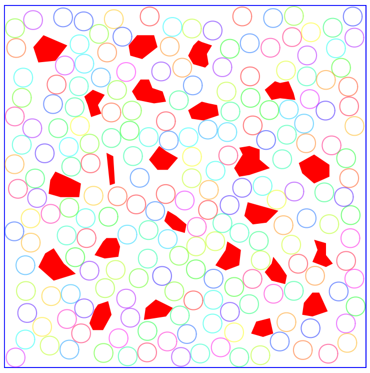
3.7 Generator of CETSP instances

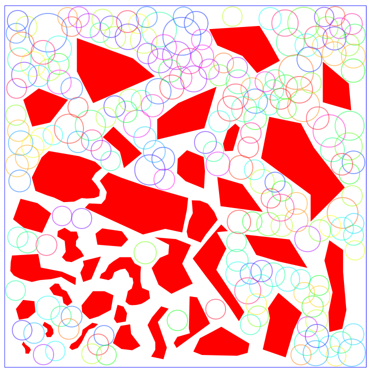
Suppose the variants $CETSP_{dis}$, $CETSP_{int}$, $CETSP_{obst\_dis}$, and $CETSP_{obst\_int}$ of CETSP problem and their *maps*. All *62* instances of *CETSP-lib* were solved by the *GLNS-CETSP* algorithm, but they fulfill the requirements only for $CETSP_{int}$ variant. *CETSP instances* that fulfill the requirements for the rest of these variants were generated by the generator of circles.

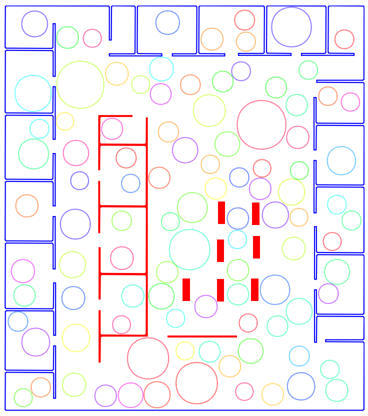
There are two types of *CETSP instances* that were generated based on the sizes of radii of circles: *random* and *fixed*. In *random* type, each $C_i \in \mathcal{C}$ has

r_i generated uniformly randomly in a given range. In *fixed* type, each $C_i \in \mathcal{C}$ has the same value of r_i. The radii of circles differ in each set $\mathcal{C}$ to fill the largest area of the traversable part of the map.

Four different maps were used to generate instances: *potholes*, *large*, *jari-huge* and *warehouse*. Used maps are shown in Fig. 3.14 together with different generated instances. Polygonal obstacles have a red color, the borders are symbolized by blue lines, and the circles have randomly assigned colors. Circles are generated on the maps with obstacles even for the $CETSP_{dis}$ variant, where the polygonal obstacles are neglected during the solving process. The generator's output is generated $\mathcal{C}$.


(a) : *Potholes* map together with 200 disjoint circles with fixed size of radii. Instance is called *disjoint_200*.


(b) : *Large* map together with 200 intersect circles with random size of radii. Instance is called *intersect_200_random*.


(c) : *Jari-huge* map together with 100 disjoint circles with random size of radii. Instance is called *disjoint_100_random*.

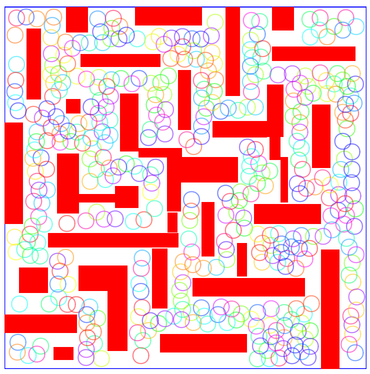

(d) : *Warehouse* map together with 400 intersect circles with fixed size of radii. Instance is called *intersect_400*.

Figure 3.14: All used *maps* describing the environment with polygonal obstacles.

The generator is described by the pseudocode Algorithm 6. The generator's inputs are the number of circles to generate m, Boolean value *random* choosing the type of CETSP instance and a map. The generator has one main loop (lines 1 - 12) that exits if the set $\mathcal{C}$ contains m circles (stopping criterion line 12). The loop starts generating a circle C_i with a center c_i inside the *map* (line 2) and radius r_i (line 3) uniformly randomly or with a fixed value. Three conditions must be fulfilled before the circle C_i is saved into $\mathcal{C}$:

1. C_i must be inside the border of *map* (line 5).
2. C_i can not collide with obstacles $\mathcal{P}$ on *map* (line 7). C_i collides with obstacle $P_j \in \mathcal{P}$ if intersect.
3. C_i can not collide with other circles in $\mathcal{C}$ (line 9). If the circles can intersect, the center of each circle can not be located inside other circles in $\mathcal{C}$. The circles can not intersect at all in the case of disjoint circles.

If one of these is not fulfilled, the iteration is skipped and the main loop is repeating.

Algorithm 6: Generator of CETSP instances

```
   Input: $Map$, $m$, $random$
   Output: Set of circles $\mathcal{C}$
 1 repeat
 2     $c_i \leftarrow$ generate_center($map$);
 3     $r_i \leftarrow$ generate($random$);
 4     $C_i \leftarrow$ create($c_i$, $r_i$);
 5     if $is\_not\_inside\_map(map, C_i)$ then
 6         continue with the next iteration;
 7     if $collide\_with\_obstacles(map, C_i)$ then
 8         continue with the next iteration;
 9     if $collide\_with\_other(\mathcal{C}, C_i)$ then
10         continue with the next iteration;
11     $\mathcal{C} \leftarrow$ save($C_i$);
12 until $stopping\ criterion\ is\ met$ ;
13 return $\mathcal{C}$
```

3.8 Code implementation

The majority of the algorithms described in the previous sections have been implemented from scratch in C++. Moreover, several libraries and implementations of other authors were used. These are described next together with their necessary modifications done to integrate them into the whole project.

3.8.1 GLNS-CETSP

The general GLNS algorithm described in Section 3.4 is initially implemented in the programming language Julia [1]. In the master thesis presented by David Woller [30], the GLNS algorithm is implemented in the programming language C++ and compared with the original version implemented in Julia. In addition, Jan Vidašič [31] in his master thesis modified the GLNS in C++ version to solve the traveling salesman problem with neighborhoods in a polygonal domain. We used both C++ versions of GLNS and modified them to solve the CETSP problem also in C++.

3.8.2 TCP

The *TPP* algorithm solving a tour on a sequence of polygons was implemented by Jan Vidašič as a part of the GLNS algorithm [31]. This implementation was used as a template for our *TCP* algorithm and completely modified to find a tour on a sequence of circles instead of polygons. The *TCP* is implemented in C++ and it is one of the parts of *the GLNS-CETSP* algorithm. Moreover, *TCP* can be used as a stand-alone algorithm.

3.8.3 LKH

The general LKH library [2] implemented in the programming language *C* was used to implement LKH in this thesis. The general LKH library requires tuning some internal parameters to achieve an effective solver tailored to our CETSP problem. It was done by supervisor Miroslav Kulich in the form of a library called *lkh* in code.

[1] Available at `https://github.com/stephenlsmith/GLNS.jl`
[2] Available at `http://webhotel4.ruc.dk/~keld/research/LKH/`

3.8.4 GSOA

The original implementation of GSOA proposed by J.Faigl [3] was used and edited to the library, which allows us to call the GSOA solver in our project. The modifications were done because the original solution is a stand-alone executable project without the option to call the solver directly from another project.

3.8.5 Polyanya solver

Polyanya solver and mesh converter [27] are implemented in the programming language C++ and provided in the form of already compiled executable files. These files can not be used for these reasons:

1. Mesh converter requires a different input map than we apply.
2. We need call of this solver directly from our algorithms.

Hence, we modified the original mesh converter to convert our type of maps to *M-CDT* mesh. In addition, we adjusted the solver to be callable in our algorithm.

[3] Available at `https://github.com/comrob/gsoa`

Chapter 4

Experiments

In this thesis, several experiments were proposed to measure the performance of implementation of *GLNS-CETSP* algorithm. Firstly, the performance of three different algorithm modes (fast, medium, and slow), four different initialization heuristics (random_insertion, random, LKH, and GSOA), and two versions of PCP solver called *simplified* and *precomputed* were compared sequentially. Secondly, the configuration of *GLNS-CETSP* with the best results is selected and compared with other algorithms. A new algorithm *GSOA+TCP* was proposed and compared with *GLNS-CETSP* and *GSOA*. Finally, *GLNS-CETSP* were tested on *CETSP* instances on four maps with polygonal obstacles.

Section 4.1 describes tools and methods including used parameters of the algorithms and the instances on which were the experiments evaluated. Parameters of *GLNS-CETSP*, *GLNS-GTSP*, and *PCP* algorithms are described in Section 4.1.1. All used instances on the maps with and without obstacles are described in Section 4.1.2. Description of the evaluation and measuring of performance of the algorithms is described in Section 4.1.3. Description of the experiments in detail is in Section 4.1.4. Section 4.2 describes and evaluates the obtained results.

4.1 Tools and methods

4.1.1 Parameters

Parameters used for the evaluation of the experiments are described in this section. Additionally, parameters, which were experimented with and changed during the evaluation, are stated. The default values of the parameters of external libraries or algorithms are not mentioned.

GLNS-CETSP algorithm

Table 4.1 shows the selected parameters of *GLNS-CETSP* that were chosen after initial experimentation with all possible values. These selected values of the parameters are the most suitable for the given configuration. *Num_circles* parameter is equal to the number of circles in the instance. *Min* is a function that finds a minimum value from two given values. The rest of the parameters is explained in Section 3.4 and Section 3.5.

	Modes		
Parameters	Fast	Medium	Slow
N_{iter}	$num\_circles$	$5 \cdot num\_circles$	$10 \cdot num\_circles$
N_{cold}	2	5	10
N_{warm}	2	4	6
$count_{first}$	$N_{iter}/4$	$N_{iter}/4$	$N_{iter}/4$
$count_{latest}$	$N_{iter}/4$	$N_{iter}/3$	$N_{iter}/3$
N_{start}	$N_{iter}/3$	-	-
N_{same}	5	-	-
e_{ref}	1×10^{-5}	1×10^{-5}	1×10^{-10}
$limit_{iter}$	1000	1000	10000
N_{move}	$min(20, 0.1 \cdot N_{iter})$	$min(100, 0.3 \cdot N_{iter})$	$0.4 \cdot N_{iter}$

Table 4.1: Parameters of GLNS-CETSP for three different modes. *Num_circles* value is the number of circles in a CETSP instance and *min* is function that selects the minimum from two given values.

GLNS-GTSP algorithm

Table 4.2 shows the selected parameters of *GLNS-GTSP* for the three modes of the algorithm.

	Modes		
Parameters	Fast	Medium	Slow
N_{iter}	$num\_circles$	$10{\cdot}num\_circles$	$20{\cdot}num\_circles$
N_{cold}	3	5	6
N_{warm}	3	3	5
$count_{first}$	$N_{iter}/6$	$N_{iter}/4$	$N_{iter}/6$
$count_{latest}$	$N_{iter}/4$	$N_{iter}/2$	$N_{iter}/3$
N_{move}	$min(20,\ 0.1{\cdot}N_{iter})$	$min(100,\ 0.3{\cdot}N_{iter})$	$0.4{\cdot}N_{iter}$

Table 4.2: Parameters of three modes of GLNS-GTSP algorithm. *Num_circles* value is the number of circles in GTSP instance and *min* is function that selects the minimum from two given values.

PCP solver

When using the *precomputed* solution of PCP solver, a three-dimensional table with precomputed values is used. The relationship between two of points and a circle are described by PCP parameters (a_y, s, ϕ) each representing one dimension in the table. Ranges of precomputed values are given by parameters of the discrete interval (lower bound l, upper bound u, and increment i). Table 4.3 shows the selected values for these pairs of parameters (interval and PCP parameters). Memory usage of the table is approximately 880MB. The specific values in the tables were assigned as follows:

- The parameters l and u of ϕ are fixed.
- The parameter l of a_y and s are fixed since in this case the radius of the circle is always equal to 1.
- A preliminary experiment was performed to estimate the upper bound u on a_y and s. In the first step of the experiment, a set of values $\{10, 20, 30, 40, 50\}$ was created. In the second step, the same value of u on a_y and s was assigned from the set. For each value the number of cases when CPP parameters were outside of the bounds set by l and u (and therefore not in the precomputed table) were counted. In the third step, the value 40 was selected as the best in terms of the ratio between the number of cases exceeding the selected bounds and the memory usage of the precomputed table. Almost 98% of the cases were in the

precomputed table. The rest of the cases (2%) were computed using the *simplified* version of solver.

- Parameter i of ϕ, a_y and s was estimated.

	PCP parameters		
Interval parameters	a_y	s	ϕ
lower bound l	1.0	1.0	$-\frac{\pi}{2}$
upper bound u	40.0	40.0	$\frac{\pi}{2}$
step (increment) i	0.1	0.1	$\frac{\pi}{720}$

Table 4.3: Interval parameters of the three-dimensional table used to precompute parameters of PCP solver.

4.1.2 CETSP instances

Two types of *CETSP* instances were used in combination with maps without obstacles. The first type contains a set of all 62 instances from *CETSP-lib* created by Mennell (2009) [1] [5]. These instances consist of 17 to 1001 intersected circles and can be divided into three categories:

1. Circles with equal-sized radii in the instance.
2. Circle with arbitrary-sized radii in the instance. These instances are labeled by *rdmRad* at the end of the instance name.
3. Instances containing circles with an overlap ratio R, which specify the extent to which the circles overlap in the given instance. Three sets of various overlap ratios were proposed: *very low* (R=0.02), *moderate* (R=0.10) and *very high* (R=0.30).

The second type contains a set of 32 instances created by the developed generator of *CETSP* instances. These instances consist of 50 to 400 *disjoint circles* and are generated on four maps with obstacles. The obstacles are later omitted. The instances are divided into two categories: instances of circles of equal-sized radii and instances of circles with arbitrary-sized radii labeled by *rdmRad.*

The measurements on the map with polygonal obstacles were carried out on a set of 64 *CETSP* instance produced by the generator of *CETSP* instances.

[1] Available at `https://drum.lib.umd.edu/handle/1903/9822`

Each set of 16 instances is generated on a different map. A set of instances on each map contains from 50 to 400 intersecting and disjoint circles with the equal- or arbitrary-sized radii.

4.1.3 Tools

Suppose that the *CETSP* instance is solved n_{repeat} times by an algorithm. The performance of the algorithm is compared based on the quality of the solutions and the measured computational times. The quality of the solutions is evaluated by two deviation equations *%PDB* and *%PDM*. These equations are adopted from Faigl (2018) [8]. *%PDB* (Eq.4.1) calculates the percentage deviation of the length of the shortest found tour from n_{repeat} solutions L_{min} from the length of the currently best found solution on the *CETSP* instance L_{opt}. *%PDM* (Eq.4.2) calculates the percentage deviation of the average length of the tour L_{avg} from L_{opt}.

$$\%PDB = \frac{L_{min} - L_{opt}}{L_{opt}} \cdot 100 \tag{4.1}$$

$$\%PDM = \frac{L_{avg} - L_{opt}}{L_{opt}} \cdot 100 \tag{4.2}$$

Computational time is computed as the median of all measured times for the n_{repeat} solutions. Almost all used algorithms were evaluated on a computer with processor Intel(R) Core(TM) i5-6300HQ CPU @ 2.30GHz and operation system Ubuntu 20.04.02 LTS. One exception is the heuristic *SZ2*, which was evaluated on a processor Intel Pentium E2220. Based on the CPU benchmark software [2] is Intel Pentium 2.0 times slower than Intel(R) Core(TM) i5-6300HQ in *single thread rating*. Therefore, the computational times of *SZ2* proposed in [5] were divided by the constant of 2.0.

4.1.4 Methods

This section describes the proposed experiments. All experiments were evaluated on *CETSP* instances. The performance of each algorithm was

[2] Available at `https://www.cpubenchmark.net/singleCompare.php`

measured by the quality of the solution defined by *%PDB* and *%PDM* values and the computation time. Due to the time consuming complexity of solving big instances, each instance containing up to 999 circles was solved 100 times while instances with 1000 and more circles were solved only 30 times.

The experiments were carried out in the following order. First, the performance of three different *GLNS-CETSP* modes (fast, medium, and slow) was compared on 13 instances containing up to 201 intersected circles with equal-sized and arbitrary-sized radii. The *GSOA* initialization heuristic, which was considered to be the best was used. Based on the results, the *fast* mode and the best algorithm configuration was selected and used from this point forward.

Second, four initialization heuristics (random_insertion, random, LKH, and GSOA) with *GLNS-CETSP* were compared on 47 instances containing intersected circles with equal-sized and arbitrary-sized radii and with three different overlap ratios. Based on the measured performance of *GLNS-CETSP* with each initialization heuristic, the *GSOA* was selected as the best initialization heuristic, and it is used in the rest of the experiments.

Third, two versions of PCP solver *simplified* and *precomputed* were tested on 32 instances containing disjoint circles and 58 instances containing intersected circles both in combination with equal-sized and arbitrary-sized radii and with three different overlap ratios. Moreover, the number of solved *PCP* problems per second for each version of *PCP* was counted. The *simplified* version of PCP solver computes the solution analytically from the implemented equations. In comparison, the *precomputed* version of PCP solver uses a table of precomputed values. The *simplified* version of PCP performed the best in terms of the number of solved problems per second and it was used in the rest of the experiments.

Fourth, the *GLNS-CETSP* algorithm with the previously selected configurations was compared with other algorithms: *SZ2* [5] based on the Steiner zones, *GSOA* [8] based on the unsupervised learning procedure and *GLNS*-based algorithm [24] (*GLNS-GTSP*) solving *GTSP* on *GTSP* instances. *GTSP* instance is created from the *CETSP* instance by approximation of each circle in the instance by 24 equally distanced points.

Next, a new algorithm called *GSOA+TCP* was proposed as a possible improved version of stand-alone *GSOA* algorithm. The algorithm is a combination of *GSOA* [8] and *TCP* algorithms, where *GSOA* finds the tour T and *TCP* shortens the length of T (improves T) on the given order of circles T^C if it is possible. *GSOA* requires the setting of several parameters.

All these parameters have default values as it was proposed in the original solution by Faigl (2018) [8]. *TCP* requires only two parameters (e_{ref} and $limit_{iter}$) that have the same values as in the *GLNS-CETSP* defined in table 4.1. The algorithm *GSOA+TCP* is compared with *GSOA* and *GLNS-CETSP* algorithms on all *CETSP* instances.

Finally, *GLNS-CETSP* were tested on 64 *CETSP* instances containing intersected and disjoint circles on four maps with polygonal obstacles (jari-huge, large, potholes and warehouse) generated by the *generator of CETSP instances*.

4.2 Results and Evaluation

This section describes the measured results. All results are in form of tables and placed in the Appendix B. The shortest computational time is highlighted for each instance in all tables that show computational times. Tables showing the quality of the found solutions contain L_{opt} value that refers to the best known solution for the given instance proposed by Mennell (2009) [5]. If any of the configurations of *GLNS-CETSP* algorithm found the new best solution or the solution is equal to currently best solution rounded to three decimal places, the L_{opt} value is modified and highlighted. Otherwise, L_{opt} value is not highlighted. The smallest *%PDB* and *%PDM* values for each instance are highlighted. The process of finding the smallest *%PDB* and *%PDM* values was carried out on values before they were rounded to two decimal places.

4.2.1 Modes of GLNS-CETSP

Tables B.1 and B.2 show the found solutions and computational times of different modes of *GLNS-CETSP* algorithm on instances containing only intersected circles with equal-sized and arbitrary-sized radii. The initialization heuristic of *GLNS-CETSP* is the *GSOA* algorithm.

GLNS-CETSP with mode *fast* is much faster than with other modes, as we can see in table B.1. As expected, the best quality of the solution is found by *GLNS-CETSP* with mode *slow* on all instances as shown in table B.2. This is because the *slow* mode prefers quality of solution over time taken to compute it.

Looking closely at *%PDB* and *%PDM* values in table B.2, we can see that these values are equal or almost the same in *slow* mode. This means that *slow* mode always finds the best solution or a solution very close to the best solution in each run of the algorithm. Comparing %PDB value of *fast* mode and *slow* mode, the %PDB values of *fast* mode are close to or almost the same as %PDB values found by *slow* mode.

Overall, these results indicate that *GLNS-CETSP* with mode *fast* should be considered the best in terms of the ratio of the quality of solution and computational requirements.

4.2.2 Initialization heuristics

The performances of four different initialization heuristics with *GLNS-CETSP* are shown in the following tables:

- Tables B.3, B.4 show the results for instances containing intersected circles with equal-sized radii.
- Tables B.5, B.6 show the results for instances containing intersected circles with arbitrary-sized radii.
- Tables B.7, B.8 show the results for instances containing intersected circles with three different overlap ratios.

The results shown in tables B.4, B.6 and B.8 revealed a significant difference between the quality of the solutions depending on initialization heuristic used. Initialization heuristic *GSOA* found the best initial tour out of the three initialization heuristics. This lead to identification of the solution with the best quality by *GLNS-CETSP* as we can see in the tables. Values %PDB and %PDM of *GLNS-CETSP* using *GSOA* are the lowest in nearly all solutions. Surprisingly, *GLNS-CETSP* using *GSOA* is not always the fastest solution as shown in tables B.3, B.5 and B.7. In more than half of the solutions *random_insertion* method used as initialization heuristic was the fastest, but the difference in speed is not so great. The possible explanation is that *GLNS-CETSP* using *random_insertion* was terminated earlier than the others with worse quality of initial tour. Example supporting this explanation is the instance *bubbles5*, where *GLNS-CETSP* using *random_insertion* found the worst quality of solution in the shortest time.

Overall, *GLNS-CETSP* using *GSOA* is considered to be the best algorithm and initialization heuristic combination despite the fact that it has not always found the best quality solution in the shortest possible time.

4.2.3 Simplified and precomputed versions of PCP

The results obtained for *GLNS-CETSP* using simplified *PCP* solver and *GLNS-CETSP* using precomputed version of *PCP* solver are shown in the following tables:

- Tables B.9, B.10 show the results for instances containing intersected circles with equal-sized radii.
- Tables B.11, B.12 show the results for instances containing intersected circles with arbitrary-sized radii.
- Tables B.13, B.14 show the results for instances containing disjoint circles with equal-sized radii.
- Tables B.15, B.16 show the results for instances containing disjoint circles with arbitrary-sized radii.

The expected results for the *simplified* and *precomputed* versions of PCP were that the *GLNS-CETSP* using *simplified* version of the solver would find the best quality results but in longer time period than *GLNS-CETSP* using *precomputed* version. Unexpectedly, the *GLNS-CETSP* using *simplified* version found the results faster and with the better quality of solutions than *GLNS-CETSP* using *precomputed* version on the majority of instances as shown in tables B.9, B.10, B.11, B.12, B.13, B.14, B.15, and B.16. There are two reasons why the *precomputed* version takes longer to find the solution than the *simplified* version:

1. The quality of solution of *precomputed* version is better than the *simplified* version at a cost of prolonged computational time.
2. Since, the *precomputed* version of PCP approximated points based on the values from precomputed table, it can lead to increased number of iterations of *GLNS-CETSP* algorithm.

The number of solved PCP problems per second for each version is the following:

- Simplified version of PCP solves approximately *22900* PCP problems per second.
- Precomputed version of PCP solves approximately *37420* PCP problems per second.

From the obtained results it is clear that the *precomputed* version of PCP solver is faster than the *simplified* version.

4.2.4 Comparison of GLNS-CETSP with other algorithms

The performance of *GLNS-CETSP* is compared with *SZ2*, *GSOA*, and *GLNS-GTSP* algorithms. The following tables show the results, where the *%PDM* value of *SZ2* heuristic is not known and therefore is not mentioned:

- Tables B.17, B.18 show the results for instances containing intersected circles with equal-sized radii.
- Tables B.19, B.20 show the results for instances containing intersected circles with arbitrary-sized radii. Computational time of *SZ2* heuristic is not known and therefore *SZ2* heuristic is not mentioned.
- Tables B.21, B.22 show the results for instances containing intersected circles with three different overlap ratios.
- Tables B.23, B.24 show the results for instances containing disjoint circles with equal-sized radii. *SZ2* heuristic has not been solved on these instances.
- Tables B.25, B.26 show the results for instances containing disjoint circles with arbitrary-sized radii. *SZ2* heuristic has not been solved on these instances.

Looking at the quality of solution, the *GLNS-CETSP* finds solution that is the new best or is equal to the currently best solution in 47 out of 62 instances consisting of intersecting circles, and 32 out of 32 instances consisting of disjoint circles, as the tables B.17, B.19, B.21, B.23, B.25 show. Instances on which the solutions have significantly improved compared to the solutions proposed by Mennell (2009) [5] are the following:

- bonus1000

- bonus1000rdmRad
- bubbles4
- bubbles6
- bubbles7
- bubbles8
- bubbles9
- team2_200rdmRad
- team6_500rdmRad
- dsj100rdmRad
- pcb442rdmRad
- pcb442 with overlap ratio R=0.10

Example of best found solution is on figures 4.1 and 4.2. On the other hand, the solved instance by *GLNS-CETSP* with the worst result is *dsj1000* with overlap ration R=0.30, where the value %PDB=7.17. The other algorithms solve the instance *dsj1000* with following %PDB values:

- *GSOA* with value %PDB=6.18
- *GLNS-GTSP* with value %PDB=228.57

Comparing the computational requirements, the *GSOA* is the fastest algorithm on all instances as we can see in the tables B.18, B.20, B.22, B.22, B.24 and B.26. Comparing *GLNS-CETSP* and *SZ2*, the *GLNS-CETSP* algorithm is faster for instances containing up to 200 intersected circles and for instances with low overlap ratio. The *GLNS-GTSP* was the slowest and worst performing algorithm of them all.

4.2.5 GSOA+TCP

The following tables show the comparison of the performance of *GSOA+TCP*with *GLNS-CETSP* and *GSOA* algorithms:

- Tables B.27, B.28 show the results for instances containing intersected circles with equal-sized radii.
- Tables B.29, B.30 show the results for instances containing intersected circles with arbitrary-sized radii.
- Tables B.31, B.32 show the results for instances containing intersected circles with three different overlap ratios.

As mentioned previously, *GSOA* algorithm found the solution in the shortest computational time on all instances. Since the stand-alone *TCP* algorithm is fast, the *GSOA+TCP* algorithm is only slightly slower than *GSOA*. In comparison, the computational time of *GLNS-CETSP* are much longer on instances with high number of circles (more than 300 circles), as we can see on tables B.28, B.30, and B.32.

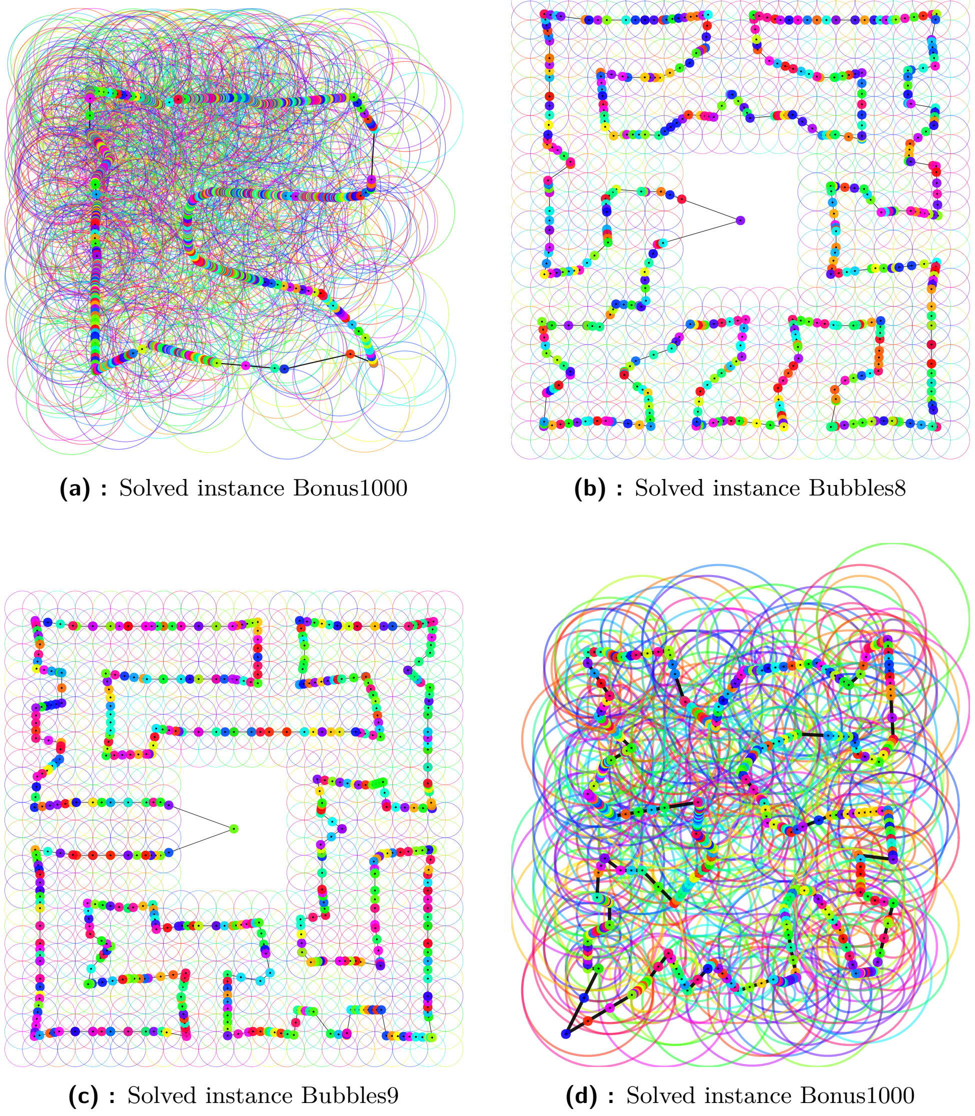
GLNS-CETSP algorithm finds the best quality solutions (lowest %PDB and %PDM values) on nearly all instances. *TCP* improves the quality of the solution, therefore *GSOA+TCP* finds better quality solutions than *GSOA*. However, the improvement is not enough to reach better quality solution than the one obtained using *GLNS-CETPS*, as shown in the tables B.27, B.29, and B.31.

There are some instances with solutions that were not expected. For example, the best quality solutions on instances *d493* and *dsj1000* with overlap ratio R=0.30 were found by *GSOA* algorithm and *GSOA+TCP* found significantly worse quality of solutions. The possible explanation is that due to the high overlap ratio, the *GSOA* found the order of circles on which *TCP* lengthened the tour. The instances are too complex even for *GLNS-CETSP* as it was not able to find the best quality solution.

4.2.6 Environment with polygonal obstacles

Table B.33 shows solved instances on the four maps with obstacles. These instances are solved only using the *GLNS-CETSP* solving obstacles. Therefore, *%PDB* value is not used.

On average, the best quality solutions with lowest %PDM values and the shortest computational times for environments with polygonal obstacles were found on the map *potholes* as the table B.33 shows. Comparing the instances with disjoint and intersected circles on all maps, the *GLNS-CETSP* algorithm found the best quality solutions (lowest %PDM) on instances consisting of intersected circles. On the other hand, the solution is found faster on instances consisting of disjoint circles.



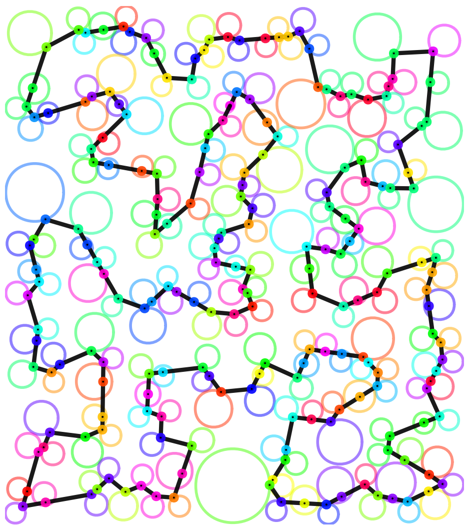
(a) : Solved instance Bonus1000

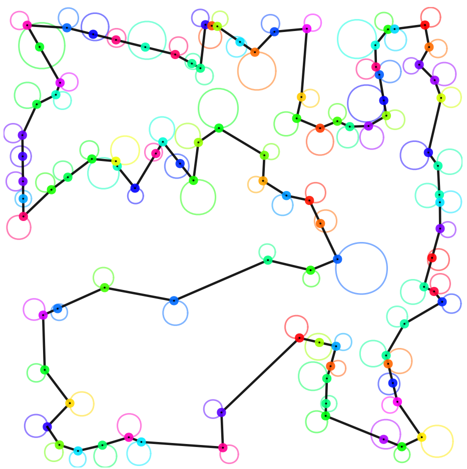
(b) : Solved instance Bubbles8

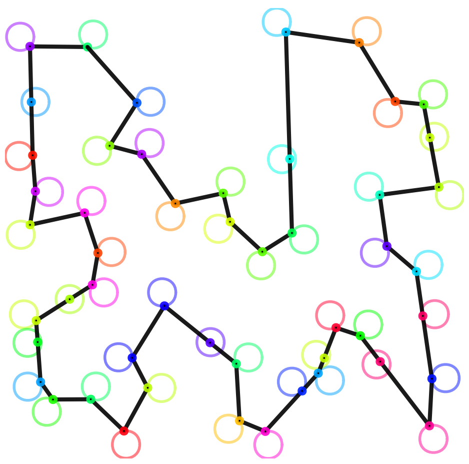
(c) : Solved instance Bubbles9

(d) : Solved instance Bonus1000

Figure 4.1: Example of best found solutions on instances with intersected circles


(a) : Solved instance Jari-huge200rdmRad


(b) : Solved instance Large100rdmRad


(c) : Solved instance Potholes50

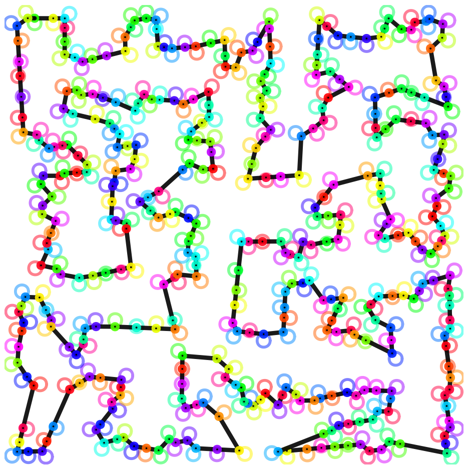

(d) : Solved instance Warehouse400

Figure 4.2: Example of best found solutions on instances with disjoint circles

Chapter 5

Discussion

The proposed *GLNS-CETSP* algorithm combines PCP, GLNS, and TCP algorithms to create an effective method to solve all types of CETSP problems ($CETSP_{dis}$, $CETSP_{int}$, $CETSP_{obst\_dis}$, and $CETSP_{obst\_int}$). The achieved results confirm that the final algorithm indeed is an effective method. Looking at the quality of the solutions, the majority of results for the $CETSP_{int}$ problem show solutions that are better than or equal to the currently best-known solution obtained by Mennell (2009) [5]. Moreover, the instances used in $CETSP_{dis}$, $CETSP_{obst\_dis}$, and $CETSP_{obst\_int}$ problems were generated and also solved by *GLNS-CETSP* for the future comparisons of results with other algorithms.

When comparing the computational times, it can be observed that the *GLNS-CETSP* algorithm is fast on instances containing up to 200 circles. In instances consisting of up to 150 intersected circles with low overlap ratio, the *GLNS-CETSP* algorithm is even faster than *SZ2* heuristic [5]. However, the *GLNS-CETSP* is slow for larger instances containing more than 200 circles which makes it suitable only in applications that are not limited by computation time or when the quality of the solution is preferred over the computational time.

A second new algorithm *GSOA+TCP* was also proposed, which is suitable for applications when the computational time is limited. The *GSOA+TCP* is a combination of *GSOA* and *TCP* algorithms, where *GSOA* finds the solution and *TCP* improve this solution with the fixed order of circles. Since stand-alone *TCP* and stand-alone *GSOA* are fast, *GSOA+TCP* is slightly slower than *GSOA*. Although the *GSOA+TCP* is slower than the *GSOA*, the significantly improved solution is worth the little extra time gained by the addition of *TCP* in the majority of cases. However, it is not suitable for applications that are not limited by time as there are algorithms such as the *GLNS-CETSP* that produce a higher quality of results.

For applications that find both time and quality important, it is advised to use *GLNS-CETSP* for environments with up to around 500 circles and *GSOA+TCP* for environments with a larger number of circles to balance the quality/time trade-off.

The experiments carried out were limited, especially in instances with more than 999 circles. It is noteworthy that the number of repetitions carried out for these instances was limited to only 30, compared to the number of repetitions carried out for smaller environments where the number of repetitions was 100. The limitation may have a negative effect on the %PDM value or the final computational time considered for these instances, as 30 repetitions and even 100 repetitions may not be enough to obtain adequate results. Some of the unexpected results obtained throughout the experiments may be attributed to the low number of repetitions.

Additionally, the results of *simplified* and *precomputed* versions of PCP solver should be studied more in the future. Looking at the speed of the PCP solvers individually and in combination with the *GLNS-CETSP*, there is a possibility of fine-tuning the *precomputed* version so that in combination with the *GLNS-CETSP* it fulfils its potential of being faster than the other version. Last but not least, further investigation of unexpected results observed in instances *d493* and *dsj1000* with overlap ratio R=0.30 is recommended. An explanation has already been proposed in the previous section, however, a solution to these situations has not yet been offered.

Chapter 6

Conclusion

In this thesis, the new heuristic method *GLNS-CETSP* was proposed to solve *CETSP* problem in polygonal domain on maps with and without obstacles. The heuristic method combines three algorithms: *PCP*, *GLNS* and *TCP*. *PCP* is a technique of finding the shortest path between two points and a circle. The main idea of *PCP* was inspired by light reflection theory and uses a set of derived equations to find the shortest path. These equations were inspired by the solution proposed by Chou (2008) [23]. However, equations from the original solution contain several transpiring errors that prevent us from using these equations. Two versions of *PCP* were proposed: *simplified* and *precomputed*. The *simplified* version finds the shortest path using the equations and the *precomputed* version finds the shortest path using the precomputed values in the table. *GLNS* is a solver of the *GTSP* problem which was modified to solve the *CETSP* problem. *TCP* improved the solution on the fixed order of circles and it was used as a tour optimization technique of the heuristic method.

Several *GLNS-CETSP* configurations were compared and the best configuration was selected. The configuration is given by the mode of *GLNS-CETSP* (fast, medium, slow), initialization heuristic (random_insertion, random, GSOA, and LKH) and two versions of *PCP*. Based on the results, the *GLNS-CETSP* containing *fast* mode, *GSOA* initialization heuristic and simplified version of *PCP* were considered as the best configuration. *GLNS-CETSP* using this configuration finds the best solutions based on the ratio of the quality of the solution and computational time.

Next, the new *CETSP* instances were generated on four maps (jari-huge, large, potholes and warehouse) by developed generator of *CETSP* instances. The *CETSP* instances contain from 50 to 400 intersected and disjoint circles with equal-sized and arbitrary-sized radii, and the number of generated instances is 64 in total. These instances can be used for CETSP problems with and without polygonal obstacles.

The results of *GLNS-CETSP* with the best configuration were compared with several state of the art algorithms such as *SZ2*, *GSOA*, and *GLNS-GTSP* on instances from *CETSP-lib* and the generated *CETSP* instances. The best-known solutions for instances in *CETSP-lib* were used from Mennell (2009) [5]. *GLNS-GTSP* is a *GLNS* solver of *GTSP* problem. In case of *GLNS-GTSP* algorithm, each *CETSP* instance is transformed into *GTSP* problem by approximation of each circle by 24 points. Comparing the quality of the solution, in 75% cases the *GLNS-CETSP* found on instances from *CETSP-lib* a new best solution or a solution that is equal to the currently best solution. *GSOA* found worse quality results in all cases. Comparing the computational time, *GSOA* was very fast even on large instances (instances containing more than 500 circles). *GLNS-CETSP* was faster than *SZ2* on instances containing up to 150 circles and with low overlap ratio. *GLNS-CETSP* is slow on large instances (computational time is approximately 220 second on instance contains 1000 circles) and is not suitable in applications where the time is limited and large instances are used. However, comparing *GLNS-CETSP* and *GLNS-GTSP*, *GLNS-CETSP* is four times faster even on large instances and finds better solutions.

Because of the time consumption of *GLNS-CETSP* on large instances, a second new algorithm *GSOA+TCP* was proposed. The *GSOA+TCP* is a combination of *GSOA* and *TCP* algorithms, where *GSOA* finds the solution and *TCP* improve this solution with the fixed order of circles. Since *TCP* is fast *GSOA+TCP* is only slightly slower than stand-alone *GSOA*. Comparing the quality of the solutions, *GSOA+TCP* finds better solutions than *GSOA*, but worse than *GLNS-CETSP*. Therefore, *GSOA+TCP* is an improved version of *GSOA* that is much faster than *GLNS-CETSP* and should be used especially in applications where the computational time is limited and where the instances contain more than 500 circles.

Finally, *GLNS-CETSP* was extended to solve the generated *CETSP* instances on maps with polygonal obstacles.

Appendix A

Bibliography

[1] O. Tekdas, V. Isler, J. H. Lim, and A. Terzis, "Using mobile robots to harvest data from sensor fields," *IEEE Wireless Communications*, vol. 16, no. 1, pp. 22–28, 2009. [Online]. Available: https://doi.org/10.1109/MWC.2009.4804365

[2] B. Yuan, M. Orlowska, and S. Sadiq, "On the optimal robot routing problem in wireless sensor networks," *IEEE transactions on knowledge and data engineering*, vol. 19, no. 9, pp. 1252–1261, 2007. [Online]. Available: https://doi.org/10.1109/TKDE.2007.1062

[3] S. Alatartsev, V. Mersheeva, M. Augustine, and F. Ortmeier, "On optimizing a sequence of robotic tasks," in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2013. [Online]. Available: https://cse.cs.ovgu.de/cse-wordpress/wp-content/uploads/2015/05/Alatartsev_IROS2013.pdf

[4] S. Dubowsky and T. Blubaugh, "Planning time-optimal robotic manipulator motions and work places for point-to-point tasks," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 3, pp. 377–381, 1989. [Online]. Available: https://doi.org/10.1109/70.34775

[5] M. William Kenneth, "Heuristics for solving three routing problems: Close-enough traveling salesman problem, close-enough vehicle routing problem, sequence-dependent team orienteering problem," Ph.D. dissertation, Digital Repository at the University of Maryland, 2009. [Online]. Available: http://hdl.handle.net/1903/9822

[6] W. P. Coutinho, R. Q. d. Nascimento, A. A. Pessoa, and A. Subramanian, "A branch-and-bound algorithm for the close-enough traveling salesman

problem," *INFORMS Journal on Computing*, vol. 28, no. 4, pp. 752–765, 2016. [Online]. Available: https://doi.org/10.1287/ijoc.2016.0711

[7] B. Behdani and J. C. Smith, "An integer-programming-based approach to the close-enough traveling salesman problem," *INFORMS Journal on Computing*, vol. 26, no. 3, pp. 415–432, 2014. [Online]. Available: https://doi.org/10.1287/ijoc.2013.0574

[8] 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. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231218306647

[9] M. Antonescu and C. Bîră, "Discrete gravitational search algorithm (dgsa) applied for the close-enough travelling salesman problem (tsp / cetsp)," in *2019 International Semiconductor Conference (CAS)*, 2019, pp. 123–126. [Online]. Available: http://doi.org/10.1109/SMICND.2019.8923719

[10] D. Sinha Roy, B. Golden, X. Wang, and E. Wasil, "Estimating the tour length for the close enough traveling salesman problem," *Algorithms*, vol. 14, no. 4, 2021. [Online]. Available: https://www.mdpi.com/1999-4893/14/4/123

[11] D. J. Gulczynski, J. W. Heath, and C. C. Price, *The Close Enough Traveling Salesman Problem: A Discussion of Several Heuristics*. Boston, MA: Springer US, 2006, pp. 271–283. [Online]. Available: https://doi.org/10.1007/978-0-387-39934-8_16

[12] J. Dong, N. Yang, and M. Chen, *Heuristic Approaches for a TSP Variant: The Automatic Meter Reading Shortest Tour Problem*. Boston, MA: Springer US, 2007, pp. 145–163. [Online]. Available: https://doi.org/10.1007/978-0-387-48793-9_10

[13] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Trans. Math. Softw.*, vol. 22, no. 4, p. 469–483, Dec. 1996. [Online]. Available: https://doi.org/10.1145/235815.235821

[14] K. Helsgaun, "An effective implementation of the lin–kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, no. 1, pp. 106–130, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221799002842

[15] M. Dror, A. Efrat, A. Lubiw, and J. S. B. Mitchell, "Touring a sequence of polygons," in *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 473–482. [Online]. Available: https://doi.org/10.1145/780542.780612

[16] C. Bliek1ú, P. Bonami, and A. Lodi, “Solving mixed-integer quadratic programming problems with ibm-cplex: a progress report,” in *Proceedings of the twenty-sixth RAMP symposium*, 2014, pp. 16–17.

[17] F. Carrabs, C. Cerrone, R. Cerulli, and M. Gaudioso, “A novel discretization scheme for the close enough traveling salesman problem,” *Computers & Operations Research*, vol. 78, pp. 163–171, 2017. [Online]. Available: https://doi.org/10.1016/j.cor.2016.09.003

[18] F. Carrabs, C. Cerrone, R. Cerulli, and C. D’Ambrosio, “Improved upper and lower bounds for the close enough traveling salesman problem,” in *Green, Pervasive, and Cloud Computing*, M. H. A. Au, A. Castiglione, K.-K. R. Choo, F. Palmieri, and K.-C. Li, Eds. Cham: Springer International Publishing, 2017, pp. 165–177. [Online]. Available: https://doi.org/10.1007/978-3-319-57186-7_14

[19] T. Kohonen, *Self-organizing maps.* Springer Science & Business Media, 2012, vol. 30.

[20] G. Reinelt, “Tsplib—a traveling salesman problem library,” *ORSA journal on computing*, vol. 3, no. 4, pp. 376–384, 1991. [Online]. Available: https://doi.org/10.1287/ijoc.3.4.376

[21] X. Wang, B. Golden, and E. Wasil, “A steiner zone variable neighborhood search heuristic for the close-enough traveling salesman problem,” *Computers & Operations Research*, vol. 101, pp. 200–219, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054818302132

[22] A. Bondy and U. Murty, *Graph Theory.* Springer, c2008.

[23] S.-Y. Chou, C.-C. Chou, and Y.-K. Chen, “A base function for generating contour traversal paths in stereolithography apparatus applications,” *Expert Syst. Appl.*, vol. 35, pp. 235–244, 07 2008.

[24] S. L. Smith and F. Imeson, “Glns: An effective large neighborhood search heuristic for the generalized traveling salesman problem,” *Computers & Operations Research*, vol. 87, pp. 1–19, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054817301223

[25] G. Gutin and D. Karapetyan, “A memetic algorithm for the generalized traveling salesman problem,” *Natural Computing*, vol. 9, no. 1, pp. 47–60, 2010. [Online]. Available: http://link.springer.com/10.1007/s11047-009-9111-6

[26] S. Ropke and D. Pisinger, “An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows,” *Transportation Science*, vol. 40, no. 4, pp. 455–472, 2006. [Online]. Available: http://pubsonline.informs.org/doi/abs/10.1287/trsc.1050.0135

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

[28] L. Paul Chew, “Constrained delaunay triangulations,” *Algorithmica*, vol. 4, no. 1-4, pp. 97–108, 1989. [Online]. Available: http://link.springer.com/10.1007/BF01553881

[29] “Fade2d documentation,” c2021. [Online]. Available: https://www.geom.at/fade2d/html/

[30] D. Woller, “Search for sources of gamma radiation,” Ph.D. dissertation, 2019. [Online]. Available: https://dspace.cvut.cz/handle/10467/83422

[31] J. Vidašič, “Travelling salesman problem with neighborhoods,” Ph.D. dissertation, 2020. [Online]. Available: http://hdl.handle.net/10467/88069

Appendix B

Tables of results

CETSP instances	N_c	Fast	Medium	Slow
		T[s]	T[s]	T[s]
bubbles1	37	**0.007**	0.162	0.712
bubbles2	77	**0.030**	1.620	6.191
bubbles3	127	**0.120**	5.525	27.482
chaoSingleDep	201	**0.331**	22.733	107.122
concentricCircles1	17	**0.002**	0.032	0.127
concentricCircles2	37	**0.007**	0.136	0.702
concentricCircles3	61	**0.018**	0.483	2.441
kroD100rdmRad	100	**0.051**	2.505	13.362
rotatingDiamonds1	21	**0.003**	0.054	0.248
rotatingDiamonds2	61	**0.016**	0.501	2.442
rotatingDiamonds3	181	**0.248**	28.396	86.377
team1_100	101	**0.069**	29.269	14.695
team1_100rdmRad	101	**0.056**	3.375	15.345

Table B.1: Comparison of computational times of three different modes of GLNS-CETSP. Initial heuristic is GSOA in all cases. The shortest time is bold for each measured CETSP instance.

CETSP instances	N_c	L_{opt}	Fast		Medium		Slow	
			%PDB	%PDM	%PDB	%PDM	%PDB	%PDM
bubbles1	37	**349.13**	0.00	0.01	0.00	0.00	**0.00**	**0.00**
bubbles2	77	**428.28**	0.00	0.12	0.00	0.00	**0.00**	**0.00**
bubbles3	127	**529.96**	0.00	0.13	0.00	0.00	**0.00**	**0.00**
chaoSingleDep	201	1039.61	0.00	1.09	0.00	0.00	**0.00**	**0.00**
concentricCircles1	17	**53.16**	0.00	0.04	0.00	0.00	**0.00**	**0.00**
concentricCircles2	37	**153.13**	0.00	1.02	0.00	0.16	**0.00**	**0.01**
concentricCircles3	61	**270.01**	0.12	0.99	0.00	0.30	**0.00**	**0.06**
kroD100rdmRad	100	**141.83**	0.05	2.31	0.02	0.21	**0.00**	**0.02**
rotatingDiamonds1	21	**32.39**	0.00	0.00	0.00	0.00	**0.00**	**0.00**
rotatingDiamonds2	61	**140.48**	0.00	0.19	0.00	0.00	**0.00**	**0.00**
rotatingDiamonds3	181	**380.88**	0.00	0.28	0.00	0.00	**0.00**	**0.00**
team1_100	101	**307.34**	0.00	0.12	0.00	0.01	**0.00**	**0.00**
team1_100rdmRad	101	**388.54**	0.00	1.95	0.00	0.00	**0.00**	**0.00**

Table B.2: Comparison of found solutions of three different modes of GLNS-CETSP. Initial heuristic is GSOA in all cases. The lowest PDB and PDM values are bold for each measured CETSP instance.

CETSP instances	N_c	Random insertion	Random	LKH	GSOA
		T[s]	T[s]	T[s]	T[s]
bubbles1	37	**0.005**	0.005	0.005	0.007
bubbles2	77	**0.025**	0.033	0.029	0.030
bubbles3	127	**0.103**	0.133	0.110	0.120
bubbles4	185	**0.310**	0.395	0.326	0.333
bubbles5	251	**0.786**	0.912	0.795	0.844
bubbles6	325	2.111	2.228	**1.802**	1.984
chaoSingleDep	201	0.342	0.409	1.769	**0.331**
concentricCircles1	17	0.001	**0.001**	0.002	0.002
concentricCircles2	37	0.004	**0.004**	0.010	0.007
concentricCircles3	61	**0.010**	0.012	0.053	0.018
concentricCircles4	105	**0.046**	0.051	0.089	0.072
concentricCircles5	149	**0.126**	0.134	0.215	0.154
rotatingDiamonds1	21	0.002	**0.002**	0.003	0.003
rotatingDiamonds2	61	0.015	**0.011**	0.033	0.016
rotatingDiamonds3	181	0.241	**0.236**	0.269	0.248
rotatingDiamonds4	321	**1.372**	1.715	1.738	1.502
team1_100	101	**0.052**	0.068	0.080	0.069
team2_200	201	**0.877**	1.242	1.283	0.882
team3_300	301	**1.498**	2.141	2.074	1.779
team4_400	401	**4.261**	5.149	4.793	4.798

Table B.3: Comparison of computation time of GLNS-CETSP algorithm for four initialize heuristics (random insertion, random, LKH and GSOA). Instances consisting of intersected circles with equal-sized radii.

CETSP instances	N_c	L_{opt}	Random insertion		Random		LKH		GSOA	
			%PDB	%PDM	%PDB	%PDM	%PDB	%PDM	%PDB	%PDM
bubbles1	37	**349.13**	0.00	0.03	0.00	0.05	0.00	0.07	**0.00**	**0.01**
bubbles2	77	**428.28**	0.00	0.15	0.00	0.37	0.00	0.23	**0.00**	**0.12**
bubbles3	127	**529.96**	0.00	0.44	0.00	0.44	0.00	0.40	**0.00**	**0.13**
bubbles4	185	**802.76**	0.23	2.36	0.77	2.53	0.56	2.49	**0.00**	**0.61**
bubbles5	251	**1040.54**	1.44	5.24	0.94	5.56	0.84	5.43	**0.00**	**3.85**
bubbles6	325	**1247.05**	**0.00**	9.57	1.83	9.51	0.81	9.62	2.80	**8.40**
chaoSingleDep	201	1039.61	0.00	1.47	0.00	1.52	0.00	1.11	**0.00**	**1.09**
concentricCircles1	17	**53.16**	0.00	0.08	0.00	0.10	0.00	0.14	**0.00**	**0.04**
concentricCircles2	37	**153.13**	0.02	2.47	0.02	2.54	0.40	2.62	**0.00**	**1.02**
concentricCircles3	61	**270.04**	**0.00**	2.34	0.20	1.97	0.31	2.68	0.10	**0.98**
concentricCircles4	105	**452.68**	1.29	4.18	0.99	3.99	1.26	4.52	**0.00**	**0.78**
concentricCircles5	149	**644.73**	0.05	3.03	0.22	3.32	0.76	3.18	**0.00**	**2.19**
rotatingDiamonds1	21	**32.39**	0.00	0.00	0.00	0.07	0.00	0.00	**0.00**	**0.00**
rotatingDiamonds2	61	**140.48**	0.00	0.81	0.00	0.98	0.00	0.84	**0.00**	**0.19**
rotatingDiamonds3	181	**380.88**	0.01	0.95	0.01	0.90	0.01	0.70	**0.00**	**0.28**
rotatingDiamonds4	321	770.66	0.02	4.43	0.02	5.18	0.03	**2.79**	**0.02**	2.85
team1_100	101	**307.34**	0.00	1.41	0.00	0.99	0.00	1.20	**0.00**	**0.12**
team2_200	201	**246.68**	0.01	0.28	0.02	0.57	0.01	0.36	**0.00**	**0.17**
team3_300	301	**462.35**	0.62	4.64	0.66	5.14	0.64	5.21	**0.00**	**4.56**
team4_400	401	**670.68**	2.78	5.25	2.41	6.53	2.52	5.61	**0.00**	**2.03**

Table B.4: Comparison of found solutions by GLNS-CETSP algorithm for four initialize heuristics (random insertion, random, LKH and GSOA). Instances consisting of intersected circles with equal-sized radii.

CETSP instances	N_c	Random insertion	Random	LKH	GSOA
		T[s]	T[s]	T[s]	T[s]
kroD100rdmRad	100	**0.044**	0.056	0.067	0.051
lin318rdmRad	318	2.816	2.973	3.069	**2.358**
pcb442rdmRad	442	8.629	11.605	8.926	**7.419**
rat195rdmRad	195	1.315	1.546	1.578	**1.301**
rd400rdmRad	400	**3.486**	5.820	3.785	4.875
team1_100rdmRad	101	**0.049**	0.068	0.078	0.056
team2_200rdmRad	201	**0.357**	0.418	0.502	0.419
team3_300rdmRad	301	**3.870**	5.281	4.811	4.124
team4_400rdmRad	401	**3.741**	5.427	4.229	4.666

Table B.5: Comparison of computation time of GLNS-CETSP algorithm for four initialize heuristics (random insertion, random, LKH and GSOA). Instances consisting of intersected circles with arbitrary radii.

CETSP instances	N_c	L_{opt}	Random insertion		Random		LKH		GSOA	
			%PDB	%PDM	%PDB	%PDM	%PDB	%PDM	%PDB	%PDM
kroD100rdmRad	100	141.84	0.49	2.83	0.35	2.91	0.31	2.97	**0.04**	**2.31**
lin318rdmRad	318	**2047.12**	**0.00**	5.92	1.60	6.67	1.58	6.89	1.52	**5.58**
pcb442rdmRad	442	**219.72**	0.83	4.77	1.48	5.15	1.13	5.24	**0.00**	**2.91**
rat195rdmRad	195	**68.22**	0.02	0.73	0.03	0.95	0.05	0.93	**0.00**	**0.08**
rd400rdmRad	400	1252.38	3.04	4.96	2.67	5.24	2.32	4.97	**1.15**	**2.93**
team1_100rdmRad	101	**388.54**	0.00	2.87	0.00	2.40	0.00	3.04	**0.00**	**1.95**
team2_200rdmRad	201	**614.90**	0.40	4.15	0.51	4.56	0.87	4.05	**0.00**	**1.78**
team3_300rdmRad	301	**378.09**	0.03	4.32	0.08	4.48	0.04	4.67	**0.00**	**2.58**
team4_400rdmRad	401	**1007.98**	2.20	4.28	2.18	4.53	1.09	4.18	**0.00**	**2.39**

Table B.6: Comparison of found solutions by GLNS-CETSP algorithm for four initialize heuristics (random insertion, random, LKH and GSOA). Instances consisting of intersected circles with arbitrary radii.

CETSP instances	N_c	Random insertion	Random	LKH	GSOA
		T[s]	T[s]	T[s]	T[s]
		Overlap ratio R = 0.02			
kroD100	100	**0.033**	0.043	0.062	0.052
lin318	318	**1.465**	2.098	1.710	1.856
pcb442	442	**4.770**	7.055	5.545	6.151
rat195	195	**0.263**	0.318	0.308	0.344
rd400	400	**3.733**	4.312	3.766	4.562
		Overlap ratio R = 0.10			
kroD100	100	**0.057**	0.084	0.093	0.064
lin318	318	2.304	3.027	2.689	**2.179**
pcb442	442	7.396	10.169	10.000	**7.251**
rat195	195	**0.563**	0.825	0.703	0.618
rd400	400	4.710	6.795	5.803	**4.656**
		Overlap ratio R = 0.30			
kroD100	100	0.136	0.210	0.222	**0.131**
lin318	318	7.943	11.503	12.474	**7.091**
pcb442	442	25.263	45.502	40.136	**23.329**
rat195	195	1.163	2.164	2.184	**1.056**
rd400	400	17.751	25.528	27.608	**17.674**

Table B.7: Comparison of computation time of GLNS-CETSP algorithm for four initialize heuristics (random insertion, random, LKH and GSOA). Instances are sorted by three different overlap ratios.

CETSP instances	N_c	L_{opt}	Random insertion		Random		LKH		GSOA	
			%PDB	%PDM	%PDB	%PDM	%PDB	%PDM	%PDB	%PDM
Overlap ratio R = 0.02										
kroD100	100	159.05	0.22	1.81	**0.03**	1.83	0.11	1.87	0.12	**1.15**
lin318	318	**2849.12**	1.14	3.72	**0.00**	4.41	0.12	4.05	0.94	**3.17**
pcb442	442	323.03	4.42	6.59	4.79	7.45	2.64	6.86	**2.54**	**4.48**
rat195	195	158.78	2.75	5.99	2.90	6.40	2.84	6.25	**1.14**	**3.05**
rd400	400	**1032.60**	1.37	4.95	2.31	5.73	2.30	5.19	**0.00**	**2.10**
Overlap ratio R = 0.10										
kroD100	100	**89.67**	0.00	0.48	0.00	0.64	0.01	0.64	**0.00**	**0.10**
lin318	318	**1395.61**	**0.00**	1.88	0.26	2.69	0.23	2.75	0.03	**1.59**
pcb442	442	**142.79**	0.82	4.08	1.68	5.50	0.69	5.46	**0.00**	**3.83**
rat195	195	**67.99**	0.03	0.48	0.10	0.68	0.03	0.50	**0.00**	**0.24**
rd400	400	**453.72**	**0.00**	3.58	0.60	4.91	0.78	4.89	0.31	**2.77**
Overlap ratio R = 0.30										
kroD100	100	**58.54**	0.03	0.32	0.04	0.27	0.07	0.32	**0.00**	**0.06**
lin318	318	765.96	0.03	0.41	0.03	0.42	0.04	0.45	**0.00**	**0.29**
pcb442	442	83.54	0.29	1.12	0.28	1.15	0.23	1.11	**0.05**	**0.67**
rat195	195	45.70	0.16	0.87	0.17	0.92	0.14	0.82	**0.01**	**0.19**
rd400	400	224.84	0.10	0.52	0.15	0.61	0.12	0.63	**0.03**	**0.34**

Table B.8: Comparison of found solutions by GLNS-CETSP algorithm for four initialize heuristics (random insertion, random, LKH and GSOA). Instances are sorted by the three different overlap ratios.

CETSP instances	N_c	L_{opt}	Simplified		Precomputed	
			%PDB	%PDM	%PDB	%PDM
bubbles1	37	**349.13**	**0.00**	0.01	0.00	**0.00**
bubbles2	77	**428.28**	0.00	0.12	**0.00**	**0.09**
bubbles3	127	**529.96**	**0.00**	**0.13**	0.00	0.14
bubbles4	185	**802.76**	**0.00**	0.61	0.07	**0.58**
bubbles5	251	**1040.54**	**0.00**	**3.85**	1.02	3.87
bubbles6	325	**1254.61**	2.18	**7.75**	**0.00**	7.76
bubbles7	407	**1622.99**	**0.00**	1.94	0.19	**1.92**
bubbles8	497	**1964.68**	0.28	**1.47**	**0.00**	1.56
bubbles9	595	**2283.41**	**0.00**	3.24	0.36	**3.10**
chaoSingleDep	201	1039.61	0.00	1.09	**0.00**	**1.08**
concentricCircles1	17	**53.16**	**0.00**	0.04	0.00	**0.03**
concentricCircles2	37	**153.13**	0.00	1.02	**0.00**	**0.90**
concentricCircles3	61	**270.32**	**0.00**	0.88	0.04	**0.85**
concentricCircles4	105	**452.68**	**0.00**	**0.78**	0.17	0.84
concentricCircles5	149	**644.73**	**0.00**	2.19	0.51	**2.13**
rotatingDiamonds1	21	**32.39**	0.00	0.00	**0.00**	**0.00**
rotatingDiamonds2	61	**140.48**	0.00	**0.19**	**0.00**	0.29
rotatingDiamonds3	181	**380.88**	**0.00**	0.28	0.00	**0.25**
rotatingDiamonds4	321	770.66	0.02	**2.85**	**0.00**	2.89
rotatingDiamonds5	681	1510.75	0.03	1.22	**0.01**	**0.99**
team1_100	101	**307.34**	**0.00**	**0.12**	0.00	0.14
team2_200	201	**246.68**	**0.00**	0.17	0.00	**0.14**
team3_300	301	**462.35**	**0.00**	**4.56**	0.28	4.74
team5_499	500	702.82	**0.86**	**2.68**	0.96	2.71
team6_500	501	225.22	**0.05**	**0.46**	0.61	0.61

Table B.9: Comparison of two types of PCP solver as a part of GLNS-CETSP on found results by GLNS-CETSP algorithm with GSOA initialization heuristic. The first one is without precomputation *Simplified* and the second one is with table of precomputed values *Precomputed.* Instances consisting of intersected circles with equal-sized radii.

CETSP instances	N_c	Simplified	Precomputed
		T[s]	T[s]
bubbles1	37	**0.007**	0.008
bubbles2	77	0.030	**0.027**
bubbles3	127	**0.120**	0.133
bubbles4	185	**0.333**	0.379
bubbles5	251	**0.844**	0.969
bubbles6	325	**1.984**	2.436
bubbles7	407	**4.471**	5.228
bubbles8	497	9.561	**9.307**
bubbles9	595	**19.957**	20.434
chaoSingleDep	201	**0.331**	0.368
concentricCircles1	17	**0.002**	0.002
concentricCircles2	37	**0.007**	0.008
concentricCircles3	61	**0.018**	0.021
concentricCircles4	105	**0.072**	0.082
concentricCircles5	149	0.154	**0.153**
rotatingDiamonds1	21	**0.003**	0.003
rotatingDiamonds2	61	**0.016**	0.022
rotatingDiamonds3	181	**0.248**	0.290
rotatingDiamonds4	321	**1.502**	1.798
rotatingDiamonds5	681	**24.159**	24.705
team1_100	101	0.069	**0.065**
team2_200	201	0.882	**0.866**
team3_300	301	1.779	**1.674**
team5_499	500	**8.664**	8.769
team6_500	501	39.053	**21.007**

Table B.10: Comparison of two types of PCP solver as a part of GLNS-CETSP on computational time of GLNS-CETSP. The first one is without precomputation *Simplified* and the second one is with table of precomputed values *Precomputed*. Instances consisting of intersected circles with equal-sized radii.

CETSP instances	N_c	L_{opt}	Simplified		Precomputed	
			%PDB	%PDM	%PDB	%PDM
d493rdmRad	493	**134.24**	0.01	1.39	**0.00**	**1.38**
kroD100rdmRad	100	141.84	**0.04**	2.31	0.11	**2.23**
lin318rdmRad	318	**2047.11**	1.52	5.58	**0.00**	**5.16**
pcb442rdmRad	442	**219.56**	0.07	2.98	**0.00**	**2.83**
rat195rdmRad	195	**68.22**	0.00	0.08	**0.00**	**0.06**
rd400rdmRad	400	1252.38	1.15	**2.93**	**0.62**	2.94
team1_100rdmRad	101	**388.54**	**0.00**	1.95	0.00	**1.73**
team2_200rdmRad	201	**614.26**	0.10	**1.89**	**0.00**	2.25
team3_300rdmRad	301	**378.09**	**0.00**	**2.58**	0.01	2.82
team5_499rdmRad	500	**446.19**	0.00	**1.42**	**0.00**	1.45
team6_500rdmRad	501	**624.01**	0.45	3.87	**0.00**	**3.34**

Table B.11: Comparison of two types of PCP solver as a part of GLNS-CETSP on found results by GLNS-CETSP algorithm with GSOA initialization heuristic. The first one is without precomputation *Simplified* and the second one is with table of precomputed values *Precomputed*. Instances consisting of intersected circles with arbitrary radii.

CETSP instances	N_c	Simplified	Precomputed
		T[s]	T[s]
d493rdmRad	493	26.458	**18.868**
kroD100rdmRad	100	**0.051**	0.054
lin318rdmRad	318	**2.358**	2.407
pcb442rdmRad	442	**7.419**	8.096
rat195rdmRad	195	1.301	**0.855**
rd400rdmRad	400	**4.875**	5.409
team1_100rdmRad	101	**0.056**	0.068
team2_200rdmRad	201	**0.419**	0.454
team3_300rdmRad	301	4.124	**2.833**
team5_499rdmRad	500	19.611	**12.159**
team6_500rdmRad	501	13.228	**11.633**

Table B.12: Comparison of two types of PCP solver as a part of GLNS-CETSP on computational time of GLNS-CETSP. The first one is without precomputation *Simplified* and the second one is with table of precomputed values *Precomputed*. Instances consisting of intersected circles with arbitrary radii.

CETSP instances	N_c	L_{opt}	Simplified		Precomputed	
			%PDB	%PDM	%PDB	%PDM
jari-huge100	100	**224.90**	**0.00**	1.99	0.21	**1.95**
jari-huge200	200	**304.86**	**0.00**	**1.98**	0.22	2.20
jari-huge400	400	**412.93**	0.21	**1.77**	**0.00**	1.83
jari-huge50	50	**156.31**	**0.00**	1.70	0.00	**1.66**
large100	100	**360.51**	0.43	2.60	**0.00**	**2.59**
large200	200	**497.02**	0.14	**2.67**	**0.00**	2.82
large400	400	**666.01**	**0.00**	3.20	1.02	**3.10**
large50	50	**272.98**	**0.00**	**0.45**	0.06	0.65
potholes100	100	**230.34**	**0.00**	3.00	0.03	**2.98**
potholes200	200	**280.70**	**0.00**	**3.04**	0.15	3.26
potholes400	400	**392.55**	**0.00**	**1.88**	0.28	1.89
potholes50	50	**169.68**	0.29	**1.61**	**0.00**	1.72
warehouse100	100	**199.56**	0.48	**2.30**	**0.00**	2.33
warehouse200	200	**266.75**	**0.00**	**2.35**	0.03	2.37
warehouse400	400	**359.91**	0.70	2.97	**0.00**	**2.73**
warehouse50	50	**146.01**	**0.00**	**1.29**	0.79	1.34

Table B.13: Comparison of two types of PCP solver as a part of GLNS-CETSP on found results by GLNS-CETSP algorithm with GSOA initialization heuristic. The first one is without precomputation *Simplified* and the second one is with table of precomputed values *Precomputed.* Instances consisting of disjoint circles with equal-sized radii.

CETSP instances	N_c	Simplified	Precomputed
		T[s]	T[s]
jari-huge100	100	0.053	**0.050**
jari-huge200	200	0.401	**0.376**
jari-huge400	400	**4.759**	5.062
jari-huge50	50	0.011	**0.011**
large100	100	**0.048**	0.051
large200	200	**0.373**	0.429
large400	400	**4.463**	5.557
large50	50	**0.009**	0.011
potholes100	100	**0.046**	0.051
potholes200	200	**0.368**	0.444
potholes400	400	**4.580**	5.106
potholes50	50	**0.008**	0.010
warehouse100	100	**0.049**	0.053
warehouse200	200	**0.398**	0.417
warehouse400	400	**4.430**	5.260
warehouse50	50	**0.009**	0.011

Table B.14: Comparison of two types of PCP solver as a part of GLNS-CETSP on computational time of GLNS-CETSP. The first one is without precomputation *Simplified* and the second one is with table of precomputed values *Precomputed*. Instances consisting of disjoint circles with equal-sized radii.

CETSP instances	N_c	L_{opt}	Simplified		Precomputed	
			%PDB	%PDM	%PDB	%PDM
jari-huge100rdmRad	100	**225.54**	0.24	**1.86**	**0.00**	1.95
jari-huge200rdmRad	200	**283.24**	**0.00**	1.86	0.34	**1.84**
jari-huge400rdmRad	400	**402.75**	0.05	1.78	**0.00**	**1.61**
jari-huge50rdmRad	50	**158.63**	**0.00**	3.30	0.00	**3.02**
large100rdmRad	100	**368.23**	**0.00**	**2.38**	0.07	2.50
large200rdmRad	200	**498.39**	**0.00**	**1.87**	0.29	2.03
large400rdmRad	400	**660.21**	**0.00**	2.83	0.17	**2.66**
large50rdmRad	50	**290.98**	0.00	**1.31**	**0.00**	1.42
potholes100rdmRad	100	**216.10**	**0.00**	**2.00**	0.64	2.02
potholes200rdmRad	200	**271.52**	**0.00**	**3.23**	1.05	3.58
potholes400rdmRad	400	**388.41**	0.26	2.11	**0.00**	**1.90**
potholes50rdmRad	50	**154.54**	**0.00**	**1.35**	0.00	1.40
warehouse100rdmRad	100	**196.55**	0.31	**2.96**	**0.00**	3.21
warehouse200rdmRad	200	**268.86**	0.19	**2.33**	**0.00**	2.44
warehouse400rdmRad	400	**355.87**	0.33	**2.86**	**0.00**	2.95
warehouse50rdmRad	50	**156.60**	**0.00**	**0.59**	0.03	0.69

Table B.15: Comparison of two types of PCP solver as a part of GLNS-CETSP on found results by GLNS-CETSP algorithm with GSOA initialization heuristic. The first one is without precomputation *Simplified* and the second one is with table of precomputed values *Precomputed.* Instances consisting of disjoint circles with arbitrary radii.

CETSP instances	N_c	Simplified	Precomputed
		T[s]	T[s]
jari-huge100rdmRad	100	**0.050**	0.052
jari-huge200rdmRad	200	0.380	**0.375**
jari-huge400rdmRad	400	**4.280**	5.121
jari-huge50rdmRad	50	**0.009**	0.010
large100rdmRad	100	**0.048**	0.052
large200rdmRad	200	**0.387**	0.461
large400rdmRad	400	**4.259**	5.203
large50rdmRad	50	**0.010**	0.011
potholes100rdmRad	100	**0.045**	0.055
potholes200rdmRad	200	**0.403**	0.449
potholes400rdmRad	400	**4.683**	5.191
potholes50rdmRad	50	**0.009**	0.010
warehouse100rdmRad	100	**0.046**	0.049
warehouse200rdmRad	200	**0.372**	0.436
warehouse400rdmRad	400	**4.270**	4.964
warehouse50rdmRad	50	**0.009**	0.010

Table B.16: Comparison of two types of PCP solver as a part of GLNS-CETSP on computational time of GLNS-CETSP. The first one is without precomputation *Simplified* and the second one is with table of precomputed values *Precomputed*. Instances consisting of disjoint circles with arbitrary radii.

CETSP instances	N_c	L_{opt}	SZ2	GLNS-CETSP		GLNS-GTSP		GSOA	
			%PDB	%PDB	%PDM	%PDB	%PDM	%PDB	%PDM
bonus1000	1001	**376.69**	6.85	**0.00**	**3.37**	56.51	59.82	8.07	12.37
bubbles1	37	**349.13**	0.00	**0.00**	**0.01**	0.22	0.44	0.19	0.75
bubbles2	77	**428.28**	0.00	**0.00**	**0.12**	0.77	0.90	0.92	1.81
bubbles3	127	**529.96**	0.15	**0.00**	**0.13**	0.30	0.37	0.23	0.80
bubbles4	185	**802.76**	3.38	**0.00**	**0.61**	1.92	2.54	1.47	2.69
bubbles5	251	**1040.54**	2.09	**0.00**	**3.85**	1.82	4.15	4.77	6.21
bubbles6	325	**1281.96**	7.89	**0.00**	**5.45**	0.35	7.17	5.95	7.76
bubbles7	407	**1622.99**	5.99	**0.00**	**1.94**	0.83	2.34	2.01	3.73
bubbles8	497	**1962.69**	7.07	**0.00**	**1.59**	0.05	1.69	1.40	3.64
bubbles9	595	**2283.41**	6.26	**0.00**	**3.21**	0.23	3.37	2.29	5.29
chaoSingleDep	201	1039.61	**0.00**	0.00	1.09	0.45	**0.94**	2.02	6.26
concentricCircles1	17	**53.16**	0.00	**0.00**	**0.04**	0.35	0.42	0.20	0.90
concentricCircles2	37	**153.13**	0.00	**0.00**	**1.02**	0.58	1.54	0.69	2.32
concentricCircles3	61	**270.32**	0.28	**0.00**	**0.88**	0.32	1.08	1.09	2.00
concentricCircles4	105	**452.68**	0.39	**0.00**	**0.78**	0.77	2.84	1.08	1.91
concentricCircles5	149	**644.73**	0.10	**0.00**	**2.19**	0.09	2.34	1.57	3.54
rotatingDiamonds1	21	**32.39**	0.00	**0.00**	**0.00**	0.76	1.18	0.46	1.18
rotatingDiamonds2	61	**140.48**	0.00	**0.00**	**0.19**	0.24	0.40	0.22	1.70
rotatingDiamonds3	181	**380.88**	0.00	**0.00**	**0.28**	0.26	0.33	0.80	2.84
rotatingDiamonds4	321	770.66	**0.00**	0.02	2.85	0.16	**2.64**	1.11	9.04
rotatingDiamonds5	681	1510.75	**0.00**	0.03	**1.22**	0.12	1.85	2.69	15.31
team1_100	101	**307.34**	0.00	**0.00**	**0.12**	2.24	2.63	1.20	2.77
team2_200	201	**246.68**	0.00	**0.00**	**0.17**	25.19	26.46	2.15	3.48
team3_300	301	**462.35**	0.84	**0.00**	**4.56**	4.54	8.79	6.85	10.02
team4_400	401	**670.68**	1.42	**0.00**	**2.03**	3.92	5.86	2.64	5.06
team5_499	500	702.82	**0.00**	0.86	**2.68**	1.30	2.93	2.97	6.34
team6_500	501	225.22	**0.00**	0.03	**0.46**	97.28	103.06	3.13	8.32

Table B.17: Comparison of four algorithms (SZ2, GLNS-CETSP, GLNS-GTSP, GSOA) on found best solution. Instances consisting of intersected circles with equal-sized radii. The best found solution for a given instance is highlighted. If the length of the solution is better of equal to the best known solution (L_{opt}), the length is highlighted in L_{opt} column.

CETSP instances	N_c	SZ2	GLNS-CETSP	GLNS-GTSP	GSOA
		T[s]	T[s]	T[s]	T[s]
bonus1000	1001	2.426	277.887	1075.338	**0.880**
bubbles1	37	0.043	0.007	0.023	**0.002**
bubbles2	77	1.329	0.030	0.169	**0.007**
bubbles3	127	0.535	0.120	0.758	**0.018**
bubbles4	185	0.387	0.333	2.186	**0.041**
bubbles5	251	2.062	0.844	5.796	**0.073**
bubbles6	325	0.680	1.984	14.302	**0.120**
bubbles7	407	1.117	4.471	30.433	**0.190**
bubbles8	497	1.469	9.904	58.793	**0.292**
bubbles9	595	2.864	19.415	117.254	**0.423**
chaoSingleDep	201	0.207	0.331	3.388	**0.049**
concentricCircles1	17	0.028	0.002	0.004	**0.001**
concentricCircles2	37	0.059	0.007	0.023	**0.003**
concentricCircles3	61	0.062	0.018	0.095	**0.007**
concentricCircles4	105	0.137	0.072	0.405	**0.019**
concentricCircles5	149	0.359	0.154	1.053	**0.035**
rotatingDiamonds1	21	0.024	0.003	0.006	**0.001**
rotatingDiamonds2	61	0.070	0.016	0.090	**0.006**
rotatingDiamonds3	181	0.250	0.248	2.259	**0.038**
rotatingDiamonds4	321	0.504	1.502	14.705	**0.121**
rotatingDiamonds5	681	1.344	24.159	205.019	**0.465**
team1_100	101	0.157	0.069	0.416	**0.012**
team2_200	201	0.317	0.882	2.955	**0.036**
team3_300	301	0.555	1.779	10.778	**0.089**
team4_400	401	0.715	4.798	25.611	**0.178**
team5_499	500	1.308	8.664	57.685	**0.288**
team6_500	501	0.543	34.950	61.013	**0.196**

Table B.18: Comparison of computational times of four algorithms (SZ2, GLNS-CETSP, GLNS-GTSP, GSOA). Instances consisting of intersected circles with equal-sized radii. The shortest computational time for a given instance is highlighted

CETSP instances	N_c	L_{opt}	GLNS-CETSP		GLNS-GTSP		GSOA	
			%PDB	%PDM	%PDB	%PDM	%PDB	%PDM
bonus1000rdmRad	1001	**942.88**	**0.00**	**2.73**	6.67	8.22	3.29	6.18
d493rdmRad	493	**134.25**	**0.00**	**1.38**	21.93	24.44	2.23	4.42
dsj1000rdmRad	1000	**630.94**	**0.00**	**2.82**	30.26	33.94	4.74	9.49
kroD100rdmRad	100	141.84	**0.04**	**2.31**	0.84	3.33	2.52	3.89
lin318rdmRad	318	**2078.28**	**0.00**	**4.00**	4.97	7.62	1.92	8.30
pcb442rdmRad	442	**219.72**	**0.00**	**2.91**	5.64	7.31	3.07	6.00
rat195rdmRad	195	**68.22**	**0.00**	**0.08**	94.23	98.49	0.79	2.23
rd400rdmRad	400	1252.38	**1.15**	**2.93**	1.87	3.37	2.44	4.00
team1_100rdmRad	101	**388.54**	**0.00**	**1.95**	1.26	2.45	3.65	4.62
team2_200rdmRad	201	**614.90**	**0.00**	**1.78**	1.29	2.59	1.30	3.50
team3_300rdmRad	301	**378.09**	**0.00**	**2.58**	24.89	27.18	3.61	8.83
team4_400rdmRad	401	**1007.98**	**0.00**	**2.39**	0.70	3.06	1.08	3.60
team5_499rdmRad	500	**446.21**	**0.00**	**1.42**	33.05	35.78	2.79	7.94
team6_500rdmRad	501	**626.29**	**0.00**	**3.16**	7.28	9.91	3.18	6.77

Table B.19: Comparison of three algorithms (GLNS-CETSP, GLNS-GTSP, GSOA) on found best solution. Instances consisting of intersected circles with arbitrary radii. The best found solution for a given instance is highlighted. If the length of the solution is better of equal to the best known solution (L_{opt}), the length is highlighted in L_{opt} column.

CETSP instances	N_c	GLNS-CETSP	GLNS-GTSP	GSOA
		T[s]	T[s]	T[s]
bonus1000rdmRad	1001	222.364	880.681	**1.090**
d493rdmRad	493	26.458	58.243	**0.205**
dsj1000rdmRad	1000	289.768	999.042	**0.937**
kroD100rdmRad	100	0.051	0.346	**0.014**
lin318rdmRad	318	2.358	12.129	**0.101**
pcb442rdmRad	442	7.419	38.461	**0.189**
rat195rdmRad	195	1.301	2.871	**0.024**
rd400rdmRad	400	4.875	18.537	**0.244**
team1_100rdmRad	101	0.056	0.368	**0.013**
team2_200rdmRad	201	0.419	2.753	**0.052**
team3_300rdmRad	301	4.124	10.784	**0.077**
team4_400rdmRad	401	4.666	20.619	**0.229**
team5_499rdmRad	500	19.611	63.823	**0.204**
team6_500rdmRad	501	14.939	60.647	**0.242**

Table B.20: Comparison of computational times of three algorithms (GLNS-CETSP, GLNS-GTSP, GSOA). Instances consisting of intersected circles with arbitrary radii. The shortest computational time for a given instance is highlighted.

CETSP instances	N_c	L_{opt}	SZ2	GLNS-CETSP		GLNS-GTSP		GSOA	
			%PDB	%PDB	%PDM	%PDB	%PDM	%PDB	%PDM
			Overlap ratio R = 0.02						
d493	493	**202.10**	0.34	**0.00**	**2.05**	1.10	2.95	1.49	4.25
dsj1000	1000	**918.58**	1.87	**0.00**	**1.47**	4.40	5.34	2.19	4.07
kroD100	100	159.05	**0.00**	0.12	**1.15**	0.22	1.32	0.98	2.58
lin318	318	2863.37	**0.00**	0.44	**2.65**	0.60	2.85	2.43	4.71
pcb442	442	323.03	**0.00**	2.54	**4.48**	2.81	4.57	4.05	5.87
rat195	195	158.78	**0.00**	1.14	**3.05**	1.01	3.57	2.06	4.28
rd400	400	**1032.60**	0.08	**0.00**	**2.10**	1.52	3.12	1.69	3.54
			Overlap ratio R = 0.10						
d493	493	**100.80**	0.92	**0.00**	**0.68**	30.08	31.69	2.04	6.11
dsj1000	1000	**373.94**	0.58	**0.00**	**0.54**	61.34	64.08	5.43	10.03
kroD100	100	**89.67**	0.00	**0.00**	**0.10**	3.28	3.95	1.14	2.52
lin318	318	**1396.07**	0.89	**0.00**	**1.56**	11.03	13.92	3.02	6.34
pcb442	442	**142.79**	3.12	**0.00**	**3.83**	13.09	16.98	5.70	9.43
rat195	195	**67.99**	0.14	**0.00**	**0.24**	7.48	8.40	1.69	3.56
rd400	400	**455.15**	2.41	**0.00**	**2.45**	9.93	12.24	3.53	6.61
			Overlap ratio R = 0.30						
d493	493	69.76	**0.00**	3.39	8.53	110.32	114.26	0.45	**2.36**
dsj1000	1000	199.95	**0.00**	7.17	14.30	228.57	234.48	6.18	**10.44**
kroD100	100	**58.54**	0.00	**0.00**	**0.06**	26.17	28.21	0.99	2.21
lin318	318	765.96	**0.00**	0.00	**0.29**	64.26	69.22	2.11	4.68
pcb442	442	83.54	**0.00**	0.05	**0.67**	90.40	95.86	2.62	6.58
rat195	195	45.70	**0.00**	0.01	**0.19**	57.38	59.45	1.18	3.41
rd400	400	224.84	**0.00**	0.03	**0.34**	87.16	90.97	3.03	6.18

Table B.21: Comparison of four algorithms (SZ2, GLNS-CETSP, GLNS-GTSP, GSOA) on found best solution. Instances contain only the intersected circles and are sorted by three different overlap ratios. The best found solution for a given instance is highlighted. If the length of the solution is better of equal to the best known solution (L_{opt}), the length is highlighted in L_{opt} column.

CETSP instances	N_c	SZ2	GLNS-CETSP	GLNS-GTSP	GSOA
		T[s]	T[s]	T[s]	T[s]
			Overlap ratio R = 0.02		
d493	493	1.262	9.369	53.819	**0.274**
dsj1000	1000	4.598	202.530	939.654	**1.209**
kroD100	100	0.148	0.052	0.335	**0.015**
lin318	318	0.739	1.856	10.598	**0.137**
pcb442	442	0.723	6.151	30.848	**0.273**
rat195	195	0.270	0.344	2.187	**0.054**
rd400	400	0.910	4.562	19.993	**0.229**
			Overlap ratio R = 0.10		
d493	493	0.891	21.383	65.101	**0.200**
dsj1000	1000	2.329	251.683	1099.317	**0.873**
kroD100	100	0.125	0.064	0.405	**0.011**
lin318	318	0.429	2.179	13.737	**0.090**
pcb442	442	0.797	7.251	39.007	**0.171**
rat195	195	0.227	0.618	2.888	**0.034**
rd400	400	0.399	4.656	29.454	**0.144**
			Overlap ratio R = 0.30		
d493	493	0.340	29.151	55.459	**0.097**
dsj1000	1000	1.301	660.775	837.807	**0.884**
kroD100	100	0.067	0.131	0.395	**0.007**
lin318	318	0.297	7.091	11.843	**0.073**
pcb442	442	0.359	23.329	34.416	**0.141**
rat195	195	0.168	1.056	2.635	**0.024**
rd400	400	0.433	17.674	24.212	**0.123**

Table B.22: Comparison of computational times of three algorithms (GLNS-CETSP, GLNS-GTSP, GSOA). Instances contain only the intersected circles and are sorted by three different overlap ratios. The shortest computational time for a given instance is highlighted.

CETSP instances	N_c	L_{opt}	GLNS-CETSP		GLNS-GTSP		GSOA	
			%PDB	%PDM	%PDB	%PDM	%PDB	%PDM
jari-huge50	50	**156.31**	**0.00**	**1.70**	0.22	1.80	2.57	4.62
jari-huge100	100	**224.89**	**0.00**	**2.02**	0.01	2.27	0.80	3.25
jari-huge200	200	**304.86**	**0.00**	**1.98**	0.75	2.26	1.37	3.35
jari-huge400	400	**413.81**	**0.00**	**1.55**	1.00	2.42	1.54	2.94
large50	50	**272.98**	**0.00**	**0.45**	0.36	1.18	1.10	2.36
large100	100	**362.06**	**0.00**	2.16	0.26	**1.90**	2.13	4.11
large200	200	**497.73**	**0.00**	**2.52**	0.15	2.57	1.40	3.79
large400	400	**666.01**	**0.00**	**3.20**	0.63	3.36	1.06	4.58
potholes50	50	**170.17**	**0.00**	**1.32**	0.47	1.60	1.72	3.63
potholes100	100	**230.34**	**0.00**	**3.00**	0.38	3.70	2.24	5.20
potholes200	200	**280.70**	**0.00**	3.04	0.38	**2.82**	1.81	4.70
potholes400	400	**392.55**	**0.00**	**1.88**	0.00	2.14	1.47	3.16
warehouse50	50	**146.01**	**0.00**	**1.29**	0.38	1.42	2.15	2.90
warehouse100	100	**200.53**	**0.00**	**1.81**	0.10	2.65	0.74	3.02
warehouse200	200	**266.75**	**0.00**	**2.35**	0.05	3.35	1.39	3.77
warehouse400	400	**362.42**	**0.00**	**2.26**	0.24	2.33	0.75	3.41

Table B.23: Comparison of three algorithms (GLNS-CETSP, GLNS-GTSP, GSOA) on found best solution. Instances contain only the disjointed circles with equal-sized radii. The best found solution for a given instance is highlighted. If the length of the solution is better of equal to the best known solution (L_{opt}), the length is highlighted in L_{opt} column.

CETSP instances	N_c	GLNS-CETSP	GLNS-GTSP	GSOA
		T[s]	T[s]	T[s]
jari-huge50	50	0.011	0.046	**0.004**
jari-huge100	100	0.053	0.300	**0.015**
jari-huge200	200	0.401	2.205	**0.055**
jari-huge400	400	4.759	18.879	**0.216**
large50	50	0.009	0.044	**0.004**
large100	100	0.048	0.309	**0.014**
large200	200	0.373	2.335	**0.055**
large400	400	4.463	19.671	**0.217**
potholes50	50	0.008	0.043	**0.004**
potholes100	100	0.046	0.296	**0.015**
potholes200	200	0.368	2.280	**0.055**
potholes400	400	4.580	19.509	**0.211**
warehouse50	50	0.009	0.044	**0.004**
warehouse100	100	0.049	0.305	**0.015**
warehouse200	200	0.398	2.344	**0.056**
warehouse400	400	4.430	19.710	**0.215**

Table B.24: Comparison of computational times of three algorithms (GLNS-CETSP, GLNS-GTSP, GSOA). Instances contain only the disjointed circles with equal-sized radii. The shortest computational time for a given instance is highlighted.

CETSP instances	N_c	L_{opt}	GLNS-CETSP		GLNS-GTSP		GSOA	
			%PDB	%PDM	%PDB	%PDM	%PDB	%PDM
jari-huge100rdmRad	100	**226.09**	**0.00**	**1.62**	0.32	1.65	1.26	3.23
jari-huge200rdmRad	200	**283.24**	**0.00**	**1.86**	0.04	1.96	1.57	3.15
jari-huge400rdmRad	400	**402.97**	**0.00**	**1.73**	0.72	2.42	0.77	2.60
jari-huge50rdmRad	50	**158.63**	**0.00**	**3.30**	0.17	3.71	0.53	4.75
large100rdmRad	100	**368.23**	**0.00**	2.38	0.19	**2.37**	1.78	4.54
large200rdmRad	200	**498.39**	**0.00**	**1.87**	0.66	2.34	1.36	3.19
large400rdmRad	400	**660.21**	**0.00**	**2.83**	1.19	3.00	1.29	3.60
large50rdmRad	50	**290.99**	**0.00**	1.31	0.16	**1.14**	0.66	3.62
potholes100rdmRad	100	**216.10**	**0.00**	**2.00**	0.23	3.14	1.17	3.33
potholes200rdmRad	200	**271.52**	**0.00**	**3.23**	0.66	3.97	1.21	4.77
potholes400rdmRad	400	**389.40**	0.00	**1.85**	0.57	2.35	**0.00**	2.58
potholes50rdmRad	50	**154.54**	**0.00**	**1.35**	0.23	1.47	1.78	4.37
warehouse100rdmRad	100	**197.17**	**0.00**	**2.64**	0.91	3.68	1.29	4.98
warehouse200rdmRad	200	**269.37**	**0.00**	**2.14**	0.31	2.40	0.67	3.24
warehouse400rdmRad	400	**357.04**	**0.00**	**2.53**	0.02	3.19	0.25	4.01
warehouse50rdmRad	50	**156.60**	**0.00**	**0.59**	0.29	1.20	0.80	2.15

Table B.25: Comparison of three algorithms (GLNS-CETSP, GLNS-GTSP, GSOA) on found best solution. Instances contain only the disjointed circles with arbitrary radii. The best found solution for a given instance is highlighted. If the length of the solution is better of equal to the best known solution (L_{opt}), the length is highlighted in L_{opt} column.

CETSP instances	N_c	GLNS-CETSP	GLNS-GTSP	GSOA
		T[s]	T[s]	T[s]
jari-huge100rdmRad	100	0.050	0.297	**0.015**
jari-huge200rdmRad	200	0.380	2.302	**0.054**
jari-huge400rdmRad	400	4.280	19.188	**0.215**
jari-huge50rdmRad	50	0.009	0.042	**0.004**
large100rdmRad	100	0.048	0.304	**0.014**
large200rdmRad	200	0.387	2.384	**0.054**
large400rdmRad	400	4.259	20.507	**0.217**
large50rdmRad	50	0.010	0.042	**0.004**
potholes100rdmRad	100	0.045	0.320	**0.015**
potholes200rdmRad	200	0.403	2.385	**0.054**
potholes400rdmRad	400	4.683	18.661	**0.214**
potholes50rdmRad	50	0.009	0.042	**0.004**
warehouse100rdmRad	100	0.046	0.320	**0.015**
warehouse200rdmRad	200	0.372	2.304	**0.055**
warehouse400rdmRad	400	4.270	21.185	**0.213**
warehouse50rdmRad	50	0.009	0.044	**0.004**

Table B.26: Comparison of computational times of three algorithms (GLNS-CETSP, GLNS-GTSP, GSOA). Instances contain only the disjointed circles with arbitrary radii. The shortest computational time for a given instance is highlighted.

CETSP instances	N_c	L_{opt}	GLNS-CETSP		GSOA		GSOA + TCP	
			%PDB	%PDM	%PDB	%PDM	%PDB	%PDM
bonus1000	1001	**376.69**	**0.00**	**3.37**	8.07	12.37	4.58	8.19
bubbles1	37	**349.13**	**0.00**	**0.01**	0.19	0.75	0.00	0.06
bubbles2	77	**428.28**	**0.00**	**0.12**	0.92	1.81	0.00	0.37
bubbles3	127	**529.96**	**0.00**	**0.13**	0.23	0.80	0.00	0.22
bubbles4	185	**802.76**	**0.00**	**0.61**	1.47	2.69	0.01	0.63
bubbles5	251	**1040.54**	**0.00**	**3.85**	4.77	6.21	3.13	3.99
bubbles6	325	**1281.96**	**0.00**	**5.45**	5.95	7.76	4.15	6.01
bubbles7	407	**1622.99**	**0.00**	**1.94**	2.01	3.73	0.43	2.02
bubbles8	497	**1962.69**	**0.00**	**1.59**	1.40	3.64	0.66	1.71
bubbles9	595	**2281.33**	0.09	3.31	2.39	5.39	**0.00**	**3.29**
chaoSingleDep	201	1039.61	**0.00**	**1.09**	2.02	6.26	0.68	4.71
concentricCircles1	17	**53.16**	**0.00**	**0.04**	0.20	0.90	0.00	0.20
concentricCircles2	37	**153.13**	**0.00**	1.02	0.69	2.32	0.00	**0.97**
concentricCircles3	61	**270.32**	**0.00**	**0.88**	1.09	2.00	0.16	1.12
concentricCircles4	105	**452.68**	**0.00**	0.78	1.08	1.91	0.04	**0.67**
concentricCircles5	149	**644.73**	**0.00**	**2.19**	1.57	3.54	0.14	2.40
rotatingDiamonds1	21	**32.39**	**0.00**	**0.00**	0.46	1.18	0.00	0.00
rotatingDiamonds2	61	**140.48**	**0.00**	**0.19**	0.22	1.70	0.00	1.21
rotatingDiamonds3	181	**380.88**	**0.00**	**0.28**	0.80	2.84	0.35	2.31
rotatingDiamonds4	321	770.66	**0.02**	**2.85**	1.11	9.04	1.12	8.83
rotatingDiamonds5	681	1510.75	**0.03**	**1.22**	2.69	15.31	2.48	14.41
team1_100	101	**307.34**	**0.00**	**0.12**	1.20	2.77	0.00	0.25
team2_200	201	**246.68**	**0.00**	**0.17**	2.15	3.48	0.01	0.22
team3_300	301	**462.35**	**0.00**	**4.56**	6.85	10.02	4.11	7.02
team4_400	401	**670.68**	**0.00**	**2.03**	2.64	5.06	0.52	2.40
team5_499	500	702.82	**0.86**	**2.68**	2.97	6.34	1.01	4.34
team6_500	501	225.22	**0.03**	**0.46**	3.13	8.32	0.97	4.60

Table B.27: Comparison of GLNS-CETSP, GSOA and GSOA+TCP algorithms based on the length of tour. Instances with intersecting circles with equal-sized radii.

CETSP instances	N_c	GLNS-CETSP	GSOA	GSOA + TCP
		T[s]	T[s]	T[s]
bonus1000	1001	277.887	**0.880**	1.000
bubbles1	37	0.007	**0.002**	0.003
bubbles2	77	0.030	**0.007**	0.008
bubbles3	127	0.120	**0.018**	0.021
bubbles4	185	0.333	**0.041**	0.046
bubbles5	251	0.844	**0.073**	0.081
bubbles6	325	1.984	**0.120**	0.131
bubbles7	407	4.471	**0.190**	0.216
bubbles8	497	9.904	**0.292**	0.333
bubbles9	595	19.415	**0.423**	0.456
chaoSingleDep	201	0.331	**0.049**	0.059
concentricCircles1	17	0.002	**0.001**	0.001
concentricCircles2	37	0.007	**0.003**	0.003
concentricCircles3	61	0.018	**0.007**	0.008
concentricCircles4	105	0.072	**0.019**	0.021
concentricCircles5	149	0.154	**0.035**	0.039
rotatingDiamonds1	21	0.003	**0.001**	0.001
rotatingDiamonds2	61	0.016	**0.006**	0.006
rotatingDiamonds3	181	0.248	**0.038**	0.042
rotatingDiamonds4	321	1.502	**0.121**	0.135
rotatingDiamonds5	681	24.159	**0.465**	0.557
team1_100	101	0.069	**0.012**	0.014
team2_200	201	0.882	**0.036**	0.044
team3_300	301	1.779	**0.089**	0.105
team4_400	401	4.798	**0.178**	0.203
team5_499	500	8.664	**0.288**	0.324
team6_500	501	34.950	**0.196**	0.210

Table B.28: Comparison of GLNS-CETSP, GSOA and GSOA+TCP algorithms based on the computational time. Instances with intersecting circles with equal-sized radii.

CETSP instances	N_c	L_{opt}	GLNS-CETSP		GSOA		GSOA + TCP	
			%PDB	%PDM	%PDB	%PDM	%PDB	%PDM
bonus1000rdmRad	1001	**942.88**	**0.00**	**2.73**	3.29	6.18	0.37	2.88
d493rdmRad	493	**134.25**	**0.00**	**1.38**	2.23	4.42	0.50	2.62
dsj1000rdmRad	1000	**630.94**	**0.00**	**2.82**	4.74	9.49	1.31	5.75
kroD100rdmRad	100	141.84	**0.04**	**2.31**	2.52	3.89	0.92	2.74
lin318rdmRad	318	**2078.28**	**0.00**	**4.00**	1.92	8.30	0.04	5.06
pcb442rdmRad	442	**219.72**	**0.00**	**2.91**	3.07	6.00	0.54	3.07
rat195rdmRad	195	**68.22**	**0.00**	**0.08**	0.79	2.23	0.05	2.43
rd400rdmRad	400	1252.38	**1.15**	**2.93**	2.44	4.00	1.31	3.22
team1_100rdmRad	101	**388.54**	**0.00**	**1.95**	3.65	4.62	2.18	2.79
team2_200rdmRad	201	**614.90**	**0.00**	**1.78**	1.30	3.50	0.10	2.26
team3_300rdmRad	301	**378.09**	**0.00**	**2.58**	3.61	8.83	0.03	4.40
team4_400rdmRad	401	**1007.98**	**0.00**	2.39	1.08	3.60	0.15	**2.37**
team5_499rdmRad	500	**446.21**	**0.00**	**1.42**	2.79	7.94	0.25	4.15
team6_500rdmRad	501	**626.29**	**0.00**	3.16	3.18	6.77	0.11	**2.96**

Table B.29: Comparison of GLNS-CETSP, GSOA and GSOA+TCP algorithms based on the length of tour. Instances with intersecting circles with arbitrary radii.

CETSP instances	N_c	GLNS-CETSP	GSOA	GSOA + TCP
		T[s]	T[s]	T[s]
bonus1000rdmRad	1001	222.364	**1.090**	1.112
d493rdmRad	493	26.458	**0.205**	0.260
dsj1000rdmRad	1000	289.768	**0.937**	1.111
kroD100rdmRad	100	0.051	**0.014**	0.015
lin318rdmRad	318	2.358	**0.101**	0.138
pcb442rdmRad	442	7.419	**0.189**	0.218
rat195rdmRad	195	1.301	**0.024**	0.036
rd400rdmRad	400	4.875	**0.244**	0.278
team1_100rdmRad	101	0.056	**0.013**	0.015
team2_200rdmRad	201	0.419	**0.052**	0.057
team3_300rdmRad	301	4.124	**0.077**	0.102
team4_400rdmRad	401	4.666	**0.229**	0.236
team5_499rdmRad	500	19.611	**0.204**	0.238
team6_500rdmRad	501	14.939	**0.242**	0.303

Table B.30: Comparison of GLNS-CETSP, GSOA and GSOA+TCP algorithms based on the computational time. Instances with intersecting circles with arbitrary radii.

CETSP instances	N_c	L_{opt}	GLNS-CETSP		GSOA		GSOA + TCP	
			%PDB	%PDM	%PDB	%PDM	%PDB	%PDM
Overlap ratio R = 0.02								
d493	493	**201.82**	0.14	**2.19**	1.63	4.40	**0.00**	2.43
dsj1000	1000	**918.58**	**0.00**	**1.47**	2.19	4.07	0.51	1.76
kroD100	100	159.05	**0.12**	**1.15**	0.98	2.58	0.21	1.74
lin318	318	2863.37	**0.44**	**2.65**	2.43	4.71	0.56	3.09
pcb442	442	323.03	**2.54**	**4.48**	4.05	5.87	2.73	4.62
rat195	195	158.78	1.14	3.05	2.06	4.28	**0.89**	**2.92**
rd400	400	**1032.60**	**0.00**	**2.10**	1.69	3.54	0.23	2.23
Overlap ratio R = 0.10								
d493	493	**100.80**	**0.00**	**0.68**	2.04	6.11	0.74	3.83
dsj1000	1000	**373.94**	**0.00**	**0.54**	5.43	10.03	1.31	5.54
kroD100	100	**89.67**	**0.00**	**0.10**	1.14	2.52	0.00	0.20
lin318	318	**1396.07**	**0.00**	**1.56**	3.02	6.34	0.48	2.15
pcb442	442	**142.79**	**0.00**	**3.83**	5.70	9.43	1.31	5.26
rat195	195	**67.99**	**0.00**	**0.24**	1.69	3.56	0.00	0.48
rd400	400	**455.15**	**0.00**	**2.45**	3.53	6.61	0.14	2.72
Overlap ratio R = 0.30								
d493	493	69.76	3.39	8.53	**0.45**	**2.36**	5.88	13.73
dsj1000	1000	199.95	7.17	14.30	**6.18**	**10.44**	22.99	35.78
kroD100	100	**58.54**	**0.00**	**0.06**	0.99	2.21	0.01	0.17
lin318	318	765.96	**0.00**	**0.29**	2.11	4.68	0.40	1.81
pcb442	442	83.54	**0.05**	**0.67**	2.62	6.58	1.97	4.93
rat195	195	45.70	**0.01**	**0.19**	1.18	3.41	0.04	0.73
rd400	400	224.84	**0.03**	**0.34**	3.03	6.18	0.51	2.63

Table B.31: Comparison of GLNS-CETSP, GSOA and GSOA+TCP algorithms based on the computational time. Instances with intersecting circles and sorted based on the overlap ratios.

CETSP instances	N_c	GLNS-CETSP	GSOA	GSOA + TCP
		T[s]	T[s]	T[s]
		Overlap ratio R = 0.02		
d493	493	9.369	**0.274**	0.305
dsj1000	1000	202.530	**1.209**	1.308
kroD100	100	0.052	**0.015**	0.017
lin318	318	1.856	**0.137**	0.152
pcb442	442	6.151	**0.273**	0.308
rat195	195	0.344	**0.054**	0.060
rd400	400	4.562	**0.229**	0.254
		Overlap ratio R = 0.10		
d493	493	21.383	**0.200**	0.236
dsj1000	1000	251.683	**0.873**	0.968
kroD100	100	0.064	**0.011**	0.012
lin318	318	2.179	**0.090**	0.107
pcb442	442	7.251	**0.171**	0.206
rat195	195	0.618	**0.034**	0.040
rd400	400	4.656	**0.144**	0.160
		Overlap ratio R = 0.30		
d493	493	29.151	**0.097**	0.118
dsj1000	1000	660.775	**0.836**	0.893
kroD100	100	0.131	**0.007**	0.010
lin318	318	7.091	**0.073**	0.092
pcb442	442	23.329	**0.141**	0.172
rat195	195	1.056	**0.024**	0.032
rd400	400	17.674	**0.123**	0.144

Table B.32: Comparison of GLNS-CETSP, GSOA and GSOA+TCP algorithms based on the computational time. Instances with intersecting circles and sorted based on the overlap ratios.

CETSP instances	jari-huge			large			potholes			warehouse		
	L_{opt}	%PDM	T[s]	L_{opt}	%PDM	T[s]	L_{opt}	%PDM	T[s]	L_{opt}	%PDM	T[s]
disjoint50	170.48	5.73	0.02	282.80	3.91	0.02	171.72	1.49	0.01	163.90	2.38	0.03
disjoint50rdmRad	206.29	12.03	0.02	306.07	2.84	0.03	156.30	1.46	0.02	169.84	3.54	0.02
disjoint100	289.09	13.28	0.08	374.70	6.12	0.08	231.89	3.74	0.07	238.05	6.38	0.09
disjoint100rdmRad	297.70	13.76	0.09	387.59	4.69	0.09	217.21	1.81	0.07	237.82	11.66	0.09
disjoint200	386.29	11.17	0.62	519.25	5.90	0.75	282.09	2.86	0.50	318.34	6.72	0.61
disjoint200rdmRad	367.92	11.12	0.54	519.16	6.05	0.57	272.44	3.75	0.51	312.55	7.98	0.78
disjoint400	525.88	8.48	5.28	682.19	5.94	5.61	392.38	2.02	5.00	404.17	8.19	5.52
disjoint400rdmRad	595.12	10.76	5.46	687.08	3.71	8.40	385.03	3.14	5.30	400.63	7.03	5.83
intersect50	122.52	3.06	0.03	258.02	1.66	0.04	128.57	0.72	0.02	138.95	1.74	0.03
intersect50rdmRad	144.81	8.90	0.03	244.08	2.93	0.04	152.57	0.19	0.02	159.90	7.00	0.03
intersect100	212.57	13.05	0.13	284.02	5.39	0.14	183.86	1.45	0.09	177.61	4.16	0.15
intersect100rdmRad	209.98	19.16	0.12	307.15	1.99	0.13	190.69	1.11	0.09	185.81	8.78	0.12
intersect200	243.91	16.11	0.74	371.62	3.76	0.89	202.11	3.53	0.68	235.67	4.38	0.71
intersect200rdmRad	298.73	12.23	0.80	392.13	4.16	0.84	230.96	2.77	0.59	264.31	5.92	0.66
intersect400	363.49	9.00	8.62	489.78	5.76	6.29	291.86	2.71	5.35	304.48	6.77	6.71
intersect400rdmRad	393.65	10.03	6.76	578.09	3.05	6.52	308.74	2.48	5.33	310.57	6.45	6.48

Table B.33: Results computed by GLNS-CETSP with simplified PCP solver and GSOA initialization heuristic on four different maps with obstacles: jari-huge, large, potholes, warehouse. Solutions on each map contain the best found tour L_{opt}, its $\%PDM$ value and execution time T in seconds. Instances contain from 50 to 400 disjointed and intersected circles with arbitrary and equal-sized radii.

Appendix C

Contents of the attached CD

```
Code ........................................ All code implementations
 ├── tables .. Continuous tables of results for given version of algorithm
 ├── report ........................... Continuous report of the theses
 ├── Matlab .........Helping matlab code to solve the complex equation
 └── CETSP ..................... Final implementation of all algorithms
├── Results ...................................... All measured results
└── The_Close_Enough_Travelling_Salesman_Problem_in_polygonal_domain.pdf
    Diploma thesis in pdf format
```