Reidar Berge

Path planning for a telescopic boom lift robot

Master's thesis in Cybernetics and Robotics
Supervisor: Lars Struen Imsland
Co-supervisor: Jørn Sandvik Nilsson
June 2022

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Reidar Berge

Path planning for a telescopic boom lift robot

Master's thesis in Cybernetics and Robotics
Supervisor: Lars Struen Imsland
Co-supervisor: Jørn Sandvik Nilsson
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Path planning for a telescopic boom lift robot

Reidar Berge

Supervisor: Lars Struen Imsland
Co-supervisor: Jørn Sandvik Nilsson

Master's thesis in Cybernetics and Robotics
Norwegian University of Science and Technology
May 2022

Abstract

This thesis present five different algorithms for path planning for a telescopic boom lift robot. The boom lift is cleaning a 2-dimensional surface with a smaller robot arm mounted to the lift basket. The smaller robot arm has a working area of circle in 2D, called a washing circle. The main task for this thesis is to find the shortest possible path between these washing circles. The circles has to be visited in a specific order, from top to bottom. The evaluation metrics for the path are computation time and euclidean length of the path. The algorithms tested are genetic algorithm(GA), nearest neighbour(NN) with 2-opt and solving TSP as ILP. The two last algorithms are modifications of NN and TSP ILP to suit the specified problem in this thesis better. The improved version of TSP ILP has a pre-procession of data, to reduce variables in the optimization problem. The algorithms are tested on two different example jobs. The best performing algorithms on these tests are the modified NN and pre-processed optimization.

Sammendrag

Denne oppgaven presenterer fem ulike algoritmer for en teleskopisk bom lift. Liften vasker en to dimensjonal overflate med en mindre robot arm montert på liften. Robotarmen har et arbeidsområde på en sirkel. Hovedproblemet i denne oppgaven er å finne korteste vei mellom disse sirklene. Sirklene må bli besøkt i en bestemt rekkefølge, fra bunnen til toppen. Vurderingskriteriene for algoritmene er beregningstid og distanse på løsningen. Algoritmene som er testet er genetisk algoritme(GA), nærmeste nabo(NN) med 2-opt og løse det som et optimaliseringsproblem(ILP). Dei to siste algoritmene er forbedring av NN og ILP, for å fungere bedre med problemet i denne oppgaven. Den forbedrede ILP reduserer antall variabler. Algoritmene er testet på to forskjellige problemer. Dei best fungerende algoritmene er forbedrede versjoner av NN og ILP.

Preface

This thesis is written as a master thesis at the Cybernetics and Robotics study program at Norwegian University of Science and Technology (NTNU), written in collaboration with nLink. The work is continuation of of my project thesis. The work presented in this thesis is performed from januar to early june 2022, by me independently. I want to thank my supervisors Professor Lars Struen Imsland and Jørn Sandvik Nilsson, from nLink.

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Motivation

Today a lot of work is done manually at construction cites. nLink[1] is a company that work with automating construction task. Earlier nLink has developed a robot for drilling holes in ceiling in corporation with Hilti, the Hilti Jaibot[2]. nLink is now developing a robot designed to perform work on building surfaces. The first application is cleaning building surfaces with pressure washers. This thesis is on the project, with focus on path planning of cleaning building surface efficient. This section is based on [3].

1.2 Overview

This master thesis is a continuation of the project report [3] by Berge. The report present algorithms for generating stops for the robot. The goal for this thesis is find the shortest possible path between these stops. The stops has to be visited in a specific order, from bottom to top.

In Section 1.3 are the robot presented in more detail, and further description of the problem is in Section 1.4. Relevant background theory is presented in Chapter 2. In Chapter 3 are software used presented. The implementation of algorithms is presented in Chapter 4. Chapter 5 shows the result from testing the algorithms. The discussion and conclusion of the report are in Chapter 6 and Chapter 7.

1.3 The robot

This section is strongly based on [3]. The robot used is a boom lift with a smaller high precision industrial robot mounted on the basket. The industrial robot is produced by Universal Robots[4]. Figure 1.1 is an illustration of the robot. The boom lift has very low precision and no precise position system integrated. Only fault safe functions, to avoid rollover and keep the man basket horizontal.

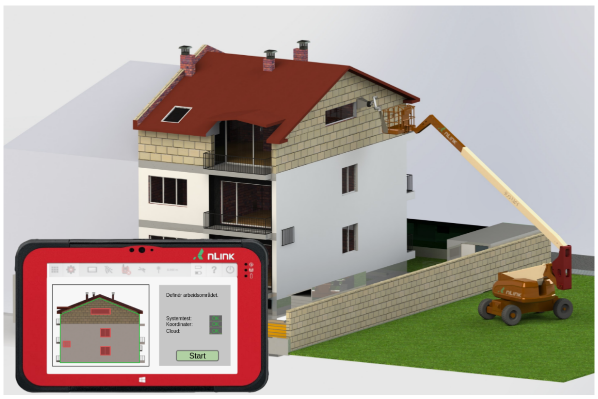


Figure 1.1: nLink telescopic boom lift robot.

1.3.1 Position system

For accurate position of the lift a Hilti PLT 300 is used[5]. It is a laser distance measurement tool. The building wall is measured with the laser tool, and estimated as a 2D-polygon. The robot position is estimated using the tool, with Prism from Hilti mounted on the lift[6]. Prism is a reflector that can be detected by the laser tool. Which can be used to determine accurate position of the robot.

1.3.2 Working principle

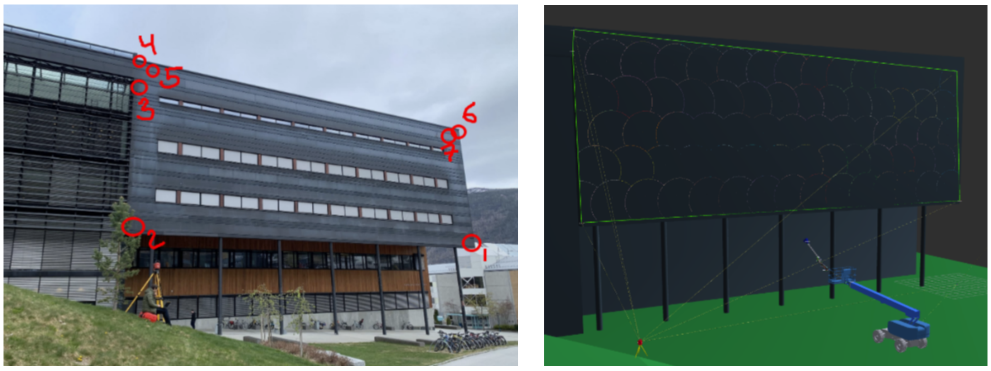
The working principle of the robot is to keep the boom lift stationary and let the industrial robot perform work within reach. Then move the boom lift to next point, where the industrial robot perform a new work. The working area of the industrial robot arm is a sphere with origin in the arm's first joint. The intersection between the surface and sphere is a circle. Hereafter called washing circle. It is assumed the circle has constant radius. Hence the distance from robots first joint to surface must be constant. Figure 1.3 illustrates a building surface with washing circles marked.

1.4 Problem Specification

This section first present the workspace of the robot. Then naming for variable and areas are described and illustrated. Last is the evaluation metric for algorithms.

1.4.1 Workspace

The workspace is the area to wash, it contain vertices of a 2D-polygon. The workspace can include no-go areas, like windows or balconies. These are also represented as 2D-polygons. The workspace is called washing surface.



(a) Building facade is measured with Hilti Plt 300 and estimated to polygon.

(b) 3D-model of robot and surface to wash

Figure 1.2: Example building, with building surface and washing circles simulated by nLink.

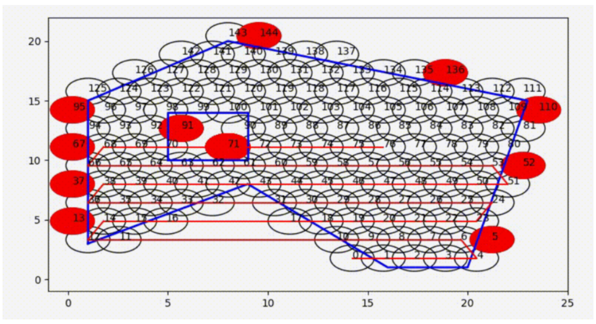


Figure 1.3: Example washing surface with washing circles, provided by nLink.

1.5 Motion planning

The evaluation metrics is to find shortest possible path for the robot.

1.5.1 Generating washing circles

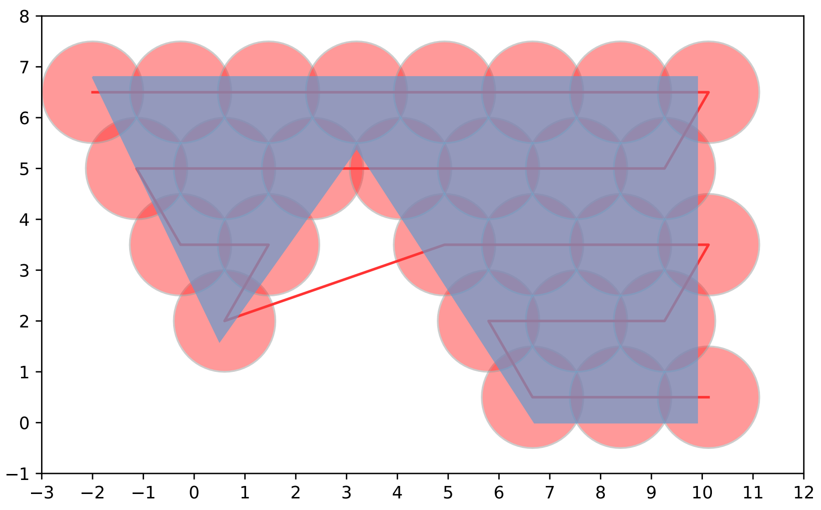
In [3] Berge present different algorithms for generating washings circles. The algorithms cover a washing surface with circles in an optimum way. According to this report, the best performing algorithm is covering the washing surface with circles placed in a hexagon pattern. I this thesis this algorithm will be used to generate washing circles. Due to it simplicity and fast computing time.

1.5.2 Generating path visiting washing circles

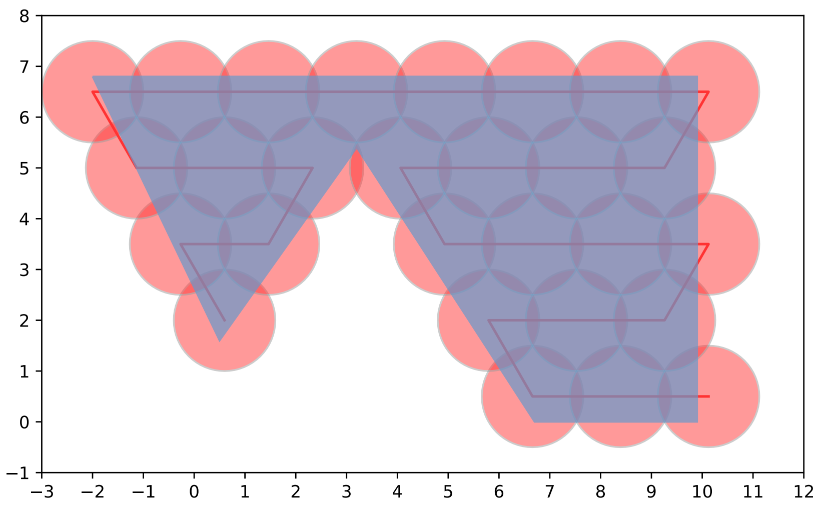
After determining washing circles, the robot needs to know the order to visit them. There are rules for which order to visit circles. It has to be from bottom to top. The reason to wash from bottom to top is to let the washing water flow in wet part of wall, to avoid marks on the wall. Fig 1.4a illustrates a valid path and Fig 1.4b an invalid path.

1.5.3 Evaluation metric of path

The speed of the boom robot between washing circles is not possible to determine accurate, because there are many joints with different limits and velocities. However would a shortest possible path most likely give the fastest path for the robot. The path should also look natural by human eye. This problem can be formulated as euclidean distance travel salesman problem(TSP). To guarantee the path to visit washing circles in correct order there has to be done some modifications to the algorithms.



(a) Valid path for visiting washing circles, circles are visited in bottom to top order.



(b) Invalid path, the path visit a circles above before circles lying under.

Figure 1.4: Two different path of visiting washing circles, fig 1.4a is valid and fig 1.4b is illustrating an invalid path.

Chapter 2

Background Theory

This chapter presents relevant background theory. Some information about general optimization problem, which can be used to solve TSP. Formulating TSP as an optimization and algorithms for solving TSP are also presented.

2.1 General optimization formulation

In [7] a general optimization problem is described, which can be rewritten as follow

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{array}{ll} c_i(x) = 0, & i \in \mathcal{E} \\ c_i(x) \geq 0, & i \in \mathcal{I} \end{array} \tag{2.1}$$

Where f is the objective function, x is the vector of variables and c_i are constraints. $\mathcal{E}$ and $\mathcal{I}$ are the sets of equality and inequality constraints.

2.1.1 Integer linear programming

Integer linear program (ILP) is optimization problem where the objective function and constraints are linear. In [8] ILP is formulated, which can be rewritten as

$$\min c^\top x \quad \text{subject to} \quad \begin{array}{r} Ax = b \\ x \geq 0 \\ x \in \mathbb{Z}^n \end{array} \tag{2.2}$$

where A is $m\ by\ n$ matrix, and x is n-dimensional column vector. b is m-dimensional column vector. x is also positive integer. c is an n-dimensional row vector.

2.2 Traveling Salesman Problem

Traveling salesman problem(TSP) is a combinatorial problem optimization problem. The problem has been studied for decades, and has roots back to 19. century[9]. The problem is simple to define but hard to solve. Given a set of cities and distance between them. What is the shortest possible route visiting every city once and returning to the start city[10]? According to [11] TSP is a NP-hard problem. Which means there is no algorithm that can solve the problem in polynomial time.

If the distance from city A to B is equal to B to A, the problem is symmetrical. On the other hand, if distance A to B is not equal to B to A, the problem is asymmetrical. In this thesis it is focusing the asymmetrical TSP.

2.2.1 Mathematical Formulation of the Symmetric TSP

The TSP can be formulated mathematical as an ILP, in this thesis the Miller–Tucker–Zemlin (MTZ) is used. The paper presenting this formulation was published in April 1960[12]. The formulation can be rewritten to match the notation of this report as follow

$$\min \sum_{i=1}^{n} \sum_{j\neq i, j=1}^{n} c_{ij} x_{ij} : \tag{2.3a}$$

$$x_{ij} \in \{0,1\} \qquad i,j = 1,\dots,n \tag{2.3b}$$

$$u_i \in \mathbf{Z} \qquad i = 2,\dots,n \tag{2.3c}$$

$$\sum_{i=1, i\neq j}^{n} x_{ij} = 1 \qquad j = 1,\dots,n \tag{2.3d}$$

$$\sum_{j=1, j\neq i}^{n} x_{ij} = 1 \qquad i = 1,\dots,n \tag{2.3e}$$

$$u_i - u_j + n x_{ij} \leq n-2 \qquad 2 \leq i \neq j \leq n \tag{2.3f}$$

$$1 \leq u_i \leq n-1 \qquad 2 \leq i \leq n \tag{2.3g}$$

$c_{i,j}$ is the cost between city i and j. $x_{i,j}$ decides which edges are part of the solution. $x_{i,j} = 1$ means that $c_{i,j}$ is part of the solution, when $x_{i,j} = 0$ is $c_{i,j}$ not part of the solution. u_i is a dummy variable keeping the order cities are entered. The constraints 2.3d and 2.3e require that every city is entered and left once. Contraint 2.3f and 2.3g eliminates sub tours and keep the solution to a Hamiltonian circle.

2.3 Traveling Salesman Problem algorithms

2.3.1 Nearest neighbour

The nearest neighbour (NN) algorithm solve TSP with a basic technique. A random starting vertex on the graph is chosen, from this the algorithm take a greedy decision. The next vertex has shortest distance from the current, and also has to be unvisted. This is repeated until every vertex is visited, then the algorithm return to starting point. The computation time of NN algorithm is $O(n^2)$[13], however it always provide a valid result without subtours.

2.3.2 Genetic algorithm

The genetic algorithm(GA) is a metaheuristic algorithm, that can solve many different problems[14]. The algorithm use natural selection, with survival of the fittest and evolution of a population. The population contains individuals with different genetic material. The focus in this thesis will be on using GA to solve TSP. When solving TSP with GA each individual contain a list of cities, the list is equal to a path. These individuals are generated random, and multiple individuals forms a population. The population is evolved using crossover and mutations.

Crossover is when the genes from two random parents are combined to form a child, in TSP is that the path from two parents are combined. One implementation is two remove some random cities from one parent. A path in TSP should always contain every city once, so these cities have to be put back in a different order. The order is determined by the other parent. The removed cities is put back in same order, as they appear in the second parent. In Figure 2.1 is crossover between two parents illustrated.

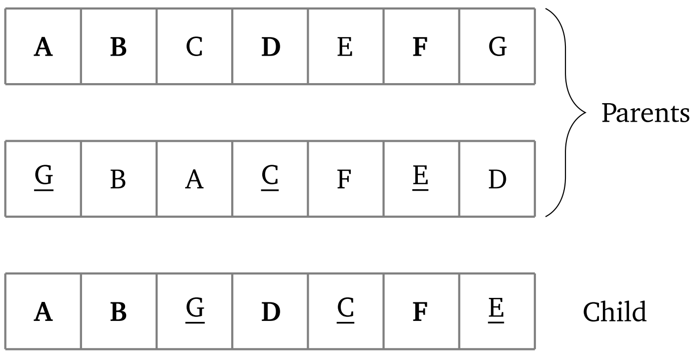


Figure 2.1: GA crossover or between two parents. The **bold** letter has the same placement in upper parent and child. The underlined characters has same order in lower parent and child. Figure inspired by [10].

Mutation is when the genetic material in an individual is changed though random events. Mutation on TSP is when two random cities change order on a individual[10]. Figure 2.2 illustrates mutation on a individual.

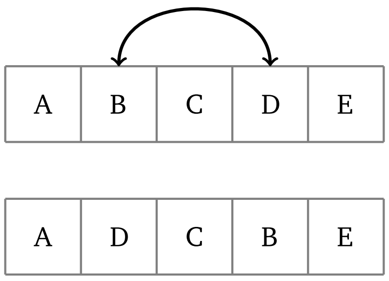


Figure 2.2: Example of mutation, two characters are swapped. Figure inspired by [10].

The next generation is created using crossover and mutation, which lead to an increased population. To select the next generation a fitness function is used. A fitness function calculate a value based on the genetic material, the distance of the path in TSP. A lower fitness value is better, so the distance is inverted. Based on the fitness function is the best individuals selected, after each generation.

The evolution of the individuals is either running a given number of generation or to the improvement between generation is below a set value. The fittest individual is the best performing, and chosen as the solution.

2.3.3 2-opt

The 2-opt algorithm is a tour improvement algorithm [15]. The algorithm improve an existing tour by trying to reordering two edges. If this change give a shorter tour it is kept. This is repeated until every valid combination is tried after last time reordering two edges. Hence the algorithm run until no improvement is made on the tour. Figure 2.3 illustrates a valid swap of edges for 2-opt algorithm.

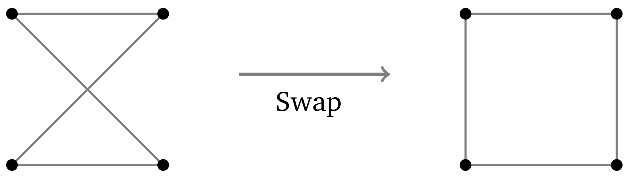


Figure 2.3: Example of 2-opt algorithm with valid swap of edges.

Chapter 3

Software

This chapter give a brief introduction to programming packages used in this thesis. All algorithms are implemented in Python. This section is partially based on [3].

3.1 Shapely

Shapely[16] are a geometry Python library used for manipulation and analysis of geometric objects in 2D. In this thesis it is used to represent washing surfaces, washing circles and paths. The results in this thesis is plotted using Shapely.

3.2 NumPy

Numpy[17] is a open source library that provides array, maatrix and linear algebra operation.

3.3 Matplotlib

Matplotlib[18] is a open source Python library used for plotting figures in Python.

3.4 SciPy

SciPy[19] is a Python package for mathematics, science, and engineering. It provide common algorithms for minimizing of constrained and unconstrained objective functions.

3.5 Pyomo

Pyomo[20] is open-source based Python package for solving and formulating optimization problems. Pyomo support different solvers, and is used to solve TSP as ILP in this thesis.

3.6 Gurobi

Gurobi[21] is a solver for optimization problems, which provide a own language for defining optimization problems. Gurobi also provide bindings to Pyhon and other common programing languages[22]. In this thesis Gurobi is used to solve optimization problems defined in Pyomo.

Chapter 4

System Implementation

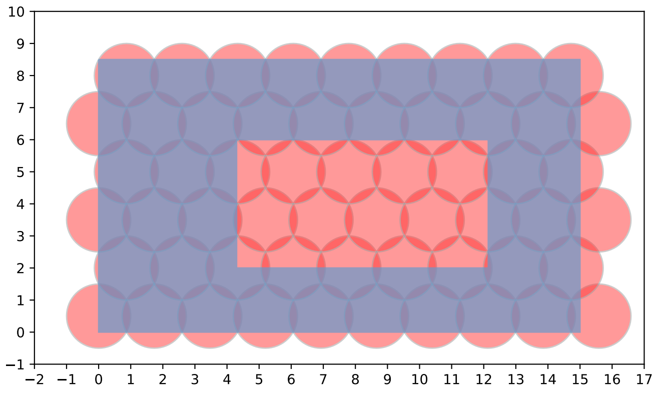
This section presents implementation of algorithms to solve path planning for boom lift. Which is solved as a traveling salesman problem.

4.1 Generating washing circles

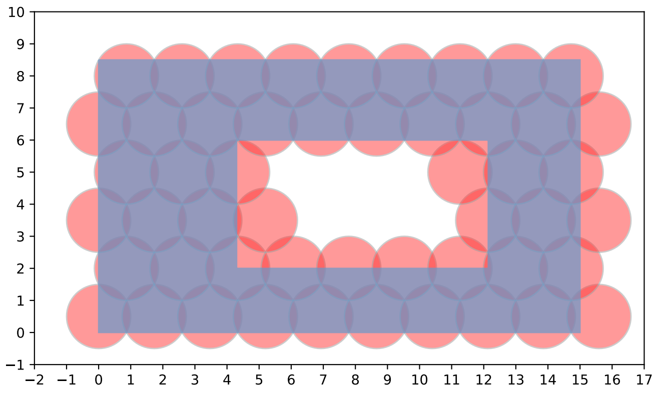
In [3] Berge present different algorithms for covering a polygon with circles of uniform size. However the report do not present any solution to remove circles from no-go areas inside polygons.

4.1.1 Remove circles from no-go areas

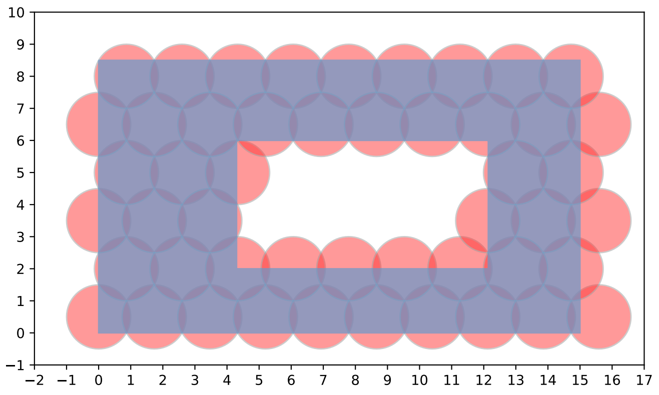
The polygon is filled with circles using algorithms presented in [3]. First step is to remove circles completely inside no-go areas removed. Next step is to remove circles partially inside no-go area. This is done by removing one circle at the time, and check if the whole polygon is still covered. If the whole polygon is still covered, the circle is removed. If not the circle is kept as washing circle. This is repeated for every circle partially covering the no-go area. The implementation of this algorithm is heavily based on Shapely, which provide manipulation and analysis of 2D-objects.



(a) Polygon is filled with circles, using algorithms from Berge[3]



(b) Circles completely inside no-go area are removed.



(c) Circles partially inside no-go area are removed, but polygon is still completely covered by circles.

Figure 4.1: Step of covering polygon with circles and remove circles inside no-go area.

4.2 Ordering washing circles

4.2.1 Representation of distance between washing circles as graph

The algorithms that solve TSP-problems need the distance between cities to compute the optimum solution. This can be represented as a nxn matrix, where n is the number of cities. Algorithm 5 implements calculation of graph.

Tab 4.1 show the distance between the four washing circles illustrated in Fig. 4.2. The circle number is added to both figure and table to improve readability.

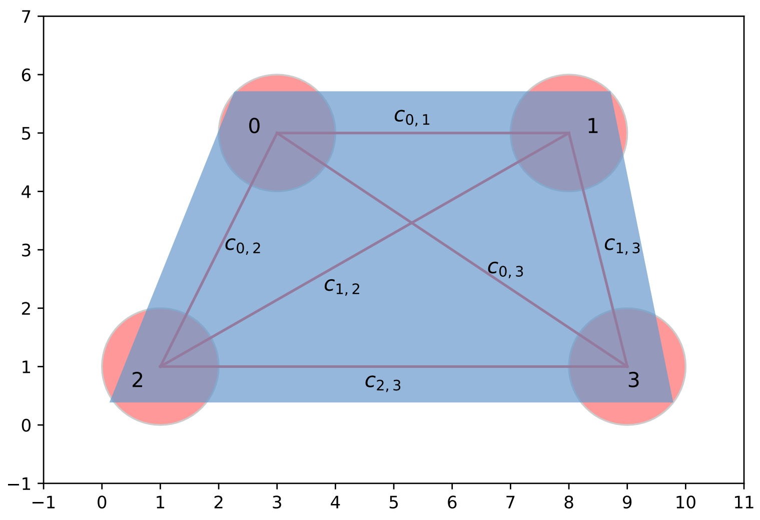


Figure 4.2: Illustration of four circles with all possible direction between, the cost of $c_{i,j}$ is equal to $c_{j,i}$, hence it a symmetric graph.

	0	**1**	**2**	**3**
0	0	5	4.5	7.2
1	5	0	8	4.1
2	4.5	8	0	8
3	7.2	4.1	8	0

Table 4.1: Graph of distance between washing circles illustrated in 4.2, represented as matrix.

4.2.2 Nearest neighbour implementation

The NN algorithm takes local greedy choice as the next position in the path. The next step is the one with minimum cost to add to the path. The starting point for the algorithm is lowest washing circles, from then it takes a greedy choice to

	0	**1**	**2**	**3**
0	4	4	4	4
1	4	4	4	4
2	4	4	4	4
3	4	4	4	4

Table 4.2: Graph for four washing circles represented as matrix

choose the rest of the path. There is boolean variable storing information about each washing circle is visited or not. When searching for the next, only washing circles with that is not visited jet are possible to visit next. This is to avoid visiting the same washing circles multiple times. The NN algorithm is implemented in Algorithm 1.

Algorithm 1 standard NN algorithm.

```
1:  function STANDARD_NN(G)
2:      $visited \leftarrow [False, False, ..., False]$
3:      $next \leftarrow find\_lowest\_WC(washing\_circles)$
4:      $path \leftarrow []$
5:      $path.append(next)$
6:      $visited[next] \leftarrow True$
7:      while not every washing_circles visited do
8:          $next = find\_closest(next, graph, visited)$
9:          $visited[next] \leftarrow True$
10:         $path.append(next)$
11:     end while
12:     return $path$
13: end function
```

4.2.3 2-opt implementation

The 2-opt algorithm is a tour improvement algorithm as stated in Section 2.3.3. In this section the implementation of the algorithm is presented. The algorithm are swapping two edges at time as illustrated at Figure 2.3. If the path is shorter after swapping the edges, it is saved. Otherwise the old path is kept. This is tried for every possible combination of swapping two edges until no improvement is made on the path. In Algorithm 2 is this implemented.

4.2.4 Modified nearest neighbour

The NN algorithm presented in 2.3.1 can be modified to solve the TSP of visiting order of washing circles presented in 1.5.2. The modification improve the performance of the algorithm.

There has to be added some extra constraints to the original NN algorithms to provide a result satisfy the requirements presented in 1.5.2. There are two requirements, start at the bottom and visit the circles below has to be visited first.

Algorithm 2 2-opt algorithm.

```
function 2-OPT(G)
    $path \leftarrow standard\_NN(G)$
    $graph \leftarrow calc\_graph(washing\_circles)$
    $starting\_circle \leftarrow find\_lowest\_circle(washing\_circles)$
    $path\_improved \leftarrow True$
    $path\_cost \leftarrow calc\_path\_length(G, path)$
    while tour_improved do
        $path\_improved \leftarrow False$
        for i in $1 : length(path) - 2$ do
            for j in $(i+1) : length(path)$ do
                $temp\_path \leftarrow 2\_opt\_swap(path, i, j)$
                $temp\_path\_cost \leftarrow calc\_path\_length(G, temp\_path)$
                if $temp\_tour\_cost < tour\_cost$ then
                    $path\_cost \leftarrow temp\_path\_cost$
                    $path\_improved \leftarrow True$
                end if
            end for
        end for
    end while
    return $path$
end function
```

Algorithm 3 Calculation of path length.

```
function CALC_PATH_LENGTH(G, path)
    $length \leftarrow 0$
    for l in $1 : length(path) - 1$ do
        $temp \leftarrow G[path[l], path[l+1]]$
        $length \leftarrow length + temp$
    end for
    return $length$
end function
```

The starting point for the algorithm is the lowest lying washing circles. It then take local greedy decision to choose next washing circle. To satisfy the requirements presented in 1.5.2 a constraint is added.

At every iteration it is checked which washing circles is possible to visit next, without breaking the requirements presented 1.5.2. This information is saved as a binary variable. The next is the one with shortest euclidean distance from last washing circle, that satisfy these requirements. In Algorithm 4 is this implemented.

4.2.5 Genetic algorithm

The first step in genetic algorithm is to generate a random population of individual. Each individual is in this case a path. The evolution of the individuals is in this implementation set to run a given number of generation. In each generation is first the fitness of each individual calculated, the euclidean distance of the path. The fittest path is kept, until next generation. Next step is crossover and mutation. The crossover is choosing two individuals, and crossing over genes. Fig-

Algorithm 4 Modified NN algorithm.

```
1:  function NN_modified(washing_circles)
2:      $graph \leftarrow calc\_graph(washing\_circles)$
3:      $visited \leftarrow [False, False, ..., False]$
4:      $next \leftarrow find\_lowest\_WC(washing\_circles)$
5:      $path \leftarrow []$
6:      $path.append(next)$
7:      $visited[next] \leftarrow True$
8:      while not every washing_circles visited do
9:          $valid\_to\_visit \leftarrow find\_allowed(washing\_circles, visited)$
10:         $next = find\_closest(next, graph, valid\_to\_visit)$
11:         $visited[next] \leftarrow True$
12:         $path.append(next)$
13:     end while
14:     return $path$
15: end function
```

Algorithm 5 Calculation of graph.

```
1:  function CALC_GRAPH(washing_circles)
2:      $graph \leftarrow zeros(washing\_circles.shape[0], washing\_circles.shape[0])$
3:      for i in $d\_samples$ do
4:          for j in $d\_samples$ do
5:              $area\_washed \leftarrow compute\_area\_washed(d, h, offset, washing\_circles)$
6:              $x \leftarrow washing\_circles[i][0] - washing\_circles[j][0]$
7:              $y \leftarrow washing\_circles[i][1] - washing\_circles[j][1]$
8:              $graph[i, j] \leftarrow hypot(x, y)$
9:          end for
10:     end for
11:     return $graph$
12: end function
```

ure 2.1 illustrated crossover. The next step is mutation, which is performed on the crossovered individuals. Part of the individuals is randomly changed, like in Figure 2.2. After a given number of generation is the fittest individual selected and returned. This is the shortest path found by the algorithm.

Algorithm 6 Genetic algorithm.

```
1: function GA(G, pop_size, generations)
2:     $pop \leftarrow []$
3:     $fitness \leftarrow []$
4:     for i in 1 : $pop\_size$ do
5:         $pop.append(generate\_random\_path(G))$
6:     end for
7:     for j in 1 : $generations$ do
8:         for j in 1 : $length(pop)$ do
9:             $ind\_1 \leftarrow pop[random(0, length(pop))]$
10:            $ind\_2 \leftarrow pop[random(0, length(pop))]$
11:            $ind\_1, ind\_2 \leftarrow crossover(ind\_1, ind\_2)$
12:            $ind\_1, ind\_2 \leftarrow mutate(ind\_1, ind\_2)$
13:            $pop.append(ind\_1)$
14:            $pop.append(ind\_2)$
15:        end for
16:        for j in $(i+1)$ : $length(path)$ do
17:            $fitness \leftarrow get\_fitness(pop)$
18:        end for
19:        $pop \leftarrow keep\_fittest(pop)$
20:    end for
21:    $path \leftarrow get\_fittest\_individual(pop)$
22:    return $path$
23: end function
```

4.2.6 Solve MTZ as ILP

In Algortithm 7 is TSP solved as an optimization problem using the formulation by MTZ presented in Section 2.2.1. The optimization problem is implemented in Pyomo a optimization framework for Python. The solver used is Gurobi, which provide solvers for ILP problems[20]. The solver finds a tour that create a Hamiltonian cycle. To create a path according to the specification in Section 4.2 the edge from top to bottom are removed before the path is returned. To always get a edge from top to bottom are the distance set to zero from top washing circles to the lowest. The algorithm return a path visiting every washing circle once, starting at bottom and ending at the top. When calculating the graph, the horizontal distance is multiplied by a factor. To get a path starting at the bottom and ending at the top, which satisfy the requirements in Section 4.2.

Algorithm 7 optimization euclidean algorithm.

```
1: function OPTIMAL_EUCLIDEANTSP(G)
2:     $sol \leftarrow Solveoptin2.3$
3:     $path \leftarrow extract\_tour\_from\_sol$
4:     $path \leftarrow remove\_edge\_top\_bottom(path)$
5:     return $path$
6: end function
```

4.2.7 Pre-processing of graph

The algorithm presented in 4.2.6 gives a huge amount of variables when solving the optimization problem with many washing circles. This can be reduced with some pre-procession of the graph. The main idea is to reduce the number of washing circles, by assuming it is naturally to visit nearby circles. Which causes lower number of variables.

The circles generated is assumed lies on horizontal lines, however they are not sorted from the algorithm determining the placement. This is done in Algorithm 9 and illustrated in Figure 4.3a.

The next step is to split the lines where the circles is not intersection each other on same line. This is areas of the washing surface where there are no go areas, for examples windows or other obstacles. In Algorithm 10 is this preformed and illustrated in Figure 4.3b.

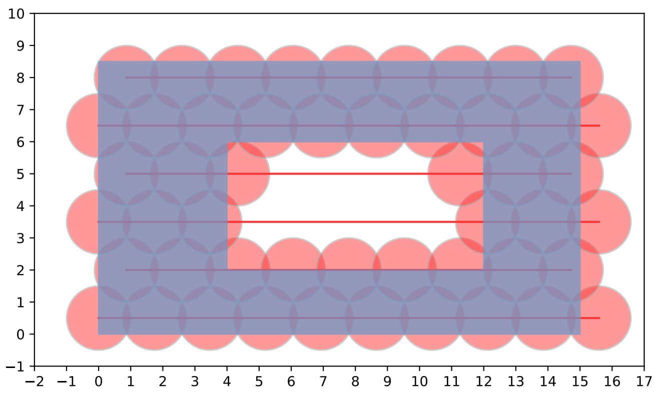
Now each line segment consist of continuous overlapping circles that lies on a horizontal line. The end circles from each line is extracted, which is used to calculate the graph. Algorithm 11 extract the end circles as illustrated in Figure 4.3c. When calculating the graph only the end circles illustrated in Figure 4.3c are used. First is all distances calculated as euclidean distance using Pythagoras. The distance between end circles on same line segment is set to zero.

To guarantee the algorithm to find a valid path, visiting washing circles in correct order, it is necessary to adjust the graph. The circles has to be visited in a bottom to top order. In Figure 4.4 all possible next stops on the path are illustrated with arrows. It is illegal to go to a washing circle below, hence there is no arrow pointing downwards. The distance in direction of the arrows is set to euclidean distance. In the opposite direction the distance is set to Inf, because it is an invalid step.

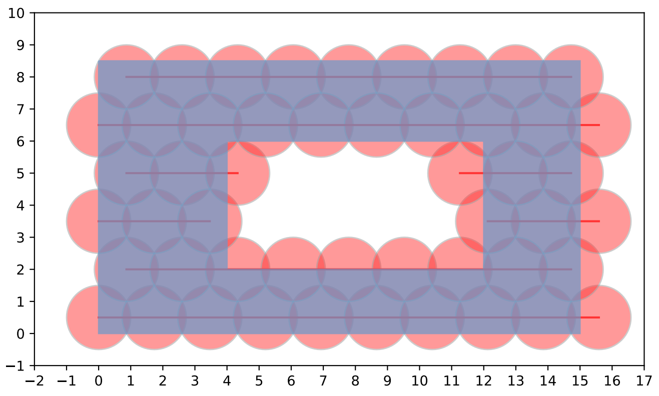
To determine the optimum path the algorithm presented in Section 4.2.6 is used. Solving TSP as an ILP with the MTZ formulation with Algorithm 7.

Algorithm 8 optimization reduce circles algorithm.

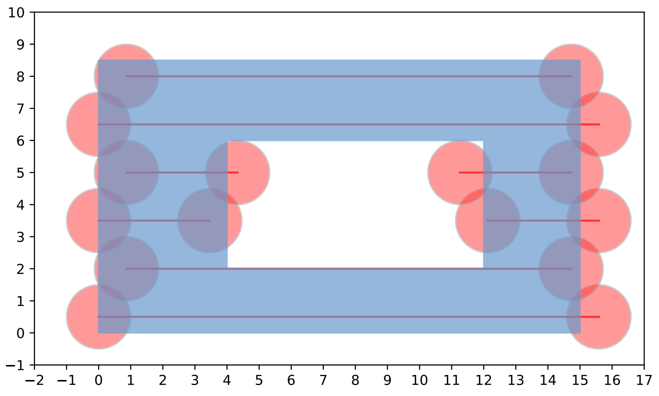
```
1: function REDUCE_CIRCLES(washing_circles)
2:     $lines \leftarrow find\_lines(washing\_circles)$
3:     $splitted\_lines \leftarrow split\_lines(lines)$
4:     $graph \leftarrow calc\_graph(washing\_circles)$
5:     $path \leftarrow optimal\_EuclideanTSP(graph)$
6:     return $path$
7: end function
```



(a) Find horizontal lines of washing circles.



(b) Split lines on blank areas, e.g. no-go-areas



(c) Extract end of lines, the distance between end of lines is set to zero in the graph.

Figure 4.3: Step of prepossession of washing circles before calculation of graph.

Algorithm 9 Find horizontal lines of washing circles.

```
function FIND_LINES(washing_circles)
    $lines \leftarrow []$
    $temp\_line \leftarrow []$
    $number\_of\_wc \leftarrow len(washing\_circles)$
    $earlier_y \leftarrow []$
    for $i \leftarrow 0$ to $number\_of\_wc$ do
        $Sum \leftarrow Sum + A_k$
        if $washing\_circles[i][1] inearlier\_y$ then
            $True$
        else
            $earlier\_y.append((washing\_circles[i])$
            for $i \leftarrow 0$ to $number\_of\_wc$ do
                if earlier_y[-1] == washing_circles[j][1] then
                    $temp\_line.append(j)$
                end if
                $lines.append(temp\_line)$
                $temp\_line \leftarrow []$
            end for
        end if
    end for
    return $lines$
end function
```

Algorithm 10 Spit lines at no-go areas.

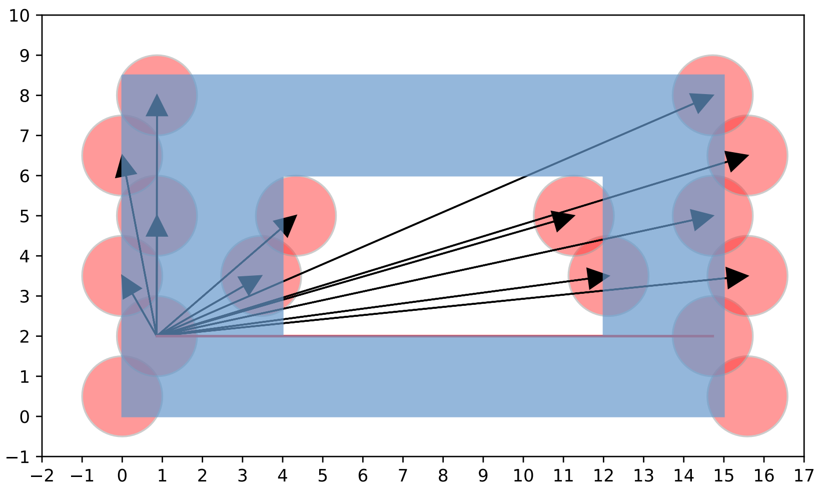
```
function SPLIT_LINES(washing_circles, lines)
    $splitted\_lines \leftarrow []$
    $temp\_line \leftarrow []$
    for $jinlines$ do
        for $kinrange(length(j))$ do
            $current\_wc \leftarrow wc[k]$
            $next\_wc \leftarrow wc[k+1]$
            $d \leftarrow distancebetweencurrentandnextwc$
            if $d > radius * 1.8splitline$ then
                $splitted\_lines.append(temp\_line)$
                $temp\_line \leftarrow []$
                $temp\_line.append(k+1)$
            else
                $temp\_line.append(k+1)$
            end if
        end for
    end for
    $splitted\_lines.append(temp\_line)$
    return $splitted\_lines$
end function
```

Algorithm 11 Extract end circles from lines.

```
1: function EXTRACT_END_OF_LINES(splitted_lines, wc)
2:     $end\_circles \leftarrow []$
3:     for $j$ in $splitted\_lines$ do
4:         if length(j) == 1 then
5:             $end\_circles.append([j[0]])$
6:         else
7:             $end\_circles.append([j[0], j[-1]])$
8:         end if
9:     end for
10:    return $end\_circles$
11: end function
```

Algorithm 12 Calculation of graph .

```
1: function CALC_GRAPH(washing_circles, end_cirles)
2:     $splitted\_lines \leftarrow []$
3:     $temp\_line \leftarrow []$
4:     $number\_of\_circles \leftarrow length(end\_cirles)$
5:     for $iinnumber\_of\_circles$ do
6:         for $jinnumber\_of\_circles$ do
7:             if $i == j$ then
8:                 $graph[i, j] \leftarrow Inf$
9:             else
10:                $graph[i, j] \leftarrow distance(number\_of\_circles[i], number\_of\_circles[j])$
11:            end if
12:        end for
13:    end for
14:    $graph \leftarrow set\_distance\_between\_circles\_on\_same\_line\_to\_zero(graph)$
15:    $graph \leftarrow set\_distance\_to\_lines\_below\_to\_Inf(graph)$
16:    return $graph$
17: end function
```



0.8

Figure 4.4: Arrows illustration euclidean distance between washing circles, to avoid visiting in wrong direction, the distance in opposite of arrow direction is set to INF.

Chapter 5

Result

The five algorithms presented in Chapter 4 is tested on two different test cases. The first is a convex washing surface. The second test is concave polygon containing a no-go area. To determine position of washing circles is the algorithm presented in Berge[3] used, the best performing one is chosen. This algorithm fill the washing area with circles in a hexagon pattern and remove circle outside. To remove circles from no-go area the algorithm presented in Section 4.1.1 is used.

5.1 Test case 1

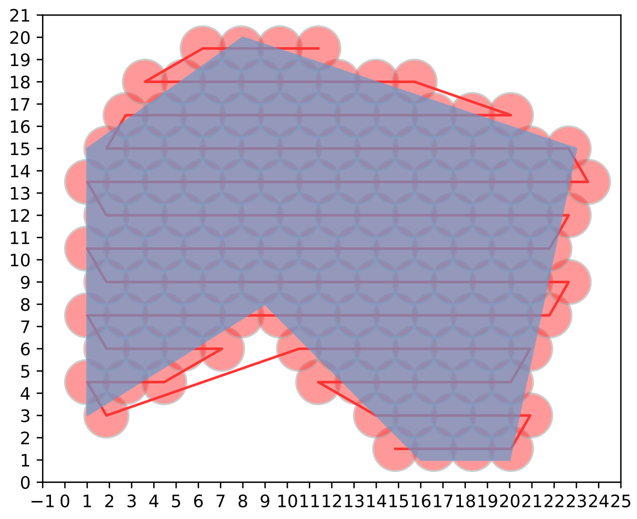


Figure 5.1: Pre-processing algorithm on test case 1, one of the best performing algorithms on this test.

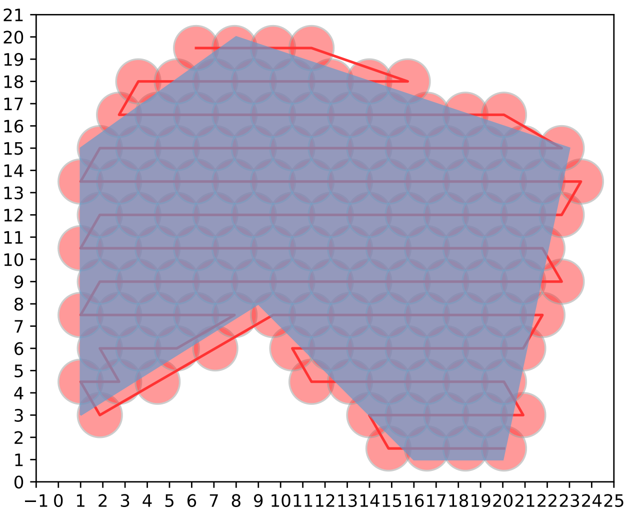


Figure 5.2: Modified NN algorithm in test case 1, generates the shortest path of all algorithms tested.

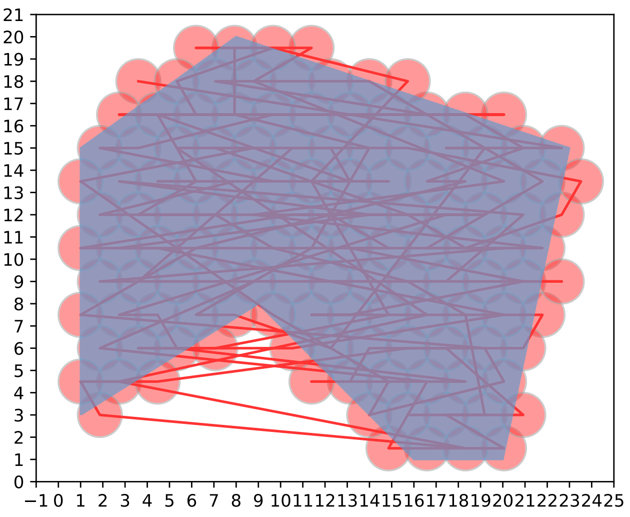


Figure 5.3: GA algorithm on test case 1 is not finding a valid solution.

In table 5.1 are the result from the first test compared. The best performing algorithms is the modified NN and pre-processed optimization. The standard TSP optimization perform well on distance, but has longer computation time. This long computation time is due to the high number of variables in the optimization

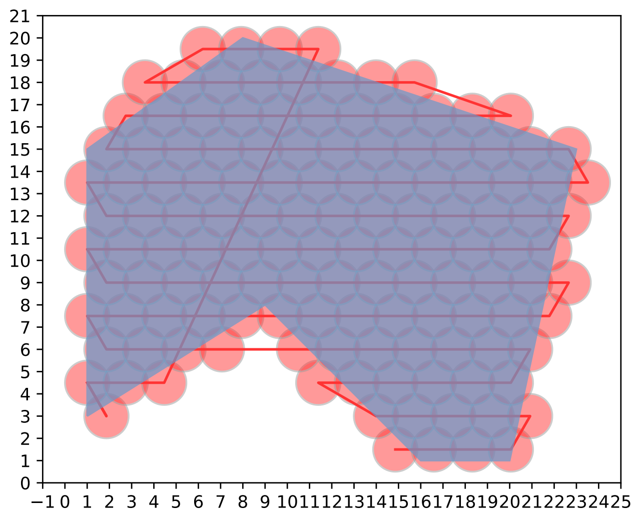


Figure 5.4: NN with 2-opt algorithm on test case 1, the path is visiting washing circles from above, hence it is an invalid path.

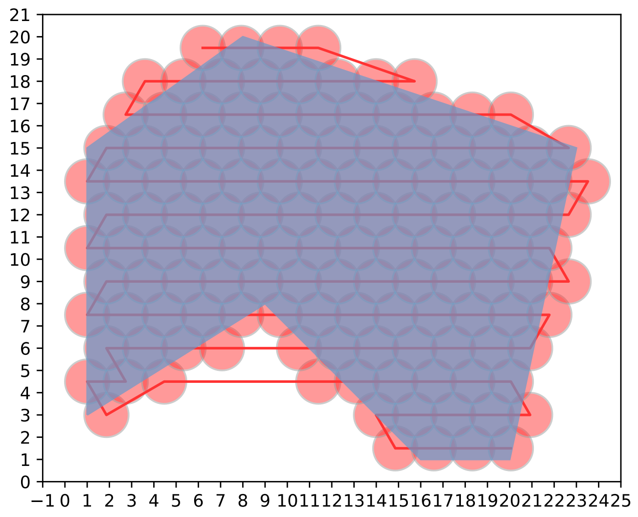


Figure 5.5: A path generated solving TSP as ILP on test 1. The calculation takes 53 seconds, way longer than the other algorithms tested.

problem. In this case the algorithm has 132 points, while the pre-processed only has 31 points. Both are solved as ILP with the same algorithm.

The GA in Fig 5.3 are not finding a valid solution. It seems that the algorithm

Algorithms	$Distance$	t_{comp}
Modified NN	236	0.15s
GA	Not valid	-
Opt	239	53
NN with 2-opt	Not valid	239
Prepossess opt	240	0.215s

Table 5.1: Table presenting result from the first test

has problem finding a better solution because the difference in distance between washing circles is so small. Which gives a small number of improvement between generation.

The standard NN with 2-opt does not find a valid path. The 2-opt has low improvement of the path, because only two edges are tried swapped at time. To get path similar to the best performing algorithms multiple edges has to be swapped at same iteration.

5.2 Test case 2

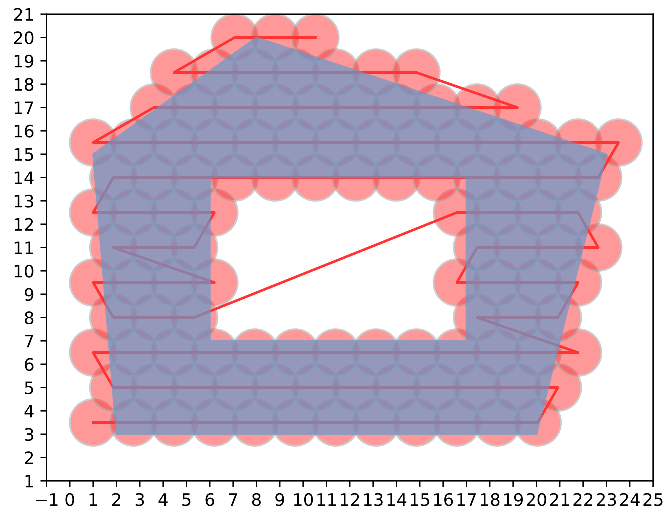


Figure 5.6: Pre-processing algorithm on test case 2, is the second best performing algorithm on both computational time and length of path.

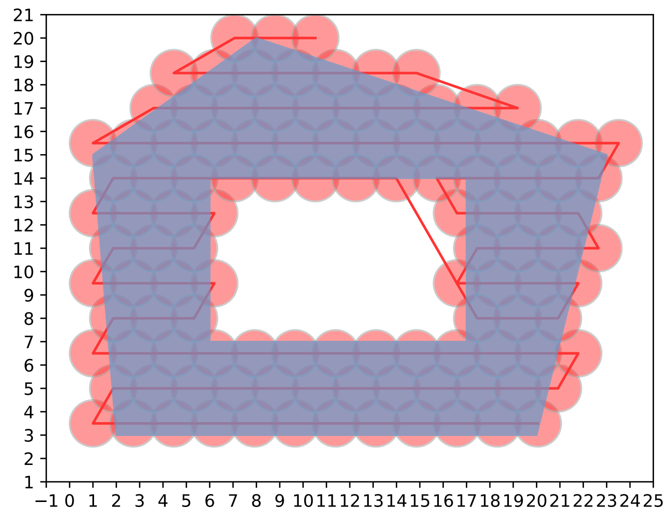


Figure 5.7: The modified NN is the best performing algorithm on test case 2. The path looks natural by human eye.

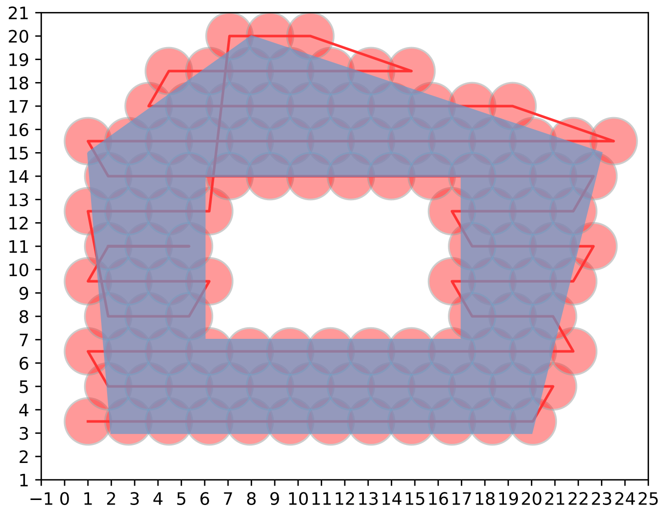


Figure 5.8: NN with 2-opt on test 2, is not generating a valid path. Because the path is going downwards on the left side.

Algorithms	*Distance*	t_{comp}
Modified NN	205	0.12s
GA	Not valid	
Opt	Not valid	
NN with 2-opt	Not valid	
Preprocess opt	215	0.23s

Table 5.2: Table presenting result from the secound test

In table 5.2 are the result presented for test 2. Only two of the algorithms are giving a valid result. The best performing algorithms in test 2 are pre-processed opt and modified NN. Which also had the best performance in the first test.

According to Table 5.2 the computation time is very low for both. However the distance for the modified NN algorithm is slightly shorter. In Figure 5.6 and Figure 5.7 are the path for these two algorithms. The reason why modified NN find a shorter path, is because pre-processed opt has to follow a whole line of circles when first entered. The path from pre-processed opt is crossing the no-go area, because it can not takes part of circles over the no-go area like the modified NN. It has to visit every on same line segment when first entered, and every below has to be visited first as well. This lead to a slightly longer path in this case.

The GA algorithm suffer from the same problems that in the first test case. The standard opt solved as ILP is not finding a valid path, it use very long time to compute.

Chapter 6

Discussion

This section discuss the result from Section 5

The GA is the worst performing algorithm, it is never finding a valid solution. It has very long computational time, since it is evolving through a lot of generations with multiple individuals. There is most likely many stops on the path with similar distance between, which gives low difference between individuals. When some of the genes are swapped between generations, there are too many possible path that can be good. Finding the optimal solution with a sort of random search are not the best for this problem. On the other hand it can maybe be possible to modify the algorithm to value different parameters when calculating fitness. Not only the length of the path, but also how well it scores on visiting washing circles in correct order.

The NN with 2-opt and opt algorithm are only standard solution for solving TSP. They are part of the thesis to show how standard implementation perform compared to more specialized algorithms. The standard algorithms has not extra constraint, so they are not finding a valid solution at every test. The standard opt algorithm suffer especially from high number of variables, which causes long computational time compared to the other algorithms.

The modified NN is an improved version of the standard NN, to work better with the problem in this thesis. When generating the path, from bottom to top, the next step is always valid. According to the requirements specified in section 1.4. This gives a valid path, and in the test cases in this thesis it has performed best of all algorithms.

The pre-processes opt algorithm are an improvement of the opt algorithm. Where the graph is simplified and only contains end of lines of washing circles. Instead of every washing circle. Invalid path segments are set to infinity in the graph. This give a significant lower number of variables in the when solving TSP as an optimization problem. Which leads to low computation time. This algorithm is performing slightly worse than modified NN, but way better than the rest of algorithms tested in this thesis.

Chapter 7

Conclusion

This thesis has implemented five different algorithms for finding a valid path between washing circles, for a telescopic boom lift robot. These algorithms solve travel salesman problem in 2D. There are some extra constraint on which order the washing circles are visited. The circles has to be visited from top to bottom.

The path planning algorithms are implemented in 2D. This algorithms can be modified to work in 3D, the modified NN are easiest to do this with.

The GA, standard NN and opt are performing worse, and are not finding a valid solution to every test. The GA is performing worst and never finding a valid path.

Further work can be a direction with more complex surfaces in 3D. This thesis has only focused on finding path in 2D, however the algorithms can most likely be modified to work in 3D. The modified NN and pre-processed optimized are most likely best suited for further work.

Thus it concluded that the modified NN algorithm and pre-pocessed optimization are performing best. They has both the shortest computation time and finding the shortest path among all algorithm tested in this thesis. The path provided from these algorithm also looks like a natural choice for human eye.

Bibliography

[1] *Nlink*, 6.12.2021. [Online]. Available: https://www.nlinkrobotics.com/.

[2] *Hilti jaibot*, 9.12.2021. [Online]. Available: https://www.hilti.com/content/hilti/W1/US/en/engineering/industry-and-trade-solutions/construction-automation/jaibot.html.

[3] R. Berge, 'Path planning for a telescopic boom lift robot,' Specialization Project Report, Norwegian University of Science and Technology, Department of Engineering Cybernetics, 2021.

[4] *Universal robots*, 6.10.2021. [Online]. Available: https://www.universal-robots.com/.

[5] *Hilti plt 300 layout tool*, 4.12.2021. [Online]. Available: https://www.hilti.com/c/CLS_MEA_TOOL_INSERT_7127/CLS_CONSTRUCTION_TOTAL_STATIONS_7127/r4728599.

[6] *Hilti prism poa 25*, 4.12.2021. [Online]. Available: https://www.hilti.com/c/CLS_MEA_TOOL_INSERT_7127/CLS_ACC_MEASURING_TOOL_AND_SCANNER_7127/CLS_RECEIVER_TGT_PLATE_REFLECTOR_7127/2115937.

[7] J. Nocedal and S. J. Wright, *Numerical Optimization, Second Edition*, 2nd ed. Springer, 2006.

[8] L. A. Wolsey, *Integer Programming*. John Wiley Sons, 1998.

[9] R. J. W. Norman Biggs E. Keith Lloyd, *Combinatorial Optimization*. Clarendon Press, 1976.

[10] J.-Y. Potvin, 'Genetic algorithms for the traveling salesman problem,' *Annals of Operations Research*, no. 63, 1996. [Online]. Available: https://www.inf.tu-dresden.de/content/institutes/ki/cl/study/summer14/pssai/slides/GA_for_TSP.pdf.

[11] B. Korte and J. Vygen, *Combinatorial Optimization*. Springer Berlin Heidelberg, 2018. [Online]. Available: https://doi.org/10.1007/978-3-662-56039-6.

[12] C. E. Miller, A. W. Tucker and R. A. Zemlin, 'Integer programming formulation of traveling salesman problems,' vol. 7, no. 4, 1960.

[13] D. S. Johnson and L. A. McGeoch, *"The traveling salesman problem: A case study in local optimization," in Local Search in Combinatorial Optimization*. E. H. L. Aarts, J. K. Lenstra, Eds. John Wiley and Sons, 1997.

[14] C. Blum and A. Roli, 'Metaheuristics in combinatorial optimization: Overview and conceptual comparison,' *ACM Comput. Surv.*, vol. 35, no. 3, 2003. [Online]. Available: `https://doi.org/10.1145/937503.937505`.

[15] G. A. Croes, 'A method for solving traveling-salesman problems,' *Operations Research*, vol. 6, no. 6, 1958. [Online]. Available: `http://www.jstor.org/stable/167074`.

[16] S. G. et al, *Manipulation and analysis of geometric objects in the cartesian plane*, 6.10.2021. [Online]. Available: `https://github.com/Toblerity/Shapely`.

[17] *Numpy*, 6.10.2021. [Online]. Available: `https://numpy.org/`.

[18] *Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in python.* 6.10.2021. [Online]. Available: `https://matplotlib.org/`.

[19] *Scipy is a python-based ecosystem of open-source software for mathematics, science, and engineering*, 6.10.2021. [Online]. Available: `https://www.scipy.org/`.

[20] *Pyomo*, 28.04.2022. [Online]. Available: `http://www.pyomo.org/about`.

[21] *Gurobi*, 28.04.2022. [Online]. Available: `https://www.gurobi.com/`.

[22] *Gurobi python*, 28.04.2022. [Online]. Available: `https://www.gurobi.com/documentation/9.5/quickstart_mac/cs_grbpy_the_gurobi_python.html`.