MŰEGYETEM 1782

DEVELOPMENT OF THE SOLUTION OF THE TRAVELING SALESMAN PROBLEM WITH A TOUR CONSTRUCTION METHOD INVOLVING DYADIC TILINGS

By

Ángela TUDELA PÉREZ
/G4N40F/

Submitted to the
Department of Fluid Mechanics of the
Budapest University of Technology and Economics
in partial fulfillment of the requirements for the degree of
Bachelor of Science in Mechanical Engineering

on the 14th of May, 2021

BSc Thesis
Final Project /BMEGEÁTBKSD/

Supervisor:
Bendegúz BAK, PhD assistant professor

Department of Fluid Mechanics
Faculty of Mechanical Engineering
Budapest University of Technology and Economics

Budapest University of Technology and Economics
Faculty of Mechanical Engineering
Department of Fluid Mechanics
Bldg. "Ae" ✱ www.ara.bme.hu

BSC FINAL PROJECT ASSIGNMENT (BSC THESIS)

IDENTIFICATION

Name(code):	**Ángela TUDELA PÉREZ (G4N40F)**	BSc Thesis Report ID number:	**GEÁT-2020-21-2-AG0-PE-G4G40F**
Student ID:	**73616390059**		
Curriculum:	**BSc in Mechanical Engineering**	Name and code of specialisation: *n.a.*	**(ERASMUS)**
Code of curriculum:	**2NAAG0-PE (ERASMUS)**		
Final project issued by:	**Department of Fluid Mechanics**	Final exam organised by: *n.a.*	
Supervisor:	**Dr. Bendegúz Dezső BAK, assistant professor (FFBGXK, bak@ara.bme.hu, +36-30-2236136)**		
Supervisor ID:	**79513977519**		

PROJECT DESCRIPTION

Title

Development of the Solution of the Traveling Salesman Problem with a tour construction method involving dyadic tilings

Az Utazó Ügynök Probléma közelítő megoldásának fejlesztése diadikus csempézést alkalmazó útösszerakó módszerrel

Details

1. Study the literature of topic! The literature survey should focus on TSP solution approaches, tilings and bisectioning.
2. Implement the basic idea of the TSP solution with dyadic tilings in Matlab environment.
3. Improve the code in at least one aspect compared to the original code (e.g. efficient bisectioning to group the points in the tiling).
3. Solve a huge amount of random TSP instances to obtain statistics.
4. Evaluate the solution statistics, test whether there is a correlation between solution quality and the properties of the tiling.
5. Summarize your work in the required document format of the BSc Thesis!

Advisor

Advisor's Affiliation:
Dept. Fluid Mechanics, Faculty Mechanical Engineering, Budapest University of Technology and Economics
Bertalan L. 4-6., BME "Ae" building, H-1111 Budapest, Hungary
Advisor: -

FINAL EXAM

1st subject (group)	2nd subject (group)	3rd subject (group)
Full official name of the subject	Full official name of the subject	Full official name of the subject
n.a.	*n.a.*	*n.a.*

AUTHENTICATION

Handed out: 8th of February 2021	Deadline: 14th of May 2021
Compiled by [illegible] Dr. Bendegúz Dezső BAK Supervisor	Controlled by BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM ÁRAMLÁSTANI TANSZÉK Gépészmérnöki KAR Prof. János VAD Head of Department

The undersigned declares that all prerequisites of the Final Project have been fully accomplished. Otherwise, the present assignment for the Final Project is to be considered invalid.

Budapest, 8th of February 2021

Signed by TUDELA PÉREZ, ÁNGELA (FIRMA) the day 31/05/2021 with a

Ángela TUDELA PÉREZ
Student

DECLARATION

Full Name (as in ID):	Ángela TUDELA PÉREZ
Neptun Code:	G4N40F
University:	Budapest University of Technology and Economics
Faculty:	Faculty of Mechanical Engineering
Department:	Department of Fluid Mechanics
Major/Minor:	BSc in Industrial Engineering ERASMUS
BSc Thesis title:	Development of the Solution of the Traveling Salesman Problem with a tour Construction Method Involving Dyadic Tilings
Academic year of submission:	2020 / 2021 – II

I, the undersigned, hereby declare that the Thesis submitted for assessment and defence, exclusively contains the results of my own work assisted by my supervisor. Further to it, it is also stated that all other results taken from the technical literature or other sources are clearly identified and referred to according to copyright (footnotes/references are chapter and verse, and placed appropriately).
I accept that the scientific results presented in my Thesis can be utilised by the Department of the supervisor for further research or teaching purposes.

Budapest, on the 14th of May, 2021

[illegible]

(Signature)

FOR YOUR INFORMATION

ABSTRACT

The objective of this paper is to briefly explore and explain what the TSP is, as well as the most adequate methods to use depending on the numbers of cities considered, and the quality of the solution desired. After this, a particular approach using dyadic tilings and genetic algorithms is explained in further detail. This method basically consists on a hierarchical heuristic that partitions the unit square into dyadic rectangles. The points considered for the TSP are grouped into clusters depending on the tile where they belong, and the geometrical barycenter is represented for each tile or cluster. A 'coarse' solution is found for the barycenters and the midpoints for each line segment is defined. Minimum tours are calculated for each cluster, connecting these together through the barycenters, resulting in a solution for the points considered for the TSP. Finally it is explained how this method has been implemented into Matlab to obtain results using this method.

CONTENTS

1. Introduction

The Traveling Salesman Problem (TSP) is a very much researched problem in our society. To explain it in a very simple way, it can be considered as a situation in which a salesman has to travel from his hometown to several other towns and finally return home, visiting each of these towns only once and following the shortest, most optimal route. The TSP can be formulated as a combinatorial optimization problem [1], this is as a graph in which the cities are represented by points (or nodes). The nodes are interconnected through line segments, also known as arcs or edges. The length of these lines can be seen as the cost of going from one node to another. Once all the nodes have been connected forming a closed loop (graph theory term is Hamiltonian cycle) it can be stated that the TSP solution is complete, as can be seen in Figure 1. From here on this closed loop will further be described as a *tour*.

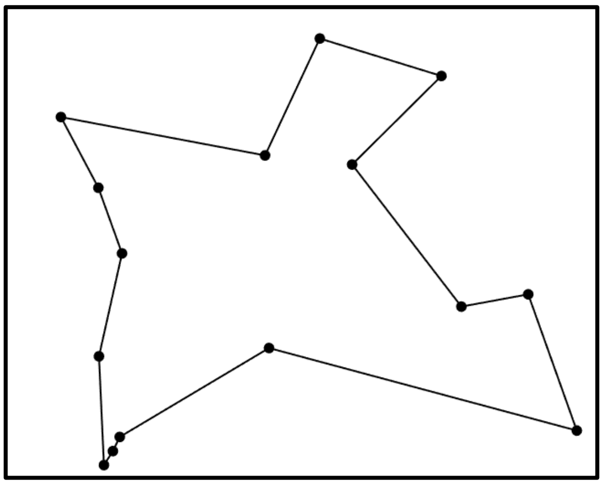


Figure 1: example of a TSP solution.

The path which the salesman must follow may seem obvious when considering a reduced number of towns. However, when hundreds or thousands of towns are considered, the solution is no longer trivial. For n towns, the number of possible combinations is

$$\frac{(n-1)!}{2}, \qquad (1)$$

which is nearly 2 million possible combinations for just 11 towns. It is obvious that a brute-force approach in which all possible combinations are tried is not viable and would require too much time. This is the reason why mathematical algorithms must be used to find an optimal solution.

The TSP belongs to the NP-complete type [2], short for nondeterministic polynomial-time complete. This can be described as any of a class of computational problems for which no optimal solution algorithm has been found that guarantees to offer a solution in polynomial time [3]. Therefore, when it comes to solving this optimization problem, a high complexity emerges. Due to this, in many cases there is no assurance that the shortest possible route or tour will be constructed, instead, a good quality solution is computed.

Depending on the number of points considered and the level of precision needed, different approaches should be used. These can be classified in two categories: exact and heuristic methods.

1.1. Exact Methods

Exact methods guarantee that an optimal solution is going to be obtained [4]. However, these methods require an extensive amount of computer time to reach the optimal solution. For this reason, exact methods are inapplicable in many cases and should only be utilized when a small number of points is considered. As Gilbert Laporte stated [5], some of the algorithms and formulations based on this method are:

- Integer linear programming formulations
- The assignment lower bound and related branch-and-bound algorithms
- The shortest spanning arborescence bound and a related algorithm
- The shortest spanning tree bound and related algorithms
- The 2-matching lower bound and related algorithms

1.1.1. Branch-and-cut method

One of the most used exact methods is the Concorde, which uses the branch and cut method to solve the TSP. The objective is to reduce the number of nodes in the search tree by generating tight bounds [6]. The upper bounds are found by implementing the branch-and-bound method, and the lower bounds by using the cutting planes method. This results in a more optimal procedure than using just the branch-and-bound method because by cutting planes, a smaller amount of information has to be stored to represent the search tree [7].

1.2. Heuristic Methods

Heuristic methods, as opposed to exact methods, offer a ‘good enough’ solution, but not precisely the optimal one. The advantage of these methods is that the computational time is significantly reduced [8], and hence satisfactory solutions can be reached for a substantial number of points in a short amount of time. The two main types of heuristic methods are Tour Construction Methods and Tour Improvement Methods, which will now be discussed.

1.2.1. Tour Construction Methods

This method consists in gradually building up a solution from scratch by starting with one node (city) and broadening the subtour by adding another vertex to it, one at a time. It consists of four steps [9]:

I. Founding of the initial small subtour (Subtour establishment rule)

This step consists of randomly choosing an initial node for the tour.

II. Next node choice (Selection rule)

From the available nodes that do not belong to the subtour that has already been constructed, the one with the minimum cost or distance to the last node in the subtour is selected.

III. Subtour Expansion

In this step, the node selected in step **II** can either be inserted or added to the existing subtour. For the insertion method it is chosen where to locate the new node, based on the cost of the subtour generated. The addition method, however, simply expands the existing subtour by placing the selected node after the last one present.

IV. Repetition of steps II and III until a closed loop is formed

Some examples of the most used Tour Construction Methods are [8]: arbitrary insertion, convex hull insertion, greatest angle insertion and ratio times difference insertion.

1.2.2. Tour Improvement Methods

Tour Improvement Methods begin with an initial solution to the TSP problem, to which a series of alterations are applied to improve it and therefore obtain a shorter distance or a smaller cost. This is an iterative process and can be applied several times until the quality of the solution obtained is suitable for its purpose.

Two different approaches can be identified for Tour Improvement, the local and the global search method [10]. Tour improvement using genetic algorithm is also commonly used.

1.2.3. Composite Methods

Composite Methods are just a combination of the two previously mentioned methods, Tour Construction and Tour Improvement. Firstly, an initial solution is computed using a tour construction method. Consecutively a tour improvement method is applied to the initial solution to obtain a more optimal one, for instance the 2-Opt algorithm.

1.3. Genetic Algorithms

This algorithm was developed by John Holland and De Jong in the 1960s and 1970s, and it basically consists in a method to solve optimization problems, based on Darwin's theory of evolution, replicating biological evolution [11]. This is done by using operators such as mutation, crossover and selection criteria. For the TSP a possible tiling is named individual, and this individual is defined by its genes (HV-Tree nodes described in 2.1.1.1). At each step the genes are recombined with operators to find a new combination. The cost of each is calculated and the best one is maintained [8].

1.4. Applications of TSP

The most popular application of the TSP is the one to which its definition refers, a salesman that has to visit N cities exactly once, and end in the initial spot. This is applicable to bus routes, package delivery routes and other types of logistic activities which require round trips with the minimum distance possible between stops. Though this are the most common, there are also many other applications related to different aspects of life. Some of them will now be explained [1][5]:

- **Genome sequencing.**

In genome sequencing, the objective is to reconstruct a missing or unknown fragment of the gene by determining the order in which sequences of nucleotides appear. This sequencing can be approached by using the TSP, in which the cities will be the local maps and the cost will be how probable it is that one map follows another one. So by combining biological and mathematical methods (TSP) it can be solved.

- **Drilling of printed circuit boards.**

In these boards, to connect one conductor layer to another one, holes of different diameters for different pins must be drilled. Each time a different diameter hole has to be drilled, the tool of the machine must be changed, which is very time consuming. The efficient way to approach the drilling is to first drill all the holes of one diameter, and then move on to the next diameter. Each of the holes can be considered a city in the TSP problem, and the cost is the amount of time required to go from one hole to another. The route which minimizes traveling time between the holes is computed using TSP.

- **Computer Wiring**

In some cases, pins which attach computer modules are linked using wires, in a way that exactly two wires are attached to each pin. It is desirable to find the shortest possible route, to minimize the length of the wire used and therefore the cost of the material.

- **Paper cutting**

When sheets have to be cut from a roll of paper by repeating a particular pattern, the objective is to minimize the waste of product. The most optimal way of cutting the sheet to make the most from the roll can be calculated by using TSP algorithms.

2. Methodology

As mentioned before, the TSP is an NP-complete type problem, but despite this, there are many existing algorithms that provide good quality solutions. The reason for developing a different algorithm, even if many others already exist, is to investigate a different approach in the method used to reach a valid solution. The particularity of this method is that dyadic tiling is used to partition the randomly generated points, forming small clusters of points (depicted in Figure 2 (b)). Each of these clusters are composed of the points belonging to the same tile. A representative point of each cluster, the geometrical barycenter, is generated and an initial TSP solution is found, connecting the barycenters exclusively, not the randomly generated points. Also, the midpoints of the segments of the solution are defined. Individual tour solutions with different endpoints are generated for each cluster, which are subsequently connected to one another through the previously defined midpoints. To finalize, the barycenters and the midpoints are excluded from the created TSP solution, and the 2-opt method is applied to improve the initial solution.

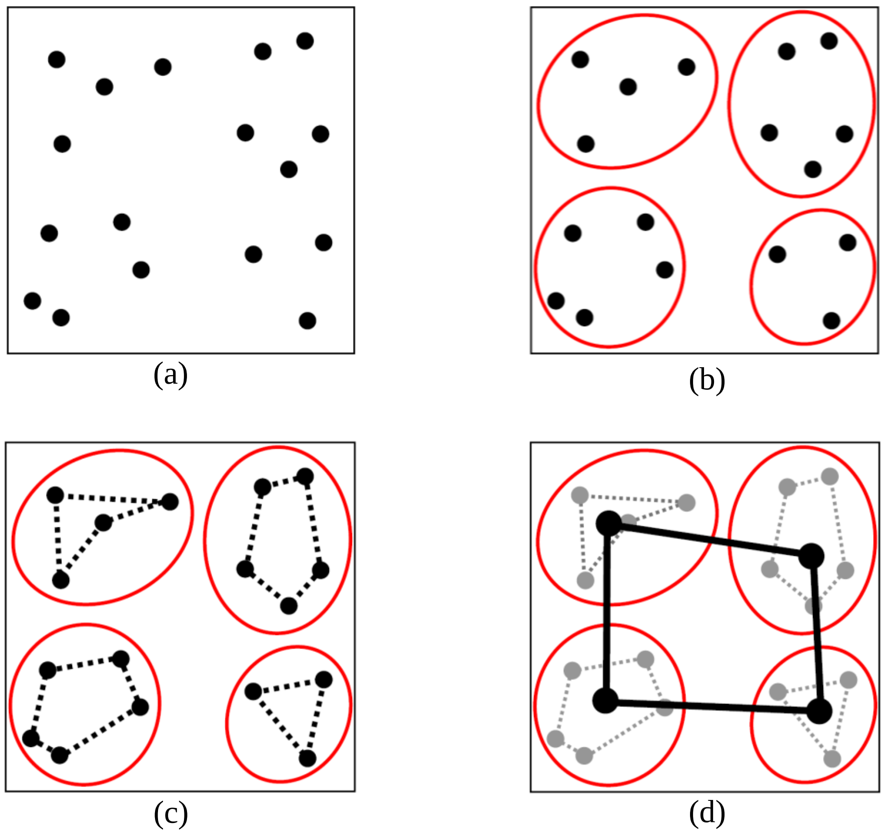


Figure 2: (a) A set of n points, (b) first clustered into disjoint sets. (c) A subsolution is obtained for each cluster and (d) these are concatenated. ***(Source: [8])***

2.1. Description of the algorithm

In this section the particularities of the method that has been used to solve the TSP will be discussed.

2.1.1. Dyadic Tiling

Tiling, also known as tessellation, is the process of completely covering a given shape with other smaller plane figures, avoiding gaps and any overlapping [8]. The objective of this thesis is to explore the resolution of the TSP using a particular type of tessellation, dyadic tiling.

In dyadic tiling, the starting domain is the unit square, to which n successive bisections are applied. The number of successive bisections by vertical or horizontal cuts, n, is known as the order of the tile. The vertical or horizontal cuts are equally probable of occurring independent of the others, and they are responsible for cutting the existing dyadic rectangle exactly in half [12].

The dyadic tiling can be defined as [13]:

$$R(a,b,s,t) = [a2^{-s}, (a+1)2^{-s}] \times [b2^{-t}, (b+1)2^{-t}], \tag{2}$$

where the following conditions must be satisfied:

$$\begin{gathered} 0 \leq a < 2s, \\ 0 \leq b < 2t. \end{gathered} \tag{3}$$

The number of dyadic rectangles into which the unit square will be partitioned is, being n the order of the tile:

$$2^{n}, \tag{4}$$

and the area of each of these dyadic rectangles being:

$$2^{-n}. \tag{5}$$

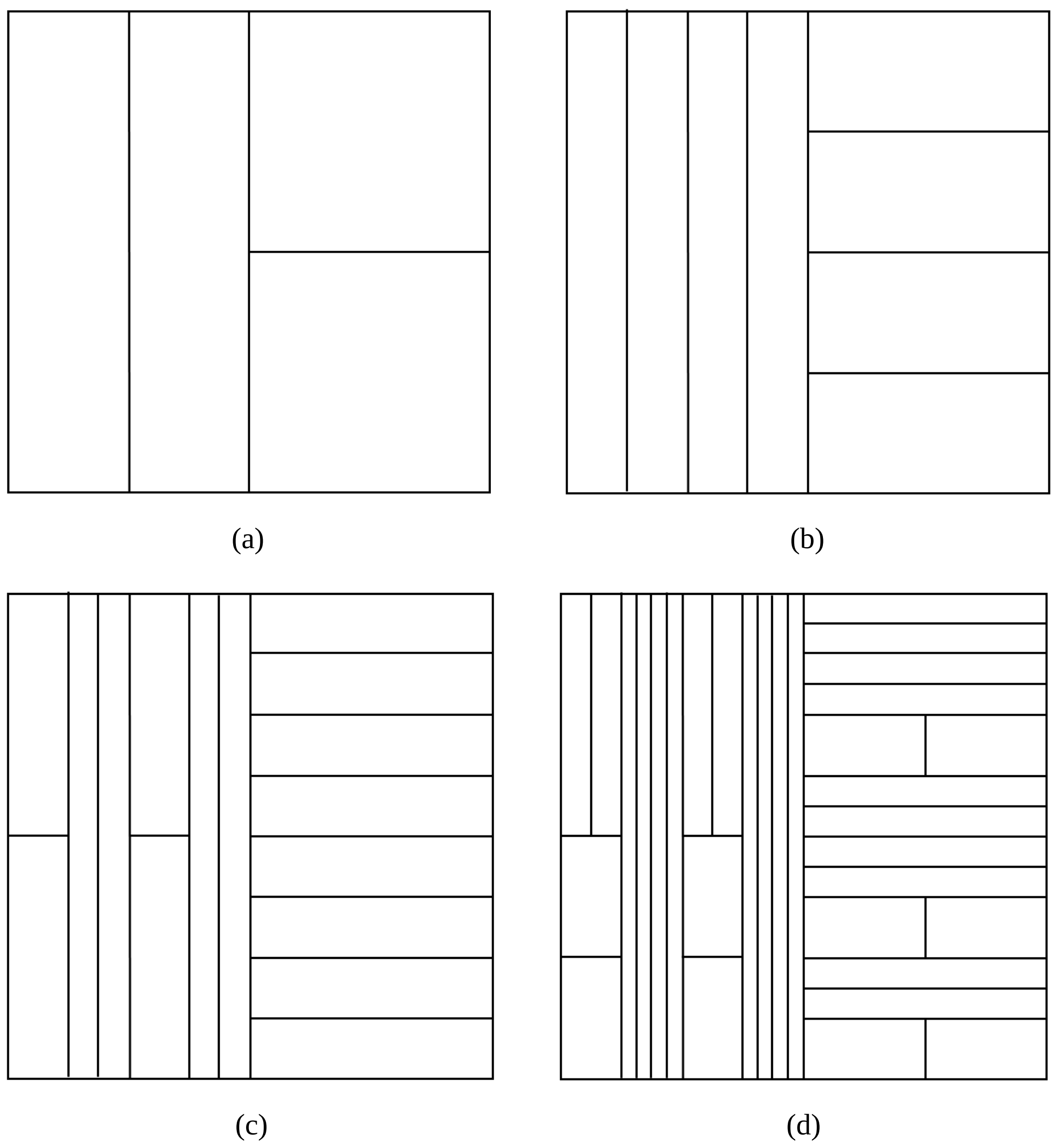


Figure 3: Dyadic tilings of (a) 2nd order, (b) 3rd order, (c) 4th order and (d) 5th order.

We can also define the type of cut that has taken place [13], either vertical, as can be seen in Figure 3 (a) or horizontal. The case where both a horizontal and a vertical cut have been done can also arise.

These cuts that are performed to the rectangles can be described using binary trees labeled with an *'H'* in the case of a horizontal cut and with a *'V'* for a vertical cut.

2.1.1.1. HV-Tree Generation

The HV-Tree that is used in this algorithm is a particular case of the general term binary tree. The term *'binary'* refers to the fact that each node has exactly two children, labeled as the left child and right child. It is called HV-Tree because each node has the label *'H'* or *'V'* to determine the type of cut that has been applied to the tile above, horizontal or vertical, respectively. The height (layers) of the HV-Tree will be equal to the order of the tiling, n, with the number of nodes present being:

$$2^n - 1. \tag{6}$$

In Figure 4 we can see the HV-Tree corresponding to the tiling in Figure 3 (a), which has an initial vertical cut. After this, another vertical cut is performed to the left child, and a horizontal cut is performed to the right child. No further cuts are executed because $n = 2$, which means it is a 2nd order tiling and that the height of the HV-Tree is 2. The number of rectangles generated is 4, according to equation (4). The number of nodes in the HV-Tree is 3 according to equation (6).

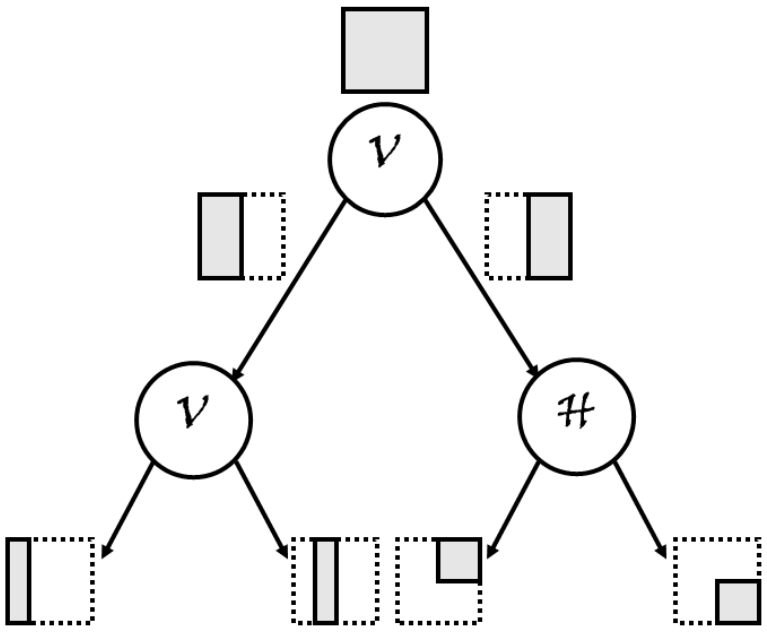


Figure 4: Illustration of the subdivision process of a 2nd order dyadic tiling corresponding to the tiling in Figure 3 (a).

The user can decide the order of the tiling, but the partitioning, whether vertical or horizontal is randomly generated, with a uniform distribution following the recursive construction [13].

In this method if the parent is labeled $'V'$, its children can be of any type; both $'V'$, both $'H'$, or one of each, and can be continued recursively. On the contrary if the parent is labeled $'H'$ its children cannot be both $'V'$, creating a dependency. This is done because each tiling has the same chance of appearing and otherwise the tiles to which most HV-Trees correspond would be overrepresented.

The node types of the HV-Tree will therefore be:

- V if the parent is labeled V, independently of its children
- H_{xy} if the node is a root (not a leaf), where x, y correspond to its children
- H_{HH} if the node is labeled H and has no children.

The probability that a particular random tiling will be generated is [13],

$$P_n = \frac{{A_{n-1}}^2}{A_n} = \frac{1}{2 - {p_{n-1}}^2} \tag{7}$$

with A_n being the total number of trees of all types $(V, H_{HH}, H_{HV}, H_{VH})$.

Therefore, the number of trees of each type can be defined as follows:

$$\begin{aligned}
&V: && {A^2}_{n-1} = p_n A_n, \\
&H_{HH}: && (A_{n-1} - {A^2}_{n-2})^2 = p_n(1 - p_{n-1})^2 A_n, \\
&H_{HV}: && {A^2}_{n-2}(A_{n-1} - {A^2}_{n-2}) = p_n p_{n-1}(1 - p_{n-1}) A_n, \\
&H_{VH}: && {A^2}_{n-2}(A_{n-1} - {A^2}_{n-2}) = p_n p_{n-1}(1 - p_{n-1}) A_n.
\end{aligned} \tag{8}$$

The 4-tuples (2) of each generated tile can be calculated knowing if the cut has been of $V\ or\ H$ type and if the child is $left\ or\ right$ (this refers to the position of the children from the same parent in the HV-Tree) [8]:

$$\begin{aligned}
&HL(a, b, s, t) = (a, 2b, s, t + 1), \\
&HR(a, b, s, t) = (a, 2b + 1, s, t + 1), \\
&VL(a, b, s, t) = (2a, b, s + 1, t), \\
&VR(a, b, s, t) = (2a + 1, b, s + 1, t).
\end{aligned} \tag{9}$$

2.1.1.2. Clustering

After creating the dyadic tile, the next step is to plot the points which are arbitrary and can be either specified or randomly generated. These points must be sorted into their corresponding tile, forming small clusters. For each non-empty tile, a representative point, which is the barycenter, is plotted.

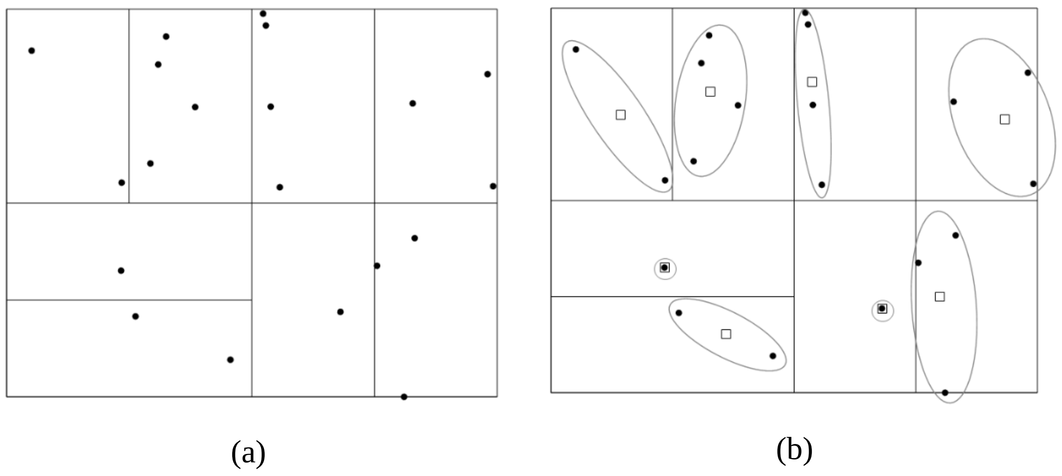


Figure 5: (a) Dyadic tiling with points, (b) clusters and barycenters.

2.1.2. Hierarchical Method to obtain the solution

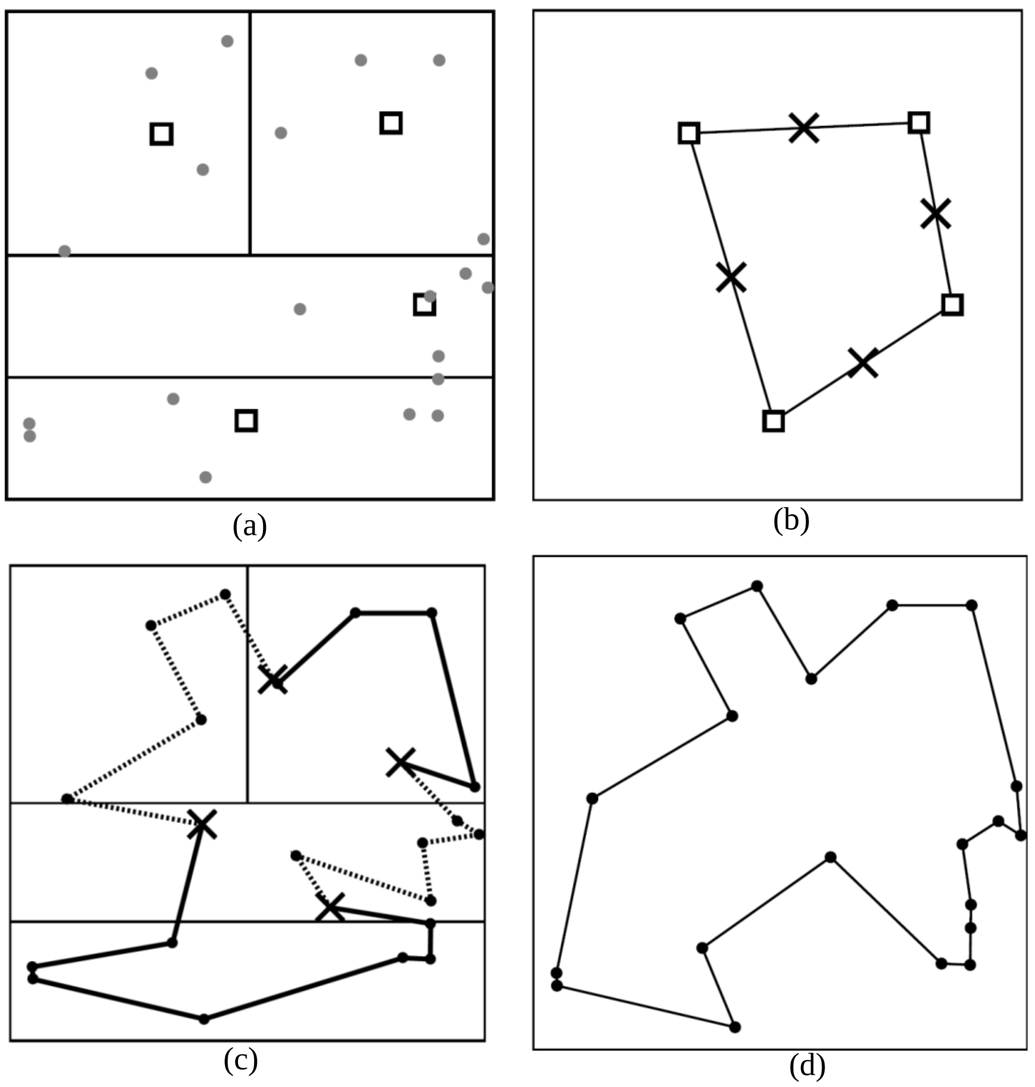


Figure 6: (a) Partitioning the points, (b) "coarse" TSP solution for the barycenters and placement of the midpoints, (c) minimum tours are calculated between midpoints and (d) the final solution using 2-opt. (Source [8])

Figure 6 is an example of a solution for the TSP problem, using a hierarchical method with one partitioning. Considering a set of points *X*, the method that will be applied to reach an optimal solution will now be explained.

(a) Tiling, point and barycenter generation

The unit square is partitioned using dyadic tiling resulting in the tiling, as the one in Figure 6 (a), named T. A set of random points (denoted in ●

Figure 6 with a), X, is generated in the tile, forming clusters of points. Each of these clusters corresponds to the group of points belonging to the same tile. A representative point, the geometrical barycenter is calculated and pl▣ted (represented by a).

(b) TSP solution for barycenters

A rough solution for the TSP problem is found but considering only the barycenters. By representing each cluster of points with their corresponding barycenter, the number of nodes that are considered for the rough initial solution is greatly reduced and the solution can be found with greater ease and speed. The midpoints of the line segments that join one barycenter to another in the solution are represented in

Figure 6 with a **x**.

(c) Minimum tours

Each barycenter is now dropped from the “coarse” solution and substituted by the cluster of points it represents, and the two midpoints to which it was connected. For each tile, a rough solution for the TSP with different starting and end points is computed. This is done in a way that the solution begins with one of the midpoints that belongs to that tile, then it goes through all the points in that particular tile and finally ends in the other midpoint belonging to that tile. The process is repeated for each tile and these subtours are connected to one another forming a closed tour going through all the points and midpoints.

(d) Final solution

To obtain the final solution, the midpoints must be deleted from the tour because they do not belong to the original set of points. This is done by simply joining the point in the solution that is before the midpoint with the point after it, for all the midpoints.

Finally, a tour improvement method is applied to this solution to make it better. In this case 2-opt has been implemented to delete the intersections between points and therefore reduce the tour length.

The method described above is only efficient for a limited number of points, a few hundred of points at maximum. If a considerable number of points is contemplated, the tiling will have to be of a greater order to avoid clusters with too many nodes in them. This leads to a great number of barycenters, which may result in two problems. The first one being that a poor-quality coarse solution in step (b) described above may be obtained, and the second one that the coarse problem may still be too complicated and therefore very time consuming to reach a good quality solution. The solution for this is to use multiple partitioning by forming new clusters with the barycenters ◘ and repeating the steps previously mentioned. The process is briefly explained in Figure 7, but it will not be explained in further detail because the purpose of the paper is to solve the TSP using a single partition.

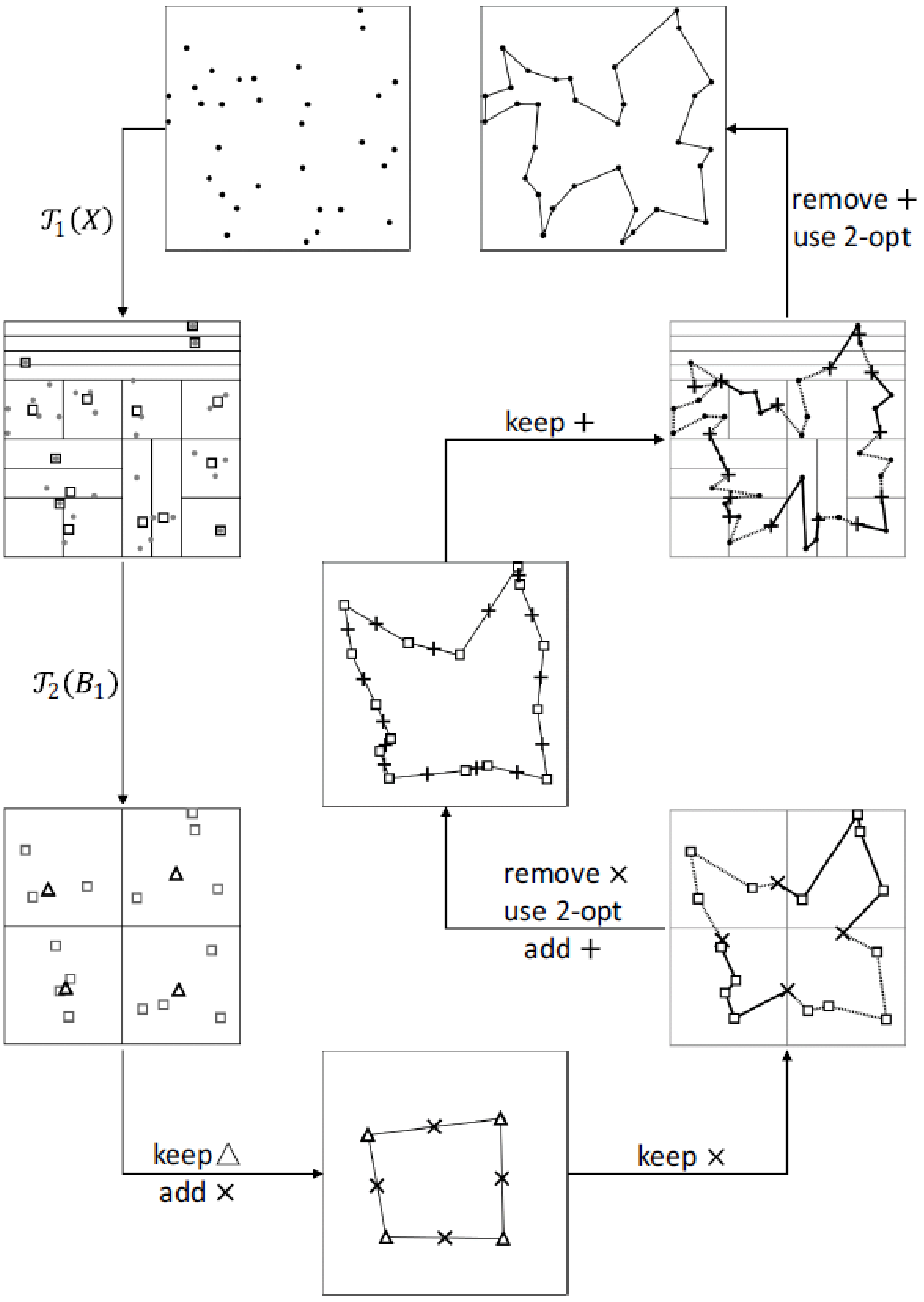


Figure 7: Method for solving the TSP using multiple partitioning. ***(Source [8])***

2.1.3. Tour Improvement: 2-Opt heuristic

The 2-Opt tour improvement method is a very simple, yet efficient, algorithm that achieves to yield a better-quality solution for the TSP. It is usually applied to an initial solution obtained from a tour construction method.

The way of working of this algorithm is by exchanging the order of two edges of the solution by another two proximal edges, in such a way that the combination will result in a tour again [14]. The resulting distance between the edges before and after the considered exchange is evaluated. Whichever combination offers the shorter route is the one kept. This step is repeated throughout all the edges, which are replaced as long as a better solution is obtained. Once no improvement is obtained from further exchanges the 2-Opt algorithm is stopped. If the existing tour cannot be improved by 2-Opt heuristic, it receives the name of 2-Optimal.

In Figure 8 a very simple example of the 2-Opt algorithm is described. In this case, in Figure 8 (a) we can see that there is an intersection between two line segments, which make the tour longer. The 2-Opt algorithm can detect this imperfection because the distance between edges after introducing the edge swap is smaller than that of the original line segments. This can be clearly seen in Figure 8 (b) where the 2-Opt has already been successfully applied.

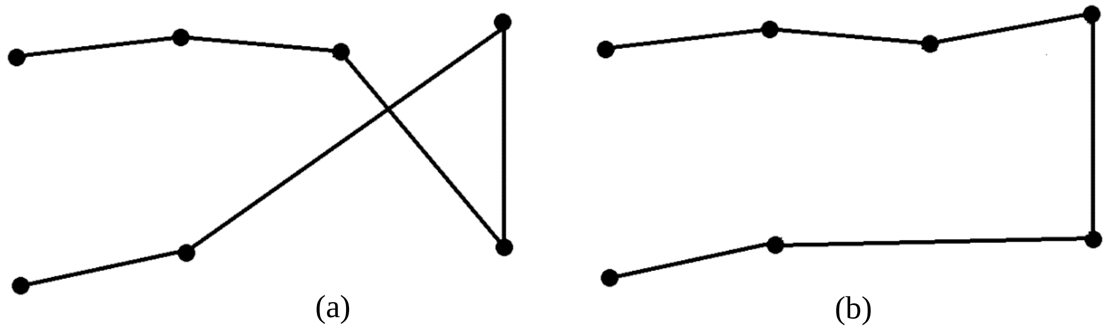


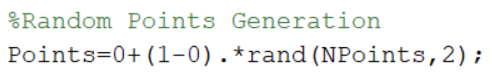
Figure 8: TSP Solution (a) before applying 2-opt and (b) after applying 2-opt.

2.2. Implementation of the algorithm to Matlab

The theoretical explanation of the method of dyadic tiling to solve the TSP has already been accomplished. From this point forward, the intention of the paper is to explain how this theory, considering only the case of single partitioning, has been applied and how it has been implemented to the computing platform *Matlab*.

2.2.1. Random Point and HV-Tree Generation

The code starts with the random generation of a specified number of points, for testing purposes, with x-y coordinates belonging to the range [0,0] × [1,1], the unit square, for simplicity reasons. However, if desired, a set of specific coordinates could also be introduced or imported from a database.

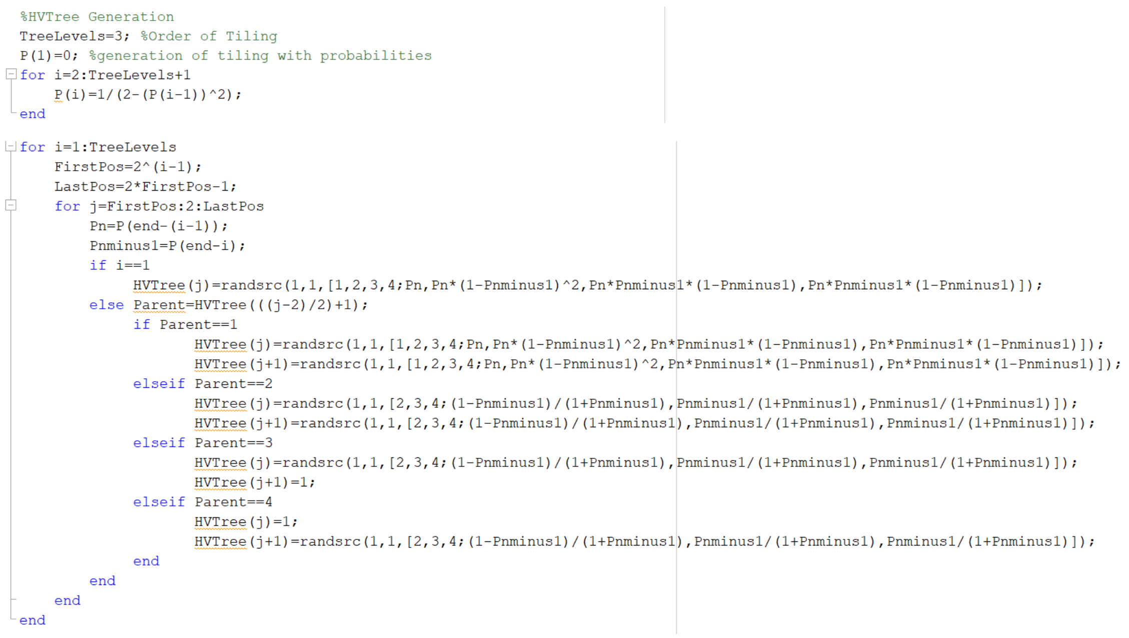


```
%Random Points Generation
Points=0+(1-0).*rand(NPoints,2);
```

Figure 9: Random point generation in Matlab.

The next step is the generation of the HV-Tree by firstly indicating the number of tree levels, this is the order of the desired tiling. This number will depend on the number of points considered for the TSP. The greater the number of points, the greater the number of tiles. An excessive number of tiles also must be avoided because this will result in too many barycenters, so a compromise between the average size of the clusters and the number of barycenters must be found.

The tessellation process includes either vertical or horizontal cuts of the unit square. Considering the conditions mentioned in 2.1.1.1, that the HV-Tree is constructed with a uniform distribution recursively. These cuts are defined by probability, in such a way that all types of cuts have the same probability of occurring. The probability equation followed to generate these cuts is (7).



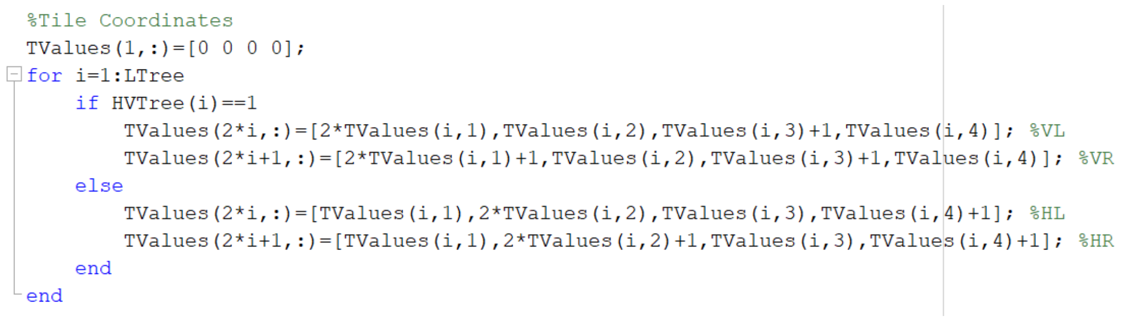
```
%HVTree Generation
TreeLevels=3; %Order of Tiling
P(1)=0; %generation of tiling with probabilities
for i=2:TreeLevels+1
    P(i)=1/(2-(P(i-1))^2);
end

for i=1:TreeLevels
    FirstPos=2^(i-1);
    LastPos=2*FirstPos-1;
    for j=FirstPos:2:LastPos
        Pn=P(end-(i-1));
        Pnminus1=P(end-i);
        if i==1
            HVTree(j)=randsrc(1,1,[1,2,3,4;Pn,Pn*(1-Pnminus1)^2,Pn*Pnminus1*(1-Pnminus1),Pn*Pnminus1*(1-Pnminus1)]);
        else Parent=HVTree(((j-2)/2)+1);
            if Parent==1
                HVTree(j)=randsrc(1,1,[1,2,3,4;Pn,Pn*(1-Pnminus1)^2,Pn*Pnminus1*(1-Pnminus1),Pn*Pnminus1*(1-Pnminus1)]);
                HVTree(j+1)=randsrc(1,1,[1,2,3,4;Pn,Pn*(1-Pnminus1)^2,Pn*Pnminus1*(1-Pnminus1),Pn*Pnminus1*(1-Pnminus1)]);
            elseif Parent==2
                HVTree(j)=randsrc(1,1,[2,3,4;(1-Pnminus1)/(1+Pnminus1),Pnminus1/(1+Pnminus1),Pnminus1/(1+Pnminus1)]);
                HVTree(j+1)=randsrc(1,1,[2,3,4;(1-Pnminus1)/(1+Pnminus1),Pnminus1/(1+Pnminus1),Pnminus1/(1+Pnminus1)]);
            elseif Parent==3
                HVTree(j)=randsrc(1,1,[2,3,4;(1-Pnminus1)/(1+Pnminus1),Pnminus1/(1+Pnminus1),Pnminus1/(1+Pnminus1)]);
                HVTree(j+1)=1;
            elseif Parent==4
                HVTree(j)=1;
                HVTree(j+1)=randsrc(1,1,[2,3,4;(1-Pnminus1)/(1+Pnminus1),Pnminus1/(1+Pnminus1),Pnminus1/(1+Pnminus1)]);
            end
        end
    end
end
```

Figure 10: HV-Tree generation in Matlab.

2.2.2. Tile Coordinates

Once the tiles have been defined and the unit square has been completely partitioned, the coordinates of each of these dyadic rectangles must be defined. This is necessary so that afterwards the points can be grouped into their corresponding tile.

Whilst the HV-Tree is being generated, a vector is also created that includes the type of cut that has taken place, whether vertical or horizontal, and whether it is the left or the right child. With the information in the previously mentioned vector and using the equations (9) , the 4-tuples of each dyadic tile can be calculated.



```
%Tile Coordinates
TValues(1,:)=[0 0 0 0];
for i=1:LTree
    if HVTree(i)==1
        TValues(2*i,:)=[2*TValues(i,1),TValues(i,2),TValues(i,3)+1,TValues(i,4)]; %VL
        TValues(2*i+1,:)=[2*TValues(i,1)+1,TValues(i,2),TValues(i,3)+1,TValues(i,4)]; %VR
    else
        TValues(2*i,:)=[TValues(i,1),2*TValues(i,2),TValues(i,3),TValues(i,4)+1]; %HL
        TValues(2*i+1,:)=[TValues(i,1),2*TValues(i,2)+1,TValues(i,3),TValues(i,4)+1]; %HR
    end
end
```

Figure 11: Generation of the values of the 4-tuples from the type of cut.

With the 4-tuples generated for the last level of the HV-Tree, which are the ones corresponding to the final tessellation, we can obtain the position of each tile. The 4 corners of the tile are defined by using the $x_{min}, x_{max}, y_{min}$ and y_{max} values, as can be seen in Figure 12.

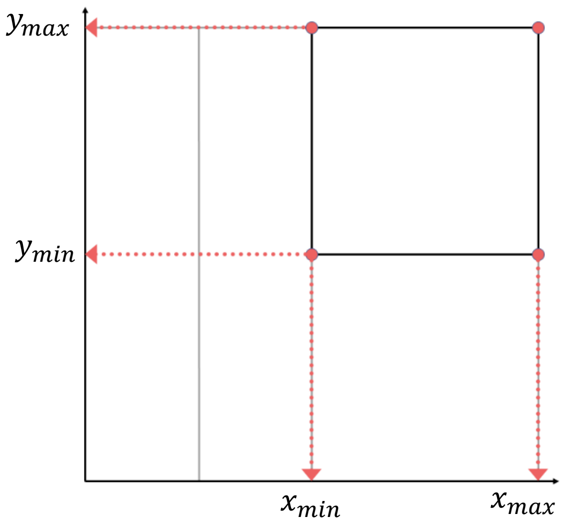
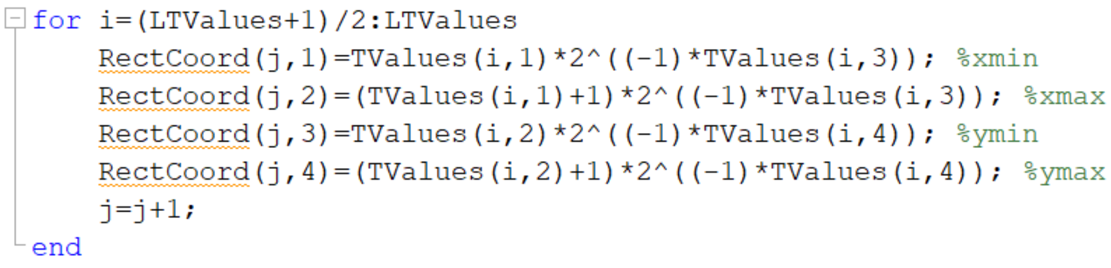


Figure 12: definition of the coordinates of a tile.

These values are defined by substituting the values generated in (2) for each tile, into the equations (10):

$$
\begin{aligned}
x_{min} &= a2^{-s}, \\
x_{max} &= (a+1)2^{-s}, \\
y_{min} &= b2^{-t}, \\
y_{max} &= (b+1)2^{-t}.
\end{aligned}
\qquad (10)
$$



```
for i=(LTValues+1)/2:LTValues
    RectCoord(j,1)=TValues(i,1)*2^((-1)*TValues(i,3)); %xmin
    RectCoord(j,2)=(TValues(i,1)+1)*2^((-1)*TValues(i,3)); %xmax
    RectCoord(j,3)=TValues(i,2)*2^((-1)*TValues(i,4)); %ymin
    RectCoord(j,4)=(TValues(i,2)+1)*2^((-1)*TValues(i,4)); %ymax
    j=j+1;
end
```

Figure 13: Tile coordinates generation in Matlab.

This process is repeated until every tile in the unit square has its corners defined. These rectangles are all plotted and using the *'hold on'* command in Matlab, the points are plotted on the same graph. This is depicted in Figure 14.

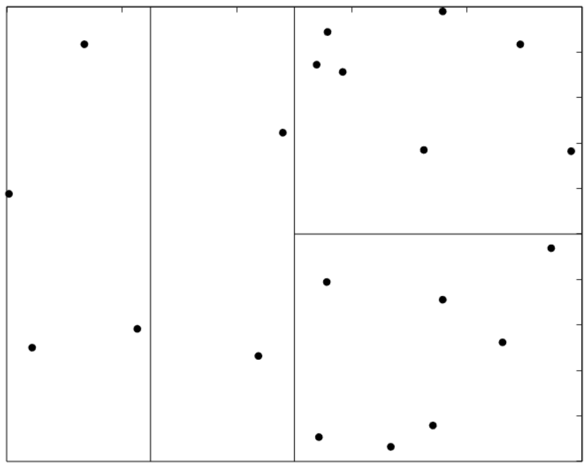


Figure 14: plotting of tiles and points.

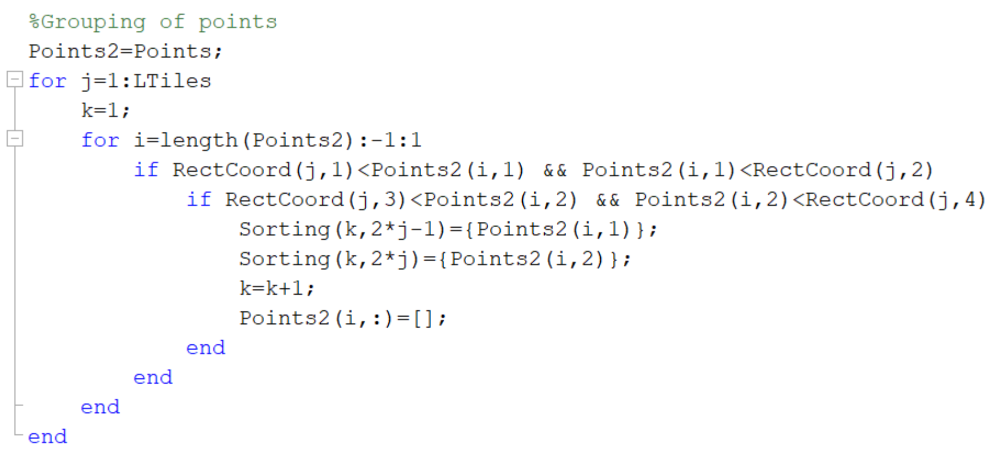
The next step is to sort the points into the corresponding tile. Although visually we can see in Figure 14 to which tile each point belongs, this is just because the is a reduced number of points and tiles. This process must be programmed so that the code in Matlab provides us with this information automatically.

2.2.3. Grouping of Points

The way of proceeding to sort each point into its corresponding tile is very simple. The coordinates of the corners of each tile are stored in a different row of a matrix, where each column represents each of the values described in (10). Taking one row at a time (this corresponds to one tile at a time), a loop is created in which each point is tested to see if it belongs to that particular tile. This is done by checking if the x-coordinate of the point is found between the x_{min} and x_{max} coordinates of the considered time, and respectively for the y-coordinates.

If both conditions are met, the point belongs to the tile and it must be stored. However, it cannot be stored in matrix form as has been done before for the rest of the data. A cell structure must be used, because not all columns will have the same number of elements in them. The reason for this is that not all tiles will necessarily have the same number of points that belong to them. In fact, the case where there are no points belonging to a tile can also arise.

To avoid going each time through all the points to see if they belong to a particular tile, once a point has been classified into a tile it is deleted from the point vector. This makes the process more efficient because less points will have to be tested each time.



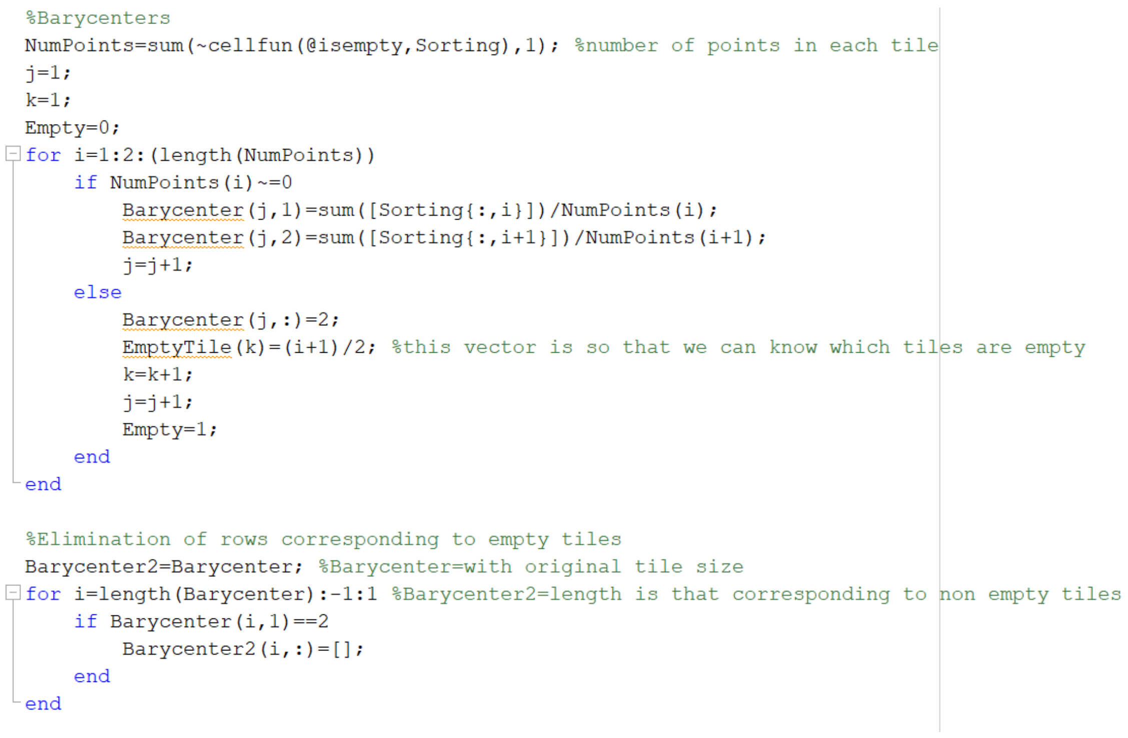
```
%Grouping of points
Points2=Points;
for j=1:LTiles
    k=1;
    for i=length(Points2):-1:1
        if RectCoord(j,1)<Points2(i,1) && Points2(i,1)<RectCoord(j,2)
            if RectCoord(j,3)<Points2(i,2) && Points2(i,2)<RectCoord(j,4)
                Sorting(k,2*j-1)={Points2(i,1)};
                Sorting(k,2*j)={Points2(i,2)};
                k=k+1;
                Points2(i,:)=[];
            end
        end
    end
end
```

Figure 15: Point classification into the corresponding tile in Matlab.

2.2.4. TSP Solution for Barycenters

After classifying the points into the corresponding tile, the geometrical center of the points in each tile is represented. This point has been named before as barycenter.

A special consideration must be made at this point, for the tiles that are empty. The rows corresponding to the empty tiles should not be considered for the barycenter calculation since it does not exist. To avoid problems further on in the code the rows corresponding to the empty tiles should be eliminated.



```
%Barycenters
NumPoints=sum(~cellfun(@isempty,Sorting),1); %number of points in each tile
j=1;
k=1;
Empty=0;
for i=1:2:(length(NumPoints))
    if NumPoints(i)~=0
        Barycenter(j,1)=sum([Sorting{:,i}])/NumPoints(i);
        Barycenter(j,2)=sum([Sorting{:,i+1}])/NumPoints(i+1);
        j=j+1;
    else
        Barycenter(j,:)=2;
        EmptyTile(k)=(i+1)/2; %this vector is so that we can know which tiles are empty
        k=k+1;
        j=j+1;
        Empty=1;
    end
end

%Elimination of rows corresponding to empty tiles
Barycenter2=Barycenter; %Barycenter=with original tile size
for i=length(Barycenter):-1:1 %Barycenter2=length is that corresponding to non empty tiles
    if Barycenter(i,1)==2
        Barycenter2(i,:)=[];
    end
end
```

Figure 16: Barycenter generation in Matlab.

At this point the barycenters of each non-empty tile are plotted joint with the tiles and the points that are being considered for the TSP solution.

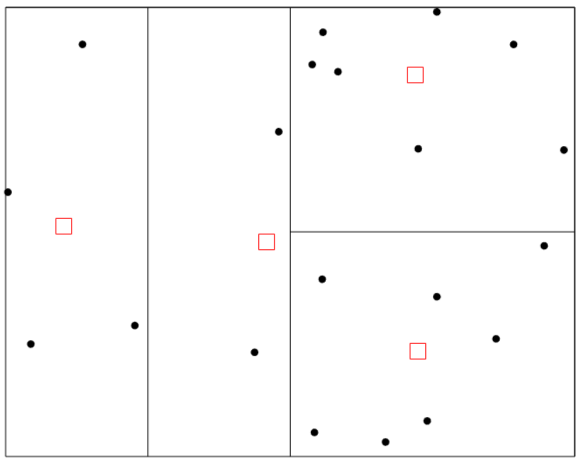


Figure 17: plotting of barycenters of each tile, ● for the points and □ for the barycenters.

Now that the barycenters have been created, a TSP solution must be found considering just these, and not the original points. At first the *Greedy method* was implemented into the code. This method basically starts the tour at a given point and calculates the distance from that point to the rest of the points that are available (that do not already belong to the constructed solution). The point that is nearest is selected and the process is repeated until all the points have been included in the tour. Finally, the last point and the first one are joined together. This method is not the most efficient one but yields a good enough solution when few points are considered, with the advantage that it is easy and quick to implement.

In the end a different approach was taken to obtain a more efficient initial *'coarse'* solution for the barycenters. An already existing TSP solver was implemented into the code. This function implemented is called TSP_GA *'Traveling Salesman Problem (TSP) Genetic Algorithm (GA)'* [15]. This code uses the previously explained in 1.3 genetic algorithm to find a route going through all the barycenters.

```
% %TSP Solver for Barycenters
xy=[Barycenter2];
userConfig=struct('xy',xy);
resultBarycenter=tsp_ga(userConfig);
OrderBarycenters=resultBarycenter.optRoute;
```

Figure 18: TSP solver for barycenters.

The data passed on to the already existing solver, is the coordinates of the barycenters, and the order of the barycenters forming the tour is returned as the solution, see Figure 19.

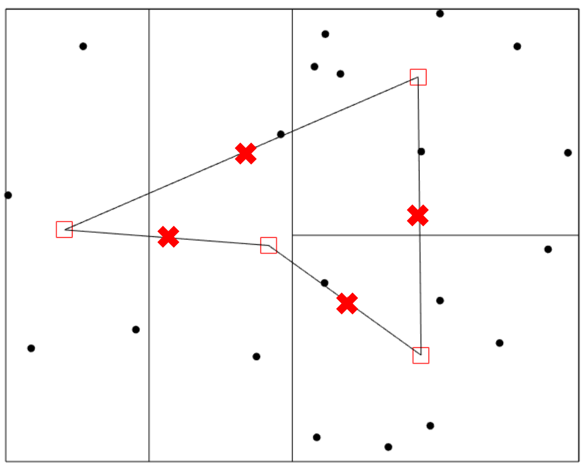


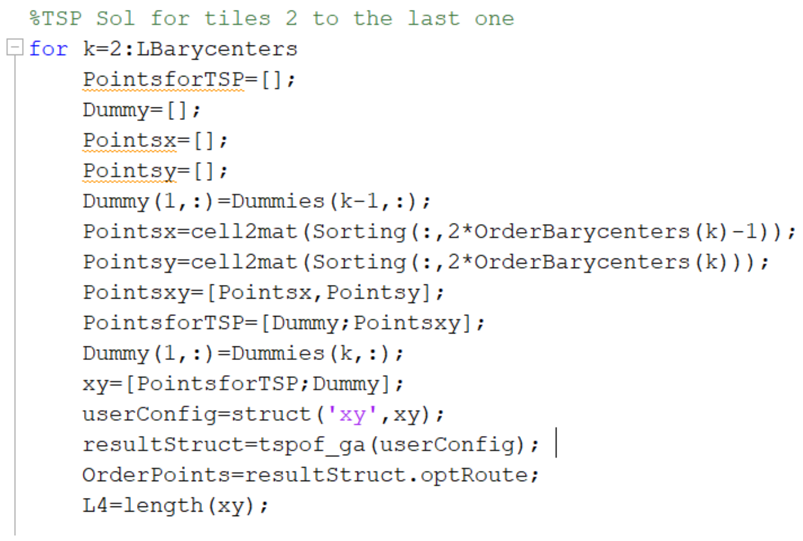
Figure 19: TSP solution for barycenters □ and midpoints ✖.

The next step is the plotting of the mid points for each line segment, which are represented in Figure 19.

2.2.5. TSP Solution for Points

At this point the barycenters are dropped from the solution, and a minimum tour is found for the cluster of points that each barycenter represented, linking them to their two corresponding midpoints.

Initially the ‘*Greedy Method’* was also going to be used, but finally it was decided to implement another existing TSP solver, because this would yield a better solution. However, this time a different function than the previously mentioned TSP_GA had to be implemented. The reason for this is that the normal TSP solver tries to find the shortest round trip. In this step however, we need the tour to start at one midpoint and end at the next one, so the first and last points are different. To overcome this inconvenience a different function was used, TSPOF_GA, *‘Fixed Open Traveling Salesman Problem (TSP) Genetic Algorithm (GA)’* [16].



```
%TSP Sol for tiles 2 to the last one
for k=2:LBarycenters
    PointsforTSP=[];
    Dummy=[];
    Pointsx=[];
    Pointsy=[];
    Dummy(1,:)=Dummies(k-1,:);
    Pointsx=cell2mat(Sorting(:,2*OrderBarycenters(k)-1));
    Pointsy=cell2mat(Sorting(:,2*OrderBarycenters(k)));
    Pointsxy=[Pointsx,Pointsy];
    PointsforTSP=[Dummy;Pointsxy];
    Dummy(1,:)=Dummies(k,:);
    xy=[PointsforTSP;Dummy];
    userConfig=struct('xy',xy);
    resultStruct=tspof_ga(userConfig);
    OrderPoints=resultStruct.optRoute;
    L4=length(xy);
```

Figure 20: Minimum tour construction between midpoints.

The process shown in Figure 20 is repeated for each cluster of points that the barycenters represented. The TSP solver returns the order of the points forming the tour, so these can be plotted.

It is important to remember to delete the midpoints that were previously generated from the overall solution because they have been created to help construct the solution but are not part of the original points considered for the solution.

At this point an approximate solution has been found for the original set of points by using dyadic tiling and a genetic algorithm TSP solver.

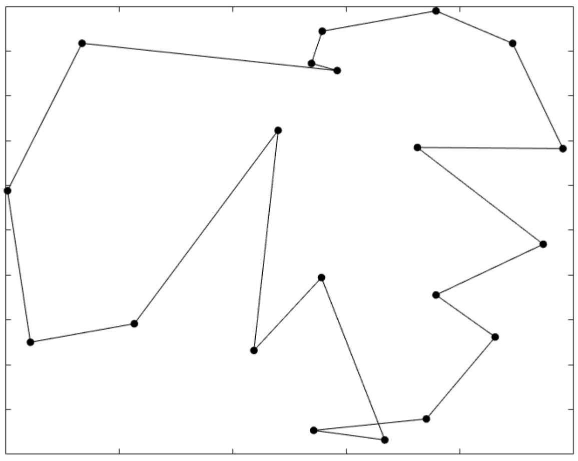
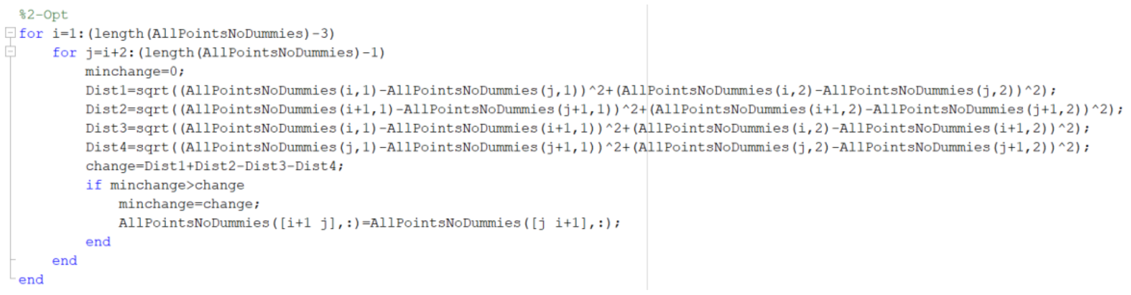


Figure 21: TSP solution for points before 2-Opt.

2.2.6. 2-Opt Implementation

This is the last step of the developed code. Although an approximate solution has been found, it can be improved by using a Tour Improvement Method (1.2.2) based on the obtained solution. In Figure 22 it can be seen that 2-Opt is a simple method to implement, in which the distances of two edges are computed, as well as the distances corresponding to the edges that would result if they were switched. If the distance of the switched edges is in fact shorter, these are swapped, otherwise they remain unchanged. By doing this the number of intersections present in the solution is reduced, and therefore the total length of the tour decreases.



```
%2-Opt
for i=1:(length(AllPointsNoDummies)-3)
    for j=i+2:(length(AllPointsNoDummies)-1)
        minchange=0;
        Dist1=sqrt((AllPointsNoDummies(i,1)-AllPointsNoDummies(j,1))^2+(AllPointsNoDummies(i,2)-AllPointsNoDummies(j,2))^2);
        Dist2=sqrt((AllPointsNoDummies(i+1,1)-AllPointsNoDummies(j+1,1))^2+(AllPointsNoDummies(i+1,2)-AllPointsNoDummies(j+1,2))^2);
        Dist3=sqrt((AllPointsNoDummies(i,1)-AllPointsNoDummies(i+1,1))^2+(AllPointsNoDummies(i,2)-AllPointsNoDummies(i+1,2))^2);
        Dist4=sqrt((AllPointsNoDummies(j,1)-AllPointsNoDummies(j+1,1))^2+(AllPointsNoDummies(j,2)-AllPointsNoDummies(j+1,2))^2);
        change=Dist1+Dist2-Dist3-Dist4;
        if minchange>change
            minchange=change;
            AllPointsNoDummies([i+1 j],:)=AllPointsNoDummies([j i+1],:);
        end
    end
end
```

Figure 22: 2-Opt implementation.

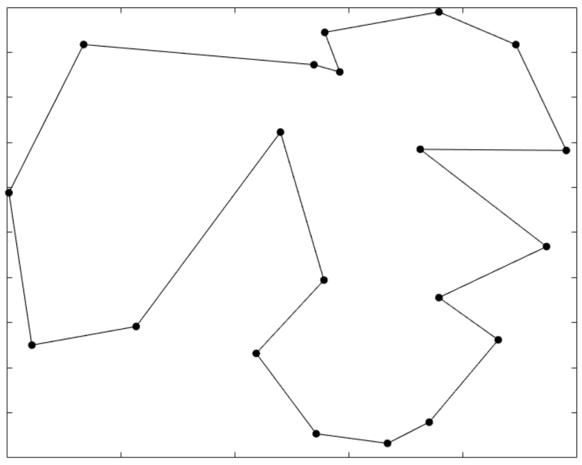


Figure 23: TSP solution for points after 2-Opt implementation.

By comparing Figure 21 and Figure 23 we can observe that indeed, the use of a tour improvement method as is 2-Opt has yielded a better quality solution than the one initially obtained.

3. Results

In this section the results obtained by testing the code with different sets of points and tilings will be presented. Depending on the number of points N, the order of tiling chosen varies to guarantee that the average estimated number of points in each tile is less than 10. The number of iterations performed for each set of points has been 100.

The following tests have been performed:

Number of Points	Order of tiling	Average number of points per cluster	
25	3rd	$\frac{25}{2^3} = 3.125$	3 or 4 points
50	4th	$\frac{50}{2^4} = 3.125$	3 or 4 points
75	4th	$\frac{75}{2^4} = 4.6875$	4 or 5 points
100	4th	$\frac{100}{2^4} = 6.25$	6 or 7 points

Table 1: **characteristics of tests performed.**

3.1. *Mean solution lengths and standard deviations*

The developed code works correctly and ensures the objective of obtaining a solution for given number of points. However, the quality of the solution obtained must also be evaluated. A possible way to do this is by using the approximation formula stated by Beardwood [17], in which the expected optimal tour length is:

$$\lim_{n\to\infty} \frac{L_{min}}{\sqrt{n}} = \beta$$
$$L_{min} = \beta\sqrt{n} + C \qquad (11)$$

The values for the constants in equations (11) that will be used are taken from [18] and are $\beta = 0.71$ and $C = 0.63$.

3.1.1. 25 Points

For $N = 25$ an example is shown in Figure 24 of the different steps undergone by the code to obtain the solution.

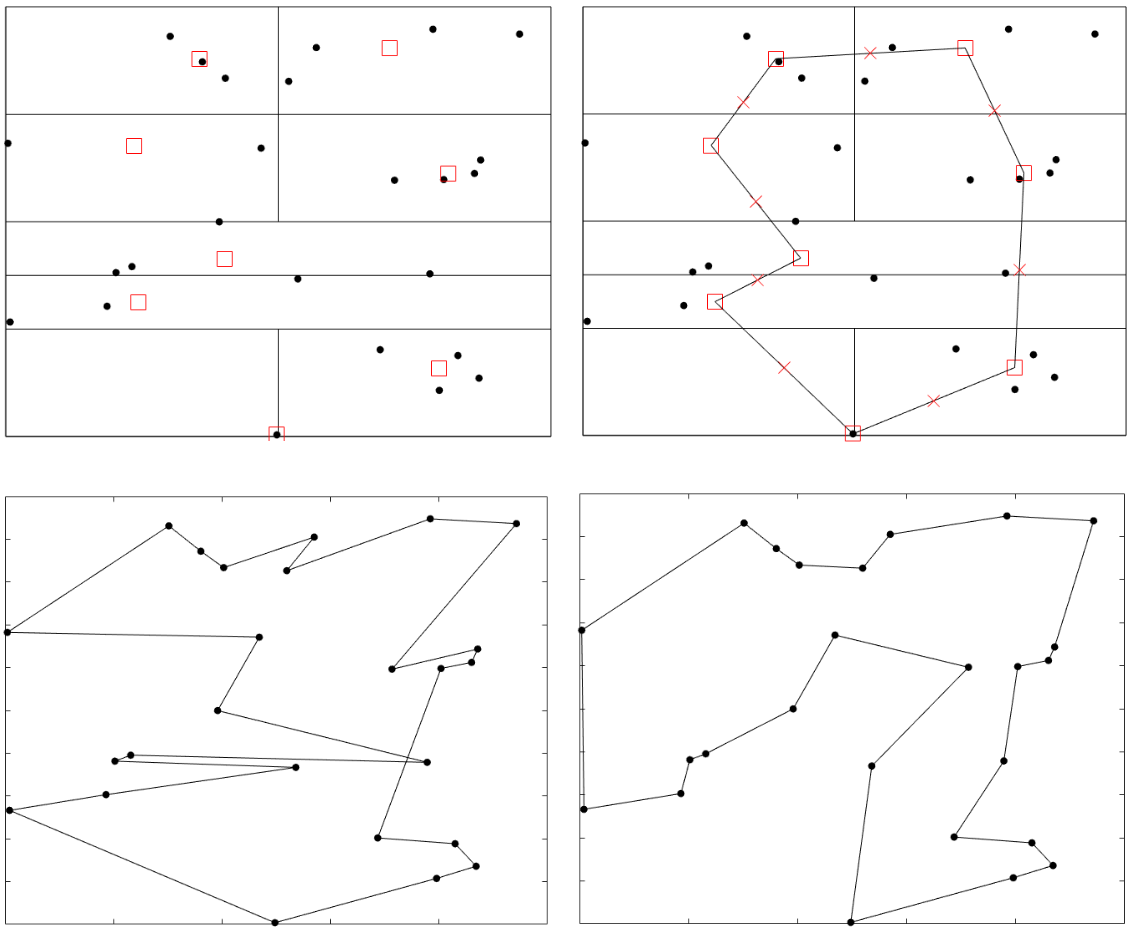


Figure 24: TSP solution construction for $N = 25$.

For the tests performed the results obtained are the following:

Mean tour length before 2-Opt	**Mean tour length after 2-Opt**	**Standard deviation before 2-Opt**	**Standard deviation after 2-Opt**	**% of tour length reduction**	**% of standard deviation reduction**
5.6460	4.7953	0.8694	0.4866	15.067	44.024

Table 2: Results for $N = 25$.

It can be observed from Table 2 that the 2-Opt algorithm efficiently accomplishes its aim, resulting in a solution with a tour length 15% shorter than the initial one. This reflects the importance of using Tour-Improvement Methods, which are easy to implement and give a better quality solution.

3.1.2. 50 Points

For the case of 50 points a higher order tiling has been used, which results in a greater number of barycenters to compute the initial ‘coarse’ solution.

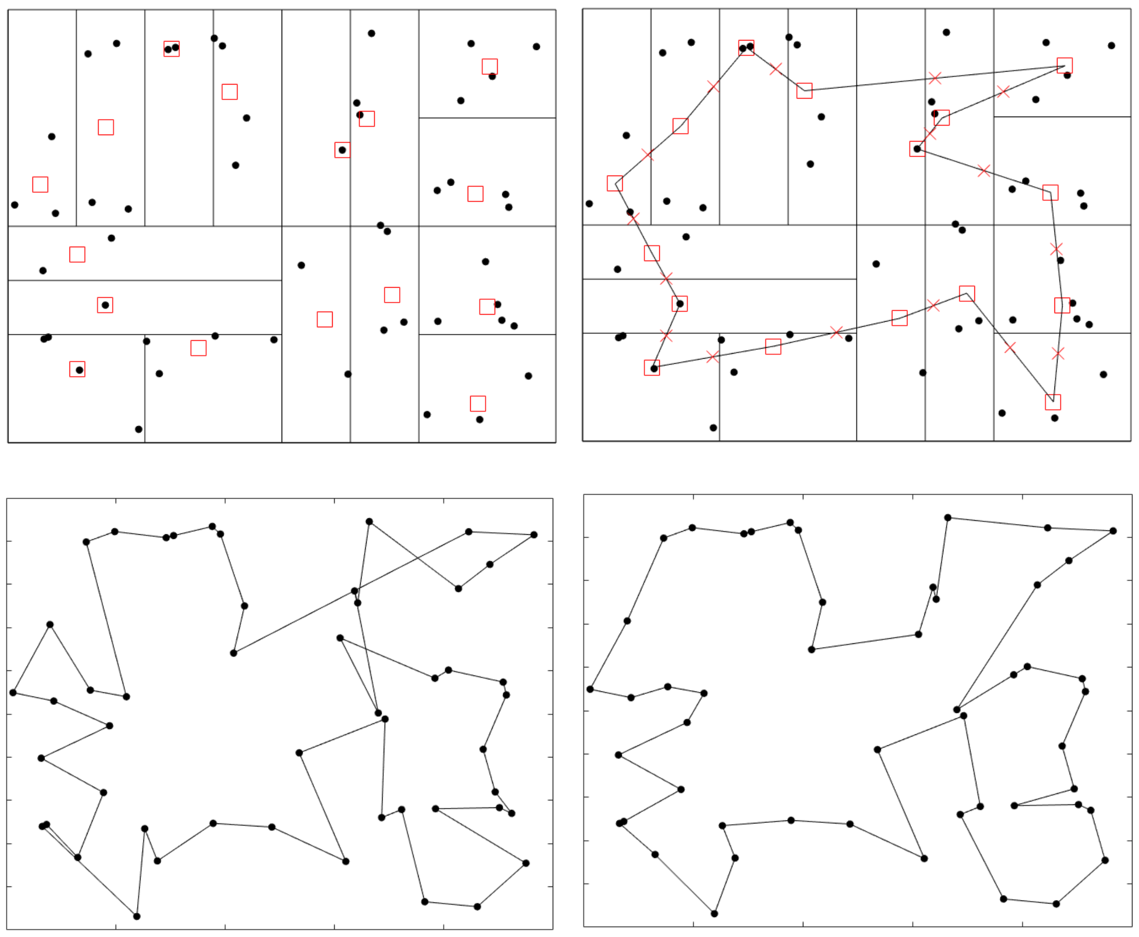


Figure 25: TSP solution construction for $N = 50$.

Mean tour length before 2-Opt	Mean tour length after 2-Opt	Standard deviation before 2-Opt	Standard deviation after 2-Opt	% of tour length reduction	% of standard deviation reduction
8.9188	6.8995	1.1837	0.5785	22.641	51.131

Table 3: Results for $N = 50$.

In this case, the 2-Opt heuristic significantly reduced the mean tour length by more than 20%.

3.1.3. 75 Points

For $N = 75$ the same order of tiling has been used as for $N = 50$ and it can be observed in Figure 26 that in this case the number of points contained in each cluster is bigger than previously, and hence a lower quality solution can be expected.

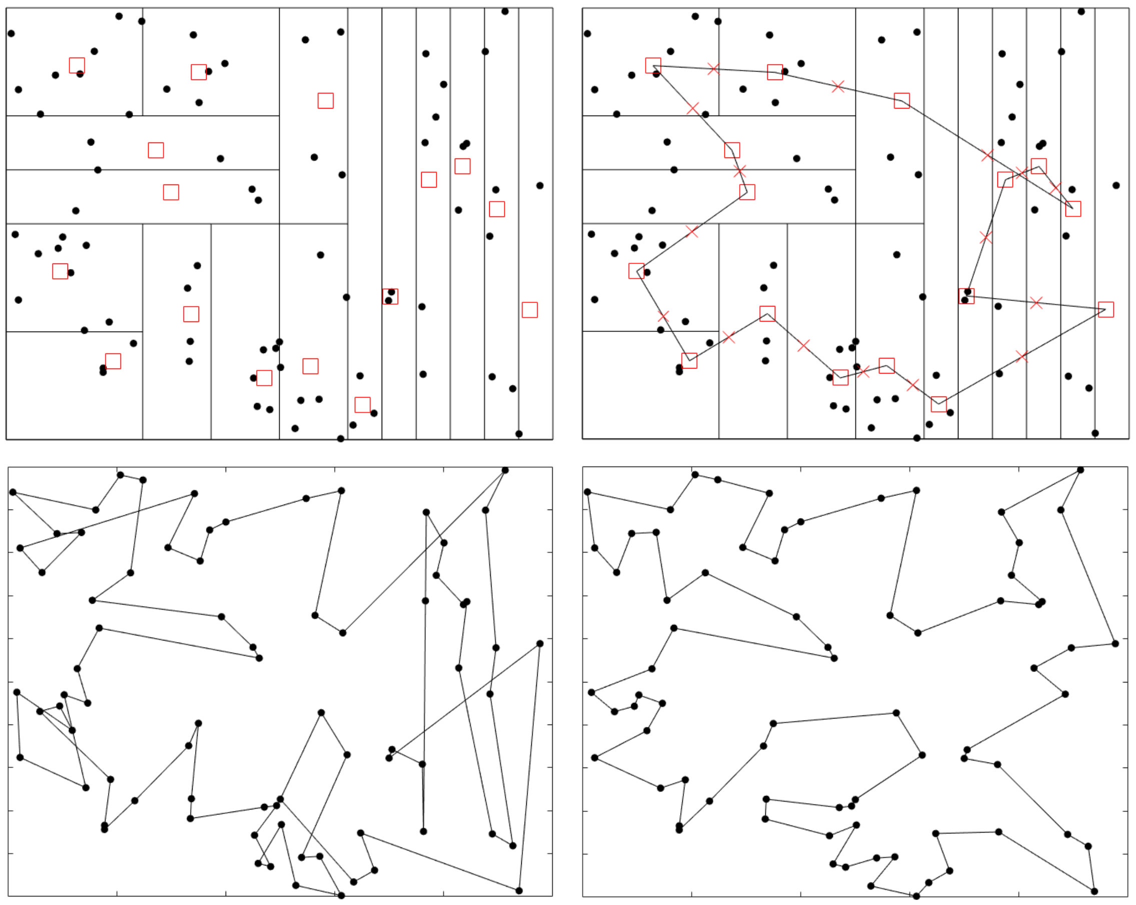


Figure 26: TSP solution construction for $N = 75$.

Mean tour length before 2-Opt	Mean tour length after 2-Opt	Standard deviation before 2-Opt	Standard deviation after 2-Opt	% of tour length reduction	% of standard deviation reduction
11.892	8.6489	1.6788	1.1120	27.271	33.761

Table 4: Results for $N = 75$.

Since there are more points in each dyadic tile, it is harder to obtain such a good solution as would be obtained with less points. This means that the quality of the initial solution is not expected to be of very good quality due to the intersections present. This is the reason why applying 2-Opt in this case has more effect than when applied to solutions for TSPs with a lower number of points.

3.1.4. 100 Points

For $N = 100$ 3rd order tiling has been used again. Although the mean number of points per cluster is still under 10, it has increased.

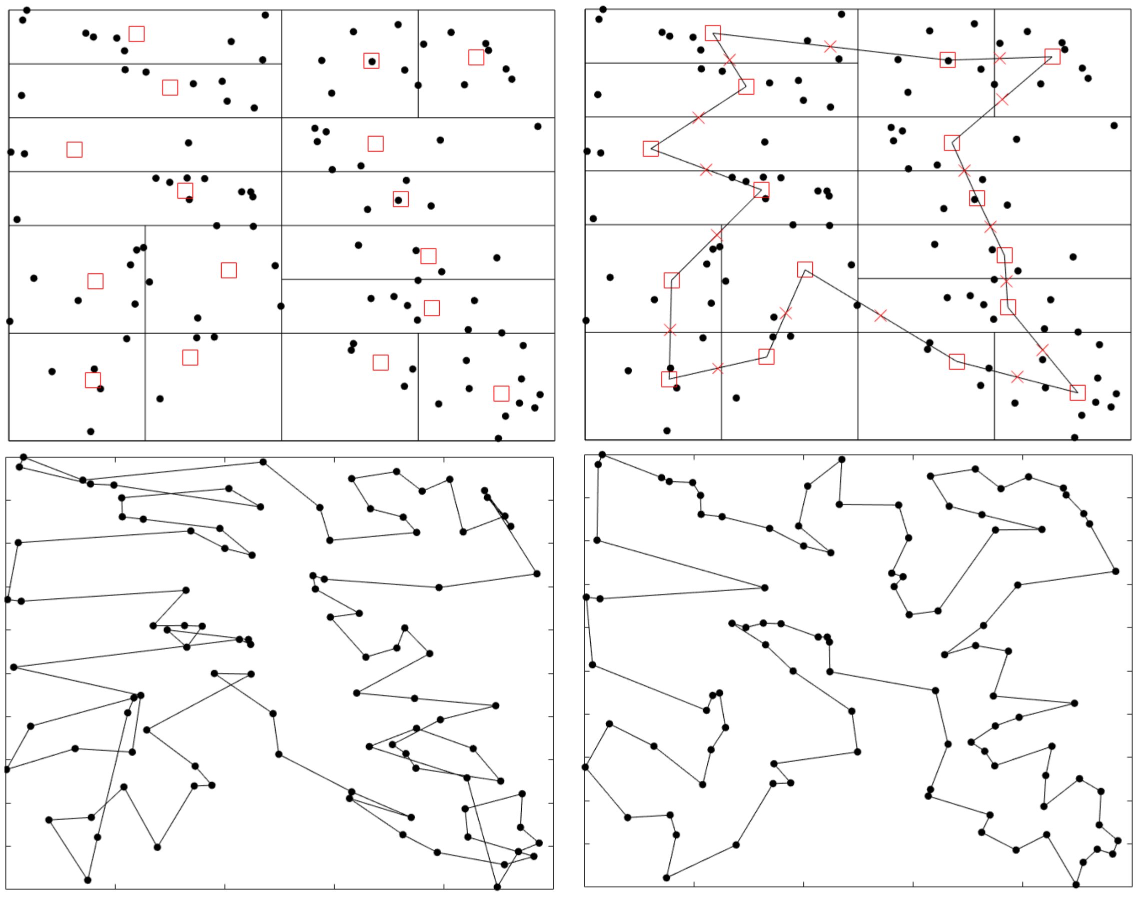


Figure 27: TSP solution construction for $N = 100$.

Mean tour length before 2-Opt	**Mean tour length after 2-Opt**	**Standard deviation before 2-Opt**	**Standard deviation after 2-Opt**	**% of tour length reduction**	**% of standard deviation reduction**
14.591	10.373	1.8798	0.7413	28.908	60.561

Table 5: Results for $N = 100$.

3.2. Comparison with expected values

In Figure 28 a graph comparing the optimal value for the tour length calculated with (11) and the results obtained with the developed Matlab code is shown, the difference between the values is also represented numerically in Table 6.

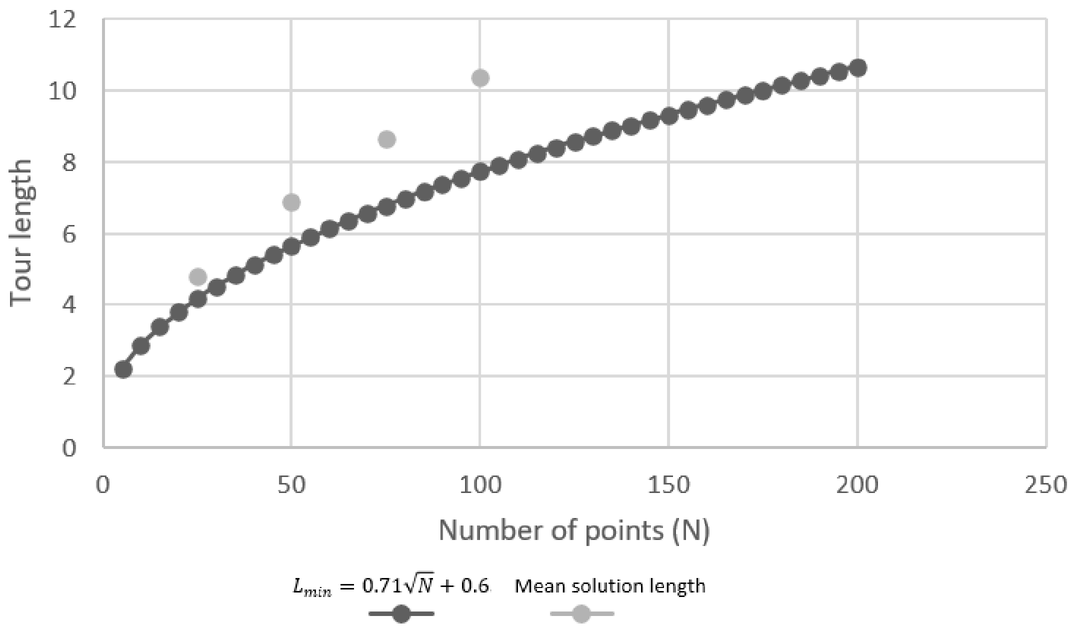


Figure 28: Comparison of optimal and obtained value.

Nº of Points	Mean Tour Length with 2-Opt	L_{min}	Mean error (%)
25	4.7953	4.1800	14.720
50	6.8995	5.6504	22.105
75	8.6489	6.7787	27.587
100	10.373	7.7300	34.191

Table 6: Comparison of expected and obtained values.

It can be observed that as the number of points has increased, the mean error between the minimum expected length and the one obtained with the code has also increased. For $N = 25$ the error was acceptable, around 14%, but for the largest number of points considered ($N = 100$) the error has increased to 34%, which shows that the solution could be of a better quality. The reasons for this and possible improvements will be explained in the next section.

3.3. Future improvements

From Table 6 it can be observed that the order of the tiling chosen has been adequate for $N = 25$, but from the high value obtained for the error in the rest of the cases, it can be deduced that perhaps a better solution could have been obtained if a higher order tiling had been used. This is because the number of points in each cluster would have been smaller. However, a higher order tiling would have also resulted in a greater number of barycenters and hence a lower quality ‘coarse’ solution.

For the case of $N = 100$ and further, to minimize the number of points per cluster, a higher order tiling should be used. To avoid getting a low quality ‘coarse’ solution for the barycenters, multiple partitioning, as explained in Figure 7, should be used.

Also, as mentioned in [8] another possible way of improving the solution would have been by using GAEHS, *Genetic Algorithm Enhanced Hierarchical Solution,* which is a method that recombines the random dyadic tiling to obtain a better solution.

Bibliography

[1] *Hoffman, Karla & Padberg, Manfred & Rinaldi, Giovanni*. **Traveling Salesman Problem**, 2013. Doi: 10.1007/978-1-4419-1153-7_1068.

[2] *Christos H. Papadimitriou,* **The Euclidean travelling salesman problem is NP-complete, Theoretical Computer Science**, Volume 4, Issue 3, 1977, Pages 237-244, ISSN 0304-3975, https://doi.org/10.1016/0304-3975(77)90012-3

[3] *Britannica, T. Editors of Encyclopaedia* (2018, January 16). **NP-complete problem**. **Encyclopedia Britannica**. https://www.britannica.com/science/NP-complete-problem

[4] *R. Martí and G. Reinelt,* **The Linear Ordering Problem, Exact and Heuristic Methods in Combinatorial Optimization 175,** doi: 10.1007/978-3-642-16729-4_2, Springer-Verlag Berlin Heidelberg 2011.

[5] *Gilbert Laporte,* **The traveling salesman problem: An overview of exact and approximate algorithms**, European Journal of Operational Research, Volume 59, Issue 2, 1992, Pages 231-247, ISSN 0377-2217, https://doi.org/10.1016/0377-2217(92)90138-Y.

[6] *C. Roos,* **Linear Optimization,** Editor(s): Robert A. Meyers, Encyclopedia of Physical Science and Technology (Third Edition), Academic Press, 2003, Pages 597-616, ISBN 9780122274107, https://doi.org/10.1016/B0-12-227410-5/00379-3.

[7] *M. Padberg, G. Rinaldi,* **Optimization of a 532-city symmetric traveling salesman problem by branch and cut,** Operations Research Letters, Volume 6, Issue 1, 1987, Pages 1-7, ISSN 0167-6377, https://doi.org/10.1016/0167-6377(87)90002-2.

[8] *T. Kalmár-Nagy and B. D. Bak,* **Hierarchical Solution of the Traveling Salesman Problem with Random Dyadic Tilings**, **Fluctuation and Noise Letters**, vol. 17, no. 1, 2018. Doi:10.1142/S0219477518500037.

[9] *Ziauddin Ursani and David W. Corne,* (2016), **Introducing Complexity Curtailing Techniques for the Tour Construction Heuristics for the Travelling Salesperson Problem**, Journal of Optimization, Volume 2016, Article ID 4786268, p. 1-15.

[10] *C.-P. Hwang, Alidaee, B., & Johnson, J.* (1999). **A Tour Construction Heuristic for the Travelling Salesman Problem**. The Journal of the Operational Research Society, 50(8), 797-809. doi:10.2307/3010339

[11] *Xin-She Yang,* Chapter 6 - **Genetic Algorithms**, Editor(s): Xin-She Yang, Nature-Inspired Optimization Algorithms (Second Edition), Academic Press, 2021, Pages 91-100, ISBN 9780128219867, https://doi.org/10.1016/B978-0-12-821986-7.00013-5.

[12] *Angel, o., holroyd, a., kozma, g., wästlund, j., & winkler, p.* (2014). **The phase transition for dyadic tilings.** Transactions of the American Mathematical Society, 366(2), 1029-1046. Retrieved May 16, 2021, from http://www.jstor.org/stable/23812977.

[13] *Janson, Svante & Randall, Dana & Spencer, Joel.* (2002). **Random Dyadic Tilings of the Unit Square**. Random Struct. Algorithms. 21. 225-251. 10.1002/rsa.10051.

[14] *Stefan Hougardy, Fabian Zaiser, Xianghui Zhong*, (2020). **The approximation ratio of the 2-Opt Heuristic for the metric Traveling Salesman Problem**, Operations Research Letters, Volume 48, Issue 4, Pages 401-404, ISSN 0167-6377, https://doi.org/10.1016/j.orl.2020.05.007.

[15] *Kirk, Joseph* (2014). **Traveling Salesman Problem - Genetic Algorithm,** MATLAB Central File Exchange.

[16] *Kirk, Joseph* (2014). **Fixed Open Traveling Salesman Problem (TSP) Genetic Algorithm (GA),** MATLAB Central File Exchange.

[17] *J. Beardwood, J. H. Halton and J. M. Hammersley,* **The shortest path through many points,** in Mathematical Proc. Cambridge Philosophical Society, Vol. 55 (Cambridge University Press, Cambridge, 1959), pp.299-327.

[18] *D. L. Applegate, R. E. Bixby, V. Chvatal and W. J. Cook*, **The Traveling Salesman Problem: A Computational Study,** (Princeton University Press, Princeton, NJ, 2011).