# Shortest Path Optimization of the CyberKnife treatment in Radiotherapy

Kortste-pad optimalisatie van de CyberKnife-behandeling bij radiotherapie

by

**Joris Mühlsteff**

in partial fulfillment of the requirements for the degree of

**Bachelor of Science**
in Applied Mathematics

at the Delft University of Technology,
to be defended publicly on Thursday July 6, 2017 at 11:00 PM.

**Supervisor**

| | |
|---|---|
| Drs. ir. M. Keijzer, | TU Delft |

**Commitee members**

| | |
|---|---|
| Dr.ir. S. Breedveld, | Erasmus MC Rotterdam |
| Dr. D.C. Gijswijt, | TU Delft |
| Dr. J.G. Spandaw, | TU Delft |

# Preface

This bachelor thesis was written as part of the curriculum for students of the bachelor Applied Mathematics at the TU Delft. For my Bachelorproject, I did an internship at the Erasmus Medical Center Rotterdam. The project itself took place at the Daniel den Hoed location, which specializes in oncology.

I would like to thank the following people: Marleen Keijzer, my supervisor from the TU Delft for bringing me in contact with the Erasmus Medical Center and for advice on the project and the thesis. Sebastiaan Breedveld, my supervisor at the Erasmus Medical Center, for guiding me through most of the project, doing plan quality calculations and for overall support during the internship. And lastly, I would like to thank my fellow research colleagues at the faculty, for making my internship at the Erasmus Medical Center a true fun experience for me to remember.

This internship was a great opportunity for me to apply knowledge of mathematics to solve real world problems.

# Summary

The main goal of this report is to improve the traveltime of the CyberKnife treatment, used for radiotherapy, without loss of plan quality. This is done by using optimization techniques, such as Dijkstra's Algorithm, as well as incorporating Hamiltonian paths and the Traveling Salesman Problem. All calculations are done using Matlab, a numerical computing software.

With the above mentioned techniques, we created OPA, an Optimal Path Algorithm, that is based on finding Hamiltonian paths, combined with the iterated process of interchanging nodes with adjacent ones. With OPA, the traveltimes for 31 patients have been brought down by 35.9% on average, without any significant loss of plan quality.

# Contents


**1 Introduction** **1**

1.1 Radiotherapy . . . 1

1.2 The CyberKnife . . . 1

**2 Problem description** **5**

2.1 Research goals . . . 5

2.2 Overview . . . 6

2.3 Study Design . . . 6

**3 Shortest Path Optimization** **9**

3.1 Asymmetrical Dijkstra Algorithm (ADA) . . . 9

3.1.1 Methods . . . 10

3.1.2 Results . . . 11

3.2 Symmetrical Dijkstra Algorithm (SDA) . . . 12

3.2.1 Methods . . . 12

3.2.2 Results . . . 14

3.3 Hamiltonian Path Algorithm (HPA) . . . 15

3.3.1 Methods . . . 15

3.3.2 Results . . . 18

3.4 Comparison of methods . . . 19

**4 Neighbour Path Algorithm (NPA)** **21**

4.1 Computing Neighbour Paths . . . 21

4.1.1 Methods . . . 21

4.1.2 Results . . . 23
4.2 Analysis of Convergence . . . 24
4.3 Plan Quality . . . 26

**5 Additional restrictions 27**
5.1 Starting Nodes . . . 27
5.1.1 Methods . . . 27
5.2 Dummy Nodes . . . 28
5.2.1 Methods . . . 28
5.3 Concluding Results . . . 28

**6 Conclusions 31**

**7 Recommendations 33**

**Bibliography 35**

**Appendices 37**
A Matlab codes . . . 39
B Traveltimes of all algorithms . . . 49
C New paths . . . 51

# Chapter 1

# Introduction

## 1.1 Radiotherapy

In the history of medicine, different treatments for cancer have been developed, such as radiotherapy, surgery and chemotherapy. However, nowadays more than 50% of the people diagnosed with cancer, undergo a radiotherapy treatment [1]. During a radiotherapy treatment, the tumor is treated locally witch ionizing radiation, which destroys the malignant cells, while sparing the surrounding healthy tissue as much as possible [2]. Some other forms of radiotherapy include stereotactic radiosurgery (focusing high-power energy on a small area of the body) and brachytherapy (placing a radiation source inside the tumor).

Whenever a patient has to undergo radiotherapy, a computed tomography scan (CT-scan) has to be made of the patient, on which a treatment plan is designed. This plan tells which parts should be irradiated and with what level [3]. After this, the treatment is delivered by a treatment device, which will be further outlined in the next section.

## 1.2 The CyberKnife

The CyberKnife is a robotic radiosurgery system that was invented by Dr. John R. Adler, a Stanford University professor and Peter and Russell Schonberg of Schonberg Research Corporation. It is used for treating a variety of tumors, such as lung, prostate, and head-and-neck tumors. For this project, we will focus on prostate cancer patients only. Since 2004, the Radiotherapy department of the Erasmus Medical Center Rotterdam started using the CyberKnife, as seen in Figure 1.1, for precise radiations.

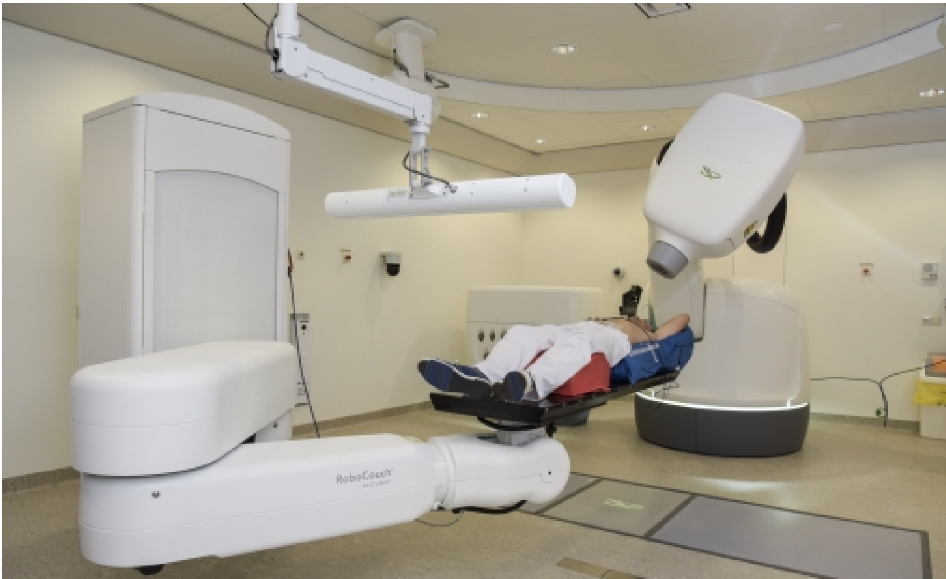


Figure 1.1: CyberKnife used at the Erasmus Medical Center Rotterdam

The treatment plan for a patient contains a selection of around 25 out of 110 nodes which is found to be best fitting for the treatment of that patient. The nodes mentioned in the treatment plan are the nodes from which radiation is delivered.

For the actual treatment of the patient, a virtual grid is placed around the patient. Since the nodes emanated from treatment plans made for prostate cancer patients, the grid is shaped like a semisphere, containing 110 candidate nodes, as seen in Figure 1.2.

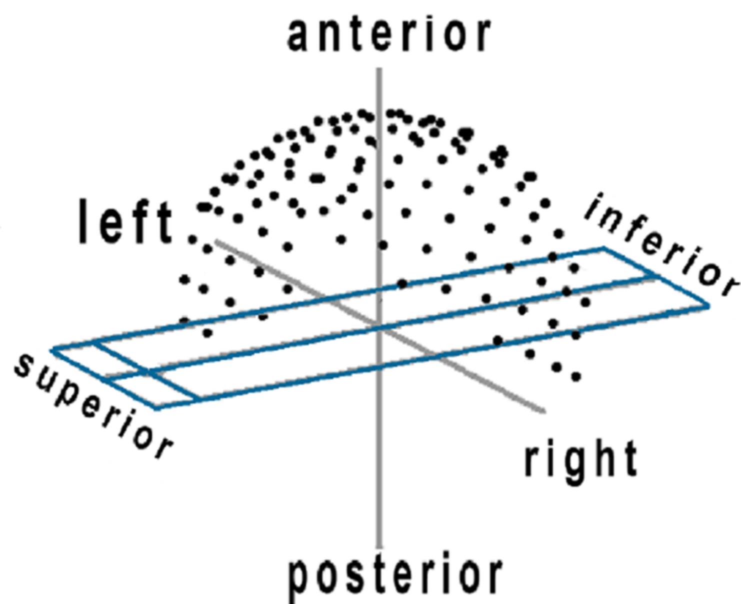



Figure 1.2: *Breedveld, S. (2013). CyberKnife search space. Towards automated treatment planning in radiotherapy.* The blue rectangle represents the operating table.

If for example a head-and-neck tumor had to be treated, the shape of the grid would resemble something close to three-quarter of a sphere, since traveling underneath the head is also possible with the CyberKnife.

The approximately 25 nodes used to radiate the patient are the so called *mustpass nodes*, while the remaining ∼85 nodes can be used as traveling nodes for the CyberKnife to move from one node to the other, the so called *inbetween nodes*. There are also 8 dummy nodes, which are used for the sole purpose of serving as inbetween nodes. Radiation from these nodes is not possible. Note that the inbetween nodes change for every patient, as each patient has a different set of mustpass nodes.

Movement of the CyberKnife is restricted to grid lines between the nodes. This has to do with safety reasons. The three-dimensional grid is placed around the patient so that the CyberKnife moves safely alongside the patient without touching him or her. If the machine were not to move around the grid lines, but simply around the fastest route from one node to another, which would be a straight line in Euclidean space, the machine could injure the patient. Therefore it travels around the patient by the pre-programmed lines. If the CyberKnife has to travel from node A to adjacent node B, it could do so directly by a straight line. However, if there is not a direct path from node A to B, the CyberKnife may first need to move to an adjacent node C. This process is repeated until the node B can be directly reached.

# Chapter 2

# Problem description

## 2.1 Research goals

Now that we have a decent understanding of the working of the CyberKnife, we can formulate our general research question:

*"How can we optimize the traveltime of the CyberKnife?"*

Before we start thinking of any strategies to answer this question, there are still some important restrictions of the CyberKnife which we will need to take into consideration:

- The CyberKnife can only travel from nodes to higher-numbered nodes. This means that if the CyberKnife for example has to travel from node 1 to 4, it could do so by going from 1 to 3 to 4, but not by going from 1 to 5 to 4. This is because of mechanical limitations, such as cables of the robotic arm getting tied up.
- The order in which the mustpass nodes are visited cannot be altered. The CyberKnife currently visits the mustpass nodes in ascending order.
- It is also determined in the treatment plan which mustpass nodes need to be visited. Deviating from these nodes may result in loss of plan quality.

Based on these restrictions, we can split up our research question into three sub-questions:

1. *How can we travel as fast as possible from node A to node B?*
2. *Does changing the order of the mustpass nodes benefit the traveltime?*
3. *Does replacing mustpass nodes with adjacent nodes benefit the traveltime, without loss of plan quality?*

Now we can start outlaying our steps to answer these questions.

## 2.2 Overview

In Chapter 3 we will take a look at the first two sub-questions. We will discuss three related methods for finding shortest paths through the mustpass nodes, and compare results. In Chapter 4, we will continue with the last sub-question: finding adjacent solutions without degrading the plan quality. We will first develop a method for finding adjacent solutions, after which we can check whether the plan quality is still acceptable. In Chapter 5 we will look at two additional restrictions that came along the research. Lastly, we will discuss our results in Chapter 6 and give further recommendations in Chapter 7.

Since this project is done almost entirely in Matlab, there will be pseudocode provided for all the Matlab programs. The actual Matlab code is added in Appendix A.

## 2.3 Study Design

The Erasmus Medical Center Rotterdam provided a dataset containing a cell with mustpass nodes of plans for the patients, as well as a traversal matrix: a matrix of traveltimes between the nodes. The cell contains nodes of 30 patients, from which 20 patients have 30 mustpass nodes, and 10 patients have 32 mustpass nodes. Besides the nodes for the 30 patients, they also provided 25 nodes for a class solution [4]. This is a path that is acceptable for all patients. Since it contains less nodes than the paths for the patients, it does not result in a optimal treatment plan, but saves more time on the other hand. We will include this path as our 'patient 0'.

The traversal matrix is a $110 \times 110$ matrix, mustpass nodes and inbetween nodes mixed, so because of its size, it is not possible to fully display the matrix, so a small section of it is displayed in Table 2.1. Note that the matrix is the same for all patients, as these times concern the traveltimes of the nodes in the grid, which is stored information.

| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 8.5 | x | x | x | x | x | 10.8 |
| **2** | x | 0 | 3.7 | 8 | 5.3 | 7 | 10.2 | 5.7 |
| **3** | x | x | 0 | 7.7 | x | 9.5 | 12.7 | 7 |
| **4** | x | x | x | 0 | 4.3 | x | 13.7 | x |
| **5** | x | x | x | x | 0 | 8 | 11.5 | 10.2 |
| **6** | x | x | x | x | x | 0 | 4.7 | 9.2 |
| **7** | x | x | x | x | x | x | 0 | 11.7 |
| **8** | x | x | x | x | x | x | x | 0 |

Table 2.1: Section of the asymmetrical traversal matrix. Traveltimes denoted in seconds.

The time it takes the CyberKnife to travel from node $A$ to node $B$, with $A < B$, can be found in matrix element $(A, B)$. Note that if $A > B$, we always find x as a result, because traveling to a lower-numbered node is not possible with this implementation. This is coherent with the restrictions mentioned in section 2.1.
There are also some x's above the diagonal. This implies that there is not a direct path between those nodes. For example, if the algorithm has to travel from node 1 to node 5, it first checks all the directly reachable nodes from node 1. From those nodes, it checks if node 5 now can be

directly reached. It keeps doing so until node 5 can be directly reached, or until a shorter path with more nodes is found. For example if $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$ would be faster than $1 \rightarrow 3 \rightarrow 5$, because traveling from node 3 to 5 is apparently very inefficient. Lastly, traveling to a node itself has no traveltime, and therefore the diagonal entries are all 0.

# Chapter 3

# Shortest Path Optimization

This chapter contains three related algorithms for finding shortest paths through the mustpass nodes. For each algorithm, an explanation of its functioning will be provided, both in words, pseudocode and visual. The chapter will be concluded with a comparison of the three algorithms.

## 3.1 Asymmetrical Dijkstra Algorithm (ADA)

In this section, we will give a description of the current programming of the CyberKnife. This method utilizes the Asymmetrical Dijkstra Algorithm (ADA). For the construction of the algorithm, an implementation of Dijkstra from Mathworks [5] was used. A pseudocode description of Dijkstra's algorithm [6] is shown in Algorithm 1.

**Input** Graph $G = (V, E)$ with traveltimes on edges; starting node $s$; finish node $f$;
**Ouput** Shortest distance from $s$ to $f$;

$W$ = set of all unvisited nodes;
**for** *all vertices* $v$ *in* $G$ **do**
  $d(s, v) = \infty$;
  $d(s, s) = 0$;
  add $v$ to $W$;
**end**
**while** $W \neq \emptyset$ **do**
  $u$ = *vertex with minimal* $d(s, u)$;
  $W = W \backslash \{u\}$;
  **for** *each neighbour* $v \in W$ *of* $u$ **do**
    $d(s, v) = min\{d(s, v), d(s, u) + d(u, v)\}$;
  **end**
**end**

**Algorithm 1:** Pseudocode description of Dijkstra's algorithm.

### 3.1.1 Methods

The shortest path problem in the CyberKnife treatment comes down to the following: find the shortest path that covers all mustpass nodes, starting from node(1), to node(2),..., to node(end). This basically means applying Dijkstra's algorithm to each sequential node pair and finding the shortest times between these pairs. If we then add up all the sequential times, we get the total traveltime to cover all the mustpass nodes.

To visualize this process, we will discuss an example with the use of Figure 3.1.

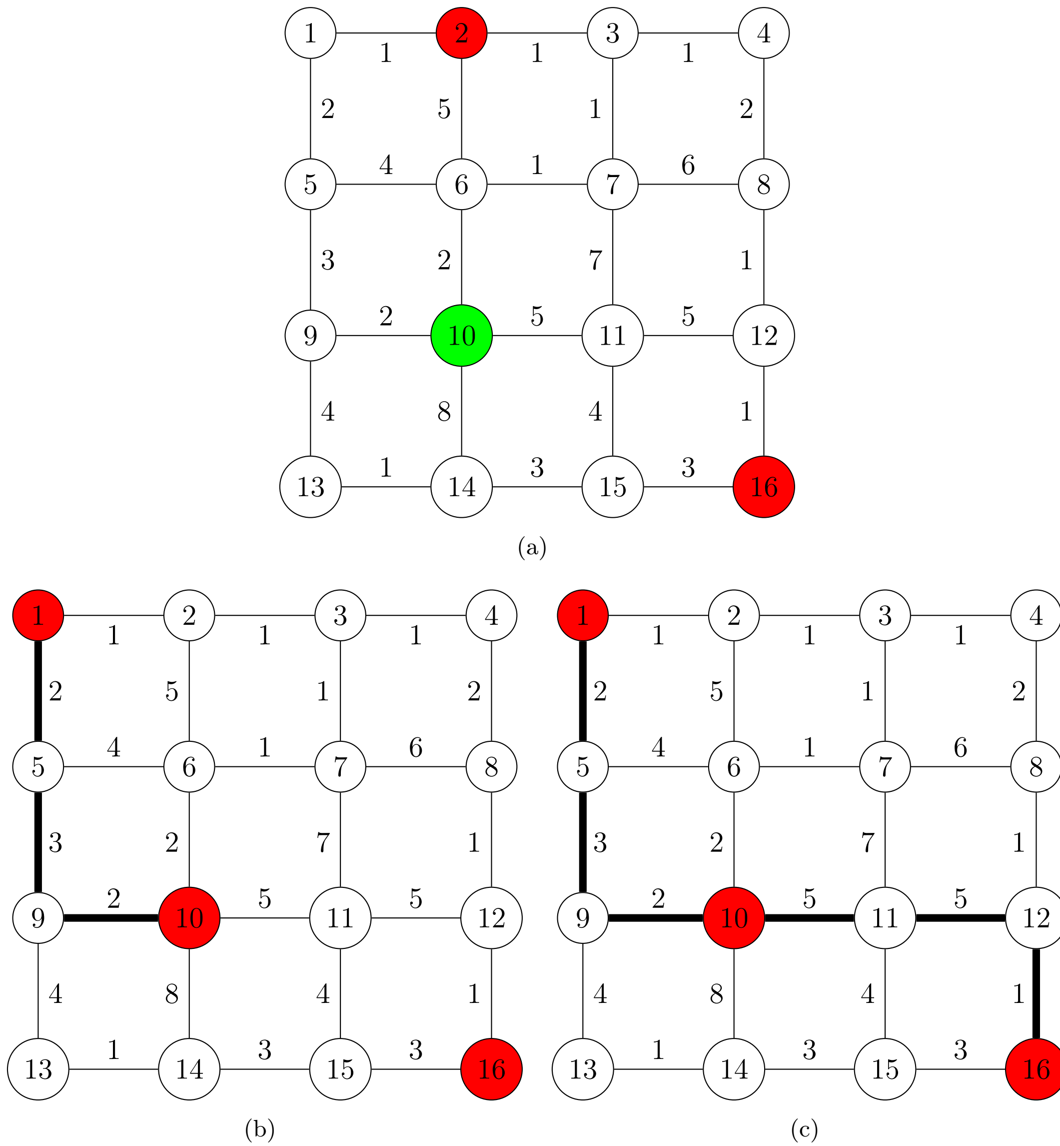



Figure 3.1: Example of the ADA.

Suppose we have a simplified grid with 13 inbetween nodes, with 3 mustpass nodes: 1, 10 and 16. Our starting node will be node 1, and our finish node will be node 16. If it is possible to travel directly from one node to the other, the nodes are labeled neighbours and are connected

with an edge with the traveltime between those node on it, as displayed in Figure 3.1(a). The ADA will now apply Dijkstra's algorithm with starting node 1 and finish node 10. It finds the shortest path as displayed in Figure 3.1(b). Note that $1 \rightarrow 2 \rightarrow 3 \rightarrow 7 \rightarrow 6 \rightarrow 10$ is actually a shorter path, but since that contains a traversal to a lower-numbered node, it is not possible. The ADA then applies Dijkstra's algorithm with starting node 10 and and finish node 16 and finds the shortest path as displayed in Figure 3.1(c). After having computed all the sequential shortest paths, the ADA then adds up all the sequential traveltimes which results in the final traveltime. In our example, this results in a path length of 18.

A pseudocode description of the ADA is displayed in Algorithm 2. The full Matlab code is added in Appendix A.1.

```
Input Mustpass odes for all patients; Traversal matrix;
Output Shortest paths and traveltimes for all patients;

for all patients do
 |  Sort all nodes in ascending order;
end
Initialize table with results ;
for all patients do
 |  Initialize table with nodes times between sequential mustpass node pairs;
 |  for nodes per patient do
 |   |  Compute traveltimes between sequential mustpass node pairs with Dijkstra
 |   |    implementation;
 |  end
 |  Sum all sequential node times;
 |  Return traveltime
end
```

**Algorithm 2:** Pseudocode for the Asymmetrical Dijkstra algorithm

### 3.1.2 Results

Using the data from the Erasmus Medical Center Rotterdam, the ADA results in the traveltimes denoted in Figure 3.2. These are the current traveltimes of the CyberKnife for the 30 patients. The full table with results is added in Appendix B.

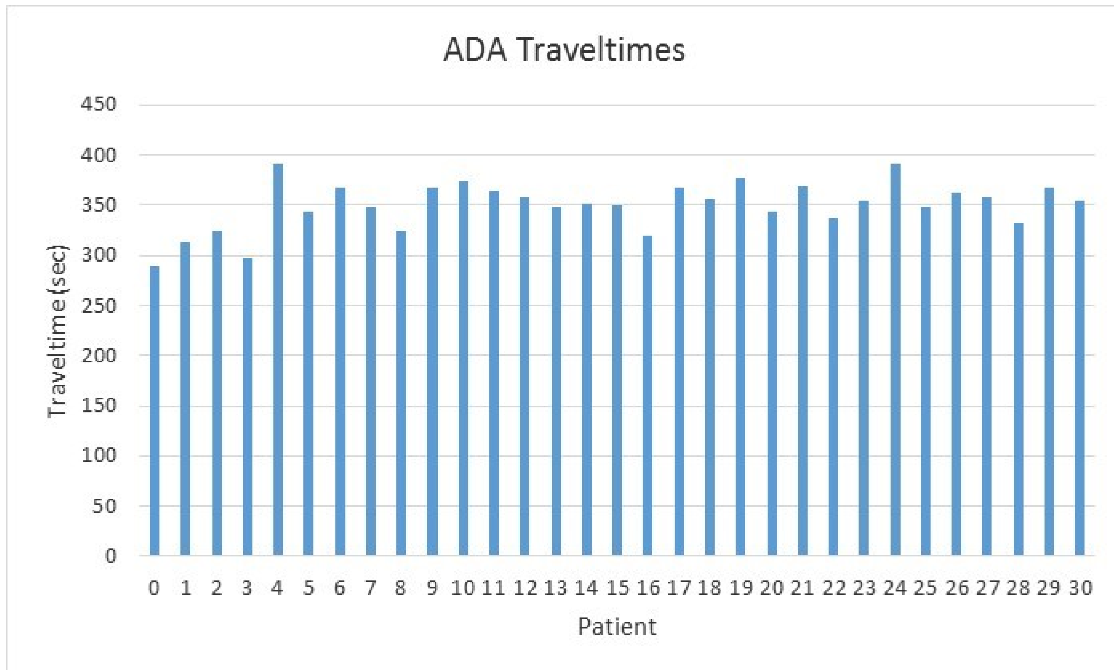



Figure 3.2: Traveltimes of 31 patients, computed with the ADA.

## 3.2 Symmetrical Dijkstra Algorithm (SDA)

We will now take our first step in improving the current programming of the CyberKnife, by removing the first restriction in Section 2.1, which states that the CyberKnife can only travel to lower-numbered inbetween nodes. The algorithm presented in this section uses the same implementation of Dijkstra's algorithm, but with a minor modification in the input, which will be explained in the next section.

### 3.2.1 Methods

We do now allow traveling to lower-numbered inbetween nodes, so we need to know the traveltime from node $A$ to node $B$, where $A > B$. The most staightforward choice would be the same traveltime from node $B$ to node $A$. This means we have to modify the matrix from Section 3.1.1, by mirroring it over its diagonal, which will result in a symmetrical matrix with zeros on the diagonal. A section of the now symmetrical traversal matrix is denoted in Table 3.1.

| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 8.5 | x | x | x | x | x | 10.8 |
| **2** | 8.5 | 0 | 3.7 | 8 | 5.3 | 7 | 10.2 | 5.7 |
| **3** | x | 3.7 | 0 | 7.7 | x | 9.5 | 12.7 | 7 |
| **4** | x | 8 | 7.7 | 0 | 4.3 | x | 13.7 | x |
| **5** | x | 5.3 | x | 4.3 | 0 | 8 | 11.5 | 10.2 |
| **6** | x | 7 | 9.5 | x | 8 | 0 | 4.7 | 9.2 |
| **7** | x | 10.2 | 12.7 | 13.7 | 11.5 | 4.7 | 0 | 11.7 |
| **8** | 10.8 | 5.7 | 7 | x | 10.2 | 9.2 | 11.7 | 0 |

Table 3.1: Small part of the symmetrical node traversal matrix. Traveltimes denoted in seconds.

The x's in the matrix are the same ones as in the matrix in Figure 2.1, meaning that there is not a direct path between the nodes. If the algorithm now has to find the shortest path from mustpass node $A$ to mustpass node $B$, it can now also pick inbetween nodes with a lower numbering than $A$.

To visualize this process, we return to our example from Section 3.1.1.

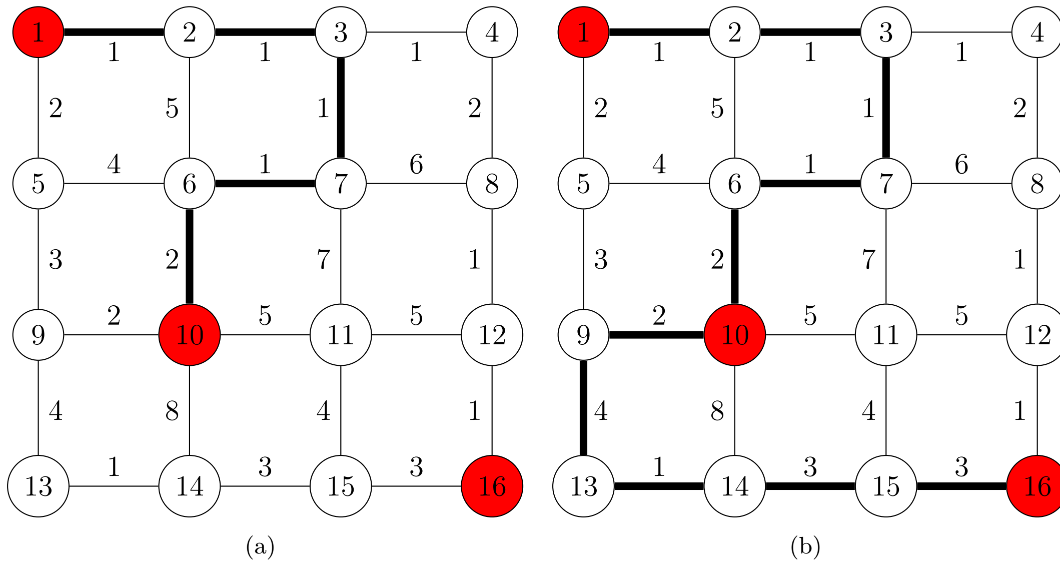



Figure 3.3: Example of the SDA.

This time, we allow traveling to lower-numbered nodes, so the SDA can pick the path that we already mentioned in 3.1.1, $1 \to 2 \to 3 \to 7 \to 6 \to 10$, as our shortest path from node 1 to node 10, as displayed in Figure 3.3(a). The SDA then does the same for start node 10 and finish node 10, which results in the shortest path $10 \to 9 \to 13 \to 14 \to 15 \to 16$, as shown in Figure 3.3(b). The total path length is now 16, which is an improvement over the ADA.

So our Symmetrical Dijkstra Algorithm (SDA) basically operates the same as the ADA, with the only difference being that we feed the algorithm the symmetrical matrix instead of the asymmetrical matrix. Therefore the pseudocode description of the SDA is the same as in Algorithm 2. The full Matlab code is added in Appendix A.2.

### 3.2.2 Results

Using the same data as in Section 3.1.1, the SDA results in the traveltimes denoted in Figure 3.4.

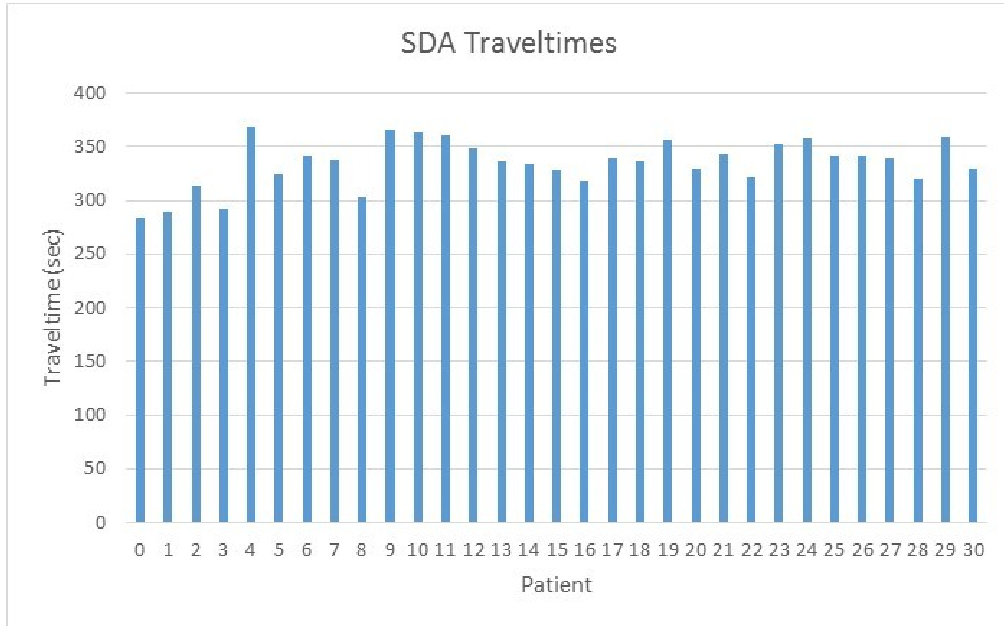



Figure 3.4: Traveltimes of 31 patients, computed with the SDA.

We would expect better results with the SDA, since more paths are allowed, thus creating a higher chance of finding a shorter path. A comparison of the ADA and SDA is shown in Figure 3.5. The full table with results is added in Appendix B.

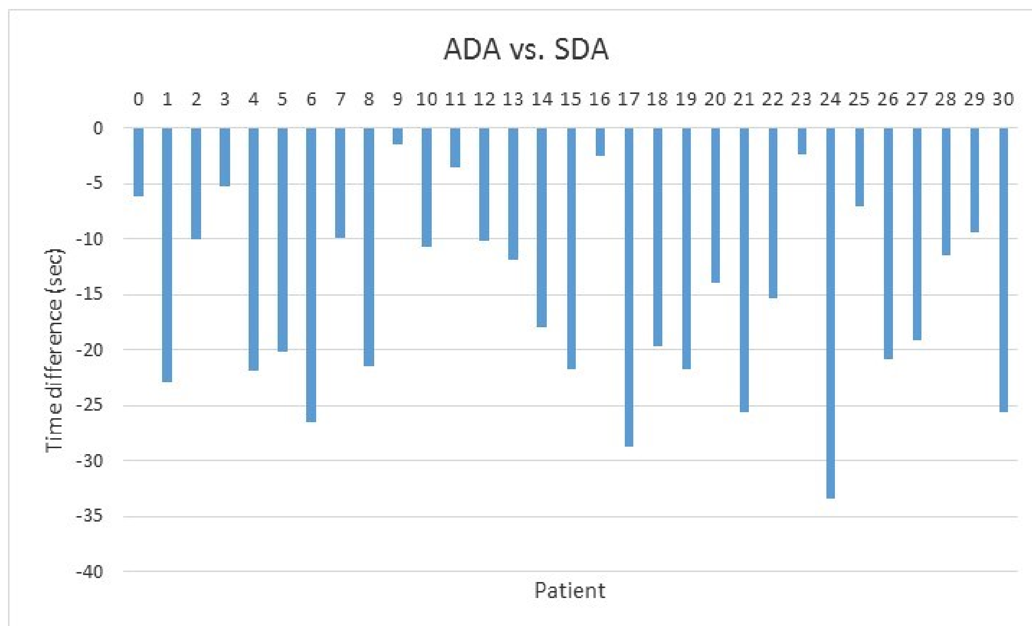



Figure 3.5: Time advantages of SDA over ADA for 31 patients.

If we look at Figure 3.5, we see that there is already a time advantage for all the patients, with an average of 15.5 seconds, which translates to a 4.4% time advantage on average. The standard deviation here amounts more than 7 seconds, which is interesting. This probably has to do with the fact that some paths simply cannot get much better by traveling to lower-numbered nodes.

## 3.3 Hamiltonian Path Algorithm (HPA)

Now we can focus on the second restriction from Section 2.1. Not only do we allow traveling to lower-numbered inbetween nodes, but also changing the order of the mustpass nodes. This means we now have a number of $n$ nodes which need to be visited as quickly as possible, regardless of the order in which we do so. This resembles one of mathematics most well known problems: The Traveling Salesman Problem (TSP), which states the following: *"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the original city?"* In our problem, the cities are nodes, and the distances are traveltimes. Note that the TSP requires that the starting point and the ending point are the same. This is required so that the postman, in the original problem, returns home instead of ending up somewhere and then having to travel all the way back home. In our problem however, it is not necessary for the CyberKnife to return to its starting position. Therefore we are free to choose our begin and end node, as long as we cover all the mustpass nodes. Again, we recognize another famous problem in mathematics, closely related to the TSP: finding a Hamiltonian path in a graph. This problem comes down to the following:

> *"A Hamiltonian path is a path in a graph that visits each vertex exactly once. Then does a given graph contain a Hamilton path?*

This problem will play a big role in constructing our new algorithm, which we will continue in the next section.

### 3.3.1 Methods

Now that we only focus on reordering the mustpass nodes, every combination of nodes is now possible. Therefore we need to extend the matrix in Section 3.2.1. This means we have to replace the remaining x's in the matrix with the accompanying traveltimes, for convenience. This can be accomplished by applying Dijkstra's algorithm on every entry in the matrix. Note that whenever we used the ADA or the SDA in the previous sections, since not all entries in the $110 \times 110$ matrix were filled with a positive traveltime, the algorithm had to compute the traveltimes for all the elements in the matrix for every single patient again. To reduce the computational time, instead of computing all the values again for every patient, we fill in the matrix in advance, so that the algorithm can simply check the value in every element. A pseudocode description of the process

is displayed in Algorithm 3. The full Matlab code is added in Appendix A.3.

**Input** Matrix used in the SDA;
**Output** Symmetrical matrix with traveltimes for all node pairs and matrix with paths between all node pairs;

Initialize empty matrix for traveltimes ;
Initialize empty matrix for paths ;

**for** *all rows* **do**
  **for** *all columns* **do**
    Compute the traveltimes and paths between every node pair with Dijkstra's algorithm ;
  **end**
**end**

**Algorithm 3:** Pseudocode for computing the traveltimes and paths between every node pair

This results in symmetrical matrix with traveltimes and a matrix with paths. Small sections of both of them are displayed in Table 3.2 and Table 3.3.

| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 8.5 | 12 | 16.5 | 13.8 | 15.5 | 18.7 | 10.8 |
| **2** | 8.5 | 0 | 3.7 | 8 | 5.3 | 7 | 10.2 | 5.7 |
| **3** | 12 | 3.7 | 0 | 7.7 | 9 | 9.5 | 12.7 | 7 |
| **4** | 16.5 | 8 | 7.7 | 0 | 4.3 | 12.3 | 13.7 | 13.7 |
| **5** | 13.8 | 5.3 | 9 | 4.3 | 0 | 8 | 11.5 | 10.2 |
| **6** | 15.5 | 7 | 9.5 | 12.3 | 8 | 0 | 4.7 | 9.2 |
| **7** | 18.7 | 10.2 | 12.7 | 13.7 | 11.5 | 4.7 | 0 | 11.7 |
| **8** | 10.8 | 5.7 | 7 | 13.7 | 10.2 | 9.2 | 11.7 | 0 |

Table 3.2: Shortest traveltimes between all node pairs.

| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|
| **1** | 1 | [1 2] | [1 12 3] | [1 2 4] | [1 2 5] | [1 2 6] | [1 2 7] | [1 8] |
| **2** | [2 1] | 2 | [2 3] | [2 4] | [2 5] | [2 6] | [2 7] | [2 8] |
| **3** | [3 12 1] | [3 2] | 3 | [3 4] | [3 2 5] | [3 6] | [3 7] | [3 8] |
| **4** | [4 2 1] | [4 2] | [4 3] | 4 | [4 5] | [4 5 6] | [4 7] | [4 2 8] |
| **5** | [5 2 1] | [5 2] | [5 2 3] | [5 4] | 5 | [5 6] | [5 7] | [5 8] |
| **6** | [6 2 1] | [6 2] | [6 3] | [6 5 4] | [6 5] | 6 | [6 7] | [6 8] |
| **7** | [7 2 1] | [7 2] | [7 3] | [7 4] | [7 5] | [7 6] | 7 | [7 8] |
| **8** | [8 1] | [8 2] | [8 3] | [8 2 4] | [8 5] | [8 6] | [8 7] | 8 |

Table 3.3: Shortest paths between all node pairs.

For the actual algorithm a TSP implementation from MathWorks [7] was used, from the same author as the Dijkstra implementation. It is slightly different than the standard TSP, as this implementation does not require the near optimal route to be a cycle, but rather a path. So the implementation finds the near optimal Hamiltonian path, which is exactly what we want. Note that the computed path will be a near optimal path instead of *the* optimal path, since solving

the TSP is an NP-complete problem. This means we can find different solutions that *approach* the optimal solution each time we run the program, because there is a random factor in the implementation. More about this is examined in Chapter 7.

We visualize the process of finding a Hamilton in our example from Section 3.1.1. We do now not need to start at node 1, but can also pick node 10 or 16. As it turns out, the shortest path is achieved by starting from node 10, as displayed in Figure 3.6, with a path length of 15, which is shorter than both lengths computed with ADA and SDA.

Figure 3.6: Example of the HPA.

A pseudocode description of the Hamiltonian Path Algorithm (HPA) is displayed in Algorithm 4. The full Matlab code is added in Appendix A.4.

**Input** List of $N$ mustpass nodes; traversal matrix between all the nodes;
**Output** Hamiltonian paths and traveltimes for all the patients;

Initialize table with results;
**for** *all patients* **do**
    Synchronize matrix dimensions to number of nodes per patient;
    Synchronize array dimensions to number of nodes per patient;
    **for** *nodes per patient* **do**
        Fill in $N \times 2$ matrix with mustpass nodes;
    **end**
    **for** *nodes per patient* **do**
        Fill in matrix with traveltimes between only the mustpass nodes;
        Make sure diagonal entries are 0;
    **end**
    Compute Hamiltonian path ;
    Display path and traveltime;
**end**

**Algorithm 4:** Pseudocode for Hamilton Path Algorithm

In the third for loop, we use a $N \times N$ matrix. By doing so, the algorithm has to look for shortest

paths in a $N \times N$ matrix, where $N \leq 32$, instead of the $110 \times 110$ matrix that both the SDA and ADA use. This is done to reduce the computational time of the HPA.

### 3.3.2 Results

Using the same data from Section 3.1.1, the HPA results in the traveltimes denoted in Figure 3.7. The full table with traveltimes is added in Appendix B.

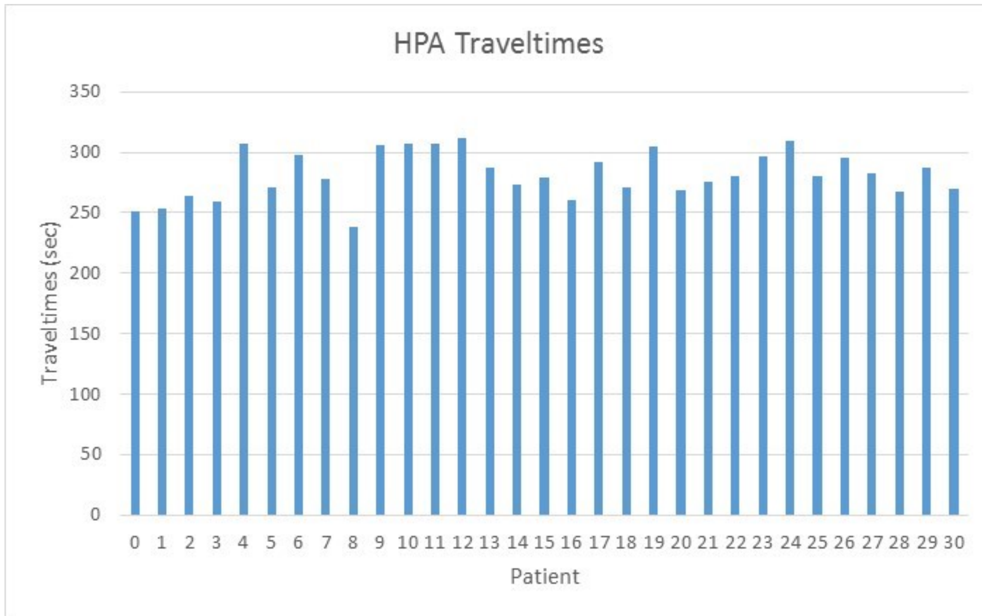



Figure 3.7: Traveltimes of 31 patients, computed with the HPA.

To compare results, a comparison of the HPA and ADA traveltimes is shown in Figure 3.8.

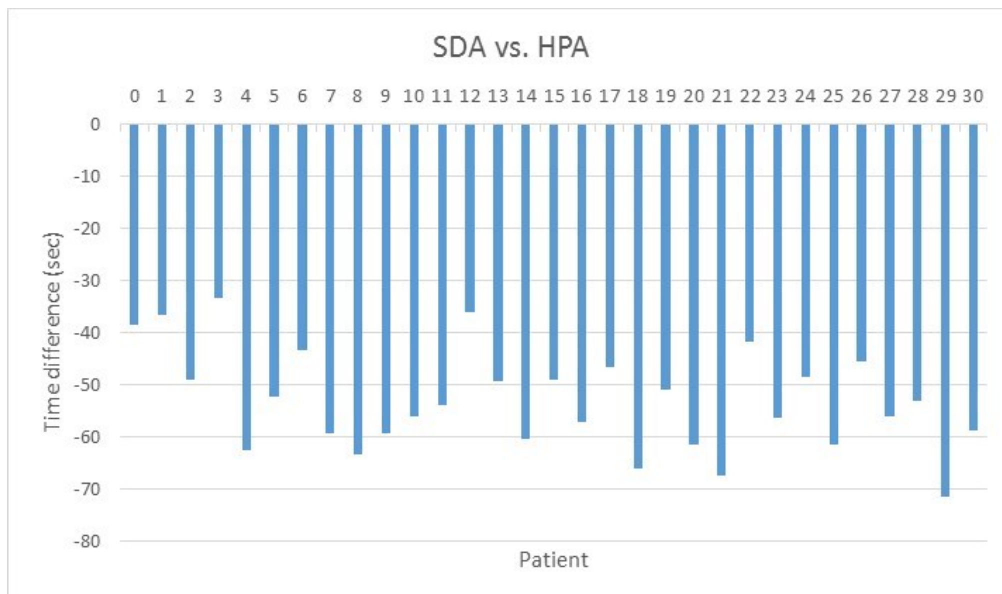



Figure 3.8: Time advantages of HPA over SDA for 31 patients.

We see more equal time advantages for all patients, with an average of 53.0 seconds , which translates to a 15.1% time advantage. This is already significantly better than the 4.4% of the SDA.

## 3.4 Comparison of methods

We have now discussed our methods for finding shortest paths. Out of all 3 of them, the HPA is clearly the best method. As seen in Figure 3.9, the method has a significant time advantage for every patient, with an average of 68.3 seconds, which translates to a 19.4% time advantage on average.

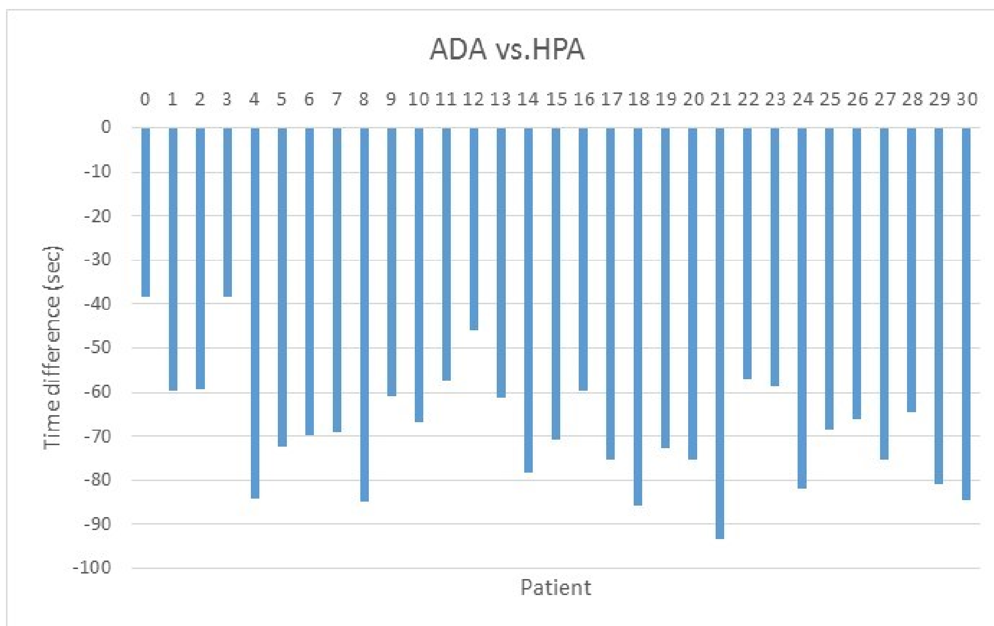



Figure 3.9: Time advantages of HPA over ADA for 31 patients.

For as far as computational time, the ADA takes less than one second per patient to compute the shortest path and traveltime, while the HPA takes around 20 seconds per patient to do so. So the HPA runs quite slower compared to the ADA, but this is no surprise, since Hamilton is a more powerful algorithm than Dijkstra. Since 20 seconds is still a very reasonable time for a program to run, we still favor the HPA over the ADA and the SDA. The full table with shorter paths is added in Appendix B.

# Chapter 4

# Neighbour Path Algorithm (NPA)

Now that we have answered the first two of our sub-questions, we can continue with the third sub-question: can we find shorter paths if we replace mustpass nodes with adjacent nodes that result in a better traveltime? We will answer this question in the section 4.1. Then, in Section 4.2 we will do a convergence analysis of our method. Lastly, in Section 4.3 we will talk about the plan quality of our newly found paths.

## 4.1 Computing Neighbour Paths

### 4.1.1 Methods

Our Neighbour Path Algorithm (NPA) takes the path computed with the HPA per patient as its standard path. It works in 2 basic steps:

1. Per node, find its neighbour nodes
2. Per node, replace the node with its neighbours, and check per neighbour if this results in a shorter path. If it does, replace the node with its neighbour.

To find neighbour nodes, we look at nodes that fall in a certain interval of degrees from the original node. We chose an interval of 15°, as we do not find any neighbours within 10°, and any number above 15 results in too many neighbours, which would take too much computational time. Choosing 15° usually leads to around 7 neighbours. The full Matlab code on choosing these neighbours is added in Appendix A.5.

Note that the NPA operates in a Greedy way. It only checks for a local minimum per node, and then moves on to the next node. More about this is examined in Chapter 7.

For computation of the shortest path after replacing a node with a neighbour node, we use Dijkstra, since the order of the mustpass nodes is fixed. This comes down to using the SDA. A pseudocode description is displayed in Algorithm 5. The full allgorithm is added in Appendix A.6.

```
Input A list of N mustpass nodes; traversal between all nodes;
Output Neighbour paths for all patients; traveltimes for all patients; potential time
  advantages;

Initialize empty tables for results;
for all patients do
    Compute the Hamiltonian path;
    Copy nodes of Hamiltonian path into new vector;
    Set initial shortest traveltime equal to HPA traveltime;
    for all N nodes in the Hamiltonian path do
        Find all neighbour nodes;
        Remember current node that is being replaced;
        for all neighbour nodes do
            if neighbour node is not already in Hamiltonian path then
                Replace current node with neighbour node;
                Compute traveltime of new path with Dijkstra;
                if Traveltime new path < initial traveltime then
                    Set new initial shortest traveltime equal to computed traveltime;
                    Set new initial shortest path equal to computed path;
                    Update current shortest path by replacing current node with neighbour node
                end
            end
        end
        If neighbour node does not result in a better path, set replaced node back to previously
          best neighbour node;
    end
    Enter results in tables;
end
```

**Algorithm 5:** Pseudocode for Neighbour Path Algorithm

### 4.1.2 Results

Using the same data from Section 3.1.1, the NPA results in the traveltimes denoted in Figure 4.1. The full table with results is added in Appendix B.

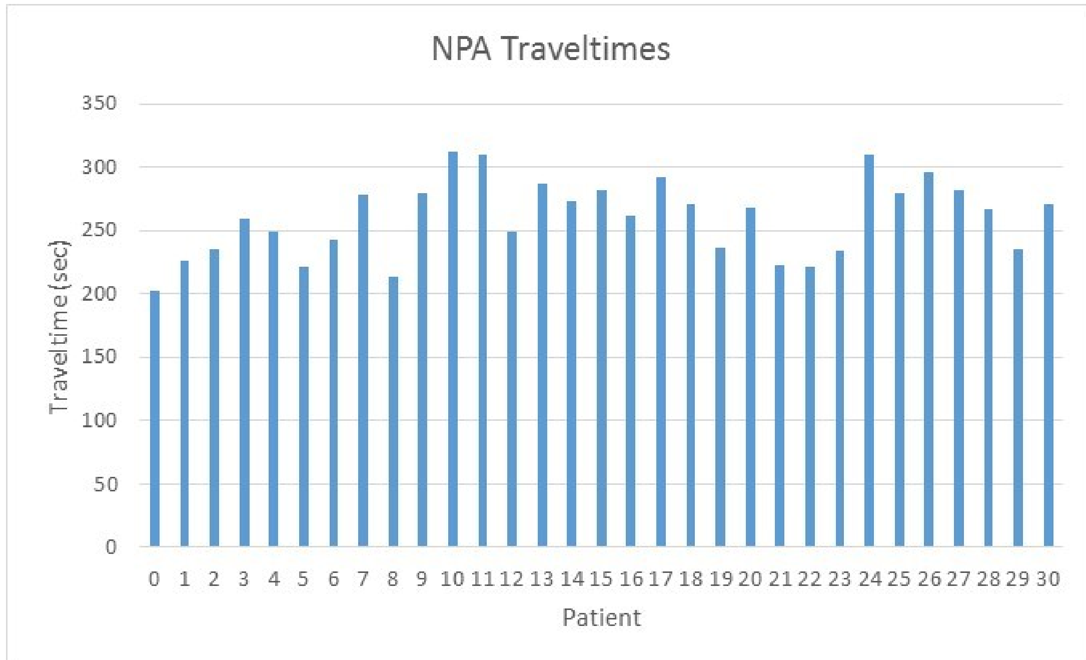



Figure 4.1: Traveltimes computed with NPA for 31 patients.

Again, we see some improvement in the traveltimes compared to those of the HPA. This is further illustrated in Figure 4.2

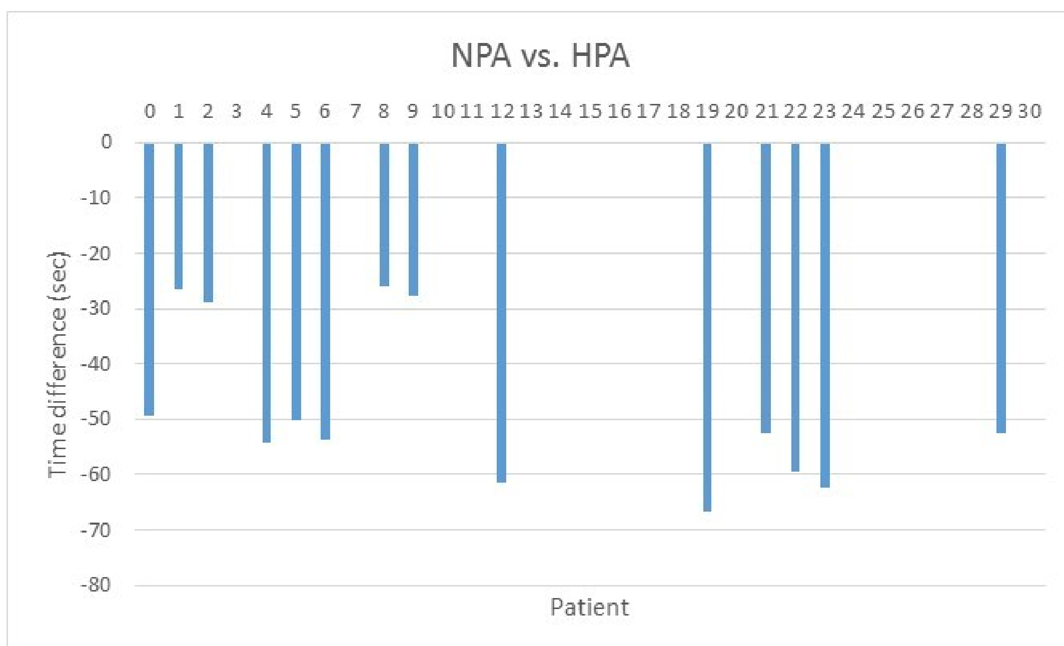


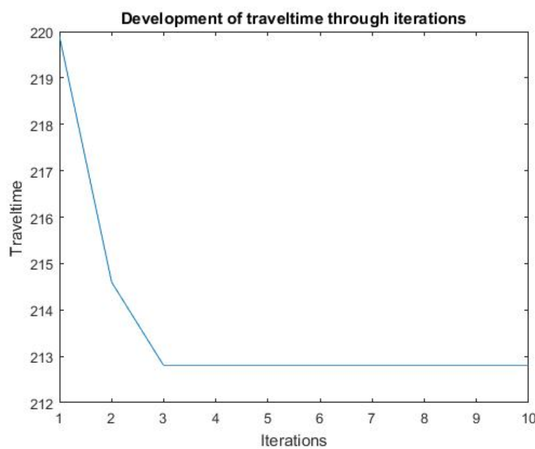

Figure 4.2: Time advantages of NPA over HPA for 31 patients.

For 13 out of 30 patients, the NPA results in an average time advantage of 48.0 seconds over the HPA, which translates to about 17.6%. On top of the 19.4% improvement the HPA made over the ADA, this means that the traveltimes for all 31 patients have been brought down already by 25.6% from the ADA already.

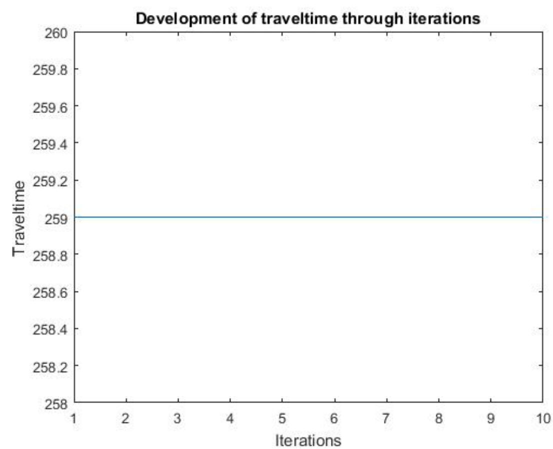
## 4.2 Analysis of Convergence

As noted before, there is a random factor in the HPA. This causes the computed path to differ a few seconds each time the program is run. This raises a question: do these traveltimes eventually do converge to an optimal traveltime?

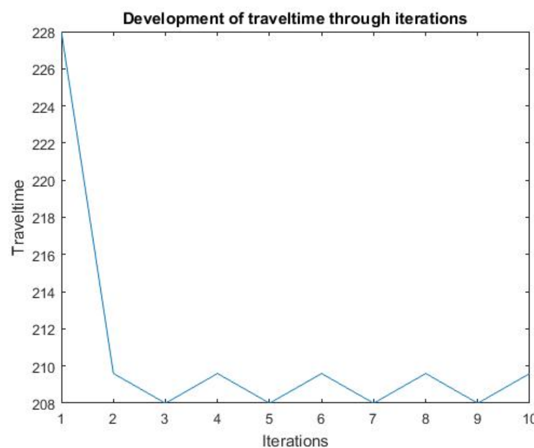
To check this hypothesis, we created the Iterated Neighbour Path Algorithm (INPA). It computes a shortest path with the NPA, takes the computed path as input and iterates the algorithm 10 times. This way, the path gets slightly altered after each iteration. The iterations resulted in three different pictures:




(a) Improvement until a few iterations.




(b) No improvement.




(c) Improvement at first, then alternating between times.

Figure 4.3: 10 Iterations for different patients.

While the the traveltimes in 4.3(a) and 4.3(b) seem to converge, the traveltimes in 4.3(c) show some more interesting behaviour. After two succesfull iterations, the traveltimes then start to alternate between 208 and 210 seconds. Unfortunately we do not have an explanation for this, but we can conclude that even though the traveltime does not always converge, it does not get any worse after a few iterations. It is fair to conclude that it is worth iterating the path found with the NPA at least 3-5 times. We should mention that there was a single case in which the traveltime got worse after one iteration and then alternated between two traveltimes. This specific result is displayed in Figure 4.4.

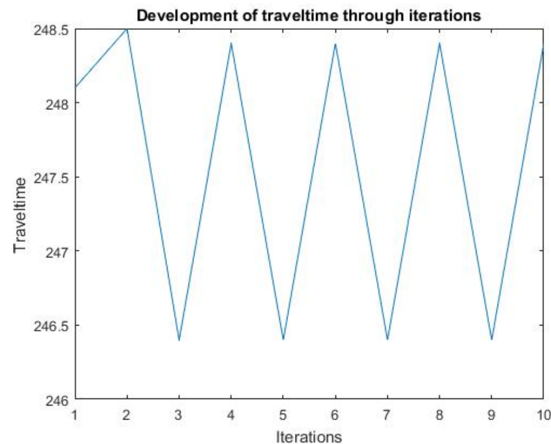



Figure 4.4: Case where the traveltime gets worse.

Note however that the degradation in time only concerns less than 0.5 seconds, which is not of any significance, while the improvement of the convergent cases can concern almost up to 20 seconds. Because of this, we will use the INPA as our next step in optimizing the traveltime. The INPA results in an average time advantage of 130.5 seconds over the ADA traveltimes, which translates to about 37.1%. This is illustrated in Figure 4.5.

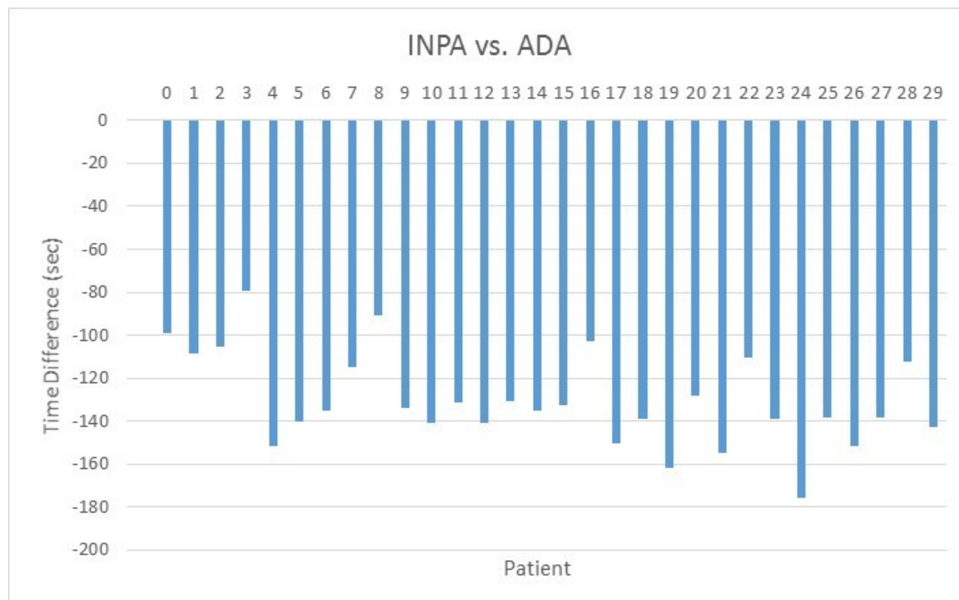



Figure 4.5: Time advantages of INPA over ADA for 31 patients.

The pseudocode for the INPA is similar to that of Algorithm 5, so it will not be included here. The full Matlab code is added in Appendix A.7.

## 4.3 Plan Quality

While a large section of this report is dedicated to shortest path optimization, it is also very important that these newly found paths do not degrade too much in plan quality, or else they will be useless. To verify this, Sebastiaan Breedveld did calculations on the plan quality each time after we found shorter paths. These calculations resulted in a Dosis-Volume-Histogram (DHV-Figure) for each patient. An example of a DHV-Figure is shown in Figure 4.6

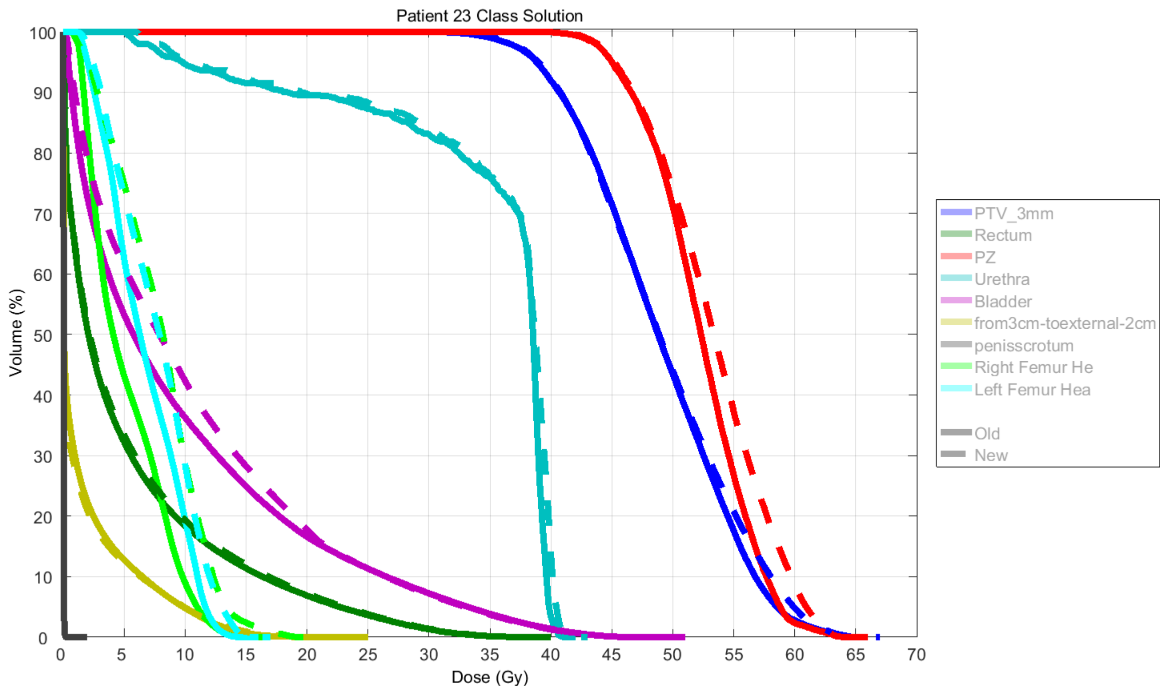



Figure 4.6: DHV-Figure for a patient.

The continuous lines represent the results with the original paths, while the dotted line represents the results with the convergent path found after 10 iterations. To not degrade in plan quality, it is important that the dotted lines stay as close as possible tot he continuous lines. As seen in Figure 4.6, most results remain the same, only the lower dose in the bladder degrades some in quality, while the high dose in the bladder improves slightly. The plan quality calculations showed that this result is quite usual for all of the patients. In fact, for some patients, the plan quality remained almost identical. This means that our optimized paths are appropriate to use, since the plan quality does not degrade significantly.

# Chapter 5

# Additional restrictions

With the INPA, we have removed all the restrictions from Section 2.1. However, as the project was progressing, two new restrictions came by: starting nodes and dummy nodes. We will discuss the nature of these restrictions and how we fixed them. We will combine these restrictions into one algorithm that will compute the final traveltimes for the patients.

## 5.1 Starting Nodes

There is a set of 45 nodes from which the CyberKnife can start its treatment. The traveltime would benefit if the first node of the path is starting node. Otherwise the CyberKnife would first have to travel to the first node of the path, which might be on the other side of the grid. This would harm the traveltime and be a waste of all time used to the compute the shortest path.

### 5.1.1 Methods

We took the result from the INPA, and added a starting node to the path. To incorporate these starting nodes in the paths, we had to distinguish three different cases of where the starting nodes can already be in the path:

1. There are no starting nodes in the path yet. The SNA then computes the traveltimes for all 45 starting nodes added to the path and picks the fastest. The path length increases with one.

2. There are starting nodes in the path, but not yet at the beginning of the path. The SNA puts all starting nodes in the path at the beginning, checks the travelime and picks the fastest. The path length does not change.

3. There are starting nodes in the path and one of them is already in the beginning. The SNA does not need to do anything.

## 5.2 Dummy Nodes

The last restriction that came along concerns dummy nodes. We briefly mentioned them in Section 1.2, but now we will fully incorporate them into the algorithm.

### 5.2.1 Methods

A brief recap, the dummy nodes are 8 nodes which are used for the sole purpose of serving as inbetween nodes. Radiation from these nodes is not possible. In the treatment plan, these nodes will never be candidate mustpass nodes. So it is only during the INPA, that we need to modify the algorithm, since they could be picked as neighbour nodes. Therefore we can use the INPA again, with the addition of one line of code that checks if the algorithm does not pick a dummy node as its neighbour for replacement. This is done in the same style as line 48 in Appendix A.7.

With the addition of a starting node and the removal of dummy nodes, we have now completed our final algorithm, the Optimal Path Algorithm (OPA). A pseudocode description of the OPA is presented in Figure 6. The full Matlab code is added in Appendix A.8.

## 5.3 Concluding Results

We can now compute the final traveltimes for all patients with the OPA. Using the same data from Section 3.1.1, the OPA results in the traveltimes denoted in Figure 5.1. The full table with results is added in Appendix B.

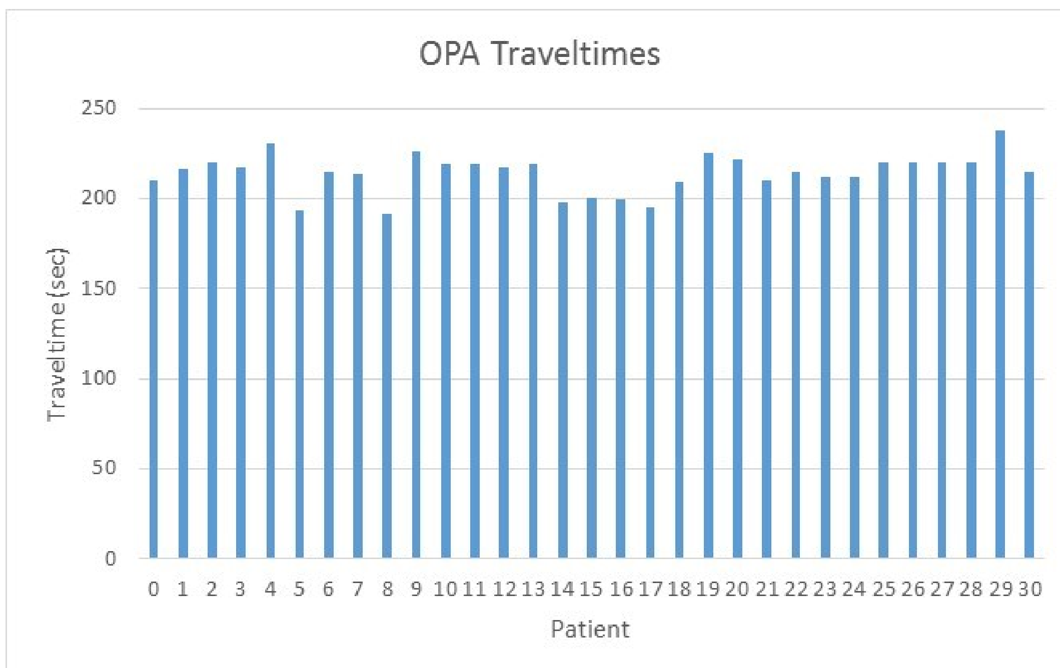



Figure 5.1: Traveltimes of 31 patients, computed with the OPA.

```
Input N Mustpass nodes for each patient; traversal matrix;
Output Shortest paths with starting nodes and without dummy nodes for all patients;
f Initialize empty tables for results;
Initialize dummy nodes and starting nodes;
for all patients do
    Compute the shortest path with the HPA;
    for 10 iterations do
        Compute the neighbour path with the NPA;
    end
    if first node of path is not a starting node then
        if there is a starting node in the path then
            Set all starting nodes in vector;
            Set shortest path distance equal to infinity;
            for all starting nodes in path do
                Define nodes that remain if starting node is placed in front;
                Compute shortest path and traveltime of remaining N − 1 nodes with HPA;
                if Traveltime new path with starting node < initial shortest path distance then
                    Save current starting node and path;
                end
            end
        else
            Set initial value of traveltime of starting node to first node equal to infinity;
            for all 45 starting nodes do
                Compute traveltime of starting node to first node of original path;
                if traveltime < initial value then
                    Save current starting node;
                end
            end
            Add best starting node to path;
        end
    else
        Program does nothing and keeps original path and traveltime;
    end
    Display results;
end
```

**Algorithm 6:** Pseudocode for the OPA.

To get a better view of our improvement, we compare the results of the OPA with the ADA, our first method of finding shortest paths. The results are illustrated in Figure 5.2.

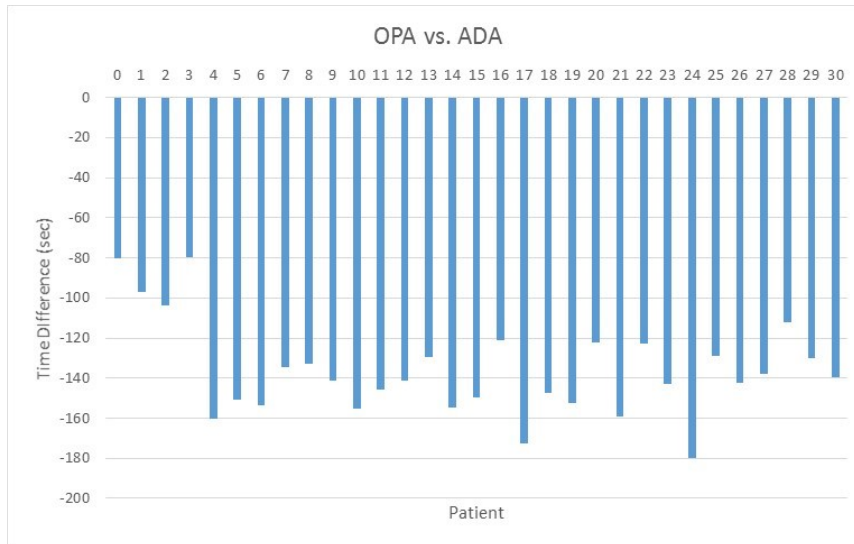



Figure 5.2: Time differences of OPA over ADA for 31 patients.

We see a notable drop in times for all patients, with an average of 136.2 seconds per patient. This translates to a 38.6% improvement on average, with a maximum of 46.9%. The full table with new paths for all 31 patients is added in Appendix C.

# Chapter 6

# Conclusions

The main goal of this project was to optimize the traveltime of the CyberKnife. We split up this goal into three sub-goals, as formulated in Section 2.1. After 2 months of research, we achieved the following results:

- With the SDA, the CyberKnife travels as fast as possible from node A to node B. We expanded this method with the HPA, which uses Hamiltonian paths. This changes the order of the mustpass nodes, which on average dropped the initial traveltimes by almost 20%. These results answered the first two of our research questions.
- Next up, we used the NPA to replace nodes in the path with adjacent nodes. This resulted in better paths for almost half of the patients, while the other half did not get worse. This result got sharpened when we noticed that iterating a new path into the NPA at least once did improve the traveltime. After a few iterations, all the paths were either converging, constant, or alternating between two better paths. So with the use of the INPA, our traveltimes had dropped 37.1% already. After these changes in the mustpass nodes, the plan quality did not lose a significant amount of quality. With these results, the last sub-question was answered as well.
- Lastly, we modified the INPA slightly, because of the addition of starting nodes and the removal of dummy nodes. This had to be done in order for the traveltime not to suffer under certain restrictions of the CyberKnife. There was an extra restriction concerning image blocking nodes, but we were not able to incorporate that restriction due to time limitations. More about this is explained in Chapter 7.

All of the previously mentioned algorithms resulted in the OPA, our final algorithm which optimized original the traveltimes with 38.6% on average, without any significant loss of plan quality.

# Chapter 7

# Recommendations

Further research on this project could be done by incorporating image blocking nodes. During the CyberKnife treatment, X-rays of the patient are made at regular intervals, to verify and/or update information on the patient's position. This is done by two X-ray detectors, as shown in Figure 7.1.

Figure 7.1: CyberKnife shown with the two X-ray detectors.

If the CyberKnife is irradiating from certain nodes, the linear accelerator or the robotic arm blocks the X-ray detectors. This causes the CyberKnife to retrieve from its current node, move to a position where it is not blocking the detectors, and then move the next node in the path. This could be prevented if the CyberKnife is always located at a detection-free node at the moment the detectors create an image. One way to do this would be to pin a node in the Hamilton implementation. This means that you could tell to the Hamilton implementation that the $k$-th node in the path has to be node 5 for example. The $k$-th node would then be the node where an image is made, and 5 would not be an image blocking node. We did not yet discover a

way to achieve this with the current implementation, but perhaps there are other more suitable implementations for this purpose.

The idea of pinning a node could also be of benefit in adding starting nodes. With the current version of the OPA, the algorithm has to identify the starting nodes, put them at the beginning and compute the traveltimes with HPA. It could save quite some for loops and thus computational time if this could be done by entering a few commands in the Hamilton implementation.

The NPA could also be more optimized. As mentioned in Section 4.1.1, the NPA operates in a Greedy way. It could for example be possible that replacing the first node with a sup-optimal neighbour results in a better overall path than replacing the first node with the optimal neighbour. Therefore the NPA is probably not the optimal method of computing neighbour solutions, but we could not come up with any better solutions. Also computational times started to rise quickly from that point, so heavier algorithms would have probably not fit in the time schedule.

One subject that was overlooked a bit is the 'random-factor' in the Hamilton implementation. We did not do any research into why Hamilton implementation result in different traveltimes each time. There also seems to be a little flaw in the OPA, as the path lengths of 4 patients were changed from 30 to 32. After we ran the OPA again for the 4 patients separately, the path length changed back to their original lengths. However, since we concluded that the plan quality did not suffer significantly under these changes, we kept the changed path lengths in Appendix C.

Lastly, the OPA takes about 40 minutes to compute traveltimes and paths for all 30 patients. With all the above mentioned improvements, it should be possible to reduce the 40 minutes computational time.

# Bibliography

[1] American Cancer Society. Radiation therapy basics. https://www.cancer.org/treatment/treatments-and-side-effects/treatment-types/radiation/basics.html, 2017.

[2] Sebastiaan Breedveld. Towards automated treatment planning in radiotherapy. *Erasmus MC*, 2013.

[3] Radiological Society of North America. Linac (linear accelerator). https://www.radiologyinfo.org/en/info.cfm?pg=linac, 2017.

[4] L. Rossi, S. Breedveld, S. Aluwini, and B.J.M Heijmen. On the beam direction search space in computerized non-coplanar beam angle optimization for imrt - prostate sbrt. *Erasmus MC*, 2012.

[5] Joseph Kirk. Dijkstra's minimum cost path algorithm. https://nl.mathworks.com/matlabcentral/fileexchange/20025-dijkstra-s-minimum-cost-path-algorithm, 2015.

[6] Christos H.Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization.* Dover Publications, Mineola, New York, 1998.

[7] Joseph Kirk. Fixed endpoints open traveling salesman problem - genetic algorithm. https://nl.mathworks.com/matlabcentral/fileexchange/21197-fixed-endpoints-open-traveling-salesman-problem-genetic-algorithm, 2014.

# Appendices

# Appendix A

# Matlab codes

## A.1 Asymmetrical Dijkstra Algorithm

```
1  clear all; close all; clc
2  load NodesForAllPatients
3  load patient1
4
5  A = cell(1,length(NodesForAllPatients));
6  for i = 1:length(NodesForAllPatients)
7      A{i} = sort(transpose(NodesForAllPatients{i}));
8  end
9
10 aptable = zeros(length(NodesForAllPatients),2);
11
12 for k = 1:length(NodesForAllPatients)
13     aptable(k,1) = k;
14     NodeTimesAsym = zeros(length(A{k})-1,1);
15     for i = 1:length(A{k}) - 1
16         [NodeTimesAsym(i),paths{i}] = PathDijkstra(TxTimeParams.
               NodeTraversalMap,TxTimeParams.NodeTraversalMap,A{k}(i),A{k
               }(i+1));
17     end
18     TotalNodeTimeAsym = sum(NodeTimesAsym);
19     aptable(k,2) = TotalNodeTimeAsym;
20 end
21 aptable
```

## A.2 Symmetrical Dijkstra Algorithm

```
1  clear all; close all; clc
2  load NodesForAllPatients
3  load patient1
4
5  A = cell(1,length(NodesForAllPatients));
```

```
6 for i = 1:length(NodesForAllPatients)
7     A{i} = sort(transpose(NodesForAllPatients{i}));
8 end
9
10 SymMat = TxTimeParams.NodeTraversalMap + transpose(TxTimeParams.
     NodeTraversalMap);
11 SymMat(end,end) = 0;
12
13 sptable = zeros(length(NodesForAllPatients),2);
14
15 for k = 1:length(NodesForAllPatients)
16     sptable(k,1) = k;
17     NodeTimesSym = zeros(length(A{k})-1,1);
18     for i = 1:length(A{k}) - 1
19         [NodeTimesSym(i),paths{i}] = PathDijkstra(SymMat,SymMat,A{k}(
             i),A{k}(i+1));
20     end
21     TotalNodeTimeSym = sum(NodeTimesSym);
22     sptable(k,2) = TotalNodeTimeSym;
23 end
24 sptable
```

## A.3 Total Matrices

```
1 TotalCosts = zeros(110,110);
2 TotalPaths = cell(110,110);
3
4 [costs, paths] = PathDijkstra(SymMat,SymMat);
5
6 for i = 1:110
7     for j = 1:110
8         TotalCosts(i,j) = costs(i,j);
9         TotalPaths(i,j) = paths(i,j);
10    end
11 end
```

## A.4 Hamiltonian Path Algorithm

```
1 clear all; close all; clc
2 load NodesForAllPatients
3 load patient1
4 load SymMat
5 run TotaleMatrices
6
7 hptable = zeros(length(NodesForAllPatients),2);
8
9 for k = 1:length(NodesForAllPatients)
10     hptable(k,1) = k;
```

```
11      m2 = zeros(length(NodesForAllPatients{k}),length(
            NodesForAllPatients{k}));
12      r2 = transpose(sort(NodesForAllPatients{k}));
13      rt2 = zeros(length(NodesForAllPatients{k}),2);
14      for m = 1:length(r2)
15          rt2(m,1) = r2(m);
16          rt2(m,2) = r2(m);
17      end
18      for j = 1:length(NodesForAllPatients{k})
19          for l = 1:length(NodesForAllPatients{k})
20              if j == l
21                  m2(j,l) = 0;
22              else
23                  m2(j,l) = TotalCosts(r2(j),r2(l));
24              end
25          end
26      end
27      userConfig = struct('xy',rt2,'dmat',m2);
28      resultStruct = tspo_ga(userConfig)
29      hptable(k,2) = resultStruct.minDist;
30  end
31  hptable
```

## A.5 Finding neighours

```
1  for k=1:size(Beams.Setup.Points, 1)
2      for j=1:size(Beams.Setup.Points, 1)
3          Angle(j, k) = acosd(sum(Beams.Setup.Points(k,:).*Beams.Setup.
               Points(j,:)));
4      end
5  end
6
7  Dummy = [12 16 20 22 26 57 73 77];
8  for j = Dummy
9      Angle = [Angle(:, 1:j-1) ones(size(Angle, 1), 1)*Inf Angle(:, j:
           end)];
10     Angle = [Angle(1:j-1, :); ones(1, size(Angle, 2))*Inf; Angle(j:
           end, :)];
11 end
```

## A.6 Neighbour Paths Algorithm

```
1  clear all; close all; clc
2  load patient1
3  load NodesForAllPatients
4  load SymMat
5  run hoektest
6  run TotaleMatrices
```

```
7
8 C = cell(length(NodesForAllPatients),1); %cell with the neighbour paths
9 nt = zeros(length(NodesForAllPatients),2); %table with traveltimes
10 ta = zeros(length(NodesForAllPatients),2); %table with the time advantages
11
12 for k = 1:length(NodesForAllPatients)
13     nt(k,1) = k;
14     ta(k,1) = k;
15     %Compute the shortest path for all patients with Hamilton
16     m2 = zeros(length(NodesForAllPatients{k}),length(NodesForAllPatients{k}));
17     r2 = transpose(sort(NodesForAllPatients{k}));
18     rt2 = zeros(length(NodesForAllPatients{k}),2);
19     for m = 1:length(r2)
20         rt2(m,1) = r2(m);
21         rt2(m,2) = r2(m);
22     end
23
24     for j = 1:length(NodesForAllPatients{k})
25         for l = 1:length(NodesForAllPatients{k})
26             if j == l
27                 m2(j,l) = 0;
28             else
29                 m2(j,l) = TotalCosts(r2(j),r2(l));
30             end
31         end
32     end
33     userConfig = struct('xy',rt2,'dmat',m2);
34     resultStruct = tspo_ga(userConfig);
35
36     %Compute the neighbour paths
37     inp = rt2(resultStruct.optRoute, 2); %copy the nodes of the Hamiltonpath into a new vector inp
38     ka = resultStruct.minDist; %set the initial shortest traveltime
39
40     for b = 1:length(inp) %check all the nodes in the Hamiltonpath
41         f = find(Angle(:,inp(b))<15); %find the neighbours of a specified node
42         bestNode = inp(b); %remember the node that is being replaced
43         for j = 1:length(f) %check all the neighbours of a node
44             if ~ismember(f(j), inp) %make sure the new neighbour node is not already in the path
45                 inp(b) = f(j); %replace the specified node with the neighbours, and calculate the new path length
46                  for i = 1:length(inp) - 1
47                      [NodeTimesSym(i),paths{i}] = PathDijkstra(SymMat,SymMat,inp(i),inp(i+1)); %calculate shortest
```

```
                    path with Dijkstra
48              end
49              TotalNodeTimeSym = sum(NodeTimesSym);
50              if TotalNodeTimeSym < ka
51                  ka = TotalNodeTimeSym; %if one the neighbour
                        nodes results in a shorter path, the minimal
                        distance is updated
52                  kp = inp; % same for the minimal path
53                  bestNode = inp(b); %if the new node results in a
                        shorter path, update the best node
54              end
55          end
56      end
57      inp(b) = bestNode; %if the new node does not result in a
            shorter path, set the beste Node back to the previously
            found best node
58  end
59  C{k} = kp;
60  nt(k,2) = ka;
61  ta(k,2) = resultStruct.minDist - ka;
62 end
```

## A.7 Iterated Neighbour Path Algorithm

```
1  clear all; close all; clc
2  load patient1
3  load NodesForAllPatients
4  load SymMat
5  run TotaleMatrices
6  run hoektest
7
8  a = 10;
9
10 TR = cell(length(NodesForAllPatients),1);
11 PR = cell(length(NodesForAllPatients),1);
12
13 for k = 1:1
14     %Compute the shortest path for all patients with Hamilton
15     m2 = zeros(length(NodesForAllPatients{k}),length(
           NodesForAllPatients{k}));
16     r2 = transpose(sort(NodesForAllPatients{k}));
17     rt2 = zeros(length(NodesForAllPatients{k}),2);
18     for m = 1:length(r2)
19         rt2(m,1) = r2(m);
20         rt2(m,2) = r2(m);
21     end
22
23     for j = 1:length(NodesForAllPatients{k})
24         for l = 1:length(NodesForAllPatients{k})
```

```
25              if j == l
26                  m2(j,l) = 0;
27              else
28                  m2(j,l) = TotalCosts(r2(j),r2(l));
29              end
30          end
31      end
32      userConfig = struct('xy',rt2,'dmat',m2);
33      resultStruct = tspo_ga(userConfig);
34
35      inp = rt2(resultStruct.optRoute, 2); %copy the nodes of the
           Hamiltonpath into a new vector inp
36      ka = resultStruct.minDist; %set the initial shortest distance
37
38      for c = 1:a
39          TR{k}(c,1) = c;
40          %Compute the neighbour paths
41          for b = 1:length(inp) %check all the nodes in the
               Hamiltonpath
42              f = find(Angle(:,inp(b))<15); %find the neighbours of a
                   specified node
43              bestNode = inp(b); %remember the node that is being
                   replaced
44              for j = 1:length(f) %check all the neighbours of a node
45                  if ~ismember(f(j), inp) %make sure the new neighbour
                       node is not already in the path
46                      inp(b) = f(j); %replace the specified node with
                           the neighours, and calculate the new path
                           length
47                      for i = 1:length(inp) - 1
48                          [NodeTimesSym(i),paths{i}] = PathDijkstra(
                               SymMat,SymMat,inp(i),inp(i+1)); %
                               calculate shortest path with Dijkstra
49                      end
50                      TotalNodeTimeSym = sum(NodeTimesSym);
51                      if TotalNodeTimeSym < ka
52                          ka = TotalNodeTimeSym; %if one the neighbour
                               nodes results in a shorter path, the
                               minimal distance is updated
53                          kp = inp; % same for the minimal path
54                          bestNode = inp(b); %if the new node results
                               in a shorter path, update the best node
55                      end
56                  end
57              end
58              inp(b) = bestNode; %if the new node does not result in a
                   shorter path, set the beste Node back to the
                   previously found best node
59          end
```

```
60          TR{c}(c,2) = ka;
61          PR{k} = kp;
62          inp = kp; %update input with the output of the previous path
63      end
64  end
```

## A.8 Optimal Path Algorithm

```
1  clear all; close all; clc
2  load patient1
3  load NodesForAllPatients
4  load SymMat
5  run TotaleMatrices
6  run hoektest
7
8  a = 10;
9  ft = zeros(length(NodesForAllPatients),1); %create table for final
       traveltimes for all the patients
10 snp = cell(length(NodesForAllPatients),1); %cell with the starting
       node paths
11 startnodes = [1 26 28 29 30 36:51 60 76 78 79:83 85:93 96:100 102
       110];
12 dummy = [12 16 20 22 26 57 73 77]; %these nodes cannot be in the path
13
14 for k = 1:length(NodesForAllPatients)
15     m2 = zeros(length(NodesForAllPatients{k}),length(
           NodesForAllPatients{k}));
16     r2 = transpose(sort(NodesForAllPatients{k}));
17     rt2 = zeros(length(NodesForAllPatients{k}),2);
18     for m = 1:length(r2)
19         rt2(m,1) = r2(m);
20         rt2(m,2) = r2(m);
21     end
22
23     for j = 1:length(NodesForAllPatients{k})
24         for l = 1:length(NodesForAllPatients{k})
25             if j == l
26                 m2(j,l) = 0;
27             else
28                 m2(j,l) = TotalCosts(r2(j),r2(l));
29             end
30         end
31     end
32     userConfig = struct('xy',rt2,'dmat',m2);
33     resultStruct = tspo_ga(userConfig);
34
35     inp = rt2(resultStruct.optRoute, 2); %copy the nodes of the
           Hamiltonpath into a new vector inp
36     ka = resultStruct.minDist; %set the initial shortest distance
```

```
37
38      for c = 1:a
39          %Compute the neighbour paths
40          for b = 1:length(inp) %check all the nodes in the
                Hamiltonpath
41              f = find(Angle(:,inp(b))<15); %find the neighbours of a
                    specified node
42              bestNode = inp(b); %remember the node that is being
                    replaced
43              for j = 1:length(f) %check all the neighbours of a node
44                  if ~ismember(f(j), inp) %make sure the new neighbour
                        node is not already in the path
45                      if ~ismember(f(j),dummy) %make sure the new
                            neighbour node is not a dummy node
46                          inp(b) = f(j); %replace the specified node
                                with the neighours, and calculate the new
                                path length
47                          for i = 1:length(inp) - 1
48                              [NodeTimesSym(i),paths{i}] =
                                    PathDijkstra(SymMat,SymMat,inp(i),inp
                                    (i+1)); %calculate shortest path with
                                     Dijkstra
49                          end
50                          TotalNodeTimeSym = sum(NodeTimesSym);
51                          if TotalNodeTimeSym < ka
52                              ka = TotalNodeTimeSym; %if one the
                                    neighbour nodes results in a shorter
                                    path, the minimal distance is updated
53                              kp = inp; % same for the minimal path
54                              bestNode = inp(b); %if the new node
                                    results in a shorter path, update the
                                     best node
55                          end
56                      end
57                  end
58              end
59              inp(b) = bestNode; %if the new node does not result in a
                    shorter path, set the beste Node back to the
                    previously found best node
60          end
61          inp = kp; %update input with the output of the previous path
62      end
63
64      %%%%%%%%%%%%%%%%%%Add a starting node
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
65      if isempty(intersect(kp(1),startnodes)) == 1 %checks if the first
            node of the computed path is not a start node
66          if ~isempty(intersect(startnodes,kp)) %scenario #1: there is
                a starting node in the computed shortest path
```

```
67            is = intersect(startnodes,kp); %define length of
                  intersection of startnodes and the computed shortest
                  path
68            spd = inf; %set the shortest path distance equal to
                  infinity
69            for i = 1:length(is)
70                newnodes = setdiff(kp, is(i)); %define the set of
                      nodes without the starting node
71                m2 = zeros(length(newnodes),length(newnodes));
72                r2 = transpose(sort(newnodes));
73                rt2 = zeros(length(newnodes),2);
74                for m = 1:length(r2)
75                    rt2(m,1) = r2(m);
76                    rt2(m,2) = r2(m);
77                end
78                for j = 1:length(newnodes)
79                    for l = 1:length(newnodes)
80                        if j == l
81                            m2(j,l) = 0;
82                        else
83                            m2(j,l) = TotalCosts(r2(j),r2(l));
84                        end
85                    end
86                end
87                userConfig = struct('xy',rt2,'dmat',m2);
88                resultStruct = tspo_ga(userConfig);
89                if costs(is(i),rt2(resultStruct.optRoute(1))) +
                      resultStruct.minDist < spd %find the best of all
                      starting nodes
90                    spd = costs(is(i),rt2(resultStruct.optRoute(1)))
                          + resultStruct.minDist;
91                    bestStart = is(i);
92                    bestRoute = rt2(resultStruct.optRoute);
93                end
94            end
95            newRoute = [bestStart bestRoute];
96            ft(k) = spd
97            snp{k} = newRoute;
98            newRoute
99        else %scenario #2: there are no starting nodes in the
              computed shortest path
100           sfc = inf; %set the Shortest Connection to the First node
                  equal to infinity
101           for n = 1:length(startnodes)
102               ck = costs(startnodes(n),kp(1)); %compute the costs
                      of the k-th starting node to the first node of kp
103               if ck < sfc
104                   sfc = ck;
105                   bestStart = startnodes(n);
```

```
106                     end
107                 end
108                 kp = [bestStart kp'];
109                 ft(k) = ka + sfc
110                 snp{k} = kp;
111                 kp
112             end
113         else %if the first node of the original path is already a
            starting node, the program does not need to do anything, and
            returns the original path
114             ft(k) = ka
115             snp{k} = kp;
116             kp
117         end
118         close all;
119 end
```

# Appendix B

# Traveltimes of all algorithms

| Patient | ADA | SDA | HPA | NPA | INPA | OPA |
|---|---|---|---|---|---|---|
| **0** | 289.9 | 283.7 | 251.4 | 202 | 191.2 | 209.8 |
| **1** | 312.8 | 289.9 | 253.2 | 226.1 | 204.6 | 215.9 |
| **2** | 323.6 | 313.5 | 264.4 | 235.6 | 218 | 219.9 |
| **3** | 297.4 | 292.2 | 259 | 259.4 | 218 | 217.6 |
| **4** | 390.9 | 369 | 306.6 | 249.3 | 239.4 | 230.6 |
| **5** | 343.8 | 323.6 | 271.3 | 221.2 | 203.5 | 193.3 |
| **6** | 368 | 341.4 | 298.1 | 243 | 232.9 | 214.3 |
| **7** | 347.7 | 337.8 | 278.5 | 278.5 | 232.9 | 213.3 |
| **8** | 323.7 | 302.2 | 238.8 | 213.3 | 232.9 | 191.1 |
| **9** | 366.7 | 365.2 | 305.8 | 279 | 232.9 | 225.7 |
| **10** | 373.9 | 363.2 | 307.2 | 312.8 | 232.9 | 218.6 |
| **11** | 364.2 | 360.6 | 306.8 | 309.9 | 232.9 | 218.6 |
| **12** | 358.4 | 348.2 | 312.3 | 249.4 | 217.6 | 217.1 |
| **13** | 348.1 | 336.2 | 286.9 | 286.9 | 217.3 | 218.8 |
| **14** | 352.2 | 334.2 | 273.8 | 273.4 | 217.3 | 197.7 |
| **15** | 349.6 | 327.8 | 278.9 | 281.7 | 217.3 | 200 |
| **16** | 320.2 | 317.7 | 260.6 | 261.7 | 217.3 | 199.1 |
| **17** | 367.8 | 339 | 292.5 | 291.7 | 217.3 | 195.2 |
| **18** | 356.3 | 336.7 | 270.6 | 270.6 | 217.3 | 208.8 |
| **19** | 377.6 | 355.9 | 305 | 237 | 215.7 | 224.9 |
| **20** | 344.1 | 330.2 | 268.8 | 268.8 | 215.7 | 222 |
| **21** | 368.7 | 343 | 275.5 | 222.9 | 213.9 | 209.7 |
| **22** | 337.4 | 322 | 280.3 | 220.9 | 227 | 214.8 |
| **23** | 355.2 | 352.8 | 296.5 | 234.6 | 216 | 212 |
| **24** | 391.6 | 358.2 | 309.6 | 309.6 | 216 | 212 |
| **25** | 348.7 | 341.6 | 280.1 | 280.1 | 210.7 | 219.7 |
| **26** | 362.2 | 341.4 | 295.9 | 295.9 | 210.7 | 219.7 |
| **27** | 357.7 | 338.5 | 282.4 | 282.4 | 219.7 | 219.7 |
| **28** | 331.8 | 320.3 | 267.3 | 267.3 | 219.7 | 219.7 |
| **29** | 368 | 358.6 | 287.2 | 234.8 | 225.5 | 237.7 |
| **30** | 354.8 | 329.1 | 270.3 | 270.3 | 223.7 | 215 |

# Appendix C

# New paths

| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 50 | 60 | 79 | 102 | 79 | 50 | 48 | 76 | 47 | 1 | 82 | 82 | 50 | 50 | 40 | 82 |
| 56 | 75 | 63 | 1 | 83 | 52 | 96 | 75 | 49 | 2 | 83 | 83 | 53 | 53 | 32 | 19 |
| 58 | 71 | 71 | 14 | 40 | 55 | 97 | 71 | 48 | 3 | 24 | 24 | 55 | 55 | 31 | 18 |
| 61 | 69 | 70 | 15 | 39 | 59 | 98 | 69 | 50 | 14 | 40 | 40 | 52 | 52 | 30 | 28 |
| 66 | 64 | 64 | 2 | 75 | 62 | 99 | 70 | 56 | 15 | 39 | 39 | 59 | 59 | 76 | 58 |
| 64 | 66 | 68 | 3 | 71 | 64 | 101 | 64 | 59 | 11 | 38 | 38 | 62 | 62 | 75 | 59 |
| 71 | 68 | 66 | 8 | 70 | 70 | 108 | 65 | 58 | 10 | 31 | 31 | 61 | 61 | 70 | 56 |
| 76 | 58 | 65 | 9 | 64 | 69 | 107 | 62 | 61 | 109 | 76 | 76 | 69 | 64 | 64 | 62 |
| 41 | 56 | 62 | 109 | 65 | 71 | 109 | 59 | 62 | 108 | 75 | 75 | 70 | 70 | 65 | 65 |
| 40 | 59 | 58 | 108 | 72 | 75 | 10 | 56 | 65 | 107 | 70 | 70 | 71 | 71 | 66 | 64 |
| 24 | 52 | 56 | 99 | 61 | 76 | 14 | 58 | 64 | 104 | 62 | 62 | 72 | 72 | 62 | 70 |
| 25 | 50 | 59 | 98 | 58 | 30 | 15 | 79 | 70 | 103 | 59 | 59 | 75 | 75 | 59 | 71 |
| 18 | 48 | 52 | 97 | 56 | 40 | 2 | 83 | 71 | 101 | 55 | 55 | 40 | 40 | 52 | 72 |
| 17 | 49 | 50 | 46 | 59 | 79 | 3 | 85 | 75 | 47 | 53 | 53 | 24 | 18 | 56 | 75 |
| 2 | 18 | 49 | 87 | 52 | 24 | 6 | 84 | 76 | 50 | 52 | 52 | 18 | 24 | 24 | 40 |
| 5 | 80 | 18 | 79 | 50 | 82 | 17 | 18 | 79 | 52 | 50 | 50 | 83 | 83 | 79 | 83 |
| 3 | 99 | 24 | 24 | 49 | 83 | 18 | 17 | 28 | 53 | 47 | 47 | 85 | 85 | 47 | 42 |
| 15 | 101 | 87 | 18 | 48 | 42 | 84 | 48 | 82 | 55 | 97 | 97 | 87 | 42 | 97 | 98 |
| 14 | 104 | 46 | 49 | 47 | 47 | 85 | 96 | 83 | 59 | 101 | 101 | 42 | 87 | 99 | 99 |
| 10 | 106 | 97 | 50 | 97 | 99 | 83 | 97 | 99 | 58 | 103 | 103 | 99 | 99 | 102 | 101 |
| 108 | 107 | 98 | 52 | 99 | 98 | 79 | 98 | 102 | 28 | 104 | 104 | 102 | 102 | 101 | 103 |
| 101 | 109 | 99 | 59 | 101 | 101 | 76 | 99 | 101 | 79 | 106 | 106 | 101 | 101 | 103 | 104 |
| 102 | 9 | 102 | 56 | 107 | 103 | 75 | 101 | 108 | 83 | 107 | 107 | 103 | 103 | 107 | 107 |
| 107 | 8 | 108 | 58 | 10 | 107 | 71 | 108 | 107 | 76 | 10 | 10 | 104 | 104 | 109 | 10 |
| 104 | 3 | 109 | 62 | 15 | 109 | 69 | 107 | 109 | 75 | 14 | 14 | 107 | 107 | 9 | 11 |
|  | 2 | 9 | 65 | 13 | 9 | 70 | 109 | 10 | 72 | 13 | 13 | 108 | 108 | 8 | 14 |
|  | 15 | 8 | 66 | 14 | 8 | 64 | 10 | 14 | 69 | 15 | 15 | 10 | 10 | 2 | 15 |
|  | 11 | 3 | 68 | 2 | 15 | 65 | 14 | 3 | 70 | 2 | 2 | 14 | 14 | 3 | 2 |
|  | 14 | 2 | 64 | 5 | 14 | 62 | 15 | 2 | 71 | 3 | 3 | 15 | 15 | 14 | 3 |
|  | 13 | 15 | 70 | 3 | 3 | 59 | 2 | 5 | 30 | 6 | 6 | 2 | 2 | 15 | 6 |
|  |  | 14 | 71 | 8 | 2 | 56 | 3 |  |  |  |  | 3 | 3 |  |  |
|  |  | 1 | 63 | 11 | 5 | 58 | 6 |  |  |  |  | 4 | 4 |  |  |

| **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 60 | 39 | 51 | 42 | 102 | 99 | 83 | 51 | 51 | 83 | 83 | 30 | 30 | 76 | 83 |
| 53 | 96 | 58 | 2 | 8 | 97 | 6 | 52 | 52 | 6 | 6 | 76 | 76 | 108 | 13 |
| 55 | 97 | 56 | 15 | 15 | 101 | 2 | 55 | 55 | 3 | 3 | 75 | 75 | 103 | 14 |
| 59 | 99 | 59 | 14 | 14 | 103 | 3 | 53 | 53 | 2 | 2 | 72 | 72 | 107 | 15 |
| 65 | 101 | 60 | 11 | 11 | 108 | 14 | 59 | 59 | 15 | 15 | 71 | 71 | 109 | 2 |
| 64 | 103 | 65 | 10 | 10 | 10 | 15 | 62 | 62 | 14 | 14 | 70 | 70 | 9 | 8 |
| 70 | 104 | 66 | 109 | 109 | 9 | 8 | 64 | 64 | 10 | 10 | 64 | 64 | 10 | 9 |
| 69 | 106 | 64 | 107 | 107 | 109 | 9 | 70 | 70 | 109 | 109 | 65 | 65 | 1 | 109 |
| 72 | 107 | 70 | 106 | 106 | 1 | 109 | 71 | 71 | 107 | 107 | 62 | 62 | 14 | 108 |
| 75 | 108 | 71 | 108 | 108 | 14 | 108 | 75 | 75 | 102 | 102 | 58 | 58 | 15 | 107 |
| 76 | 109 | 30 | 101 | 101 | 3 | 107 | 72 | 72 | 101 | 101 | 59 | 59 | 6 | 104 |
| 30 | 1 | 29 | 102 | 99 | 4 | 104 | 28 | 28 | 98 | 98 | 56 | 56 | 3 | 103 |
| 39 | 14 | 79 | 99 | 98 | 85 | 103 | 79 | 79 | 99 | 99 | 50 | 50 | 2 | 101 |
| 40 | 15 | 83 | 98 | 42 | 83 | 101 | 82 | 82 | 47 | 47 | 48 | 48 | 84 | 99 |
| 79 | 2 | 85 | 83 | 83 | 79 | 102 | 83 | 83 | 40 | 40 | 18 | 18 | 21 | 98 |
| 83 | 84 | 87 | 82 | 82 | 24 | 99 | 40 | 40 | 24 | 24 | 80 | 80 | 18 | 88 |
| 85 | 79 | 96 | 24 | 25 | 23 | 42 | 42 | 42 | 18 | 18 | 85 | 85 | 23 | 87 |
| 87 | 40 | 95 | 25 | 24 | 18 | 40 | 87 | 87 | 82 | 82 | 87 | 87 | 24 | 42 |
| 98 | 28 | 97 | 40 | 40 | 49 | 24 | 98 | 98 | 79 | 79 | 103 | 103 | 79 | 41 |
| 99 | 58 | 99 | 30 | 30 | 52 | 18 | 99 | 99 | 76 | 76 | 102 | 102 | 28 | 40 |
| 102 | 59 | 101 | 75 | 75 | 56 | 82 | 101 | 101 | 71 | 71 | 101 | 101 | 72 | 29 |
| 101 | 60 | 107 | 72 | 72 | 59 | 79 | 103 | 103 | 64 | 64 | 108 | 108 | 75 | 30 |
| 103 | 65 | 109 | 71 | 71 | 58 | 76 | 104 | 104 | 65 | 65 | 109 | 109 | 71 | 70 |
| 107 | 63 | 10 | 69 | 69 | 61 | 75 | 106 | 106 | 66 | 66 | 9 | 9 | 70 | 64 |
| 109 | 64 | 14 | 70 | 70 | 62 | 72 | 107 | 107 | 62 | 62 | 10 | 10 | 64 | 69 |
| 1 | 70 | 15 | 64 | 64 | 65 | 70 | 108 | 108 | 59 | 59 | 15 | 15 | 65 | 68 |
| 14 | 71 | 1 | 65 | 65 | 64 | 64 | 10 | 10 | 56 | 56 | 14 | 14 | 62 | 66 |
| 3 | 72 | 2 | 62 | 62 | 70 | 68 | 11 | 11 | 55 | 55 | 2 | 2 | 58 | 56 |
| 2 | 75 | 3 | 59 | 59 | 71 | 66 | 14 | 14 | 54 | 54 | 8 | 8 | 59 | 59 |
| 6 | 76 | 6 | 56 | 56 | 72 | 65 | 2 | 2 | 53 | 53 | 13 | 13 | 56 | 58 |
|  |  |  | 51 | 51 | 75 | 59 | 3 | 3 |  |  |  |  | 55 |  |
|  |  |  | 52 | 52 | 76 | 55 | 15 | 15 |  |  |  |  | 53 |  |