Multileaf collimator optimisation for column generation in radiotherapy

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 Tuesday July 1, 2014 at 2:00 PM.

Copyright © 2014 by R. N. Jacobs. All rights reserved
MULTILEAF COLLIMATOR OPTIMISATION FOR COLUMN GENERATION IN RADIOTHERAPY

by

R. N. Jacobs

Supervisors: Dr. ir. S. Breedveld, Erasmus MC
Dr. ir. M. Keijzer, TU Delft

Thesis committee: Dr. ing. F. Jeltsema, TU Delft
Drs. E. M. van Elderen, TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.
**Abstract**

Radiation therapy is a form of therapy for cancer treatments, where ionizing radiation is used to eradicate malignant cells. The patient is treated with a device called a linear particle accelerator. It contains a source, from which photons are emitted. Constructing an optimal treatment plan for radiation therapy involves optimising the angles from which the patient is irradiated. Other degrees of freedom arise from the use of a multileaf collimator, allowing beam shaping.

It is quite common that the tumour lies between other organs. With the multileaf collimator we can adjust the beam shape in such a way that we expose the healthy organs to a minimal amount of radiation. There are many shapes we cannot make using the multileaf collimator.

Column generation is a method for mathematical optimisation, that can be used to optimise treatment plans for radiation therapy. At some point during the column generation process, a suitable aperture must be found. This aperture is restricted because of the requirements of the multileaf collimator.

In this project we have developed an algorithm that finds a suitable aperture for the column generation process. At first this algorithm creates an optimal aperture, while only taking some basic requirements of the multileaf collimator into account. Afterwards, the aperture found is adapted into an aperture that complies with all common requirements of multileaf collimators.

The algorithm that we have developed, does not guarantee an optimal aperture. Even though better apertures might in the end deliver a better treatment plan, it is not a necessity. The column generation process can correct itself in the next step. How the column generation process responds to the apertures created during this research project, has not yet been tested.
This report is the result of my bachelor project. This bachelor project is part of my bachelor Technische Wiskunde at the University of Technology Delft. For this project I have done an internship at Erasmus MC at the Department of Radiation Oncology, Medical Physics.

I found this project very interesting and have worked on it with joy every day. The environment at Erasmus MC was also a positive stimulant. Especially the daily lunches with the other interns and staff members made working on this project a pleasure.

Through this preface I want to thank Sebastiaan Breedveld and Marleen Keijzer for their supervision and guidance. I also want to thank Dion Gijswijt, for his ideas on creating an optimal solution. Furthermore I want to thank Frank Tabak and Ronald van Etten for their support and suggestions.

Finally I want to wish the readers of this report a pleasant read.

R. N. Jacobs
Delft, June 2014
# CONTENTS

1 Introduction .......................... 1
   1.1 General overview .......................... 2

2 Multileaf Collimator .................. 3
   2.1 Mathematical representation for the multileaf collimator .................. 3
   2.2 Restrictions of multileaf collimators .......................... 5
   2.3 Step-and-shoot versus dynamic radiation therapy .................. 6

3 Problem formulation ................. 9
   3.1 Column Generation .......................... 9
   3.2 Setup of the research project .................. 10

4 Constructing a solution ............. 11
   4.1 Reduction to directed graphs .......................... 11
   4.2 Bellman-Ford .......................... 13
   4.3 Column generation matrix .......................... 14
   4.4 Results of the Bellman-Ford algorithm .................. 15

5 Solutions when positive elements are allowed .... 19
   5.1 Results first extension Bellman-Ford algorithm .................. 20

6 Solutions when multiple openings are allowed .... 23
   6.1 Results second extension Bellman-Ford algorithm .................. 25

7 Creating a feasible solution ......... 27
   7.1 Method I: Add a margin around the solution .................. 28
   7.2 Method II: Remove a margin from the domain, then add it again after optimisation .................. 28
   7.3 Method III: Remove a margin from the optimised aperture, then add a margin .................. 28
   7.4 Results of correcting the solutions .................. 29

8 Results for all 45 cases ............ 31

9 Spot jumping .......................... 35
   9.1 Examples of currently not optimally solvable cases .................. 35
   9.2 Spot Jumping .......................... 36

10 Conclusion .......................... 39

A Tables with results of Bellman-Ford and extensions .... 41

B Images of 4 test cases ............... 45

C Matlab files .......................... 51
   C.1 Finding all simple spots .......................... 51
   C.2 Including spots with positive elements .......................... 53
   C.3 Creating a graph from the possible spots .......................... 54
   C.4 Creating a graph when positive elements are allowed .................. 56
   C.5 Finding the shortest path using Bellman-Ford .......................... 57
   C.6 Solution matrix using the simple Bellman-Ford approach .................. 59
   C.7 Solution matrix using the Bellman-Ford approach when positive elements are allowed .................. 60
   C.8 Solution matrix using the Bellman-Ford approach with multiple openings allowed .................. 61
   C.9 Transforming a collection of spots back to a matrix .................. 62
   C.10 Finding a feasible solution: method I .......................... 63
   C.11 Finding a feasible solution: method II .......................... 65
   C.12 Finding a feasible solution: method III .......................... 67
There are several treatments for cancer, for instance chemotherapy, hormonal therapy and radiation therapy. Frequently, different treatment methods are combined to achieve the best treatment for a patient. The subject of this report is optimisation in radiation therapy.

Radiation therapy is a form of therapy where ionizing radiation is used to eradicate malignant cells. During a session of radiation therapy, the irradiation is aimed at the tumour from several different angles. Due to the crossfire effect, the tumour is exposed the most to radiation, exposure for surrounding tissue is reduced. This is important because it results in sparing healthy tissue, while the tumour will receive a much higher dose of radiation.

The patient is treated with a device called a linear particle accelerator. It contains a source, from which electrons are emitted. The linear particle accelerator is a machine with an arm, which can rotate around the patient. This arm is called a the gantry. The electrons are accelerated, pass through the gantry, and at the end of the gantry they are directed at the patient, as can be seen in figure 2.1. By placing a tungsten plate at the exit, the electrons are converted into photons. After this tungsten plate, the photons pass through a multileaf collimator, which has moving parts that enable more precise irradiation. Since the gantry can move around the patient, the radiation can be directed at the patient from different angles. It is also possible to rotate the treatment couch, to create non-coplanar irradiation directions.

Constructing an optimal treatment plan for radiation therapy involves optimising the angles from which the patient is irradiated. Other degrees of freedom arise from of the use of a multileaf collimator, allowing beam shaping.

Optimising the treatment plan of a patient is essential. Optimising the treatment plan can increase the chance for a successful treatment and can reduce collateral damage to healthy organs. A secondary objective is reduction of the treatment time. This is more comfortable for the patient, introduces less scatter and can in some cases also be cost-effective if more patients can be treated per day.

The optimisation of the treatment plan is a difficult problem to solve. Due to the existence of many degrees of freedom, it is very difficult to find a good method to calculate the general optimal treatment plan, within a reasonable amount of time. That is why the search for methods that are better in treatment time as well as optimisation time, is an ongoing project. In this report we will discuss such a method, which might be faster than the methods that are currently used.
1.1. GENERAL OVERVIEW

In this report we will first explain the workings of a multileaf collimator in section 2 and then explain how the multileaf collimator can be mathematically represented in section 2.1. We will define our exact problem in section 3. After having defined the problem, we explain all the different methods of finding an optimal solution to our problem in sections 4, through 7. Finally we will present a summary of all the results and discuss the results.
The multileaf collimator is a device which is attached to the opening of the gantry, depicted in figure 2.1. The multileaf collimator works with leaves, which are radiation resistant slabs that can move independently of each other. A multileaf collimator is depicted in figure 2.2. The leaves are positioned at the right and at the left of the opening of the linear particle accelerator, and all the leaves can move towards the centre, blocking part of the ionising beam. Thus the beam of electrons can be shaped to fit our needs better. We call a certain combination of the positions of the leaves an aperture.

It is quite common that the tumour lies between other organs. With the multileaf collimator we can adjust the beam shape in such a way that we expose the healthy organs to a minimal amount of radiation.

There are many shapes we cannot make using the multileaf collimator. Because all the leaves slide in from the left or the right restrictions follow. For instance, a horseshoe shape cannot be created in one go. Such a shape can only be created in two steps. First we would make the right half of the horseshoe and turn on the linear particle accelerator, then after irradiation we would turn the machine off and move the leaves to new positions. When the leaves form the left half of the horseshoe we would turn the machine on again and irradiate the other half of the shape. A rule of thumb is that the treatment time grows linearly with the number of apertures required, thus our goal is to keep the number of apertures to a minimum.

### 2.1. Mathematical representation for the multileaf collimator

To be able to optimise the multileaf collimator we need a mathematical representation. To begin with, we place an equidistant grid on the square opening of the linear particle accelerator. We represent this grid by an $n \times m$ matrix with $n$ leaves on the right and $n$ leaves on the left. Since the multileaf collimator effectively turns the radiation beam off in certain places, we notate ‘beam on’ in a certain place as 1, and ‘beam off’ in a certain place as 0. So all leaves fully withdrawn means that our matrix has ones everywhere, and when the leaves fully block the beam, we have a matrix with zeros everywhere.

Note that we can move the leaves within a millimeter precision, but there are only a limited number of leaves in the multileaf collimator. As a result our matrix has many more columns than rows. For instance, with five leaves on both sides and ten possible positions for each leaf, the matrix representing a certain aperture is:
Figure 2.1: Linear accelerator

Figure 2.2: Multileaf collimator
2.2. Restrictions of Multileaf Collimators

There are more restrictions on which apertures can be created, but this can differ between different multileaf collimators. Common requirements for shapes are:

- Infeasibility of horseshoe-like shapes, all multileaf collimators
- Minimum area requirement for an opening, all multileaf collimators
- Minimum height requirement for an opening, some multileaf collimators
- Minimum width requirement for an opening, some multileaf collimators
- A maximum number of openings per aperture, some multileaf collimators

Now we will show examples of apertures that cannot be achieved because of requirements in this list.

A multileaf collimator may not be allowed to open a pair of leaves less than $x$ millimeters, it may however close a pair of leaves completely. This is called the minimum width requirement. Suppose that $x$ millimeters corresponds to 3 elements in the matrix. When this is the case the aperture given by matrix (2.2) would not be allowed. Because of the red elements, this aperture would then be infeasible.

Another, quite common limitation is that each opening in the aperture must have a minimal height: the minimal height requirement. For example when this minimal height of the opening corresponds to the size of two leaves, the opening must be at least two rows everywhere. An example of an aperture that would be infeasible is given in matrix (2.3). All the red elements in matrix (2.3) should have to be changed to create an aperture which does comply with this vertical demand.
Yet another common requirement is that each aperture can only have one opening, or with multiple openings possible, that a minimum number of leaves must be fully closed between those openings. When either of these demands are required, the aperture given by matrix \( \text{(2.4)} \) is not feasible.

\[
\begin{bmatrix}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
\end{bmatrix}
\] \( \text{(2.4)} \)

In the case that multiple openings are allowed but that there must at least be one row between openings a feasible alternative for matrix \( \text{(2.4)} \) is given in matrix \( \text{(2.5)} \):

\[
\begin{bmatrix}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
\end{bmatrix}
\] \( \text{(2.5)} \)

When only one single connected opening per aperture is allowed a feasible alternative to \( \text{(2.4)} \) is given in \( \text{(2.6)} \):

\[
\begin{bmatrix}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\] \( \text{(2.6)} \)

A multileaf collimator can have even more restrictions than the ones explained in this section, but those restrictions can vary widely between multileaf collimators. This is why we develop our theory in this report for a random multileaf collimator, for which only horseshoe-like shapes are infeasible. In section 7, we present ways of taking these more common restrictions imposed by various multileaf collimators into account.

### 2.3. Step-and-Shoot versus Dynamic Radiation Therapy

In the use of the multileaf collimator there are two general approaches. The first approach is called *step-and-shoot*, and this means that the leaves of the multileaf collimator do not move when the patient is irradiated. To change the aperture during the treatment, the linear particle accelerator is turned off. Then the leaves are moved and then the linear particle accelerator is turned on again. The second approach is called *dynamic*, and with this approach the leaves move continuously during the treatment session. The linear accelerator is only turned off at the end of the treatment session.

In the case of the step-and-shoot approach, each time the leaves stand still, their exact positions are automatically checked. This guarantees that the leaves are in the correct position within the millimeter accuracy. When the dynamic approach is used however, this cannot be checked. This is because the leaves move continuously. Because the position of the leaves are not checked, the dynamic approach results in a much shorter treatment time. But it is impossible to know how well adjusted the positions of the leaves are during the treatment. There is no consensus among experts on which approach is better: just because the positions in the
step-and-shoot approach can be checked does not mean it is necessary.

In this research project, only step-and-shoot approach is considered.
3

PROBLEM FORMULATION

In the case of the step-and-shoot approach for irradiation, two optimisation approaches can be used. The first optimises a continuous fluence map, and the result is segmented in deliverable apertures. However, this results in a mismatch between the optimised result, and the deliverable result. An alternative is column-generation, which allows direct optimisation of apertures. When column generation is used to optimise the treatment plan, it automatically generates a plan which is deliverable. No quality of the solution is lost due to converting the optimal solution to a deliverable plan. However, due to the sequential approach, the result may converge to a suboptimal solution. This concept of using column generation in optimisation for radiotherapy was initially developed by Romeijn et al. [1]. In section 3.1 the concept of column generation and how it can be used for beam optimisation is explained. In section 3.2 we will formulate the object of this research project: how to find the best feasible aperture from the outcome of a column generation iteration.

3.1. COLUMN GENERATION

Column generation is a method used in mathematical optimisation. As explained in [2] the general idea is that when a problem is too large or too time consuming to solve directly. The original problem is called the master problem. Instead of trying to solve this large master problem, we consider a different problem, where a subset of the large amount of variables in the master problem is considered. We call this problem with less variables the restricted master problem. This restricted master problem is solvable in a fraction of the time it would take to solve the master problem. This optimal solution for the restricted problem is of course not automatically the optimal solution for the master problem.

After optimizing only the variables in the restricted master problem, a new set of variables is added that will have negative costs. Finding which variables will add to the quality of the solution is called the pricing problem. The variables which are expected to have the most negative costs are added to the restricted master problem. This way the set of variables is expanded until the optimal solution is found.

In radiotherapy each beamlet in every different aperture of the treatment plan, is considered a different variable. Each element in the aperture matrix corresponds to a beamlet in the actual beam of the linear accelerator. To initialise the algorithm we set all the beamlets to 1 as the starting variables for column generation. Then we solve the restricted master problem, and use Langrange multipliers to find gradient descent information for all the individuals beamlets. In column generation for radiotherapy, the restricted master problem returns each time only the beamlets in the next aperture as variables. The pricing problem is concerned with grouping the most promising beamlets into a feasible aperture, and add this aperture to the solution.

Now there is a problem. The goal is to add as many negative elements from the matrix at once, because each of them will improve the quality of the solution. This means that this problem is a minimisation problem,
where we try to minimize the total weight of the beamlets. It is however, very likely that it is impossible to create a feasible aperture that contains only negative numbers and no positive numbers. A compromise has to be found between a large aperture and additional positive elements. When a suitable aperture is found, it can then be added to the solution, and in a further iteration of the column generation, for each beamlet it is again calculated how much addition of it to the last solution will decrease or increase the objective function further.

3.2. SETUP OF THE RESEARCH PROJECT

As explained in section 3.1, column generation will return which beamlets are best to add to the solution. But as explained in section 2, there are limitations to the shapes of the apertures. This means that we must find the best feasible aperture from the matrix returned by column generation algorithm. This report will explain a method to do this.

We start with a simplified problem in section 4.1. Instead of using the matrix returned by column generation, we transformed it to a matrix of only zeros and ones. This was done by replacing each negative element in the original matrix by a one and all other elements by a zero. Then the goal was to find the largest connected feasible area. Zeros were not yet allowed in the resulting aperture.

After analysing the simplified matrices, with only zeros and ones, in section 4.1 we have looked at actual matrices containing zero, positive and negative elements. The goal here was not to find the largest feasible area, but to find the feasible area that has the largest negative weight. The weight of an area is calculated as the sum of its elements. The approach here was very similar to the approach for the discrete matrix. Still only the negative elements were considered and no positive elements were allowed in the resulting aperture. This is explained in section 4.3.

It can be more profitable to allow some positive elements, in order to construct a larger negative area by connecting the two or more negative areas. In this extension of the previous approach only specific situations of negative elements are considered. Only the positive elements that have negative elements to their right and left are allowed in the aperture. This is described in section 5.

In all the previous approaches only connected apertures were allowed. This is not a restriction of all multileaf collimators. Allowing apertures that are not completely connected could in many cases return a better solution than when only connected apertures are allowed. So in the next extension of the algorithm, described in section 6 also non-connected apertures are allowed.

As explained in section 2.1 there are requirements for the aperture a multileaf collimator can make and these requirements can differ between different multileaf collimators. In section 7 it is described how some of these requirements can be met.

In section 8 the results off all these approaches are presented and compared. Finally a method is presented that could prove to be a significant improvement, compared to the other approaches that where presented. There was however insufficient time to implement this method during this project. Because of this there are no results available for this approach, that is explained in section 9.2.
In this section we will discuss the most simple way to construct a feasible solution. This construction will be explained step by step. In section 4.4 we will discuss the results of this method.

4.1. Reduction to Directed Graphs

In this section we consider the simplified aperture matrix with zeros and ones, as explained in section 3.2. The goal is to find the largest feasible aperture where the shape which we create is connected. In order to find this aperture we reduce our matrix representation of the multileaf collimator to a directed graph, so we can work with vertices and edges. The construction of such a graph is explained using an example, matrix $\mathcal{D}:

\begin{align*}
\mathcal{D} &= \begin{matrix}
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
\end{matrix}
\end{align*}

(4.1)

Each node in graph $\mathcal{G}$ belonging to matrix $\mathcal{D}$ corresponds to a connected row of ones in a row of $\mathcal{D}$. The directed graph starts with vertex zero. Each row in the matrix is assessed, and to each connected row of ones in a row a number is assigned. For clarity a connected row of ones is called a spot. The spots are enumerated from left to right and from top to bottom. In matrix (4.2) the first spot is coloured red, in matrix (4.3) the second spot is coloured red and in matrix (4.4) the fourth spot is coloured red.

\begin{align*}
\begin{array}{cccccccccccccccccccccccccccc}
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
\end{array}
\end{align*}

(4.2)

\begin{align*}
\begin{array}{cccccccccccccccccccccccccccc}
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
\end{array}
\end{align*}

(4.3)
When one spot is connected to a spot in the row above or below that spot, their vertices in $G$ are connected by an edge: see figure 4.1. All vertices that correspond to spots that are not connected to another spot in the row above that spot are connected to node zero. Vertices that correspond to spots that are not connected to another spot in the row below that spot are simply not connected to any further vertex.

Note that the graph cannot contain any cycles, and that every path through the graph corresponds to a feasible aperture. A path through this graph could for instance be as in figure 4.2. The corresponding aperture matrix would then be matrix (4.5).

Since the goal is to find the largest connected feasible area, the weight that a certain vertex adds to the path has to be incorporated in the graph. In $G$ the weight a certain edge has, corresponds to the size of the corresponding spot in $\mathcal{D}$. The weight a certain node has, is assigned to the edge that points to that node, see figure 4.3. This is convenient when later an algorithm is used to find the longest path through $G$.

In order to find the largest feasible aperture in the matrix, the longest path in that graph has to be found. Generally longest path problems are hard to solve, because they are NP-hard. This means that it is impossible to solve them in polynomial time, and therefore solving our large complicated problem would take too

Figure 4.1: Graph $G$ corresponding to matrix $\mathcal{D}$

Figure 4.2: Path through graph $G$

\[
\begin{pmatrix}
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0
\end{pmatrix}
\]

(4.4)

\[
\begin{pmatrix}
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
\end{pmatrix}
\]

(4.5)

Figure 4.3: Graph $G$ with the weights added to the edges
4.2. Bellman-Ford

The most commonly used shortest path algorithm is Dijkstra’s. In this case however, Dijkstra’s shortest path algorithm will not work. Dijkstra’s algorithm cannot deal with negative distances in a graph. As stated in [3], the Bellman-Ford algorithm is an appropriate algorithm for solving a shortest path problem in which negative distances occur. Bellman-Ford does have a restriction: the graph may not contain any negative cycles, because then the shortest path is automatically minus infinity. (Explained intuitively, one could endlessly walk through the cycle, each time creating an even shorter path.) Fortunately, as can be seen in section 4.1, our graphs cannot contain any cycles.

The general approach of the algorithm is as follows. The algorithm starts at iteration zero at one starting node, and the distance from this start node to itself is set to zero. At this time the distances from that vertex to all other nodes is set to infinity. Now the algorithm checks for each edge whether the vertex that is reached by that edge has a larger distance than the distance to the vertex the edge was coming from plus the weight of the edge. In order to reach guaranteed optimality, the process must be repeated as many times as there are edges. The pseudo code of the Bellman-Ford algorithm is given in Algorithm 1. Here $V$ is the collection of all nodes, and $A$ is the set of all edges.

**Algorithm 1:** Pseudo-code Bellman-Ford algorithm

```
Data: A directed graph $D = (V, A)$ and a vertex $s \in V$
Result: $\text{dist}(s, v)$ for all $v \in V$
For each $v \in V$ set $\text{dist}(s, v) := \infty$
Set $\text{dist}(s,s) := 0$
while the collection of distances changes do
    check all edges in the graph one by one for an edge from $u$ to $v$
do
    Set $\text{dist}(s,v):=\min(\text{dist}(s,u)+\text{weight}(u,v),\text{dist}(s,v))$

Note that Bellman-Ford calculates the distances from the start node to all other nodes. The shortest path
then corresponds to the node with the minimal distance, that is not the begin node. To calculate the shortest path in the graph, generated by reducing the matrix representation of the multileaf collimator to a graph, the Bellman-Ford algorithm is quite useful. Then, when the algorithm found the vertex which has the smallest distance to the vertex zero, the path that was taken needs to be backtracked through the graph. When the correct path is found this can be translated back to the corresponding spots and thus back to the original matrix.

4.3. COLUMN GENERATION MATRIX

In reality, column generation does not return a matrix with zeros and ones. As explained in Chapter 3.1 the matrix contains real numbers. A negative element means that it is profitable to add the corresponding beamlet and a positive element means that adding that beamlet to the solution will decrease the quality of the solution. When an element is more negative or positive, its corresponding beamlet will increase or decrease the quality of the solution more when it is added. Adding a beamlet that corresponds to an element in the matrix that is zero would mean that adding the corresponding beamlet will neither increase nor decrease the quality of our solution. The result is that the goal changes. Instead of wanting to find the largest area of ones, as the goal was in the simplified case, the goal now is to find the connected aperture where the sum over the elements is the most negative.

This problem has a lot of parallels to the simplified case. In order to show how this non-discrete matrix is dealt with, we again take an example matrix, $M$. $M$ is a typical matrix which could be the result of column generation.

$$M =
\begin{pmatrix}
0 & 0 & -0.1 & -0.5 & -0.6 & 1.3 & 1.4 & 0.3 & -0.8 & 0 \\
-0.5 & 0.3 & -1.3 & -2.4 & 3.3 & 1.2 & -1.3 & -0.6 & 0.5 & 0 \\
0.6 & 0.8 & -1.3 & -2.5 & 0.1 & 1.1 & -1.3 & -1.3 & -1.2 & 0 \\
1.5 & 1.3 & -0.1 & -0.6 & -1.7 & -1.5 & -2.6 & 1.5 & 0 & 0 \\
0 & -1.4 & -2.6 & 1.3 & 0.4 & 1.6 & -0.6 & -0.2 & -0.1 & -0.7 \\
-0.7 & -1.4 & -1.7 & 0.7 & 0.5 & 0.6 & -0.3 & -1.2 & -1.4 & -1.2
\end{pmatrix}
$$

(4.6)

As in the discrete case, the matrix is reduced to a directed graph in order to find the most negative connected aperture. Instead of creating a vertex for each row of ones, now a vertex is created for each connected row of negative elements. The graph is shown in figure 4.5.

In this approach, the weights that need to be assigned to the edges, should not be equal to the size of the corresponding area, like in the discrete case. Instead, the sum of the elements of a spot is added as the weight of the edge. In matrix (4.7), the first spot is coloured red. The sum over that spot is then $-0.1 - 0.5 - 0.6 = -1.2$. This means that the distance from the start node to the first node is set to $-1.2$.

1The zeros around the edges however are different from a zero in the centre of the matrix. The zeros around the edges do not represent the possible increase or decrease of the solution. They represent the edge of the beams eye view of the tumour site.
4.4. RESULTS OF THE BELLMAN-FORD ALGORITHM

Now we will discuss the results of reducing the problem to a shortest path problem and applying the Bellman-Ford algorithm. In order to produce results, appropriate test problems must be found. Researchers at Erasmus MC are currently developing an algorithm that applies column generation to the planning of radiotherapy. As explained in section 3.1, column generation returns a matrix that indicates which beamlets are profitable to add to the solution. At Erasmus MC, 45 of these matrices where produced to use as test cases.

The quality of the solution aperture can be measured as the total weight of the beamlets. In the matrix returned by the column generation process, negative elements mean that their corresponding beamlets add...
to the quality of the solution. Also, elements that are more negative, add more value to the final solution. Accumulating the weights of all these negative elements gives an indication of the quality of the solution presented. So when the sum over solution aperture \( A \) is more negative then the sum over solution aperture \( B \), solution \( A \) is better than \( B \). We shall measure the quality of a solution in the value of the sum of its elements.

In order to comment on the quality of the solution found, a benchmark is needed. Since the absolute feasible optimum is unknown for our test cases, we look at the non-feasible optimum as a benchmark score: the sum of all negative elements in the matrix. This optimum is not feasible in almost all cases. It is however an indication of the minimum value of the actual feasible optimum.

During this research an attempt was made to create a better benchmark value. An easy solution would be to simply test the matrices for all possible apertures by exhaustive search. Then all the feasible apertures could be selected and the aperture with the most negative weight would be the optimal solution. However, the test matrices are of significant size and trying to solve our problem this way turned out to take too long.

In appendix A, the benchmark values and the values given by using the simple Bellman-Ford approach is presented for all 45 test matrices (second and third columns).

In order to better understand the results, we show 4 cases in detail. Figures 4.8a, 4.8c, 4.8e and 4.8g are representations of matrices that result from column generation. The white parts of the images represent the zeros in the images. These zeros lie outside the area that must be treated and are not to be considered in the radiation plan.

In order to clearly see which elements in the matrix correspond to beamlets that are profitable to add to our solution, we have used two different color ranges. The hot colour range that goes from black to white is used to display the positive elements in the matrix. The cool colour range that goes from blue to green is used to display negative elements in the matrix. Note that the scales of the legends differ.

In figures 4.8b, 4.8d, 4.8f and 4.8h the apertures are presented that result of applying Bellman-Ford to the test matrices. The gray in the figures represents the shape of the original matrix. Note that these solutions indeed allow only negative elements, and all resulting apertures are connected and can be made by a multileaf collimator in one go. In table 4.1 the sums over the elements of each of the four test cases is given. Note that indeed all the solutions have either increased in quality because their weight is more negative in the new solution, or remained the same quality. Especially in the first three cases many negative elements could not be included in the solution.
4.4. **RESULTS OF THE BELLMAN-FORD ALGORITHM**

Figure 4.8: On the left the original apertures are shown. On the right the apertures resulting from the Bellman-Ford algorithm are shown.
5

SOLUTIONS WHEN POSITIVE ELEMENTS ARE ALLOWED

In section 4.3 a method is described to find the most negative area, which is feasible and connected. It is imaginable that an even more negative area can be found when some positive elements are allowed. As a demonstration of this fact, example matrix $\mathcal{M}_e$ is used. In matrix (5.1) spots 5 and 6 are coloured red. When only series of negative elements are allowed as a spot, only one of these two spots can be chosen. This is because the multileaf collimator cannot irradiate both spots 5 and 6, while also blocking the positive elements in between.

$$\mathcal{M}_e = \begin{bmatrix}
0 & 0 & -0.1 & -0.6 & -0.3 & 1.2 & 1.2 & 0.3 & -0.7 & 0 \\
-0.6 & 0.4 & -1.5 & -2.9 & -1.0 & -1.5 & -2.3 & -0.6 & 0.5 & 0 \\
0.6 & 0.8 & -1.3 & -2.6 & 0.1 & 1.1 & -1.3 & -1.3 & -1.2 & 0 \\
1.7 & 2.3 & -0.1 & -0.7 & -1.7 & -1.5 & -2.6 & 1.5 & 0 & 0 \\
0 & -1.4 & -2.6 & 1.2 & 0.4 & 1.6 & -0.6 & -0.2 & -0.1 & -0.7 \\
-0.7 & -1.4 & -1.6 & 0.7 & 0.8 & -0.3 & -1.2 & -1.4 & -1.2 & 0
\end{bmatrix} \quad (5.1)$$

The weight of spot 5 equals $-1.3 - 2.6 = -3.9$ and the weight of spot 6 equals $-1.3 - 1.3 - 1.2 = -3.8$. So when positive numbers are not allowed, adding spot 5 will give the best result. However the positive weight between the two spots is quite small in comparison to the weights of spot 5 and spot 6. In total, adding both spots 5 and 6 and the positive elements in between these spots, all the elements coloured red in matrix (5.2), is more beneficial. As a whole, this will give the more negative area of $-6.5$.

$$\mathcal{M}_e = \begin{bmatrix}
0 & 0 & -0.1 & -0.6 & -0.3 & 1.2 & 1.2 & 0.3 & -0.7 & 0 \\
-0.6 & 0.4 & -1.5 & -2.9 & -1.0 & -1.5 & -2.3 & -0.6 & 0.5 & 0 \\
0.6 & 0.8 & -1.3 & -2.6 & 0.1 & 1.1 & -1.3 & -1.3 & -1.2 & 0 \\
1.7 & 2.3 & -0.1 & -0.7 & -1.7 & -1.5 & -2.6 & 1.5 & 0 & 0 \\
0 & -1.4 & -2.6 & 1.2 & 0.4 & 1.6 & -0.6 & -0.2 & -0.1 & -0.7 \\
-0.7 & -1.4 & -1.6 & 0.7 & 0.8 & -0.3 & -1.2 & -1.4 & -1.2 & 0
\end{bmatrix} \quad (5.2)$$

Translating this concept to the graph is done by adding extra vertices to the graph. Where in the case without positive elements two vertices were added for spots 5 and 6, now third vertex is added. This vertex corresponds to allowing both spot 5 and spot 6, including the positive numbers in between. These areas can only be found when all of these possible combinations are added to the graph. So an extra step is added to the construction of the graph. When there are multiple spots in one row, all of them are added to the graph as vertices, and all combinations of these spots are also added. For example, when there are two spots in a row, a vertex is added that corresponds with allowing both spots and the positive numbers in between. When there were three spots in the original row, all three possible combinations of these spots are added as extra vertices to the graph.
In figure 5.1 the graph is shown that corresponds to $\mathcal{M}$ without the added vertices for positive elements, in figure 5.2 the graph is shown that corresponds to $\mathcal{M}$ where the extra vertices (red) are added. The weights of the edges for the added vertices are again simply the sum over the corresponding elements in the matrix.

Now the weight of an edge can be positive. This happens when the sum over the positive numbers between two spots is greater than the absolute value of the weight of the spots themselves. This presents no challenge in finding the shortest path. Bellman-Ford will automatically choose the path with the most negative distance in total. Since all the old vertices and edges are still present in the new graph, the shortest path in the new graph is always shorter than or equal to the shortest path in the old graph. So once again Bellman-Ford can be used to find the shortest path in this extended graph.

5.1. RESULTS FIRST EXTENSION BELLMAN-FORD ALGORITHM

The full list of sums over the elements of the matrix, that are the result of applying Bellman-Ford while allowing positive elements is given in appendix A. In figures 5.3b, 5.3d, 5.3f and 5.3h the resulting apertures are shown for the same four test cases as in section 4.4.

Note that in some cases the more advanced approach finds a better solution by allowing positive elements. In for cases 19 and 37, some of the hot colour scheme (black) is visible and this means that here positive elements have been included. In those solutions it was more profitable to allow some positive elements in order to achieve a more negative weight in total. In table 5.1 you can see that as expected, the quality of the solutions for cases 19 and 37 has improved, from $-14.0$ to $-15.3$ and from $-0.59$ to $-0.60$.

<table>
<thead>
<tr>
<th>case</th>
<th>Benchmark value</th>
<th>Result simple Bellman-Ford</th>
<th>Result Bellman-Ford allowing positive elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>-26.25</td>
<td>-14.00</td>
<td>-15.32</td>
</tr>
<tr>
<td>21</td>
<td>-19.82</td>
<td>-11.46</td>
<td>-11.46</td>
</tr>
<tr>
<td>26</td>
<td>-19.09</td>
<td>-11.71</td>
<td>-11.71</td>
</tr>
<tr>
<td>37</td>
<td>-0.6214</td>
<td>-0.5870</td>
<td>-0.6018</td>
</tr>
</tbody>
</table>

Table 5.1: Result application Bellman-Ford when positive elements are allowed
5.1. RESULTS FIRST EXTENSION BELLMAN-FORD ALGORITHM

(a) Case 19, solution, no positive elements allowed

(b) Case 19, with positive elements allowed

(c) Case 21, no positive elements allowed

(d) Case 21, with positive elements allowed

(e) Case 26, no positive elements allowed

(f) Case 26, positive elements allowed

(g) Case 37, no positive elements allowed

(h) Case 37, positive elements allowed

Figure 5.3: On the left the apertures resulting from simple Bellman-Ford. On the right the apertures resulting from the Bellman-Ford with positive elements allowed.
In all previous sections, only connected apertures where considered feasible. Some multileaf collimators however, permit several openings in one aperture. Of course the openings have to be in different sets of rows to be feasible for the leaves. Allowing non-connected apertures can give better solutions. A simple example where this is the case is given in example matrix $E$.

$$
E = 
\begin{bmatrix}
0 & 0 & -0.1 & -0.5 & -0.6 & 0 & 0 & 0 & 0 & 0 \\
-0.6 & 0.4 & -1.3 & -2.5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.6 & 0.8 & -1.3 & -2.5 & 0 & 1.1 & 1.3 & 1.3 & 1.2 & -0.6 \\
1.5 & 1.3 & -0.1 & 0.6 & 1.7 & 1.5 & -0.6 & -0.5 & 0 & 0 \\
0 & 0 & 0 & 1.3 & 0.4 & 1.6 & -0.6 & -0.2 & -0.1 & -0.7 \\
0 & 0 & 1.7 & 0.7 & 0.5 & 0.7 & -0.3 & -1.2 & -1.5 & -1.2 \\
\end{bmatrix}
$$

(6.1)

In matrix (6.1) it can be seen that the negative elements do not form a single connected area. When the extended approach of section 5 would be used it would find one single connected aperture, as highlighted in matrix (6.2).

$$
E = 
\begin{bmatrix}
0 & 0 & -0.1 & -0.5 & -0.6 & 0 & 0 & 0 & 0 & 0 \\
-0.6 & 0.4 & -1.3 & -2.5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.6 & 0.8 & -1.3 & -2.5 & 0 & 1.1 & 1.3 & 1.3 & 1.2 & -0.6 \\
1.5 & 1.3 & -0.1 & 0.6 & 1.7 & 1.5 & -0.6 & -0.5 & 0 & 0 \\
0 & 0 & 0 & 1.3 & 0.4 & 1.6 & -0.6 & -0.2 & -0.1 & -0.7 \\
0 & 0 & 1.7 & 0.7 & 0.5 & 0.7 & -0.3 & -1.2 & -1.5 & -1.2 \\
\end{bmatrix}
$$

(6.2)

The lower right corner of $E$ has another high concentration of negative elements. Adding this area is physically possible for some multileaf collimators, but this area will not show up in the solution aperture when only connected apertures are allowed.

When we want to find a non-connected aperture, we must find multiple paths through the graph. These paths cannot contain vertices that correspond to spots in the same rows. In our graphs, there is no way to see in which row the spot corresponding to a vertex is situated. This means that in order to prevent that multiple spots in a single row are allowed, we must look at the original matrix as a whole.

A simple approach to prevent multiple rows in one graph is to first find the most negative area, using the previous approach. After the first area has been found and approved, it is added to the solution matrix. Then, all rows that are nonzero in the solution matrix are set to zero in the original matrix and the whole process...
is repeated. The areas the algorithm finds are checked each time for a minimal area requirement and when they suffice, they are added to the solution matrix. Then all rows that the resulting area covers, are set to zero in the original matrix. This process is repeated until no more areas with negative weight can be found in the matrix.

When this approach is applied to matrix $E$, the algorithm will at first return matrix (6.3).

$$E = \begin{bmatrix} 0 & 0 & -0.1 & -0.5 & -0.6 & 0 & 0 & 0 & 0 & 0 \\ -0.6 & 0.4 & -1.3 & -2.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1.3 & -2.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(6.3)

After setting the rows that are covered by this preliminary solution to zero, the residual matrix $R_1$ is given in matrix (6.4).

$$R_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.3 & 0.4 & 1.6 & -0.6 & -0.2 & -0.1 & -0.7 \\ 0 & 0 & 1.7 & 0.7 & 0.5 & 0.7 & -0.3 & -1.2 & -1.5 & -1.2 \end{bmatrix}$$

(6.4)

The most negative weighted area in this residual matrix is highlighted in matrix (6.5).

$$E_{alt} = \begin{bmatrix} 0 & 0 & -0.1 & -0.5 & -0.6 & 0 & 0 & 0 & 0 & 0 \\ -0.6 & 0.4 & -1.3 & -2.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.6 & 0.8 & -1.3 & -2.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.6 & -0.2 & -0.1 & -0.7 \\ 0 & 0 & 0 & 0 & 0 & -0.3 & -1.2 & -1.5 & -1.2 \end{bmatrix}$$

(6.5)

When this area consists of a sufficient number of elements, it is added to the solution matrix $E$. This opening is then added to the solution aperture, resulting in matrix (6.6).

$$E_{alt} = \begin{bmatrix} 0 & 0 & -0.1 & -0.5 & -0.6 & 0 & 0 & 0 & 0 & 0 \\ -0.6 & 0.4 & -1.3 & -2.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.6 & 0.8 & -1.3 & -2.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.6 & -0.2 & -0.1 & -0.7 \\ 0 & 0 & 0 & 0 & 0 & -0.3 & -1.2 & -1.5 & -1.2 \end{bmatrix}$$

(6.6)

Note that this is not actually an optimal solution aperture, because matrix (6.7) is also feasible and has a more negative weight. The sum over the fourth row of matrix (6.6) equals $-0.1$, the sum over the fourth row of matrix (6.7) equals $-1.1$. Since all other rows are exactly the same, this means that the total sum of matrix (6.7) is more negative than the total sum over (6.6).

$$E_{alt} = \begin{bmatrix} 0 & 0 & -0.1 & -0.5 & -0.6 & 0 & 0 & 0 & 0 & 0 \\ -0.6 & 0.4 & -1.3 & -2.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1.3 & -2.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.6 & -0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.6 & -0.2 & -0.1 & -0.7 \\ 0 & 0 & 0 & 0 & 0 & -0.3 & -1.2 & -1.5 & -1.2 \end{bmatrix}$$

(6.7)
6.1. RESULTS SECOND EXTENSION BELLMAN-FORD ALGORITHM

The weights of the solutions of the four test cases are given in table 6.1. The figures 6.1b, 6.1d, 6.1f and 6.1h depict the solutions of the multiple openings approach, side by side with the approach where negative elements are allowed but only in connected apertures. Again the hot colour scheme is used for the positive elements, the cool colour scheme is used for the negative elements and the gray represents the shape of the original matrix.

From the images 6.1b, 6.1d and 6.1f, it is clear that allowing more openings in a aperture can result in a larger area. Furthermore, from table 6.1 we can see that for cases 19, 21 and 26 the new weight is over 50% larger than that of the previous best solution. This is a significant improvement from the previous approach. In these four test cases, the resulting weight is no more then 15% less than the benchmark weight.

<table>
<thead>
<tr>
<th>case</th>
<th>Benchmark value</th>
<th>Result Bellman-Ford</th>
<th>Result allowing positive elements</th>
<th>Result allowing multiple openings</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>-26.25</td>
<td>-14.00</td>
<td>-15.32</td>
<td>-24.33</td>
</tr>
<tr>
<td>21</td>
<td>-19.82</td>
<td>-11.46</td>
<td>-11.46</td>
<td>-19.39</td>
</tr>
<tr>
<td>26</td>
<td>-19.09</td>
<td>-11.71</td>
<td>-11.71</td>
<td>-18.84</td>
</tr>
<tr>
<td>37</td>
<td>-0.6214</td>
<td>-0.5870</td>
<td>-0.6018</td>
<td>-0.6018</td>
</tr>
</tbody>
</table>

Table 6.1: Results of different Bellman-Ford approaches

So with this simple approach of allowing more openings in one aperture optimality is not automatically guaranteed. It does however in many cases give a better result than when only one opening is allowed.
6. Solutions when multiple openings are allowed

(a) Result case 19, positive elements allowed
(b) Result case 19, not connected apertures allowed
(c) Result case 21, positive elements allowed
(d) Result case 21, not connected aperture allowed
(e) Result case 26, positive elements allowed
(f) Result case 26, not connected apertures allowed
(g) Result case 37, positive elements allowed
(h) Result case 37, not connected apertures allowed

Figure 6.1: On the right, the apertures resulting from the Bellman-Ford algorithm with positive elements allowed. On the right the apertures resulting from the Bellman-Ford algorithm with multiple openings allowed.
CREATING A FEASIBLE SOLUTION

All heuristics described so far were explained under the assumption that the multileaf collimator has no limitations besides the limitations that where marked requirements for all multileaf collimators in section 2.1. The most important one was the infeasibility of horseshoe-like shapes. In reality, all multileaf collimators have more limitations.

These requirements can be a consequence of the biological aspects of radiation therapy. Around the edges of the radiated area, the radiation is scattered. When the opening of a aperture is very small, scatter will dominate. Therefor most multileaf collimators may only be used when each opening has a certain minimal area. Even if the safety settings of the multileaf collimator allow smaller areas, the medical doctors will require such a minimal area constraint. Usually however, the minimal area is small and every optimal opening already complies with this requirement. When the optimal area does not comply with this constraint, a good heuristic for finding a feasible area is to set the optimal area in the matrix to zero, and to search for the second best optimal area. When all areas of negative elements are too small to irradiate, there is no feasible solution for that matrix.

For demonstration purposes, we will now choose a random multileaf collimator with some random but common restrictions. We say that our multileaf collimator requires that each opening has an area of minimally 75 squared millimeters. Furthermore it only allows apertures that have a single connected opening. In order to demonstrate how often such a minimal area requirement would not be met, we assess our 45 test cases. In the test cases that are presented here, none of the optimal connected apertures have an opening that has an area smaller then 75 squared millimeters. In order to test our algorithm we set the minimal area constraint on 300 squared millimeters, and in all 45 cases there was a possible opening with negative weight that had an area larger then 300 square millimeters. Note that in this case the feasible opening was not always the optimal opening, but can be a second or third best option.

Another common requirement of a multileaf collimator is that there are a minimal height and width for an opening. These limitations are also described in section 2.2. We assume that our multileaf collimator has a minimum horizontal opening of 7.0 millimeters, and a minimum vertical opening of 9 millimeter. Furthermore we assume that our multileaf collimator has 39 leaves on each side, which are all 3 millimeter wide. This means that the opening of the solution aperture must be 3 matrix elements high everywhere. Our test matrices are all 44 elements wide. When we assume that the multileaf collimator is 132 millimeter wide, each element in the matrix corresponds to 3 millimeter in reality. This means that everywhere we have a minimal opening of 3 matrix elements high and 3 matrix elements wide. For example the last optimal aperture of case 37 as shown in figure 7.1a does not meet the these requirements: the width in the middle is only one beamlet and the minimum height is not reached in five places.
Meeting this requirement in the solution aperture is not as easy as the minimal area constraint. It is difficult to construct a solution that automatically meets this requirement, therefore it is easier to create a solution that is not feasible using the method described earlier, and then adapt it into a feasible solution. When adapting a optimal solution into a feasible solution, there are three approaches:

- add elements until the requirements are met
- remove elements until the requirements are met
- adds and remove elements based on what is optimal until the requirements are met

We investigate three methods that can be implemented quickly.

### 7.1. Method I: Add a Margin Around the Solution

If an aperture is in some places too narrow or extends over to few leaves, this aperture can be made feasible by simply adding a margin of one beamlet around the whole of the aperture, see figure 7.1c. This border must be four connected, which means that each margin element borders to two other margin elements on either the North, South, West or East side of the element. This aperture is feasible automatically.

However, the bright green pixels are beamlets that lie outside the beams eye view of the tumour. Adding them is certainly a bad idea, because other organs will be exposed more. Adding this overall margin of one beamlet also has the drawback that now many positive beamlets are unnecessarily included (the red, and most of the black elements). So this approach makes any aperture feasible, but the aperture will be much less optimal and include beamlets outside the beams eye view, so we will not use this approach.

### 7.2. Method II: Remove a Margin from the Domain, then Add it Again after Optimisation

In this approach we use the fact that adding a margin makes any aperture feasible, in a different way. We start with removing a margin of one element from the initial domain. Then we calculate the optimal aperture using Bellman-Ford with positive elements allowed, for that domain. And then, we again ad a margin of one around this new optimal aperture.

The result of this process on test case 37 is shown in figure 7.1c. It is clear that with this approach no elements that lie outside the desirable domain are allowed in the feasible solution. However this feasible solution still has unnecessary positive elements (again the red elements and most of the black elements).

### 7.3. Method III: Remove a Margin from the Optimised Aperture, then Add a Margin

In a third approach to make all apertures feasible, we start with the optimised aperture (see figure 7.1a) and remove a margin of one beamlet on all sides. Then we make the computer trace the border and add again a margin of one element.

As a result of this process it is possible that the aperture is now no longer connected, and that more openings are created. In that case we just choose the connected opening with the largest negative weight.
7.4. Results of Correcting the Solutions

Having to correct the solution after optimising the aperture will decrease the quality of the solution. How much the quality will decrease however defines the quality of the process of making the aperture feasible. Since we assumed that the quality of a solution is defined by the weight of the aperture found, the process that results in a feasible aperture with the smallest weight is the best process. In table 7.1 the weights of the resulting apertures are presented. In appendix B graphs of the results of method II and III for all the four test cases are presented. The results of method I is not presented in this table. This is because we do not know the weights of the elements outside the domain. We can see that for the four test cases that where highlighted in this report not approach is best in all cases. In appendix A a table with all the weights for both approaches for all 45 test cases is given.

<table>
<thead>
<tr>
<th>case</th>
<th>Optimal connected solution</th>
<th>Result II method</th>
<th>Result III method</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>-11.46</td>
<td>-10.34</td>
<td>-11.06</td>
</tr>
<tr>
<td>26</td>
<td>-11.70</td>
<td>-10.79</td>
<td>-11.67</td>
</tr>
<tr>
<td>37</td>
<td>-0.6018</td>
<td>-0.5219</td>
<td>-0.4204</td>
</tr>
</tbody>
</table>

Table 7.1: Results of methods II and III to meet minimal height and width requirement

For case 37, the result is shown in figure 7.1 d. The minimum height requirement is now met, for example the extending bemalets in the top left and middle right of figure 7.1a are removed in figure 7.1d. However you can see that we lost a significant part of our solution due to non-connectedness.
(a) Case 37 optimal connected solution, but minimum width and height requirements are not met

(b) Case 37 solution made feasible with method I

(c) Case 37 solution made feasible with method II

(d) Case 37 solution made feasible with method III

Figure 7.1: The top figure is the result of the Bellman-Ford approach with positive elements allowed. The second figure is the result of method I, the third figure is the result of method II and the fourth figure is the result of method III
RESULTS FOR ALL 45 CASES

In order to compare the results from all 45 test cases we have produced the histogram depicted in figure 8.1. Here the weights of each resulting aperture is shown as a percentage of the benchmark value for that case. Note that, as we would expect, the simple solution is always less or equal in quality to the solution that allows positive elements. Furthermore the solution that allows positive elements is less or equal in quality to the solution that allows multiple openings in an aperture. This is what we expect because in our extensions of the original simple Bellman-Ford approach, we only add vertices to the graph. At the same time all the old vertices remain, so the previous optimal path is always a part of the graph in the next extension.

In figure 8.1 you can see that in all test cases the approach that allows multiple openings, the result has over 65% of the benchmark value.

In figure 8.2 the weights of the feasible solutions are shown as a percentage of the optimal connected solutions. As expected, not one of the two approaches reaches 100% when compared to the optimal not feasible aperture. This is a logical consequence of the fact that it was necessary to change the solution to make it feasible for a real multileaf collimator. It is also clear that, not one of the two methods is better than the other for all cases.

The fact that the actual feasible solutions are of less quality then the optimal not feasible solutions need not be a problem. Finding the optimal aperture is only part of the column generation process. Because of this it is possible that the column generation process can correct the solution in the next aperture. Adding higher quality apertures to the final treatment plan can decrease the number of aperture necessary to irradiate a patient. This can then reduces the irradiation time, which makes the treatment more comfortable for the patient, and is more efficient.
Figure 8.1: Histogram of the weights of the solutions of the simple Bellman-Ford approach, the approach where positive elements are allowed and the approach where multiple openings are allowed, expressed as percentage of the benchmark weights.
Figure 8.2: Histogram of the weights of the feasible solutions expressed as percentage of the optimal connected solution.
Now we will discuss some specific situations. We will show that the found solution are not guaranteed to be optimal with two examples. Then we will show that one of the cases described by the examples can be solved optimally using something called spot jumping.

9.1. Examples of Currently Not Optimally Solvable Cases

As explained in section 2 a multileaf collimator can have multiple restrictions. In section 6 we showed that when one or more openings where allowed, our solution is not guaranteed to be optimal. In section 5 spots on the same row could be connected by af few positive elements. A similar connection over columns is not possible with our method. Also, when only connected apertures are allowed, our approach can not guarantee optimality.

Considering the positive elements as rows, as done in the approach where positive element are allowed does in some cases take into account that allowing positive elements in order to allow even more negative elements can be beneficial. But the case where allowing positive elements that are connected and spread over several rows in order to allow more negative elements is not considered here. As an example we will present a simple discretised case: $E$.

\[
E = \begin{pmatrix}
0 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & 0
\end{pmatrix}
\]  

(9.1)

When only one opening is allowed in a aperture we can choose between connected group of ones in the upper right corner of the matrix or the group in the lower left corner. But since these two groups are quite close together it would be more profitable to allow all the red elements in matrix $E$.

\[
E = \begin{pmatrix}
0 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 0
\end{pmatrix}
\]  

(9.2)
In the approach where multiple openings are allowed per aperture optimality is also not guaranteed. This is because our construction of the graphs does not allow the solution to jump between different openings. When an opening is optimal, it is chosen and the other openings that lie in the same rows as that optimal opening are no longer considered. This does not necessarily return an optimal solution. Two openings can partly lie in the same rows and then the approach explained in this report simply chooses the opening that has the most negative weight in total. We will give an example matrix $E_2$ to explain this situation. In matrix (9.3) there are two groups of negative elements next to each other. In the rows one to four, the group of ones on the left hand side is clearly more negative. In rows five and six however, the group of ones on the right hand side is more negative. The approach discussed in this report would simply choose the red elements in matrix (9.3) as the optimal opening.

\[
E_2 =
\begin{bmatrix}
0 & 0 & 0 & -1 & -1 & -1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 0 & 0 \\
0 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & 1 \\
0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 & -1 \\
0 & 0 & 0 & 0 & 1 & 1 & -1 & 1 & -1 & -1 & -1 & 0 & 0
\end{bmatrix}
\]  

(9.3)

The red elements however do not form the optimal opening. When multiple elements are allowed, it would be more optimal to open the multileaf collimator for the beamlets corresponding to the red elements in matrix (9.4).

\[
\begin{bmatrix}
0 & 0 & 0 & -1 & -1 & -1 & -1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 & 1 & 0 & 0 \\
0 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & 1 \\
0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 & -1 \\
0 & 0 & 0 & 0 & 1 & 1 & -1 & 1 & -1 & -1 & -1 & 0 & 0
\end{bmatrix}
\]  

(9.4)

We call this a jump between two openings.

### 9.2. SPOT JUMPING

As explained in section 9.1, the solution that is found when the non-connected aperture approach is used is not always optimal. During the research an idea was developed which could be used to achieve optimality. In all the approaches that are explained in this report, some information about the data is not used. In our current algorithms, we only check whether a certain spot is connected to a spot above or below that spot. What we know, but do not use, is in which row a certain spot lies. Being able to jump between spots, of different rows, would ensure that the solution found is in fact optimal. Unfortunately all multileaf collimators have a minimal area constraint for an opening. This means that allowing the algorithm to jump freely between nodes that are associated to spots in different rows will give a solution that is not feasible in most cases. The algorithm would simply choose the best opening for each row and the solution could become highly fragmented. Such a fragmented solution could not easily be transformed into a solution that is in fact feasible, and still has a total negative weight. We have thought of a way to allow the algorithm between openings, but to retain the minimal area demand of the multileaf collimator. We will now explain how this method would work.

Completely restructuring the graphs used could be the answer to achieve optimality. To each vertex in the graph a second marker is added. Instead of only writing down to which vertices it must be connected, we add information in which row in the matrix the spot corresponding to that vertex is situated. Using this information we can decide between which vertices an extra edge can be added, without creating an infeasible matrix. We do this by only adding an extra edge between all vertices that only differ one row from each other. This extra edge represents jumping from one opening to another. The only problem is that now we do not know
whether a certain jump will create a solution that is not feasible with respect to the minimum size requirements. For example, the solution now can return that it is optimal to jump after only one row with which has a relatively small opening. This is the same case as matrix (9.4) in section 9.1. One of the openings in this matrix could be very small and therefore the resulting aperture could become infeasible.

In order to prevent this infeasibility problem we need to add more data to our graph. Each vertex will need to become a group of vertices. Each vertex in the group will represent the same spot in the matrix, but has some additional information. It counts what the area is of the path the opening it belongs to. When the area is sufficient a so called jump edge can be added and the area counter of the vertex that the edge goes to is set to zero. This way the solution can only jump when the area constraint is met. This does mean that for each spot, we need to add as many vertices to its corresponding group, as there are possibilities for the area counter.

To give an idea of what this graph will look like, we give a small example. As example matrix we have matrix $E_3$.

$$E_3 = \begin{array}{cccccccccccc}
0 & 0 & 0 & -1 & -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & -2 & -3 & -1 & 1 & 1 & 1 & -1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & -1 & -2 & -1 & 1 & -1 & -1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & -1 & -2 & -4 & -3 & 1
\end{array} \quad (9.5)$$

The graph corresponding to matrix $E_3$ is given in figure 9.1. In this figure you can see that the vertices are grouped according to the spot they belong to: all vertices that have as a first coordinate number $i$, corresponds to spot $i$. A spot can correspond to several vertices, one for each possible path that can be taken to reach it. The second coordinate indicates what the area is so far. For example, vertex (4.7) refers to the combination of spots 2 and 4 with a total area of 7 beamlets. We assume that the minimal area must be 5 beamlets in this example.

Note that, in figure 9.1 the vertices have been grouped in columns according to the row of the corresponding
Now we are going to add new edges that represent spot jumps. Spot jumps must start in vertices with enough area (≥ 5), go from one row to the next row (a column to the right in the graph), and should end in a vertex that is directly connected to the start vertex (0, 0). In this case, tree spot jumps remain, shown as dashed red arrows in figure 9.1.

Now there remains only one problem. The endpoints in the graph can well be part of an opening that has insufficient area. When the optimal path for instance would end in vertex (6, 4) this last opening is not large enough to create a feasible solution (4 < 5). In order to prevent this another step must be taken in the construction of a correct graph, that will present optimal feasible solutions. We look at all the nodes that have no outgoing edges. Then we check whether these nodes have a sufficiently large area. When the area does not meet the requirement the entire vertex is deleted. This process of identifying all vertices that have no outgoing nodes and the checking them is repeated until no infeasible vertices are found. When this process is applied to the graph depicted in figure 9.1, the result is the graph depicted in figure 9.2.

The optimal path now is clearly the path shown in figure 9.3, which corresponds to the red elements the matrix given in matrix (9.6):

$$ E_3 = 
\begin{bmatrix}
0 & 0 & 0 & -1 & -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0
0 & -1 & -2 & -3 & -1 & 1 & 1 & 1 & 1 & -1 & 1 & 0 & 0 & 0
0 & 1 & 1 & -1 & -2 & -1 & 1 & 1 & -1 & -1 & 1 & 0 & 0 & 0
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & -1 & -2 & -4 & -3 & 1
\end{bmatrix} \quad (9.6) $$

There was insufficient time to implement this approach during this research. Therefore we are not able to present any results of this approach.
Conclusion

An algorithm has been created, that finds a feasible area of sufficient quality in an aperture matrix resulting from a column generation program. So far however we have not reached optimality. This has several different causes.

First of all, it is very hard to find an optimal aperture, which is also feasible. Because of these feasibility requirements, the solution found needs to be corrected after the optimisation. Changing the optimal solution can reduce the quality of that solution. The methods that were used to change the optimal solution into a feasible solution are far from optimal, which means that optimality of the resulting feasible solution cannot be ensured.

There are also some feasibility requirements that may be taken into account during the optimisation, that where not taken into account in the algorithms used for this report. As described in section 9.2, it is possible to take the minimum area requirement into account during the optimisation. This approach would, in theory, give an optimal aperture. There was however not sufficient time to implement this approach during this project.

Even though optimality was not reached during this project, the result may be of sufficient quality for its application. In column generation the *pricing problem* returns which beamlets are beneficial to add to the irradiation plan. Adding as many of these as possible in one aperture can reduce the number of apertures necessary to achieve sufficient coverage of the tumour. However, when the aperture is not optimal, but does have a negative weight in total, the treatment plan will improve by adding this aperture. Then in the next step the column generation process can correct itself based on what beamlets where actually added.

The focus of this project was to find the optimal feasible aperture, and optimality was measured as the total weight of the beamlets allowed in the aperture. Whether this will result in the optimal treatment plan is unknown. At this moment we do not yet know how the column generation process responds to these optimal apertures.

In this project, we have not been able to establish what is actually the weight for the truly optimal apertures. Trying to calculate the apertures by an exhaustive search would take years. Not knowing what the actual optimum is means that our results might not be entirely accurate.
A

TABLES WITH RESULTS OF BELLMAN-FORD AND EXTENSIONS
<table>
<thead>
<tr>
<th>Case</th>
<th>Benchmark value</th>
<th>Simple Bellman-Ford</th>
<th>Bellman-Ford with positive elements</th>
<th>Bellman-Ford allowing multiple openings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-4359.2</td>
<td>-3745.9</td>
<td>-3745.9</td>
<td>-3745.9</td>
</tr>
<tr>
<td>2</td>
<td>-4181.6</td>
<td>-3584.4</td>
<td>-3584.4</td>
<td>-3584.4</td>
</tr>
<tr>
<td>3</td>
<td>-3173.2</td>
<td>-2531.9</td>
<td>-2531.9</td>
<td>-2531.9</td>
</tr>
<tr>
<td>4</td>
<td>-2926.2</td>
<td>-2199.3</td>
<td>-2199.3</td>
<td>-2199.3</td>
</tr>
<tr>
<td>5</td>
<td>-4300.9</td>
<td>-3359.7</td>
<td>-3359.7</td>
<td>-3388.9</td>
</tr>
<tr>
<td>6</td>
<td>-4269.3</td>
<td>-3459.4</td>
<td>-3459.4</td>
<td>-3459.4</td>
</tr>
<tr>
<td>7</td>
<td>-2854</td>
<td>-2276.7</td>
<td>-2276.7</td>
<td>-2276.7</td>
</tr>
<tr>
<td>8</td>
<td>-3148.4</td>
<td>-2621.9</td>
<td>-2621.9</td>
<td>-2621.9</td>
</tr>
<tr>
<td>9</td>
<td>-4177.7</td>
<td>-3646.3</td>
<td>-3646.3</td>
<td>-3646.3</td>
</tr>
<tr>
<td>10</td>
<td>-3670.2</td>
<td>-3631.8</td>
<td>-3631.8</td>
<td>-3631.8</td>
</tr>
<tr>
<td>11</td>
<td>-3512.2</td>
<td>-3476.6</td>
<td>-3476.6</td>
<td>-3476.6</td>
</tr>
<tr>
<td>12</td>
<td>-2525.8</td>
<td>-2462.8</td>
<td>-2462.8</td>
<td>-2462.8</td>
</tr>
<tr>
<td>13</td>
<td>-2217.6</td>
<td>-2148.8</td>
<td>-2148.8</td>
<td>-2148.8</td>
</tr>
<tr>
<td>14</td>
<td>-3362.9</td>
<td>-3292.3</td>
<td>-3292.3</td>
<td>-3292.3</td>
</tr>
<tr>
<td>15</td>
<td>-3428</td>
<td>-3378.9</td>
<td>-3378.9</td>
<td>-3378.9</td>
</tr>
<tr>
<td>16</td>
<td>-2263.3</td>
<td>-2214.3</td>
<td>-2214.3</td>
<td>-2214.3</td>
</tr>
<tr>
<td>17</td>
<td>-2595.1</td>
<td>-2543.2</td>
<td>-2543.2</td>
<td>-2543.2</td>
</tr>
<tr>
<td>18</td>
<td>-3565.1</td>
<td>-3535.8</td>
<td>-3535.8</td>
<td>-3535.8</td>
</tr>
<tr>
<td>24</td>
<td>-32.029</td>
<td>-18.875</td>
<td>-21.606</td>
<td>-29.245</td>
</tr>
<tr>
<td>28</td>
<td>-6.2597</td>
<td>-6.0893</td>
<td>-6.1261</td>
<td>-6.1261</td>
</tr>
<tr>
<td>29</td>
<td>-5.714</td>
<td>-4.5336</td>
<td>-4.5336</td>
<td>-5.6451</td>
</tr>
<tr>
<td>30</td>
<td>-4.4329</td>
<td>-3.3653</td>
<td>-3.3653</td>
<td>-4.3938</td>
</tr>
<tr>
<td>31</td>
<td>-4.1755</td>
<td>-2.8596</td>
<td>-2.8596</td>
<td>-4.1201</td>
</tr>
<tr>
<td>32</td>
<td>-6.3359</td>
<td>-6.1298</td>
<td>-6.1603</td>
<td>-6.1603</td>
</tr>
<tr>
<td>33</td>
<td>-6.768</td>
<td>-4.3803</td>
<td>-4.3803</td>
<td>-6.2029</td>
</tr>
<tr>
<td>34</td>
<td>-4.0881</td>
<td>-2.9438</td>
<td>-2.9438</td>
<td>-4.0614</td>
</tr>
<tr>
<td>35</td>
<td>-4.4308</td>
<td>-3.4263</td>
<td>-3.4263</td>
<td>-4.3688</td>
</tr>
<tr>
<td>36</td>
<td>-5.9878</td>
<td>-4.6259</td>
<td>-4.6259</td>
<td>-5.8662</td>
</tr>
<tr>
<td>37</td>
<td>-0.62143</td>
<td>-0.58702</td>
<td>-0.60183</td>
<td>-0.60183</td>
</tr>
<tr>
<td>38</td>
<td>-0.56864</td>
<td>-0.39245</td>
<td>-0.39245</td>
<td>-0.42124</td>
</tr>
<tr>
<td>39</td>
<td>-0.36415</td>
<td>-0.20439</td>
<td>-0.22038</td>
<td>-0.36491</td>
</tr>
<tr>
<td>40</td>
<td>-0.3697</td>
<td>-0.2499</td>
<td>-0.25001</td>
<td>-0.25827</td>
</tr>
<tr>
<td>41</td>
<td>-0.63111</td>
<td>-0.39261</td>
<td>-0.39276</td>
<td>-0.49087</td>
</tr>
<tr>
<td>42</td>
<td>-0.67779</td>
<td>-0.38625</td>
<td>-0.38625</td>
<td>-0.42012</td>
</tr>
<tr>
<td>43</td>
<td>-0.39368</td>
<td>-0.13532</td>
<td>-0.13532</td>
<td>-0.37648</td>
</tr>
<tr>
<td>44</td>
<td>-0.40873</td>
<td>-0.21017</td>
<td>-0.23203</td>
<td>-0.38755</td>
</tr>
<tr>
<td>45</td>
<td>-0.54218</td>
<td>-0.40098</td>
<td>-0.40098</td>
<td>-0.40758</td>
</tr>
<tr>
<td>Case</td>
<td>Optimal connected solution</td>
<td>Result feasible aperture, method II</td>
<td>Result feasible aperture, method III</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>---------------------------</td>
<td>------------------------------------</td>
<td>------------------------------------</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-3745.9</td>
<td>-3720.8</td>
<td>-3744.9</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-3584.4</td>
<td>-3552.9</td>
<td>-3583.9</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-2531.9</td>
<td>-2512</td>
<td>-2529</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-2199.3</td>
<td>-2183.3</td>
<td>-2198.6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-3359.7</td>
<td>-3330.1</td>
<td>-3358.8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-3459.4</td>
<td>-3426.6</td>
<td>-3459.1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>-2276.7</td>
<td>-2256.1</td>
<td>-2276.1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>-2621.9</td>
<td>-2600.7</td>
<td>-2621.4</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>-3646.3</td>
<td>-3620.7</td>
<td>-3644.6</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>-3631.8</td>
<td>-3615.2</td>
<td>-3630.7</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>-3476.6</td>
<td>-3464.9</td>
<td>-3475.9</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>-2462.8</td>
<td>-2449.9</td>
<td>-2462.6</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>-2148.8</td>
<td>-2141.9</td>
<td>-2148.2</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>-3292.3</td>
<td>-3276.2</td>
<td>-3291.2</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>-3378.9</td>
<td>-3364.5</td>
<td>-3377.8</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>-2214.3</td>
<td>-2203.4</td>
<td>-2214.1</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>-2543.2</td>
<td>-2535.2</td>
<td>-2542.6</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>-3535.8</td>
<td>-3523.1</td>
<td>-3535.3</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>-15.324</td>
<td>-14.6</td>
<td>-14.121</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>-11.46</td>
<td>-10.34</td>
<td>-11.065</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>-14.198</td>
<td>-12.751</td>
<td>-13.77</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>-11.705</td>
<td>-10.791</td>
<td>-11.665</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>-12.757</td>
<td>-11.301</td>
<td>-12.614</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>-6.1261</td>
<td>-4.3176</td>
<td>-4.6786</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>-4.5336</td>
<td>-4.2676</td>
<td>-4.36</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>-3.3653</td>
<td>-2.9358</td>
<td>-3.3335</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>-2.8596</td>
<td>-2.457</td>
<td>-2.7754</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>-6.1603</td>
<td>-3.8194</td>
<td>-4.073</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>-4.3803</td>
<td>-4.0064</td>
<td>-4.2937</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>-2.9438</td>
<td>-2.5569</td>
<td>-2.8646</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>-3.4263</td>
<td>-2.9621</td>
<td>-3.3907</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>-4.6259</td>
<td>-4.3386</td>
<td>-4.4643</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>-0.60183</td>
<td>-0.52186</td>
<td>-0.42035</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>-0.39245</td>
<td>-0.15611</td>
<td>-0.20035</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>-0.22038</td>
<td>-0.11741</td>
<td>-0.12121</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>-0.25001</td>
<td>-0.094535</td>
<td>-0.1231</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>-0.39276</td>
<td>-0.338</td>
<td>-0.37424</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>-0.38625</td>
<td>-0.16731</td>
<td>-0.1884</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>-0.13532</td>
<td>-0.09945</td>
<td>-0.12083</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>-0.23203</td>
<td>-0.12036</td>
<td>-0.12663</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>-0.40098</td>
<td>-0.34676</td>
<td>-0.18736</td>
<td></td>
</tr>
</tbody>
</table>
B

IMAGES OF 4 TEST CASES
(a) Case 19

(b) Result case 19, simple Bellman-Ford

(c) Case 19, positive elements allowed

(d) Result case 19, multiple openings allowed

(e) Case 19 feasible solution, method II

(f) Case 19 feasible solution, method III

Figure B.1
(a) Case 21

(b) Result case 21, simple Bellman-Ford

(c) Case 21, positive elements allowed

(d) Result case 21, multiple openings allowed

(e) Case 21 feasible solution, method II

(f) Case 21 feasible solution, method III

Figure B.2
Figure B.3

(a) Case 26
(b) Result case 26, simple Bellman-Ford
(c) Case 26, positive elements allowed
(d) Result case 26, multiple openings allowed
(e) Case 26 feasible solution, method II
(f) Case 26 feasible solution, method III
Figure B.4

(a) Case 37
(b) Result case 37, simple Bellman-Ford
(c) Case 37, positive elements allowed
(d) Result case 37, multiple openings allowed
(e) Case 37 feasible solution, method II
(f) Case 37 feasible solution, method III
C.1. FINDING ALL SIMPLE SPOTS

In this program the goal was to find all spots. The program starts with a matrix that has ones for all negative elements and a zero for all non-negative elements in the original matrix. The matrix is searched for rows of ones, and all the spots are saved as three coordinates in the matrix y. The coordinates are the row number the spot lies in, the setting of the left leaf and the setting of the right leaf.

```matlab
function y = pos_row(x)
num_pos=1;
y(1,num_pos)=0;
y(2,num_pos)=0;

% The case where only the first digit is a one
if x(1) == 1
    y(1,num_pos)=0;
    j = 1;
    while x(j)>0 && j< length(x)
        j = j+1;
    end
% catch exeption where all the apertures are switched off
if j == length(x) && x(j)>0
    j = j+1;
end
y(2,num_pos)=j;
num_pos = num_pos +1;
end

% Now we look at the rows with first digit 0
for i=1:length(x)-2
% we zeeken de eerste 1
    if sum(x(1:i))==0 && sum(x(1:i+1))==1
        y(1,num_pos)=i;
        j=i+1;
        while x(j)>0 && j< length(x)
            %
```
j = j + 1;
end
% catch exception where the last digit in the row is a
% one
if j == length(x) && x(j) > 0
j = j + 1;
end

y(2, num_pos) = j;
num_pos = num_pos + 1;
end

% find the first 1 that is not on coordinate 1
if sum(x(1:i)) > 0
  if sum(x(1:i)) == sum(x(1:i + 1))
    % now we search if after this zero we can find another 1.
    % when we find it,
    % we check the length of this sequence of one's
    if sum(x(1:i+2)) > sum(x(1:i + 1))
      y(1, num_pos) = i + 1;
      j = i + 3;
      while j < length(x) && x(j) > 0
        j = j + 1;
      end
      % catch exception where the last digit in the row is
      % a one
      if j == length(x) && x(j) > 0
        j = j + 1;
      end
      y(2, num_pos) = j;
      num_pos = num_pos + 1;
    end
  end
end
end
end

if x(length(x)) == 1 && x(length(x) - 1) == 0 && sum(x(1:length(x) - 1)) == 0
  y(2, num_pos) = length(x) + 1;
y(1, num_pos) = length(x) - 1;
num_pos = num_pos + 1;
end
end
C.2. **Including spots with positive elements**

In this function we start by calling the function `pos_row` in order to find all the spots that only allow negative elements. Then we check all the rows, and add all possible combinations of spots that lie in the same row. Again the coordinates of the spots are saved as a vector $y$ that has three coordinates for each spot.

```matlab
% Add extra options created by allowing positive elements to the collection of possible spots

function y = pos_row_t_ext(x)
    y = [0;0;0];
    [~,y_begin] = Treeww_t_ext(x);
    teller =1;
    for j = 1:size(y_begin,2)
        y_t = y_begin';
        k = find(y_t(:,1)==j);
        for p=1:size(k,1)
            for q=p:size(k,1)
                y(1,teller) = j;
                y(2,teller) = y_begin(2,k(p));
                y(3,teller) = y_begin(3,k(q));
                teller = teller+1;
            end
        end
    end
end
```
C.3. Creating a graph from the possible spots

In this function we start with the original matrix again. Then we call a vector that contains the three coordinates of the spots we would like to use to form the graph. After that, we check which spots are connected and add them to the adjacency matrix. This matrix is then returned.

```matlab
% creates the graph of a matrix x
function [tree, y] = Treeww_t_ext(x)
% x is the original matrix

x_disc = (x < 0);  % save begin values
tel = 0;
prev_num_opt = 0;
prev2_num_opt = 0;
begin = 2;
while tel == 0
    tel = pos_row(x_disc(:, begin - 1));
    tree(begin, begin) = 1;
    begin = begin + 1;
end
begin = begin - 1;
% create begin option matrix
y = pos_row(x_disc(:, begin - 1));
ind1 = (begin - 1) * ones(1, size(y, 2));
y = [ind1; y];

% create rest option matrix
for i = begin: size(x, 2)
    % look at all the rows of the matrix
    prev_num_opt = size(y, 2);
    % add extra possibilitys to y
    pos = pos_row(x_disc(:, i));
    ind = [i * ones(1, size(pos, 2)); pos];
    y = [y ind];

    % check for possibilitys for connections
    for j = (prev_num_opt + 1): size(y, 2)
        tree(j + 1, j + 1) = 1;
        for k = (prev2_num_opt + 1): prev_num_opt
            if not(y(3, j) - 1 <= y(2, k) || y(3, k) - 1 <= y(2, j))
                tree(j + 1, k + 1) = sum(x((y(2, j) + 1):(y(3, j) - 1), y(1, j)));  
            end
        end
    end
    prev2_num_opt = prev_num_opt;
end

% attach to start node
for i = 2: size(y, 2)
    if sum(tree(i, 1:i - 1)) == 0
```

```
tree(i,1)=\min(\text{sum}(x((y(2,i-1)+1):(y(3,i-1)-1),y(1,i-1))),0);

% make diagonal zero
for i=1:size(tree,1)
    tree(i,i) = 0;
end
C.4. Creating a graph when positive elements are allowed

In this function we again start with the original matrix. Then the vector with the coordinates of the spots is called. Here we use the vector which also contains the spots with positive elements. After that, we check which spots are connected and add them to the adjacency matrix. This matrix is then returned.

```matlab
%create graph with extra positive elements added
function [graph,y] = Treeww_t_addopt(x)
graph = zeros(10,10);
y = pos_row_t_ext(x);
if sum(y(3,:))>0
    prev2_num_opt = 0;
    %Create rest optionmatrix
    for i=min(y(1,:)):max(y(1,:))-1
        if find(y(1,:)==i)>0
            prev_num_opt = find(y(1,:)==i);
            prev_num_opt = prev_num_opt(end);
            next_num_opt = find(y(1,:)==i+1);
            %Check for connections with old possibilities
            for j = (prev_num_opt+1):next_num_opt(end)
                graph(j+1,j+1) = 1;
                for k = (prev2_num_opt+1):prev_num_opt
                    if not(y(3,j)-1<=y(2,k) || y(3,k)-1<= y(2,j))
                        graph(j+1,k+1)=sum(x((y(2,j)+1):(y(3,j)-1),y(1,j)));
                    end
                end
            end
            prev2_num_opt = prev_num_opt;
        end
    end
    %Attach to start node
    for i=2:size(y,2)
        if sum(graph(i,1:i-1))==0
            graph(i,1)=min(sum(x((y(2,i-1)+1):(y(3,i-1)-1),y(1,i-1))),0);
        end
    end
    %Make diagonal zero
    for i=1:size(graph,1)
        graph(i,i) = 0;
    end
```
C.5. FINDING THE SHORTEST PATH USING BELLMAN-FORD

This function first calls the adjacency matrix of the given matrix. Then it calculates all the distances from the start node to all other nodes. After all minimal distances have been calculated, the optimal path is backtracked through the adjacency matrix and the path returned. This algorithm works with the adjacency matrix with, and without the positive elements.

% recognising and calculating the shortest path in a graph using the Bellman-Ford algorithm

function [pad] = Bel_Ford_ext(x)
%
creating matrix with weights
for i = 1:sum(sum(x < 0))
    weights(:, i) = [0; inf * ones(size(x, 2), 1)];
end
for i = 1:sum(sum(x < 0)) - 1
    if (i > 1 && sum(not(weights(:, i) == weights(:, i - 1)) == 0))
        % We repeat the process until there are no longer improvements possible. The maximal amount of repetitions is the number of edges in the graph.
        for j = 1:size(x, 2)
            for k = 1:size(x, 1)
                if x(k, j) < 0
                    if weights(j, i) + x(k, j) < weights(k, i + 1)
                        weights(k, i + 1) = weights(j, i) + x(k, j);
                    end
                end
            end
        end
    end
check = exist('weights');
if check == 0
    weights = 0;
end
% Now we backtrace our route to find the path that delivers the smallest distance
[minimum, pad(size(weights, 2))] = min(weights(:, end));
i_n = pad(size(weights, 2));
i_old = i_n;
for k = 1:size(weights, 2) - 1
    if i_n > 1
        i = find(not(x(i_n, :) == 0));
        for j = 1:size(1, 2)
            if weights(i(j), size(weights, 2)) + x(i_old, i(j)) == minimum
                i_n = i(j);
            end
        end
    end
pad(size(weights, 2) - k) = i_n;
i_old = i_n;
minimum = weights(i_n, size(weights,2)-k);
end
end
C.6. SOLUTION MATRIX USING THE SIMPLE BELLMAN-FORD APPROACH

This function combines multiple functions. It returns the optimal matrix for the simple Bellman-Ford approach. It begins with an aperture matrix, then calls for the possible spots. After that the adjacency matrix is called and finally the Bellman-Ford algorithm is called. Then it translates the path back to spots and then the spots are translated back to a matrix. This function is the function used to calculate the results of the simple Bellman-Ford approach. All images of the apertures in section 4.4 were produced using this function.

```matlab
function [matrix, y_opt] = big_opp_t_ext (x)
%create and save graph of matrix x
[~, y] = Treeww_t_ext (x);

%Find optimal path through the graph of matrix x
pad = Bel_Ford_ext (Treeww_t_ext (x));
pad = pad - 1
indices = find (not (pad <= 0));
pad_cor = pad (indices);
y_opt = y(:, pad_cor)
%recreate matrix
matrix = Matrix_t (y_opt, x);
```
This function combines multiple functions. It returns the optimal matrix for the Bellman-Ford approach when positive elements are allowed. It begins with a aperture matrix, then calls for the possible spots, using the function that takes spots with positive elements in account. After that the adjacency matrix is called and finally the Bellman-Ford algorithm is called. Then it translates the path back to spots and then the spots are translated back to a matrix. This function was used to calculate the apertures resulting from using the Bellman-Ford approach when positive elements are allowed. All the images of the apertures in section 5.1 where produced using this function.

```matlab
% calculate optimal aperture when positive elements are allowed
function [matrix, y_opt] = big_opp_t_addopt(x)
% create graph of matrix
[~, y] = Treeww_t_addopt(x);
% create optimal path
pad = Bel_Ford_ext(Treeww_t_addopt(x));
pad = pad - 1;
indices = find(not(pad<=0));
pad_cor = pad(indices);
y_opt = y(:, pad_cor);
% create matrix from path
matrix = Matrix_t(y_opt, x);
```
C.8. Solution matrix using the Bellman-Ford approach with multiple openings allowed

This function starts with calling the optimal solution from the approach that allowed positive elements. Then that section is removed from the original matrix and the next best solution is sought for. When it is found, it is checked for the minimal area constraint and when it is sufficient the second area is added to the solution. When the area is to small, the area is also removed from the original matrix. This is repeated until there are not sufficient negative elements left to form a opening that has sufficient area. Then the collection of spots is again transformed back to a matrix. This function was used to calculate the apertures resulting from using the Bellman-Ford approach when multiple openings are allowed. All the images of the apertures in section 6.1 where produced using this function.

```matlab
function [ matrix , y_opt , x ] = big_opp_t_multsurf ( x )
    %read in standard constants
    matrix = zeros(size(x));
    surface = Inf;
    M_next = x;
    %get most negative linked aperture
    [M, y_opt ] = big_opp_t_addopt ( x ) ;
    while sum(sum(M) ) <0
        %Add new surface to old aperture when surface is sufficient
        if surface >5
            matrix = matrix + M;
        end
        %Remove prev aperture from matrix
        M1 = sum(M, 1 ) ;
        for i =1: size ( matrix , 2 )
            if not (M1(i)==0)
                M_next(:,i) = zeros(size(M_next,1),1);
            end
        end
        if sum(sum(M_next<0) ) >0
            %Find next best aperture
            [M, ~ ] = big_opp_t_addopt ( M_next ) ;
        else
            M = zeros(size(M));
        end
        %calculate surface
        surface = sum(sum(M~=0));
    end
```
C.9. **TRANSFORMING A COLLECTION OF SPOTS BACK TO A MATRIX**

This program simply adds all solution spots to an empty matrix, resulting in the final matrix.

```matlab
function M = Matrix_t(y, x)

M = zeros(size(x));

i = 1;
while i <= max(y(1,:))
    j = find(y(1,:) == i);
    for k = 1:length(j)
        M(y(2,j(k)) + 1:(y(3,j(k)) - 1, i) = x(y(2,j) + 1:(y(3,j) - 1), i);
    end
    i = i + 1;
end
```

C.10. FINDING A FEASIBLE SOLUTION: METHOD I

This function implements method I for creating a feasible solution. It returns a warning when there are not sufficient negative elements to create a feasible solution. Method I is explained in section 7.1.

```matlab
% In this approach an attempt is made to create a feasible aperture
% In this approach zeros from outside the domain are allowed in the
% solution, which makes the solution infeasible. The solution does however
% comply with the rest of the requirements.

function [matrix] = aperture3_doc(x)

% config multileaf collimator
DEV.MLC.LeafLayout = 3.0 * ones(1, 39);
DEV.MLC.Dimension = [132 sum(DEV.MLC.LeafLayout)];
DEV.MLC.Overtravel = Inf;
DEV.MLC.LeafGap = 0;
DEV.MLC.MinAperture = 75; % mm²
DEV.MLC.MinXOpening = 7; % mm
DEV.MLC.MinYOpening = 3 * 3.0; % mm 3 leaves, at any point there should be 3 leaves open.
DEV.MLC.AperturesPerSegment = 1; % 1 opening per segment

% transform distances to elements in matrix
hor_dist_element = DEV.MLC.Dimension(1) / size(x, 1);
min_hor_gap = ceil(DEV.MLC.MinXOpening/hor_dist_element);
min_ver_gap = DEV.MLC.MinYOpening/DEV.MLC.LeafLayout(1);
min_surf = ceil(DEV.MLC.MinAperture/(DEV.MLC.LeafLayout(1)*hor_dist_element));

% calculate how many borders must be removed and later added
num_border = ceil(0.5*(DEV.MLC.MinXOpening−hor_dist_element)/hor_dist_element);
y2=x;
se = strel('square',3);

% Find optimal solution for figure without outer border
[matrix,~] = big_opp_t_addopt(y2);

% redraw border(s)
for i=1:num_border
  % Search boundary of figure
  non_zero = find(abs(x)>0);
  Boundary = cell2mat(bwboundaries(matrix ~=0,4));
  BM = zeros(size(matrix));
  for i=1:size(Boundary,1)
    BM(Boundary(i,1),Boundary(i,2))=1;
  end
end

% expand border
BM = imdilate(BM,se);
dialatie =imdilate(not(matrix==0),se);
```
% Add outer border
for i=1:size(matrix,1)
    for j=1:size(matrix,2)
        if dilation(i,j)==1
            matrix(i,j)=x(i,j);
        end
        if dilation(i,j)==1 && x(i,j)==0
            matrix(i,j)=-10;
        end
    end
end

% checking for minimal area constraint
surf = sum(sum(abs(matrix)>0));
if surf<min_surf
    warning('case' + k1 + ' has a too small area')
end
C.11. FINDING A FEASIBLE SOLUTION: METHOD II

This function implements method II for creating a feasible solution. It returns a warning when there are not sufficient negative elements to create a feasible solution. Method II is explained in section 7.2

```matlab
%Create a feasible aperture
%In this approach elements where added to the optimal aperture
function [matrix] = aperture_doc(x)

%config multileaf collimator
DEV.MLC.LeafLayout = 3.0*ones(1, 39);
DEV.MLC.Dimension = [132 sum(DEV.MLC.LeafLayout)];
DEV.MLC.Overtravel = Inf;
DEV.MLC.LeafGap = 0;
DEV.MLC.MinAperture = 75; % mm^2
DEV.MLC.MinXOpening = 7; % mm
DEV.MLC.MinYOpening = 3*3.0; % mm 3 leafs, at any point there should be 3 leafs open.
DEV.MLC.AperturesPerSegment = 1; % 1 opening per segment

%transform distances to elements in matrix
hor_dist_element = DEV.MLC.Dimension(1)/size(x,1);
min_hor_gap = ceil(DEV.MLC.MinXOpening/hor_dist_element);
min_ver_gap = DEV.MLC.MinYOpening/DEV.MLC.LeafLayout(1);
min_surf = ceil(DEV.MLC.MinAperture/(DEV.MLC.LeafLayout(1)*hor_dist_element));

%calculate how many borders must be removed and later added
num_border = ceil(0.5*(DEV.MLC.MinXOpening-hor_dist_element)/hor_dist_element);
y2=x;

%Save figure without outer border(s)
for i=1:num_border
    se = strel('square',3);
    y = imerode(abs(y2)>0,se);
    y2 = zeros(size(y));
    for j=1:size(y,2)
        for i=1:size(y,1)
            if y(i,j)==1
                y2(i,j)=x(i,j);
            end
        end
    end
end

%Find optimal solution for figure without outer border
[matrix,-] = big_opp_t_addopt(y2);

%redraw border(s)
for i=1:num_border
    %Search boundary of figure
    non_zero = find(abs(x)>0);
    Boundary = cell2mat(bwboundaries(matrix ~=0,4));
```

BM = zeros(size(matrix));
for i = 1:size(Boundary,1)
    BM(Boundary(i,1), Boundary(i,2)) = 1;
end

% expand border
BM = imdilate(BM, se);
dialatie = imdilate(~(matrix == 0), se);

% Add outer border
for i = 1:size(matrix, 1)
    for j = 1:size(matrix, 2)
        if dialatie(i, j) == 1
            matrix(i, j) = x(i, j);
        end
        if dialatie(i, j) == 1 && x(i, j) == 0
            matrix(i, j) = -10;
        end
    end
end

% checking for minimum area constraint
surf = sum(sum(abs(matrix) > 0));
if surf < min_surf
    warning('case has a too small area')
end
**C.12. FINDING A FEASIBLE SOLUTION: METHOD III**

This function implements method III for creating a feasible solution. It returns a warning when the resulting area has an insufficiently large area, and returns an error when there are not enough negative elements present to create a sufficiently large area. Method III is explained in section 7.3.

```matlab
%Create a feasible aperture with removing elements from the optimal matrix.
%This function also takes into account that removing elements may mean
%that the solution is no longer connected.

function [matrix] = aperture2_doc(x)

%configure multileaf collimator
DEV.MLC.LeafLayout = 3.0*ones(1, 39);
DEV.MLC.Dimension = [132 sum(DEV.MLC.LeafLayout)];
DEV.MLC.Overtravel = Inf;
DEV.MLC.LeafGap = 0;
DEV.MLC.MinAperture = 75; % mm^2
DEV.MLC.MinXOpening = 7.0; % mm
DEV.MLC.MinYOpening = 3*3.0; % mm 3 leaves, at any point there should be 3 leaves open.
DEV.MLC.AperturesPerSegment = 1; % 1 opening per segment

%transform distances to elements in matrix
hor_dist_element = DEV.MLC.Dimension(1)/size(x,1);
min_hor_gap = ceil(DEV.MLC.MinXOpening/hor_dist_element);
min_ver_gap = DEV.MLC.MinYOpening/DEV.MLC.LeafLayout(1);
min_surf = ceil(DEV.MLC.MinAperture/(DEV.MLC.LeafLayout(1)*hor_dist_element));

%calculate how many borders must be removed and later added
num_border = ceil(0.5*(DEV.MLC.MinXOpening-hor_dist_element)/hor_dist_element);
y2=x;

%Find optimal solution for figure without outer border
[_,~] = big_opp_t_addopt(y2);
y2=matrix;

%give error when there are not enough negative numbers left in the matrix to
%create an opening that is sufficiently large
if sum(sum(x<0))<min_surf
    error('No area is large enough')
end

%Save figure without outer border(s)
for i=1:num_border
    se = strel('square',3);
y = imerode(abs(y2)>0,se);
y2 = zeros(size(y));
    for j=1:size(y,2)
        for i=1:size(y,1)
            if y(i,j)==1
                y2(i,j)=x(i,j);
        end
    end
end
```
for i=1:num_border
    %Search boundary of figure
    non_zero = find(abs(x)>0);
    Boundary = cell2mat(bwboundaries(matrix ~=0 ,4));
    BM = zeros(size(matrix));
    for i=1:size(Boundary,1)
        BM(Boundary(i,1),Boundary(i,2)) =1;
    end

    % expand border
    BM = imdilate(BM, se);
    dialatation =imdilate(not(matrix==0), se);

    %Add outer border
    for i=1:size(matrix,1)
        for j=1:size(matrix,2)
            if dialatation(i,j)==1
                matrix(i,j)=x(i,j);
            end
            if dialatation(i,j)==1&&x(i,j)==0
                matrix(i,j)=-10;
            end
        end
    end

    %check connectedness and remove all not connected areas that are not %optimal
    CC = bwconncomp(abs(matrix)>0,4);
    for i=1:size(CC.PixellIdxList,2)
        if length(CC.PixellIdxList(i))<min_surf
            CC.PixellIdxList{i}=0;
            CC.Weight{i}=100;
        else
            CC.Weight{i} = sum(sum(matrix(ind2sub(size(matrix), CC.PixellIdxList{i}))));
        end
    end
    [~,index] = min(cell2mat(CC.Weight));
    large_weight = CC.PixellIdxList{index};
    matrix2 = zeros(size(matrix));

    %check whether the found solution has a large enough area
    if large_weight==0
        warning('No area is large enough, return to original matrix')
    end
\begin{verbatim}
107  \( x(\text{not}(\text{matrix}==0))=0; \)
108  \text{matrix} = \text{aperture2}(x);
109  \text{else}
110
111
112
113
114  \textbf{for} i=1:length(large_weight)
115  \hspace{1em} J = \text{ceil}(\text{large_weight}(i)/\text{size}(\text{matrix},1));
116  \hspace{1em} I = \text{mod}([\text{large_weight}(i),\text{size}(\text{matrix},1)));
117  \hspace{1em} \text{if} \ I==0
118  \hspace{2em} I=\text{size}(\text{matrix},1);
119  \hspace{1em} \text{end}
120  \hspace{1em} \text{matrix2}(I,J) = x(I,J);
121  \text{end}
122
123
124
125
126  \%\text{checken for minimal area constraint}
127  \text{surf} = \text{sum}([\text{sum}(|\text{matrix}|>0));
128  \text{if} \ \text{surf}<\text{min}\_\text{surf}
129  \hspace{1em} \text{disp('current case has a to small area')}
130  \hspace{1em} x(\text{not}(\text{matrix}==0))=0;
131  \hspace{1em} \text{matrix} = \text{aperture2}(x);
132  \text{end}
133
134  \text{end}
\end{verbatim}
