**Delft University of Technology**
**Faculty Electrical Engineering, Mathematics and Computer Science**
**Delft Institute of Applied Mathematics**

# The Classroom Problem.

**An algorithm to find an optimal classroom lay-out.**

**(Dutch title: Het Klaslokaal-probleem.**

**Een algoritme voor een optimale klaslokaalindeling.)**

A thesis submitted to the
Delft Institute of Applied Mathematics
as part to obtain the degree of

**BACHELOR OF SCIENCE**
**in**
**APPLIED MATHEMATICS**

**by**

**MORTEN WESSELS**

**Delft, the Netherlands**
**July 2023**

BSc thesis APPLIED MATHEMATICS

# "The Classroom Problem. An algorithm to find an optimal classroom lay-out."

(Dutch title: "Het Klaslokaal-probleem. Een algoritme voor een optimale klaslokaalindeling.")

MORTEN WESSELS

Delft University of Technology

**Supervisor**

Dr.ir. M. Keijzer

**Other members of the committee**

Dr. D.C. Gijswijt Dr. J.G. Spandaw

July, 2023 Delft

# Abstract

"Who do you want to sit next to?" In this thesis we develop an algorithm that optimizes a classroom lay-out based on this question. Besides the couples that sit next to each other, it also matters who sits in front (or behind). Even the relations across the aisles matter in the lay-out optimization. Each student chooses three other students and divides ten point points between them, thus creating a weighted graph with potential matches. To fill the classroom we have to address three problems: "Who are the couples?", "Where do the couples sit?" and "How do the couples sit?". First, three algorithms for creating couples are explained. Two of these will, ultimately, not be used. The third is the Blossom algorithm. This algorithm is the one we will be using to create the couples. To determine the positions of the couples, we first use the Repeated Nearest Neighbor algorithm to create a string of couples. Secondly, we use a brute-force approach to determine in which direction this string fills the classroom. For the last problem another brute-force approach is used to determine the orientation of each of the couples: "Who sits at the left / right table?" Finally, the created algorithm is tested on an actual class.

# Contents

**1 Introduction** **9**

**2 The Marriage Problem** **13**
- 2.1 The algorithm . . . 13
- 2.2 Example . . . 14
- 2.3 Application to the Classroom Problem . . . 16

**3 The Roommate Problem** **19**
- 3.1 Phase 1 of the algorithm . . . 21
  - 3.1.1 Proposing . . . 21
  - 3.1.2 When proposing ends in state 1: No stable matching . . . 21
  - 3.1.3 When proposing ends in state 2: Phase 1 reduction . . . 22
- 3.2 Phase 2 of the algorithm . . . 22
  - 3.2.1 Recognition of an all-or-nothing cycle . . . 22
  - 3.2.2 Phase 2 reduction . . . 23
- 3.3 Example . . . 24
- 3.4 Application to the Classroom problem . . . 26

**4 Phase 1: The Blossom Algorithm** **27**
- 4.1 The algorithm . . . 29
- 4.2 Algorithm: Initialisation . . . 30
- 4.3 Algorithm: Primal updates . . . 31
  - 4.3.1 Augment . . . 31
  - 4.3.2 Grow . . . 31
  - 4.3.3 Shrink . . . 32
  - 4.3.4 Expand . . . 32
- 4.4 Algorithm: Dual updates . . . 33
- 4.5 Algorithm: Completion . . . 33
- 4.6 Algorithm: Complexity . . . 33
- 4.7 Example . . . 34

**5 Phase 2: The Repeated Nearest Neighbor Algorithm** **41**
- 5.1 The algorithm . . . 42
- 5.2 Example . . . 43
- 5.3 Application to the classroom problem . . . 46

**6 Phase 3: String Orientation and Selection** **47**
- 6.1 The algorithm . . . 48
- 6.2 Example . . . 49

**7 Phase 4: Couple Orientation** **51**
7.1 The algorithm . . . 52
7.2 Example . . . 53

**8 Example: A class of 28 students** **55**
8.1 Phase 1: Blossom . . . 55
8.2 Phase 2: RNNA . . . 57
8.3 Phase 3: String orientation . . . 59
8.4 Phase 4: Couple orientations . . . 61

**9 Discussion** **65**
9.1 Triplets . . . 65
9.2 Phase 2, 3 and the RNNA . . . 66
9.3 Possible improvements . . . 66
9.4 Further research questions . . . 66

**10 Conclusion** **69**
10.1 Complexity . . . 69
10.2 Phase 1 . . . 70
10.3 Phase 2 and Phase 3 . . . 70
10.4 Phase 4 . . . 70
10.5 Data . . . 70

**Appendices** **71**

**Appendix A: Matlabcode for Gale-Shapley algorithm** **73**

**Appendix B: Matlabcode for the Roommate Problem** **75**

**Appendix C: Matlabcode for the Repeated Furthest Neighbor algorithm** **79**

**Appendix D: Matlabcode for phase 3** **81**

**Appendix E: Matlabcode for phase 4** **83**

# Chapter 1

# Introduction

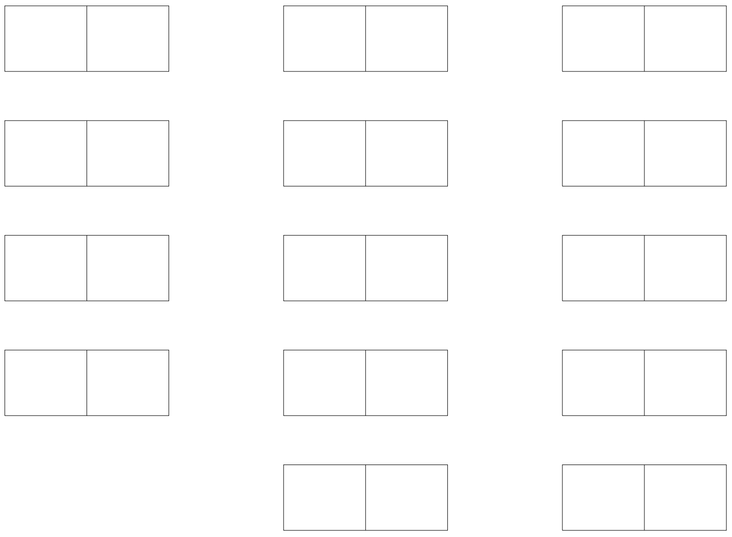
A few years ago I decided to become a math teacher. A colleague of mine posted a story on her Facebook timeline.[9] The story was about a math teacher in the US. This teacher gave her classes two questions at the end of the week: “Who do you want to sit next to next week?” and “Who would you like to nominate for excellent classroom behaviour?” The children were aware that it is not guaranteed that they will sit next to the person of their choice. These questions seem like a great way to motivate the children, but the actual plan with the answers is even more astonishing. The teacher did not use the answers to make the children happy. She used the answers to find out which children were struggling socially. Based on the answers she would make a new classroom lay-out, placing socially weaker children next to socially stronger children.

When I had to pick a subject for my thesis, I wanted to choose a subject that fits with my ambition to become a math teacher. I remembered the story above and wondered if it would be possible to develop a tool that would make a classroom lay-out based on the answers of the story. This tool could be used by every teacher to make an efficient classroom lay-out for their class.

The two questions the teacher in the above story uses are generally used for making sociograms. Sociograms are used to map social relations within a group. Many questions can be used to make sociograms representing different information. It is also possible to have different optimums for the classroom lay-out. You could make a classroom lay-out to maximize the social cohesion of the class or to maximize the learning potential during class. Different goals for the classroom lay-out will still use the same algorithm. The difference will be that the information going into the algorithm must be specified to the goal that is wished.

Instead of developing the tool entirely this thesis will limit itself to the algorithm. The actual tool requires, besides from the algorithm itself, different questions, different optimums and different parameters for the questions belonging to each optimum. The development of this part is not a mathematical problem. Because all the possible optimums will use the same algorithm, this thesis will be limited to developing the algorithm. The algorithm will be developed using the question “‘Who do you want to sit next to?” as an example.

The goal of the algorithm is to determine an optimal classroom lay-out. Throughout this thesis we are trying to decide who sits where in a classroom with the seats located as follows:



To achieve this, the algorithm is split up into four phases. In the first phase we try to create couples. So we need an algorithm to determine which students sit next to each other. In the second phase we need to determine how the couples are linked together, so we can create a string. This string has two possible orientations. Therefore we will determine which orientation is best in the third phase of the algorithm. At this point the couples will all have received their locations in the classroom. In the fourth and final phase we will determine, for each couple, who sits at the right table and who sits at the left table.

In the search of the correct algorithm to make the couples (children sitting next to each other) a few algorithms were considered. The two algorithms I did consider but which in the end didn't make the cut, will be discussed in Chapters 2 and 3. The first algorithm I decided not to use is the Gale-Shapley algorithm which solves the Marriage Problem. The second algorithm is an adaptation of the Gale-Shapley algorithm that solves the Roommate Problem.

From Chapter 4 onward, the phases of the algorithm for solving the classroom problem are discussed. The algorithm consists of four phases. The first two phases use known algorithms, with a slight adaptation for phase 2. The final two phases are brute-force calculations. Schematically, the algorithm for solving the classroom problem consists of the following phases:

Classroom problem:

| phase | problem | algorithm |
|---|---|---|
| 1 | create couples | Blossom |
| 2 | order couples into a string | Repeated Nearest Neighbor |
| 3 | orientation of the string | Brute-force |
| 4 | couple orientation | Brute-force |

In Chapter 4, the Blossom algorithm for solving a minimum cost perfect matching problem is discussed. This algorithm provides the couples that will fill up the classroom. Chapter 5 is about the Repeated Nearest Neighbor algorithm. This algorithm gives an approximation of the shortest hamiltonian circuit in a complete graph. With a little adaptation, this algorithm provides us with a string that orders the couples. This string holds the information which couples are sitting behind / in front of each other. Once we have this string, phase 3 is a brute-force calculation to decide in which direction the string must fill the classroom. The final phase is a brute-force calculation to decide who sits at the right table and who sits at the left table within each couple. After discussing the different parts of the algorithm an example for an actual class is worked out in Chapter 8.

# Chapter 2

# The Marriage Problem

At first the problem to create couples in a classroom to sit next to each other seems similar to the Marriage Problem. In this chapter the problem and the solution algorithm are explained. However, this algorithm will turn out not to be applicable to the Classroom Problem.

Consider a community with $n$ men and $n$ women, all unmarried. Each person, man or woman, ranks those of the opposite sex in accordance to their preferences for a marriage. The question to solve is: Is it possible to find a stable set of marriages for any pattern of preferences? In this, use the following definition:

**Definition 1.** *A set of marriages is called* ***unstable*** *if one of the men and one of the women are not married to each other but prefer each other to their current spouses.*

**Definition 2.** *A set of marriages is called* ***stable*** *if the set of marriages is not unstable.*

In 1962 D. Gale and L.S. Shapley [3] introduced an algorithm that would produce a stable set of marriages (or matches) for this problem (of any size $n$). This is known today as the Gale-Shapley algorithm.

In the next section the algorithm will be explained and afterwards the algorithm will be used to prove that a stable set of marriages exists for any given pattern of preferences.

## 2.1 The algorithm

The Gale-Shapley algorithm consists of a number of iterations:

***Step* 1:** Each man proposes to his favorite woman. Each woman who received one or more proposals rejects all but her favorite among the proposals she received (rejecting nobody if she received one proposal). Instead of accepting the proposal right away each woman will hold the proposal in consideration, to allow for a possible better partner who may come along.

***Step* 2:** Each man either has his proposal in consideration or has been rejected. Every man who has been rejected proposes to his next favorite woman. Each woman again takes the proposal of the man she most prefers into consideration while rejecting the other proposals (if any).

***End*:** If every man has a proposal in consideration and therefore every woman is holding a proposal in consideration, the algorithm stops and all proposals become engagements. Otherwise go back to step 2.

A Matlabcode for the Gale-Shapley algorithm is included in Appendix A.

**Theorem 1.** *There always exists a stable set of marriage.*

*Proof.* Suppose that John and Jane are not married to each other but John prefers Jane to his current wife. According to the algorithm it must have happened that John proposed to Jane at a certain point and has been rejected by Jane in favor of someone else. It is clear that Jane prefers her current husband to John and there is no instability. This argument is valid for every man. Therefore the set of marriages produced by the Gale-Shapley algorithm is stable. □

It is important to notice that a stable set of marriages is not unique by default. The same algorithm could be used where instead of men proposing to women, the women are proposing to the men. Generally this results in a different stable set of marriages. The algorithm can be used to find a male-optimal or a female-optimal stable set of marriages. When there is a unique stable set of marriages the male-optimal solution equals the female-optimal solution.

## 2.2 Example

With the next example the working of the algorithm will be shown step by step. Consider the following ranking:

| | $w_1$ | $w_2$ | $w_3$ | $w_4$ |
|---|---|---|---|---|
| $m_1$ | 1, 3 | 2, 3 | 3, 2 | 4, 3 |
| $m_2$ | 1, 4 | 4, 1 | 3, 3 | 2, 2 |
| $m_3$ | 2, 2 | 1, 4 | 3, 4 | 4, 1 |
| $m_4$ | 4, 1 | 2, 2 | 3, 1 | 1, 4 |

The first number of each pair in the table above gives the ranking of women by the men and the second number is the ranking of men by the women. Thus $m_1$ ranks $w_1$ first, and $w_1$ ranks $m_4$ first.

ITERATION 1: Each man proposes to his favorite woman. Thus $m_1$ proposes to $w_1$, $m_2$ proposes to $w_1$, $m_3$ proposes to $w_2$ and $m_4$ proposes to $w_4$ (see figure 2.1($a$)). Now $w_1$ has been proposed to twice (the other women have been proposed to just once or not), thus $w_1$ takes the proposal of $m_1$ into consideration because she prefers him to $m_2$ and rejects $m_2$. $w_2$ and $w_4$ take the proposals of $m_3$ and $m_4$ (respectively) into consideration (figure 2.1($b$)).

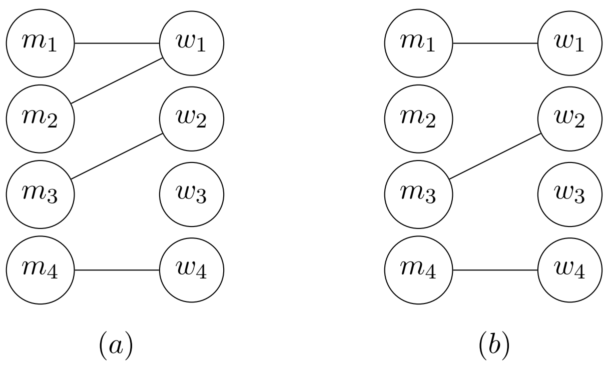
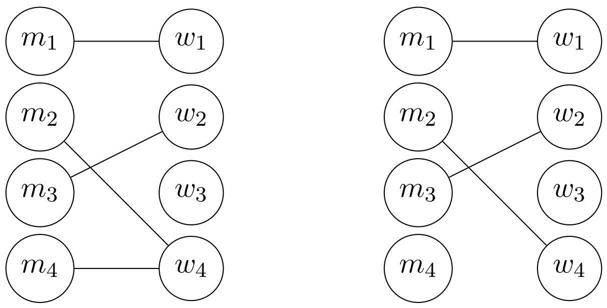
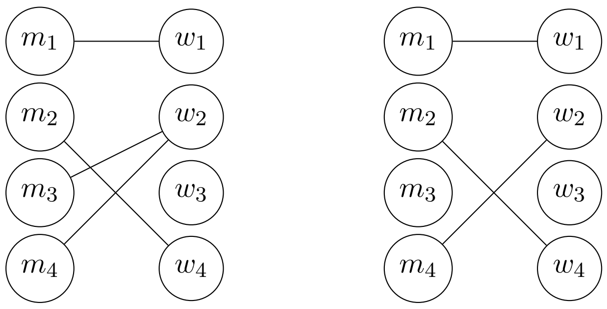


Figure 2.1: Iteration 1 ($a$) the men propose to the women. ($b$) the women take the best proposals into consideration

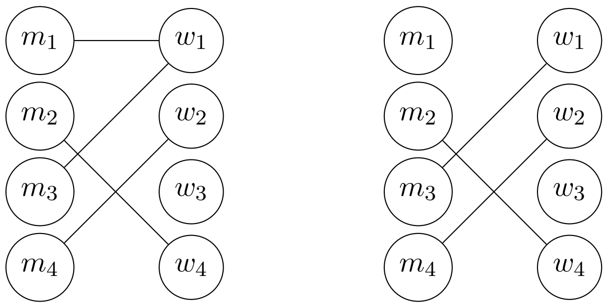
ITERATION 2: Only $m_2$ has been rejected in the previous iteration. So $m_2$ proposes to his next favorite woman $w_4$. Now $w_4$ holds two proposals. $w_4$ takes the proposal from $m_2$ into consideration and rejects $m_4$.



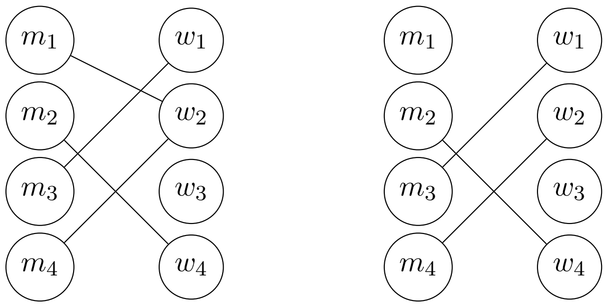
ITERATION 3: Only $m_4$ has been rejected in the previous iteration. $m_4$ now proposes to $w_2$. $w_2$ now has two proposals, because she prefers $m_4$ to $m_3$, she takes the proposal from $m_4$ into consideration and rejects $m_3$.



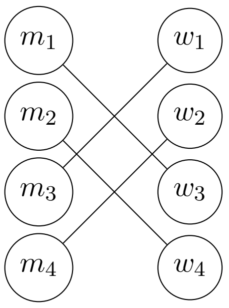
ITERATION 4: $m_3$ proposes to $w_1$. $w_1$ takes the proposal from $m_3$ into consideration and rejects $m_1$.



ITERATION 5: $m_1$ proposes to $w_2$. $w_2$ rejects the proposal of $m_1$ and keeps the proposal of $m_4$ into consideration.



ITERATION 6: $m_1$ now proposes to $w_3$. $w_3$ takes the proposal into consideration. Because every man has a proposal being considered the algorithm stops and the couples become engaged.



The Gale-Shapley algorithm results in the pairs:
$< m_1, w_3 >$, $< m_2, w_4 >$, $< m_3, w_1 >$ and $< m_4, w_2 >$.
Notice that this is indeed a stable set of marriages and that nobody has been matched to the spouse of their first preference.

## 2.3 Application to the Classroom Problem

Comparing to the matching process of the Classroom Problem the main difference with the Marriage Problem is that the Classroom Problem works with preferences within the same group instead of preferences to an other distinct group. Although it seems one might overcome this difference by 'copying' the initial group, the opposite is true. This will become apparent after applying this to an example.

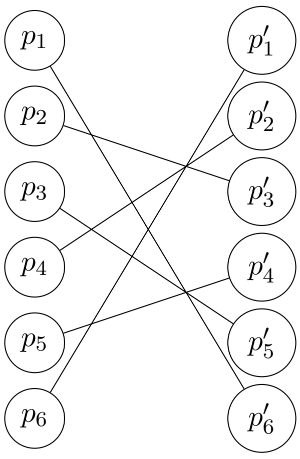
*EXAMPLE*
Lets consider the following preference list in a group of six.

| | $p_1$ | $p_2$ | $p_3$ | $p_4$ | $p_5$ | $p_6$ |
|---|---|---|---|---|---|---|
| $p_1$ | x | 3 | 5 | 1 | 4 | 2 |
| $p_2$ | 4 | x | 2 | 5 | 3 | 1 |
| $p_3$ | 3 | 5 | x | 1 | 2 | 4 |
| $p_4$ | 4 | 1 | 5 | x | 3 | 2 |
| $p_5$ | 5 | 2 | 3 | 1 | x | 4 |
| $p_6$ | 2 | 4 | 5 | 3 | 1 | x |

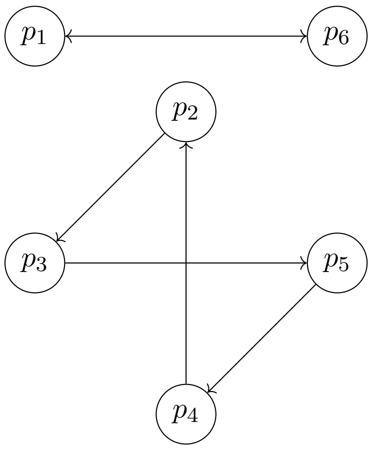
Where $p_1$ ranks $p_4$ first, $p_6$ second, $p_2$ third, $p_5$ fourth and $p_3$ last. Obviously someone can't rank him/herself.
To make this table like the table of the previous example, we transpose the table and add it as the second number. Also there will be a copied group of $p'_n$ to act as a 'distinct' group. These two actions result in:

| | $p'_1$ | $p'_2$ | $p'_3$ | $p'_4$ | $p'_5$ | $p'_6$ |
|---|---|---|---|---|---|---|
| $p_1$ | x | 3, 4 | 5, 3 | 1, 4 | 4, 5 | 2, 2 |
| $p_2$ | 4, 3 | x | 2, 5 | 5, 1 | 3, 2 | 1, 4 |
| $p_3$ | 3, 5 | 5, 2 | x | 1, 5 | 2, 3 | 4, 5 |
| $p_4$ | 4, 1 | 1, 5 | 5, 1 | x | 3, 1 | 2, 3 |
| $p_5$ | 5, 4 | 2, 3 | 3, 2 | 1, 3 | x | 4, 1 |
| $p_6$ | 2, 2 | 4, 1 | 5, 4 | 3, 2 | 1, 4 | x |

Applying the Gale-Shapley algorithm to this problem results in the following solution:



This appears to be a workable solution. However when we bring the problem back to a single group we see that copying the group will indeed not result in a matching:



It is clear that applying the Gale-Shapley algorithm to a single group ranking a copied group does not necessarily result in a matching. In the next chapter '*The Roommate Problem*' an algorithm will be discussed that solves this.

# Chapter 3

# The Roommate Problem

In the previous Chapter we thought that creating couples in a class was similar to the Marriage Problem and we tried to use the Gale-Shapley algorithm. We saw that this algorithm only works for two distinct groups. Instead of the Marriage Problem, creating couples in a class is similar to the Roommate Problem. In this chapter the problem and the solution algorithm, which is an augmentation of the Gale-Shapley algorithm, are explained. We will see that this algorithm is applicable to the Classroom Problem. Due to the existence of a better matching algorithm, this algorithm will not be used to solve the Classroom Problem.

Consider a group of $n$ (we assume $n$ even) persons. Everyone ranks the other $n-1$ persons in order of preference. The problem of matching these $n$ persons in pairs is known as the Roommate Problem. We consider a group of $n$ freshmen students going to college / university for the first time. On campus two persons share a room. So every new student is asked which other (new) student they want as their roommate.
In 1985 Robert W. Irving [5] published an article with an algorithm which would determine either a stable solution or the result that no stable solution exists.

In the next section the algorithm, which is an extension of the Gale-Shapley algorithm, is explained. Afterwards the same example as in section 2.3 is used to show that this algorithm does indeed produces a solution (if one exists).
The preference list is now notated in this form:

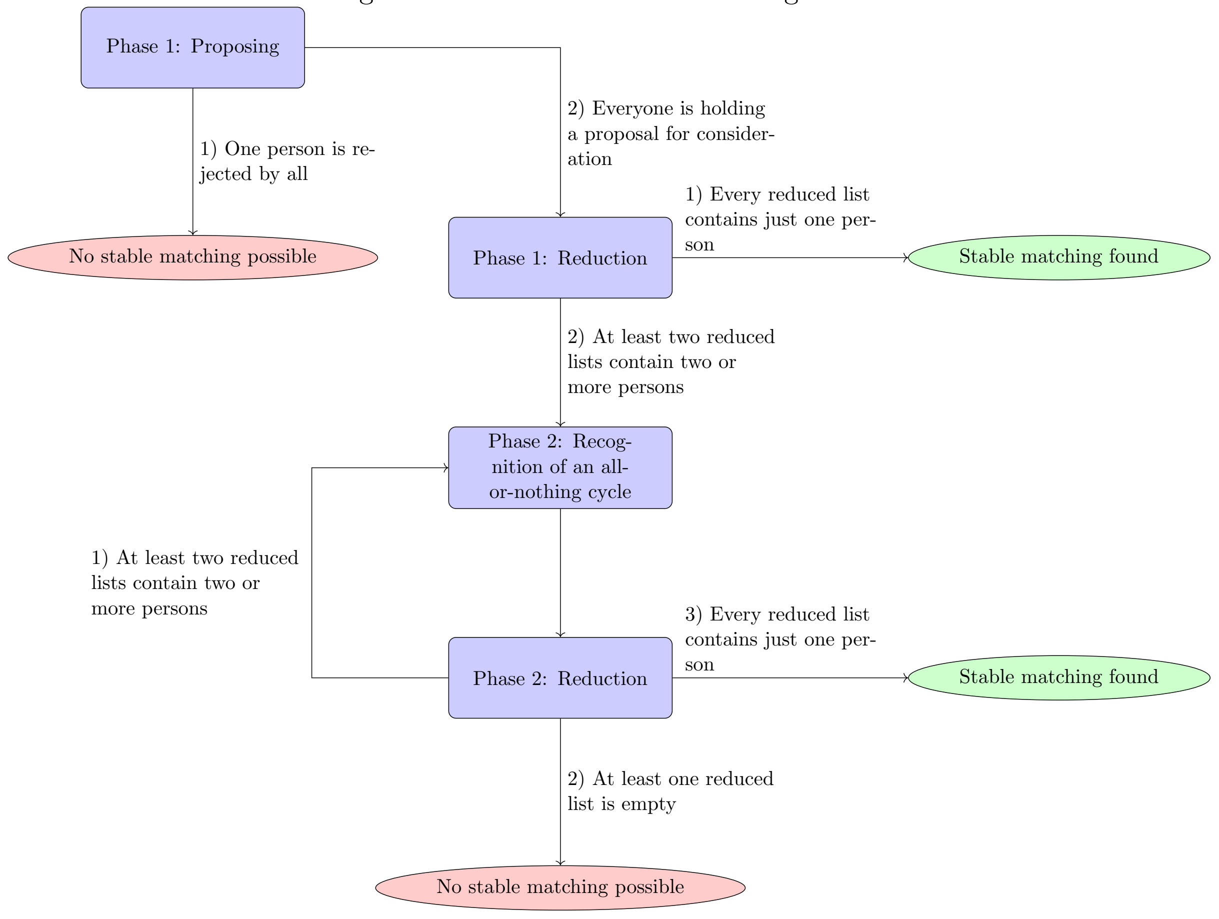
| | | | | | |
|---|---|---|---|---|---|
| $p1$ | $p4$ | $p6$ | $p2$ | $p5$ | $p3$ |
| $p2$ | $p6$ | $p3$ | $p5$ | $p1$ | $p4$ |
| $p3$ | $p4$ | $p5$ | $p1$ | $p6$ | $p2$ |
| $p4$ | $p2$ | $p6$ | $p5$ | $p1$ | $p3$ |
| $p5$ | $p4$ | $p2$ | $p3$ | $p6$ | $p1$ |
| $p6$ | $p5$ | $p1$ | $p4$ | $p2$ | $p3$ |

This means that $p1$ prefers $p4$ first, then $p6$, then $p2$ and $p5$ and finally $p3$.

The algorithm consist of two phases. The first phase is almost identical to the Gale-Shapley algorithm, followed by a reduction stage. The second phase tackles the problems we found in the final example of the previous chapter, where we found a circular matching.

A Matlabcode for the Roommate Problem algorithm is included in Appendix B.

Figure 3.1: The flowchart for the algorithm.


Phase 1: Proposing
1) One person is rejected by all
No stable matching possible
2) Everyone is holding a proposal for consideration
Phase 1: Reduction
1) Every reduced list contains just one person
Stable matching found
2) At least two reduced lists contain two or more persons
Phase 2: Recognition of an all-or-nothing cycle
1) At least two reduced lists contain two or more persons
Phase 2: Reduction
3) Every reduced list contains just one person
Stable matching found
2) At least one reduced list is empty
No stable matching possible

## 3.1 Phase 1 of the algorithm

### 3.1.1 Proposing

This phase is almost identical to the Gale-Shapley algorithm. However, since we are no longer considering men (or women) proposing to the other sex, a new formulation of the algorithm is given.
The individuals start their process of proposals one at a time ($p1$ begins). Each individual pursues the following strategy during the sequence of proposals:

**1.** If $y$ receives a proposal from $x$

- **a.** $y$ rejects $x$ immediately if $y$ is holding a better proposal in consideration. (i.e., $y$ holds a proposal from someone who $y$ prefers over $x$).
- **b.** otherwise $y$ holds the proposal in consideration and, if $y$ was already holding a proposal, $y$ rejects the proposal it was holding before.

**2.** $x$ proposes to the others in the order in which they appear in $x$'s preference list, stopping when a proposal is taken into consideration. Any following rejection causes $x$ to continue down his/her preference list.

This stage ends in one of two states:

1) One person has been rejected by all the others. In this case a stable matching doesn't exist.

2) Everyone is holding a proposal for consideration.

### 3.1.2 When proposing ends in state 1: No stable matching

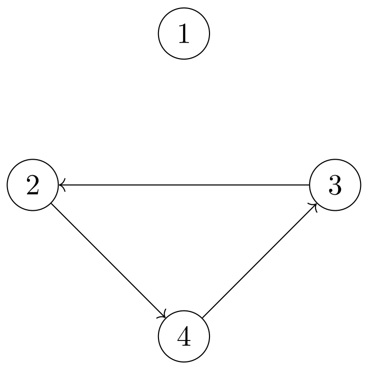
Note that circular matchings are still possible, and they are not stable.
Consider the following preference list:

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 2 | 4 | 3 | 1 |
| 3 | 2 | 4 | 1 |
| 4 | 3 | 2 | 1 |

By following the algorithm above we find the next sequence of proposals:

| | |
|---|---|
| 1 proposes to 2; | 2 holds 1; |
| 2 proposes to 4; | 4 holds 2; |
| 3 proposes to 2; | 2 holds 3 and rejects 1; |
| 1 proposes to 3; | 3 holds 1; |
| 4 proposes to 3; | 3 holds 4 and rejects 1; |
| 1 proposes to 4; | 4 rejects 1; |

Which results the following outcome:



As in the previous chapter we see a circular patterns between 2, 3 and 4. Person 1 got isolated because he was rejected by all others. In this case it is not possible to find a stable matching, because 1 isn't matched. When we would force a match and make 4 and 1 a couple, this will make an unstable couple. The same goes for forced matches of 1 with 2 or 3.

### 3.1.3 When proposing ends in state $2$: Phase $1$ reduction

After the proposing stage it still is possible to get cycles, like with the example of section 2.3. Now that everyone is holding a proposal, we are going to reduce the preference lists by deleting all possibilities that would lead to an unstable match.
When $y$ is holding a proposal from $x$, the following persons are removed from $y$'s preference list:

**i)** All $z$ who $y$ does not prefer over $x$

**ii)** All $z$ who are holding a proposal from someone who $z$ prefers over $y$

Reducing the preference lists like this will have as a result:

1. If $y$ is first on $x$'s list, then $x$ is last on $y$'s list.

2. $x$ is on $y$'s list if and only if $y$ is on $x$'s list.

There are two possible results after reducing the preference lists:

1) Every list contains just one person. In this case we have found a stable solution to the problem and we don't have to proceed to phase 2.

2) Several (at least two and not necessarily all) lists contain two or more persons. In this case we proceed to phase 2.

## 3.2 Phase $2$ of the algorithm

### 3.2.1 Recognition of an all-or-nothing cycle

Within the reduced preference lists from phase 1 we are searching for sequences $a_1, \ldots, a_r$ and $b_1, \ldots, b_r$ of different persons such that:

**1.** For $i = 1, \ldots, r-1$, the second person on $a_i$'s list ($b_{i+1}$) is the first person on $a_{i+1}$'s list.

**2.** The second person on $a_r$'s list ($b_{r+1} = b_1$) is the first person on $a_1$'s list.

A sequence like this is called an all-or-nothing cycle. Finding an all-or-nothing cycle can be done using the following method:

Let $x_1$ be an arbitrary person whose preference list has more than one person in it. Generate sequences $(x_i)$ and $(y_i)$ so that:

$y_i =$ the second person in $x_i$'s current reduced preference list

$x_{i+1} =$ the last person in $y_i$'s current reduced preference list

Continue this proces until the sequence $(x_i)$ cycles and call

$$a_i = x_{s+i-1}$$

where $x_s$ is the first occurence of the first element to be repeated in $(x_i)$. The sequence $(a_i)$ now is an all-or-nothing cycle and the sequence for $(b_i)$ can be produced.

### 3.2.2 Phase 2 reduction

Once a cycle like this is found the preference lists will be reduced again. For $i = 1, \dots r$ all $b_i$ are forced to reject the proposal they are holding into consideration from $a_i$. As a result $a_i$ will propose to $b_{i+1}$ (with $b_{r+1} = b_1$). From the preference lists can be removed:

**i)** all successors of $a_i$ from $b_{i+1}$'s list

**ii)** $b_{i+1}$ from the lists of the successors of $a_i$ in $b_{i+1}$'s list. Because if $a_i$ cannot have a better roommate than $b_{i+1}$, for the sake of stability, $b_{i+1}$ cannot do worse than $a_i$.

After the reduction one of the following conditions is met:

1) At least two reduced lists contain two or more persons. Phase 2 is repeated.

2) At least one list is empty and a stable matching does not exist

3) Every list contains precisely one person and a stable matching is found.

## 3.3 Example

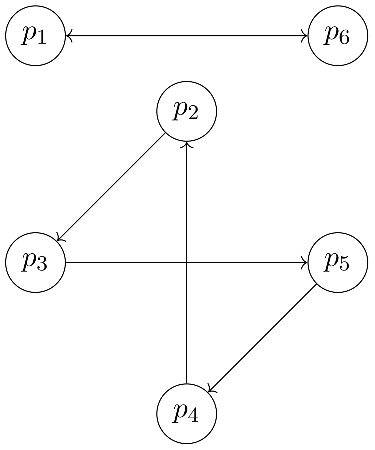
To show that this algorithm works (if a stable matching exist) we will look at the same example that did not get a solution in the previous chapter. In the new representation:

| | | | | | |
|---|---|---|---|---|---|
| $p1$ | $p4$ | $p6$ | $p2$ | $p5$ | $p3$ |
| $p2$ | $p6$ | $p3$ | $p5$ | $p1$ | $p4$ |
| $p3$ | $p4$ | $p5$ | $p1$ | $p6$ | $p2$ |
| $p4$ | $p2$ | $p6$ | $p5$ | $p1$ | $p3$ |
| $p5$ | $p4$ | $p2$ | $p3$ | $p6$ | $p1$ |
| $p6$ | $p5$ | $p1$ | $p4$ | $p2$ | $p3$ |

During phase 1 the following proposals and rejections happen:

| | |
|---|---|
| $p1$ proposes to $p4$; | $p4$ holds $p1$; |
| $p2$ proposes to $p6$; | $p6$ holds $p2$; |
| $p3$ proposes to $p4$; | $p4$ rejects $p3$; |
| $p3$ proposes to $p5$; | $p5$ holds $p3$; |
| $p4$ proposes to $p2$; | $p2$ holds $p4$; |
| $p5$ proposes to $p4$; | $p4$ holds $p5$ and rejects $p1$; |
| $p1$ proposes to $p6$; | $p6$ holds $p1$ and rejects $p2$; |
| $p2$ proposes to $p3$; | $p3$ holds $p2$; |
| $p6$ proposes to $p5$; | $p5$ rejects $p6$; |
| $p6$ proposes to $p1$; | $p1$ holds $p6$; |

The result is shown in the following graph.



In the figure (we have seen before) it is clear that the second condition for this stage to end is met, namely that everyone holds a proposal for consideration.

However we do not have a matching in pairs. So the preference list above will be reduced by removing all possibilities for unstable matches. After reducing by reduction rule **i)** from 3.1.3 we get:

| | | | | | |
|---|---|---|---|---|---|
| $p1$ | $p4$ | $p6$ | | | |
| $p2$ | $p6$ | $p3$ | $p5$ | $p1$ | $p4$ |
| $p3$ | $p4$ | $p5$ | $p1$ | $p6$ | $p2$ |
| $p4$ | $p2$ | $p6$ | $p5$ | | |
| $p5$ | $p4$ | $p2$ | $p3$ | | |
| $p6$ | $p5$ | $p1$ | | | |

When reduced by reduction rules **i)** and **ii)** from 3.1.3, the preference list becomes:

| | | | |
|---|---|---|---|
| $p1$ | $p6$ | | |
| $p2$ | $p3$ | $p5$ | $p4$ |
| $p3$ | $p5$ | $p2$ | |
| $p4$ | $p2$ | $p5$ | |
| $p5$ | $p4$ | $p2$ | $p3$ |
| $p6$ | $p1$ | | |

From these reduced preference lists the method described in 3.2.2 will be used to find an all-or-nothing cycle. Let $x_1 = p2$ we find:

$$\begin{array}{ll} x_1 = p2 & y_1 = p5 \\ x_2 = p3 & y_2 = p2 \\ x_3 = p4 & y_3 = p5 \\ x_4 = p3 & \end{array}$$

The first person the be repeated in the sequence $(x_i)$ is $p3$ with its first occurence at $x_2$, so $s = 2$. This gives us the all-or-nothing cycle:

$$a_1 = p3,\ a_2 = p4 \quad \text{with } b_1 = p5 \text{ and } b_2 = p2$$

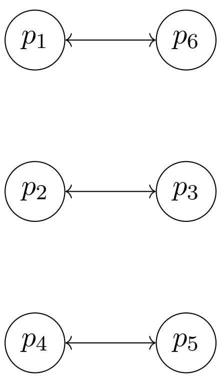
Applying reduction rule **i)** from 3.2.2 the preference lists are reduced to:

| | | |
|---|---|---|
| $p1$ | $p6$ | |
| $p2$ | $p3$ | |
| $p3$ | $p5$ | $p2$ |
| $p4$ | $p2$ | $p5$ |
| $p5$ | $p4$ | |
| $p6$ | $p1$ | |

Using reduction rule **ii)** from 3.2.2 the preference lists are reduced to:

| | |
|---|---|
| $p1$ | $p6$ |
| $p2$ | $p3$ |
| $p3$ | $p2$ |
| $p4$ | $p5$ |
| $p5$ | $p4$ |
| $p6$ | $p1$ |

As we see the third condition from section 3.2.2 is met and we can conclude that we have found a stable solution with the pairs:



$$\begin{tikzcd} p_1 \arrow[r, leftrightarrow] & p_6 \\ p_2 \arrow[r, leftrightarrow] & p_3 \\ p_4 \arrow[r, leftrightarrow] & p_5 \end{tikzcd}$$

## 3.4 Application to the Classroom problem

We can see in the example that this algorithm does indeed produce a stable solution for the roommate problem (if it is possible). However, this algorithm comes with a few drawbacks. First of all this particular algorithm requires all the students in a class to strictly rank all other classmembers. In the Netherlands it is not uncommen for a class to have thirty students or even more. Ranking each classmate as strictly as required for this algorithm is a very tedious task. For big groups it also becomes unlikely that they can rank themself as strictly as required.
To allow for a less strict ranking, W. Irving and David F. Manlove published an algorithm [6] for the roommate problem that allows ties and incomplete lists in the ranking. Note that an incomplete list can be completed by adding several ties at the end of the list.
Although there are possibilities to use an algorithm based on this one. There is still one major downside, besides the possibility no stable matching exists, to this method. The preference lists used in this algorithm hold no information about the difference between the preferences. For example: John is best friends with Kevin closely followed by Nathan. So John's first two preferences are Kevin and Nathan. His third preference is Mike, but John's not even remotely as close with Mike as he is with Kevin or Nathan. According to the algorithm the difference of John's preference between Kevin and Nathan is as big as the difference between John's preference of Nathan and Mike. So important information is lost. Because of this reason an algorithm for matching the classmembers is used that considers these differences, allows ties and incompleteness.

Starting in the next chapter the different phases of the algorithm for solving the classroom problem are discussed. We begin with the Blossom algorithm for minimum cost perfect matching.

# Chapter 4

# Phase 1: The Blossom Algorithm

The previous two chapters are documentations of two algorithms that were considered for the matching part of the Classroom Problem. From this chapter onward the four phases of the algorithm for the Classroom Problem are described. This chapter describes an algorithm that determines the couples. The next two chapters (chapters 5 and 6) describe the algorithms that determine where in the classroom the couples will sit. And finally chapter 7 describes the method of determining which students sit at the left tables and which students sit at the right tables.

In the previous chapters we were trying to match people by using preference lists. From now on, instead of a preference list, a graph is used. Let $G = (V, E, c)$ be an undirected weighted graph with $V$ representing the students, $E$ the potential matches and $c$ the costs of those edges. For a potential match between $a$ and $b$ the cost $c_{ab}$ is determined by combining the preference scores of student $a$ towards student $b$ and from student $b$ towards student $a$. The lower the cost the higher the preference of two persons willing to sit next to each other. We are searching for a set of edges such that every student has a partner and the cost of those edges is minimal.

In 1965 J. Edwards[1],[2] invented the Blossom algorithm which, when implemented, solves the above problem in polynomial time $\mathcal{O}(n^2m)$ with $n$ the number of vertices and $m$ the number of edges in $G$. The goal of the algorithm is to grow a number of *trees* in the graph improving the current matching until a perfect matching is achieved.

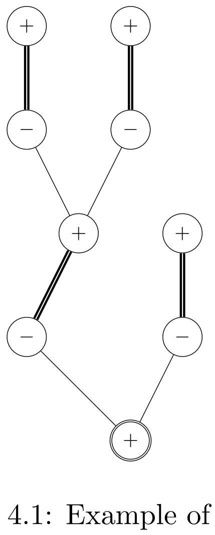


Figure 4.1: Example of a *tree*:

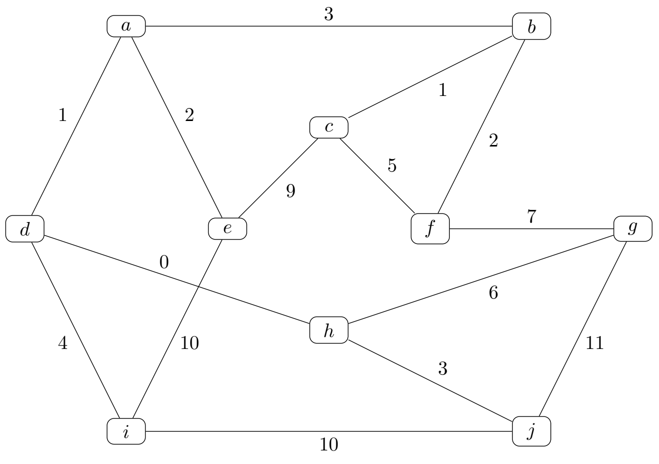


Figure 4.2: An example of an undirected weighted graph:

Before discussing the algorithm, a few definitions, remarks and examples are given.

**Definition 3.** *A* ***matching*** *$M$ is a subset of $E$ such that no $v \in V$ is connected to more than one edge in $M$. A matching $M_p$ is called* ***perfect*** *if each vertex $v \in V$ is connected to exactly one edge in $M_p$.*

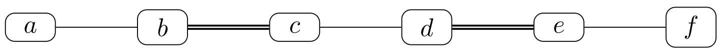
**Definition 4.** *A vertex is called* ***exposed*** *if it does not connect to any edge in $M$.*

**Definition 5.** *A path $P \subset V$ is called an* ***alternating path*** *if each vertex $v \in P$, except the starting and ending vertices, is connected to one edge in $M$ and to one edge in $E \backslash M$.*

**Definition 6.** *A alternating path $A$ is called an* ***augmenting path*** *if the starting and ending vertex are exposed vertices.*

**Remark 1.** *Alternating and augmenting paths fully depend on the current matching $M$.*

A simple example of an augmenting path is:

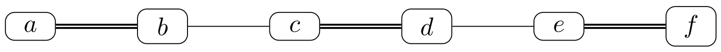


Edges in $M$ are being represented with double lines and edges in $E \backslash M$ with single lines.

Notice that this augmenting path has exactly one edge more in $E \backslash M$ than edges in $M$. This is true for every augmenting path. By exchanging the edges in $M$ with the edges in $E \backslash M$ we gain a new matching improving the previous matching by one edge.

**Remark 2.** *A matching $M$ is a perfect matching if and only if $(G, M)$ does not contain any augmenting paths.*

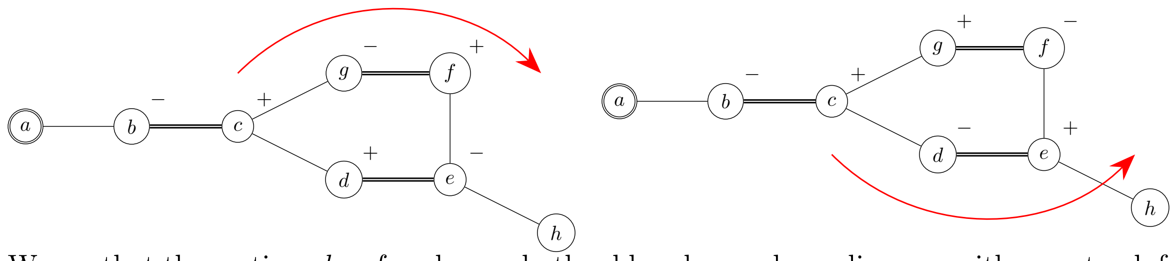
Augmenting the example we find:



**Definition 7.** *A vertex $v$ is* ***odd/even*** *if there is an alternating path from an exposed vertex to $v$ of odd/even length.*

**Remark 3.** *An augmenting path starts from an exposed vertex and alternates between odd and even vertices till reaching another exposed vertex.*

Problems may occur when reaching odd circuits which are as dense as possible with matched edges. Tracking an augmenting path may become impossible, because vertices might become odd and even at the same time. For example with $a$ as starting vertex (odd vertices are marked with $-$ and even vertices as $+$)



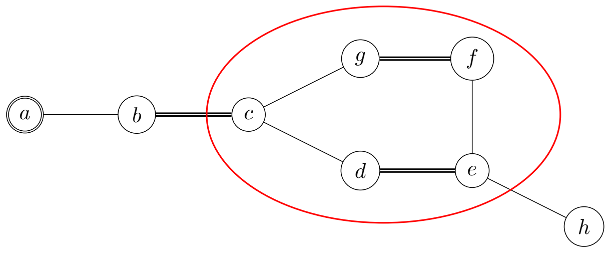
We see that the vertices $d, e, f$ and $g$ are both odd and even depending on weither we track from the top or from the bottom. Later on we will exchange odd and even for labels and one vertex can't have more than one label. So a vertex is not allowed to be odd and even at the same time. These problems occur because of so-called blossoms.

**Definition 8.** *A **blossom** is an odd circuit with $2k+1$ vertices and $k$ edges in $M$.*

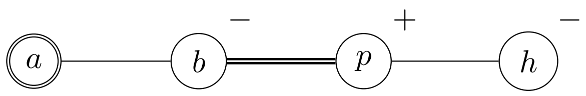
In this example $M = \{bc, fg, de\}$ and the blossom is circuit $\{cd, de, ef, fg, cg\}$ and $k = 2$.

**Remark 4.** *The existance of blossoms, as the existance of alternating/augmenting paths, fully depends on the current matching $M$.*

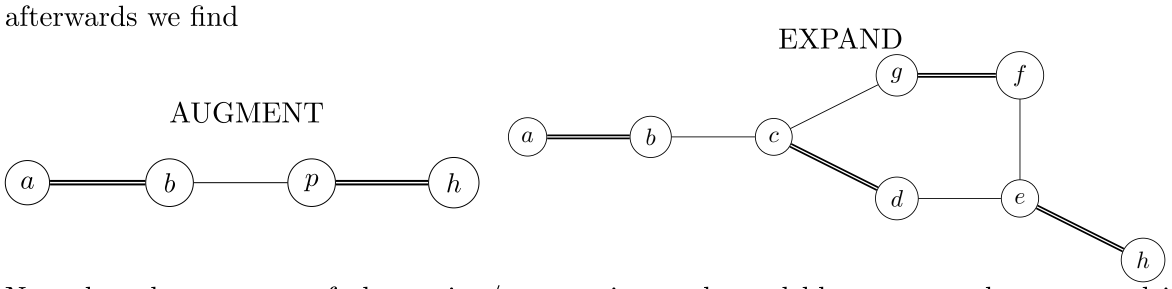
So in this case we have found the following blossom:



We can 'bypass' this problem by shrinking the blossom to a (pseudo)vertex $p$.



This results (in this example) in an augmenting path. So augmenting and expanding the blossom afterwards we find



Now that the concepts of alternating/augmenting paths and blossoms are demonstrated it is time to go deeper into the algorithm.

## 4.1 The algorithm

In 2009 V. Kolmogorov [7] documented a new implemation of the Blossom algorithm including a very good overview of the algorithm.

Let $G = (V, E, c)$ be an undirected weighted graph (as described above). We are looking for a perfect matching $M$ of minimum cost $c(M) = \sum_{e \in M} c_e$. To indicate the current matching we use a incidence vector $x \in \{0,1\}^E$ with

$$\begin{aligned} x_e &= 0 \quad \text{if} \quad e \notin M \\ x_e &= 1 \quad \text{if} \quad e \in M \end{aligned}$$

So $c(M) = \sum_e x_e c_e$.
We define $\mathcal{U}$ as the set of all subsets of $V$ of odd cardinality with a minimum of three vertices. An example of a set of $\mathcal{U}$ in figure 4.2 is $\{a, c, f, h, j\}$. These are the potential blossoms we might find throughout the algorithm. For a subset $S \subseteq V$ we define the set of boundary edges of $S$ by $\beta(S) = \{(u,v) \in E | u \in S, v \in V \backslash S\}$. For a single vertex $v$ we define

$\beta(v) = \{(u,v) \in E | u \in V \backslash \{v\}\}$. As an example let $S$ be $\{a,b,c\}$ in figure 4.2, $\beta(S)$ then becomes $\{(a,d),(a,e),(b,f),(c,e),(c,f)\}$.

The LP formulation is given by

PRIMAL

$$\begin{array}{rll} \min & \sum_{e \in M} c_e x_e & \\ \text{subject to} & x(\beta(v)) = 1 & \forall v \in V \\ & x(\beta(S)) \geq 1 & \forall S \in \mathcal{U} \\ & x_e \geq 0 & \forall e \in E \end{array}$$

DUAL

$$\begin{array}{rll} \max & \sum_{v \in V} y_v + \sum_{S \in \mathcal{U}} y_S & \\ \text{subject to} & \text{slack}(e) \geq 0 & \forall e \in E \\ & y_S \geq 0 & \forall S \in \mathcal{U} \end{array}$$

For edge $e = (u,v)$ we define:

$$\text{slack}(e) = c_e - y_u - y_v - \sum_{S \in \mathcal{U} : e \in \beta(S)} y_S$$

as the reduced cost of edge $e$. Edges with slack$(e) = 0$ are called *tight*.

During this algorithm a feasible vector $y$ is maintained in the dual part while a non-feasible (integer-valued) vector $x$ is maintained in the primal part of the algorithm. Note that the vector $x$ corresponds to a matching. With each iteration these vectors are updated such that the cardinality of the matching vector $x$ is gradually increased until it becomes a (minimum cost) perfect matching.

Before continuing it is important to make the distinction between *interior* and *exterior* vertices. (Pseudo)Vertices inside a shrunk blossom are called *interior* vertices. Other vertices are called *exterior* vertices. This distinction is important because this algorithm only works with exterior vertices and ‘ignores’ the vertices that are contained within a blossom (except when expanding a blossom). Note that updating the dual vector $y$ at any given iteration does not affect the $y$ values of interior vertices. Therefore also the slack of the edges within a blossom remain unchanged.

From this point onwards we will call exterior vertices just vertices, unless stated otherwise.

Each vertex $v$ is assigned a label $l(v) \in \{+,-,0\}$. If a vertex $v$ has the label $l(v) = 0$ then $v$ is called a *free* vertex and every free vertex is matched (via a tight edge) to an other free vertex. If a vertex $v$ is not free (so either $l(v) = +$ or $l(v) = -$) then $v$ is part of an alternating tree (path).
Vertices $v$ with the label $l(v) = -$ are always preceded by a vertex with the label ‘+’ connecting to $v$ via a tight <u>unmatched</u> edge. Similarly, vertices $v$ with the label $l(v) = +$ are always preceded by a vertex with the label ‘−’ connecting to $v$ via a tight <u>matched</u> edge, with the exception for exposed vertices. Exposed vertices are the roots of the trees and are labeled ‘+’. Note that the number of trees equals the number of exposed vertices. The algorithm alternates between primal and dual updates.

## 4.2 Algorithm: Initialisation

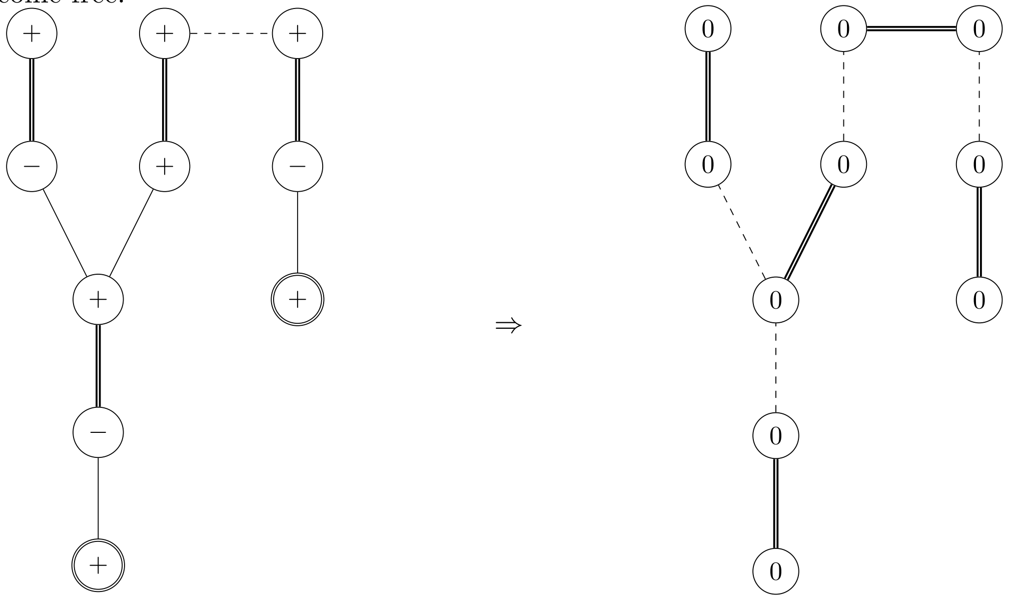
All vertices are exposed. So all vertices start as roots of trees and are labeled as ‘+’ vertices. Each dual variable is set to zero (i.e. $y_v = 0\ \forall v \in V$ and $y_S = 0\ \forall S \in \mathcal{U}$).

## 4.3 Algorithm: Primal updates

During the primal updates the algorithm looks for edges that are tight, with one exception for expanding blossoms. When, initially or after a dual update, an edge $(u, v)$ becomes tight one of the following operations is performed: Augment, Grow, Shrink or Expand.

### 4.3.1 Augment

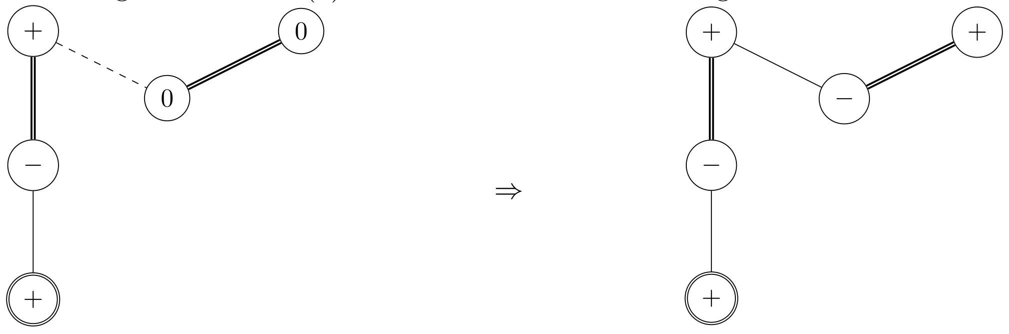
If $l(u) = l(v) = +$ and $u$ and $v$ do not belong to the same tree then the cardinality of the vector $x$ can be improved (by one) by changing $x_e = 0$ to $x_e = 1$ and vice versa for all the edges in the path connecting the two roots (exposed vertices) of the trees. Afterwards all vertices of the two trees become free.



A tight unmatched edge is drawn by a solid single line if the edge is part of an alternating tree and as a dashed line if the edge is not part of any tree. Matched (tight) edges are drawn by double lines. The roots of the trees are noted as vertices that we circled.

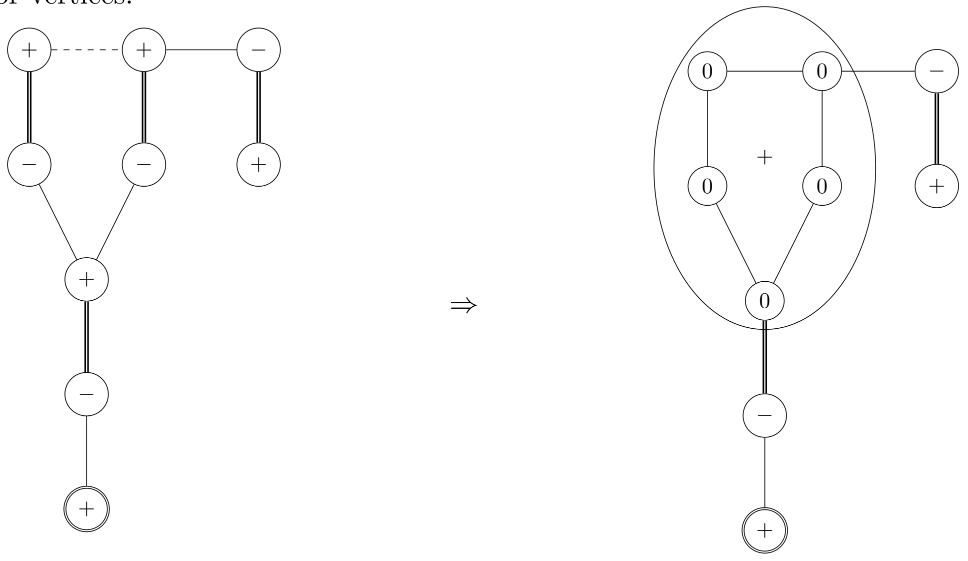
### 4.3.2 Grow

If $l(u) = +$ and $l(v) = 0$ then tree with vertex $u$ can grow by obtaining node $v$ and its matched vertex. Vertex $v$ gets the label $l(v) = -$ and its matched vertex gets the label '+'.



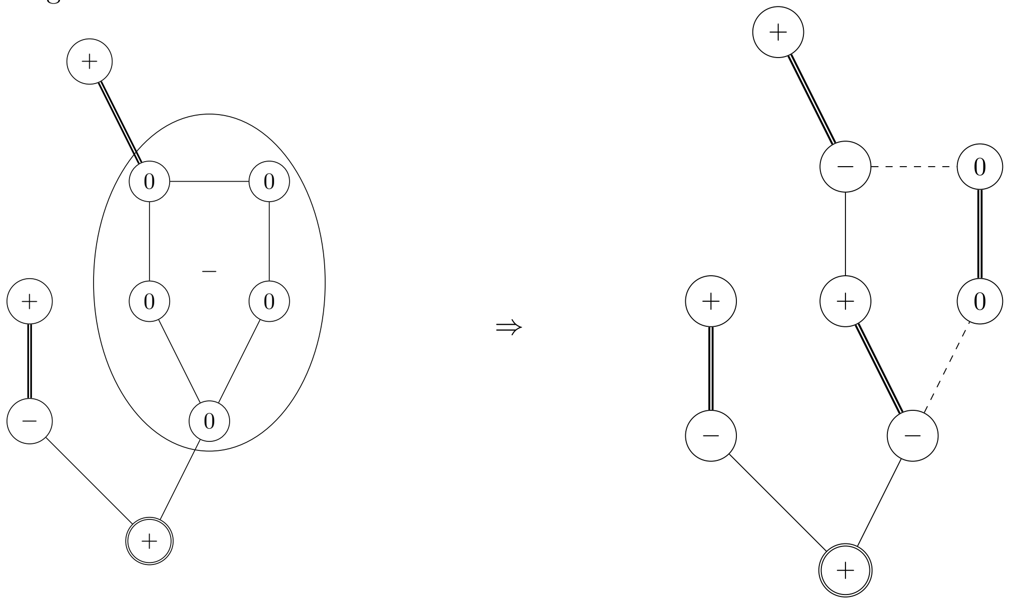
### 4.3.3 Shrink

If $l(u) = l(v) = +$ and $u$ and $v$ belong to the same tree then we have found an odd circuit (blossom) that can be shrunk to a pseudovertex. The dual variable for this pseudovertex is set to zero and the pseudovertex is labeled '+'. Note that the vertices that now become interior vertices are labeled '0', this ensures that the dual updates will not alter the dual variables for these interior vertices.



### 4.3.4 Expand

Note that in this case we do not have a new tight edge.
If for a pseudovertex $v$ (which is a blossom) $y_v = 0$ and $l(v) = -$ then it can be expanded. Matched edges are tracked back to the root of the tree.



**Remark 5.** *The operation 'Augment' is the only operation that changes (improves) the cardinality of the current matching $x$.*

**Remark 6.** *Note that the algorithm is not limited to just one operation per iteration. The prefered operation is 'Augment'.*

## 4.4 Algorithm: Dual updates

During the dual phase we will adjust the vector $y$ with a value $\delta$ such that trees and blossoms remain intact and the new $y$ is still a feasible (dual) solution. For each tree $T$, $\delta$ is constrained by:

$$\begin{array}{rcllr} \delta & \leq & \text{slack}(u,v) & (u,v) \text{ is a } (+,0) \text{ edge with } u \in T & \text{(a)} \\ \delta & \leq & \frac{1}{2}\,\text{slack}(u,v) & (u,v) \text{ is a } (+,+) \text{ edge} & \text{(b)} \\ \delta & \leq & y_v & v \text{ is a blossom and } l(v) = - & \text{(c)} \end{array}$$

The value $\delta$ is the maximum value complying to these constraints for all edges. Resulting in either a new tight edge $(u,v)$ or a blossom that needs to be expanded.

The dual variables $y_v$ are now updated for each (pseudo)vertex $v$ as follows:

$$\begin{array}{llll} \text{if} & l(v) = + & \text{then} & y_v := y_v + \delta \\ \text{if} & l(v) = - & \text{then} & y_v := y_v - \delta \\ \text{if} & l(v) = 0 & \text{then} & y_v := y_v \end{array}$$

**Remark 7.** *The value of the objective function of the dual part of the LP formulation is increased by $\#T \cdot \delta$, with $\#T$ the number of trees. This is true because in each tree there is always one "+" vertex more than there are "−" vertices.*

**Remark 8.** *If (a) is the limiting constraint for $\delta$ we can use the primal update operator "Grow" during the next iteration. With (b) being the limited constraint, we either use "Augment", when $u$ and $v$ are not in the same tree, or "Shrink", when $u$ and $v$ are in the same tree. If (c) is the limiting constraint, the operator "Expand" will be used on the blossom $v$ from the constraint.*

## 4.5 Algorithm: Completion

Once all (pseudo)vertices are labeled as '0' vertices (this can only occur after augmenting) the algorithm comes to a stop, because there are no more exposed vertices. For reaching the solution it might still be necessary to expand existing blossoms. The following conditions are now satisfied:

$$\begin{array}{rclll} \text{slack}(e) & > & 0 & \Rightarrow & x_e = 0 & \forall e \in E \\ y_S & > & 0 & \Rightarrow & x(\beta(S)) = 1 & \forall S \in \mathcal{U} \end{array}$$

## 4.6 Algorithm: Complexity

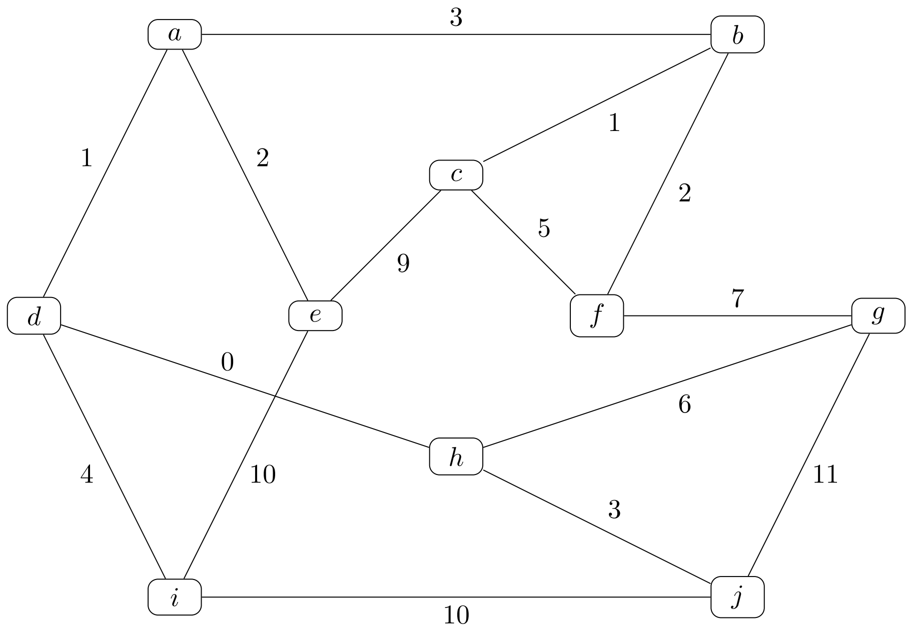
During this algorithm we want to use the primal operator 'Augment' to increase our matching. So let us call the iterations between two augmentations a *stage*. Because with each augmentation two exposed vertices 'disappear', there can only be at most $n/2$ stages.

Because the operator 'Shrink' lowers the number of exterior vertices by at least two, we can use the 'Shrink' operation at most $n/2 + 1$ times during a stage. Because free vertices always come in multitudes of two, we can use the 'Grow' operation at most $n/2$ times during a stage. Because we cannot expand more blossoms than we have shrunk, the number of 'Expand' operations that can be used during a stage is at most $n/2 + 1$.
The number of primal operations per stage is therefore bounded by $\mathcal{O}(n)$.

During each dual iteration at most $m$ edges have to be checked for their slack to determine the value of $\delta$. So each stage takes at most $\mathcal{O}(nm)$ time, bringing the total time for this algorithm to run to at most $\mathcal{O}(n^2m)$.

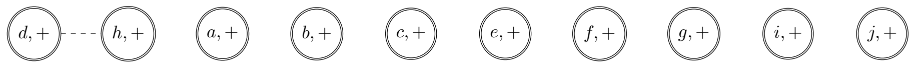
## 4.7 Example

Recall the graph (figure 4.2) at the beginning of this chapter



We are going to use this graph as an example of how the algorithm works.

INITIALISATION: All vertices are exposed roots, labeled '+', and all dual variables are set to zero.



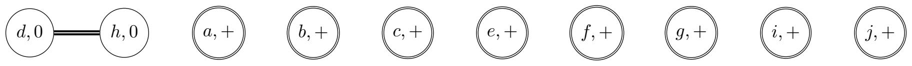
$$\begin{array}{rclrcl} y_a &=& 0 & y_f &=& 0 \\ y_b &=& 0 & y_g &=& 0 \\ y_c &=& 0 & y_h &=& 0 \\ y_d &=& 0 & y_i &=& 0 \\ y_e &=& 0 & y_j &=& 0 \end{array}$$

$$\forall S \in \mathcal{U} \quad y_S = 0 \text{ because no blossoms exist yet.}$$

Notice that the edge $(d,h)$ is already tight, because

$$\begin{array}{rcccccccl} \text{slack}((d,h)) = & c_{(d,h)} & - & y_d & - & y_h & - & \sum_{S\in\mathcal{U}:(d,h)\in\beta(S)} y_S & \\ = & 0 & - & 0 & - & 0 & - & 0 & = 0 \end{array}$$

ITERATION 1 (PRIMAL UPDATE): edge $(d,h)$ is an $(+,+)$ edge and $d$ and $h$ do not belong to the same tree. So we can use the 'Augment' operation. $d$ and $h$ become matched and free.

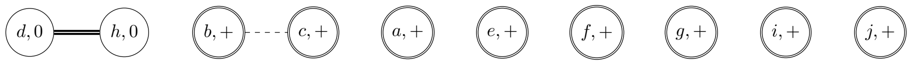


ITERATION 2 (DUAL UPDATE): From constraint (b) for edge $(b,c)$ we get $\delta \le \frac{1}{2}\text{slack}(b,c) = \frac{1}{2}\cdot 1 = 0.5$. Setting $\delta = 0.5$, the new dual variables become:

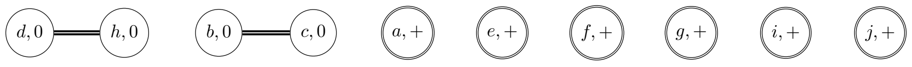
$$\begin{array}{rcl@{\qquad}rcl} y_a &=& 0.5 & y_f &=& 0.5 \\ y_b &=& 0.5 & y_g &=& 0.5 \\ y_c &=& 0.5 & y_h &=& 0 \\ y_d &=& 0 & y_i &=& 0.5 \\ y_e &=& 0.5 & y_j &=& 0.5 \end{array}$$

With these values edge $(b,c)$ becomes tight, because

$$\begin{array}{rccccccccl} \text{slack}((b,c)) = & c_{(b,c)} & - & y_b & - & y_c & - & \sum_{S\in\mathcal{U}:(b,c)\in\beta(S)} y_S & & \\ = & 1 & - & \frac{1}{2} & - & \frac{1}{2} & - & 0 & & = 0 \end{array}$$



ITERATION 3 (PRIMAL UPDATE): edge $(b,c)$ is an $(+,+)$ edge and $b$ and $c$ do not belong to the same tree. So we can use the 'Augment' operation. $b$ and $c$ become matched and free.



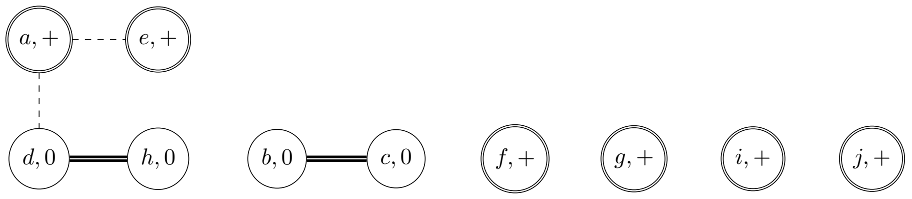
ITERATION 4 (DUAL UPDATE): From contraint (b) for the edges $(a,d)$ and $(a,e)$ follows $\delta \le \frac{1}{2}\text{slack}(a,d) = \frac{1}{2}\text{slack}(a,e) = \frac{1}{2}\cdot 1 = 0.5$. Setting $\delta = 0.5$, the new dual variables become:

$$\begin{array}{rcl@{\qquad}rcl} y_a &=& 1 & y_f &=& 1 \\ y_b &=& 0.5 & y_g &=& 1 \\ y_c &=& 0.5 & y_h &=& 0 \\ y_d &=& 0 & y_i &=& 1 \\ y_e &=& 1 & y_j &=& 1 \end{array}$$

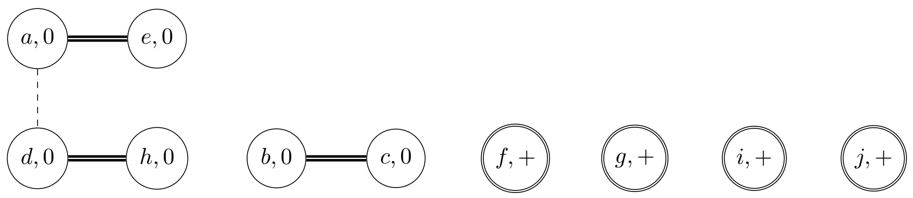
With these values edges $(a,d)$ and $(a,e)$ become tight.

$$\begin{array}{rccccccccl} \text{slack}(a,d) = & c_{(a,d)} & - & y_a & - & y_d & - & \sum_{S\in\mathcal{U}:(a,d)\in\beta(S)} y_S & & \\ = & 1 & - & 1 & - & 0 & - & 0 & & = 0 \end{array}$$

$$\begin{array}{rccccccccl} \text{slack}(a,e) = & c_{(a,e)} & - & y_a & - & y_e & - & \sum_{S\in\mathcal{U}:(a,e)\in\beta(S)} y_S & & \\ = & 2 & - & 1 & - & 1 & - & 0 & & = 0 \end{array}$$



ITERATION 5 (PRIMAL UPDATE): edge $(a, e)$ is an $(+, +)$ edge and $a$ and $e$ do not belong to the same tree. So we can use the 'Augment' operation. Because we prefer to use 'Augment' over 'Grow' we will not be growing the tree with root $a$ with vertices $d$ and $h$. $a$ and $e$ become matched and free.

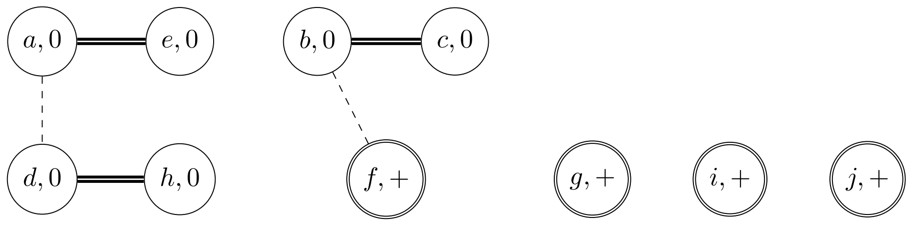


ITERATION 6 (DUAL UPDATE): From contraint (a) for edge $(b, f)$ follows
$\delta \leq \text{slack}(b, f) = 0.5$. Setting $\delta = 0.5$, the new dual variables become:

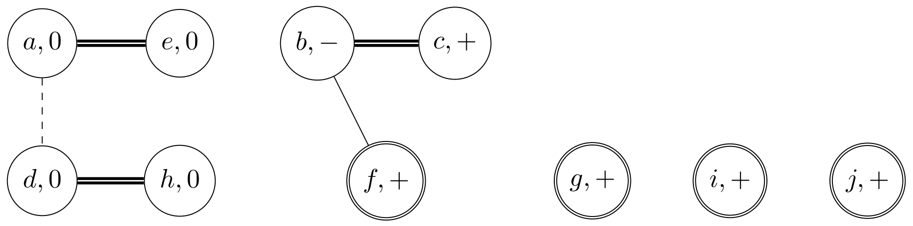
$$\begin{array}{rcl rcl} y_a &=& 1 & y_f &=& 1.5 \\ y_b &=& 0.5 & y_g &=& 1.5 \\ y_c &=& 0.5 & y_h &=& 0 \\ y_d &=& 0 & y_i &=& 1.5 \\ y_e &=& 1 & y_j &=& 1.5 \end{array}$$

With these values edge $(b, f)$ becomes tight.

$$\begin{array}{rccccccccl} \text{slack}(b,f) = & c_{(b,f)} & - & y_b & - & y_f & - & \sum_{S \in \mathcal{U}:(b,f)\in\beta(S)} y_S & \\ = & 2 & - & 0.5 & - & 1.5 & - & 0 & = 0 \end{array}$$



ITERATION 7 (PRIMAL UPDATE): edge $(b, f)$ is an $(+, 0)$ edge. So we can use the 'Grow' operation. The tree with root $f$ is grown with the vertices $b$ and $c$ and their labels will become $l(b) = -$ and $l(c) = +$.



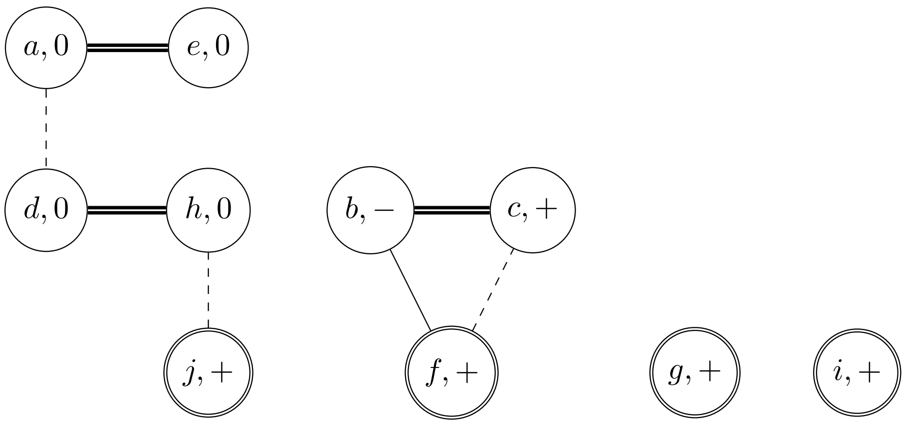
ITERATION 8 (DUAL UPDATE): From contraint (a) for edge $(h, j)$ and from constraint (b) for edge $(c, f)$ follows $\delta \leq \text{slack}(h, j) = 1.5$ and $\delta \leq \frac{1}{2}\text{slack}(c, f) = \frac{1}{2} \cdot 3 = 1.5$.
Setting $\delta = 1.5$, the new dual variables become:

$$\begin{array}{rcl rcl} y_a &=& 1 & y_f &=& 3 \\ y_b &=& -1 & y_g &=& 3 \\ y_c &=& 2 & y_h &=& 0 \\ y_d &=& 0 & y_i &=& 3 \\ y_e &=& 1 & y_j &=& 3 \end{array}$$

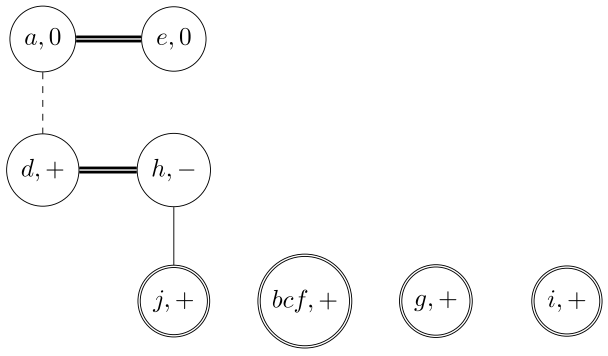
With these values edges $(c, f)$ and $(h, j)$ become tight.

$$\begin{array}{rcccccccl} \text{slack}(c,f) = & c_{(c,f)} & - & y_c & - & y_f & - & \sum_{S\in\mathcal{U}:(c,f)\in\beta(S)} y_S & \\ = & 5 & - & 2 & - & 3 & - & 0 & = 0 \end{array}$$

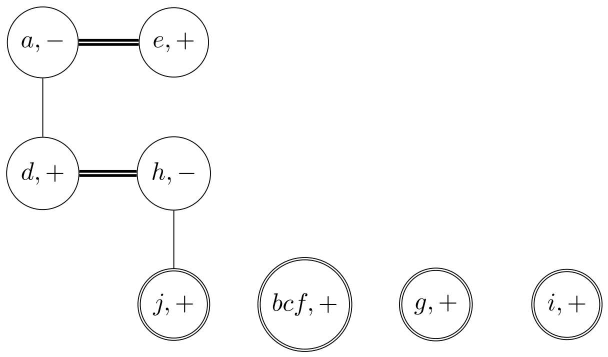
$$\begin{array}{rcccccccl} \text{slack}(h,j) = & c_{(h,j)} & - & y_h & - & y_j & - & \sum_{S\in\mathcal{U}:(h,j)\in\beta(S)} y_S & \\ = & 3 & - & 0 & - & 3 & - & 0 & = 0 \end{array}$$



ITERATION 9 (PRIMAL UPDATE): Edge $(j, h)$ is an $(+, 0)$ edge. So we can use the 'Grow' operation. The tree with root $j$ is grown with the vertices $h$ and $f$ and their labels will become $l(h) = -$ and $l(d) = +$.
Edge $(c, f)$ is an $(+, +)$ edge and $c$ and $f$ belong to the same tree. The operation 'Shrink' will be used, creating a pseudovertex $bcf \in \mathcal{U}$, set the dual variable $y_{bcf} = 0$ and set the label $l(bcf) = +$. Note that $bcf$ is an exposed vertex and therefore a root of a tree.



Notice that by growing the tree with root $j$ edge $(d, a)$ now has become an $(+, 0)$ edge. Therefore the operation 'Grow' can be applied again. The vertices $a$ and $e$ are added to the tree of root $j$ and their labels become $l(a) = -$ and $l(e) = +$.



ITERATION 10 (DUAL UPDATE): From contraint (b) for the edges $(d, i)$ and $(f, g)$ we get $\delta \leq \frac{1}{2}\text{slack}(d, i) = \frac{1}{2}\text{slack}(f, g) = 0.5$. Setting $\delta = 0.5$, the new dual variables become:

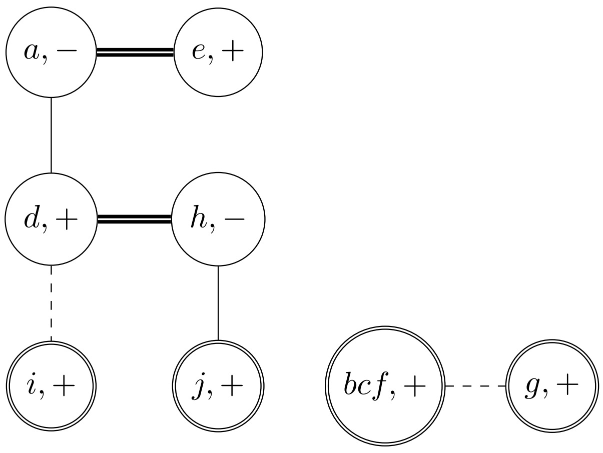
$$\begin{array}{rclcrcl} y_a &=& 0.5 && y_f &=& 3 \\ y_b &=& -1 && y_g &=& 3.5 \\ y_c &=& 2 && y_h &=& -0.5 \\ y_d &=& 0.5 && y_i &=& 3.5 \\ y_e &=& 1.5 && y_j &=& 3.5 \\ \\ y_{bcf} &=& 0.5 &&&& \end{array}$$

With these values edges $(d, i)$ and $(f, g)$ become tight.

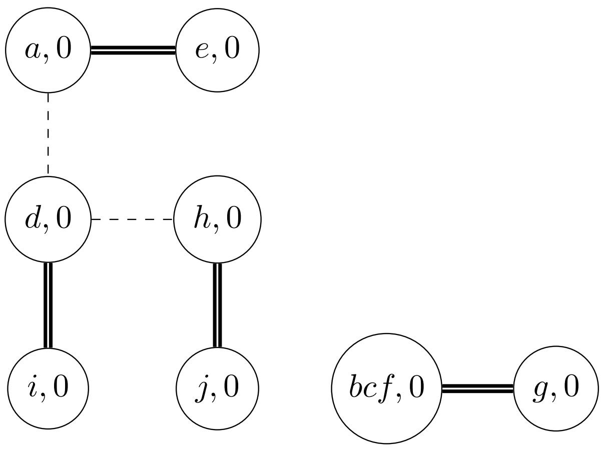
$$\begin{array}{rccccccccl} \text{slack}(d, i) = & c_{(d,i)} & - & y_d & - & y_i & - & \sum_{S \in \mathcal{U}:(d,i)\in\beta(S)} y_S & & \\ = & 4 & - & 0.5 & - & 3.5 & - & 0 & & = 0 \end{array}$$

$$\begin{array}{rccccccccl} \text{slack}(f, g) = & c_{(f,g)} & - & y_f & - & y_g & - & \sum_{S \in \mathcal{U}:(f,g)\in\beta(S)} y_S & & \\ = & c_{(f,g)} & - & y_f & - & y_g & - & y_{bcf} & & \\ = & 7 & - & 3 & - & 3.5 & - & 0.5 & & = 0 \end{array}$$

Because $f$ is an interior vertex of the blossom $bcf$, edge $(bcf, g)$ is therefore now tight.

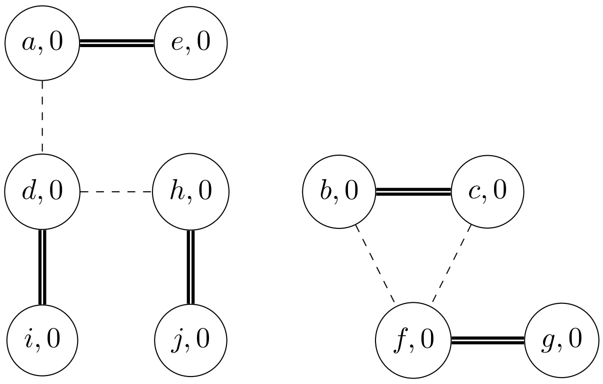


ITERATION 11 (PRIMAL UPDATE): edges $(i, d)$ and $(bcf, g)$ are $(+, +)$ edges with $i$ and $d$ not belonging to the same tree and $bcf$ and $g$ not belonging to the same tree. So we can use the 'Augment' operation twice. $i$, $d$, $bcf$ and $g$ become matched and free. Also the tree with root $j$ becomes free.

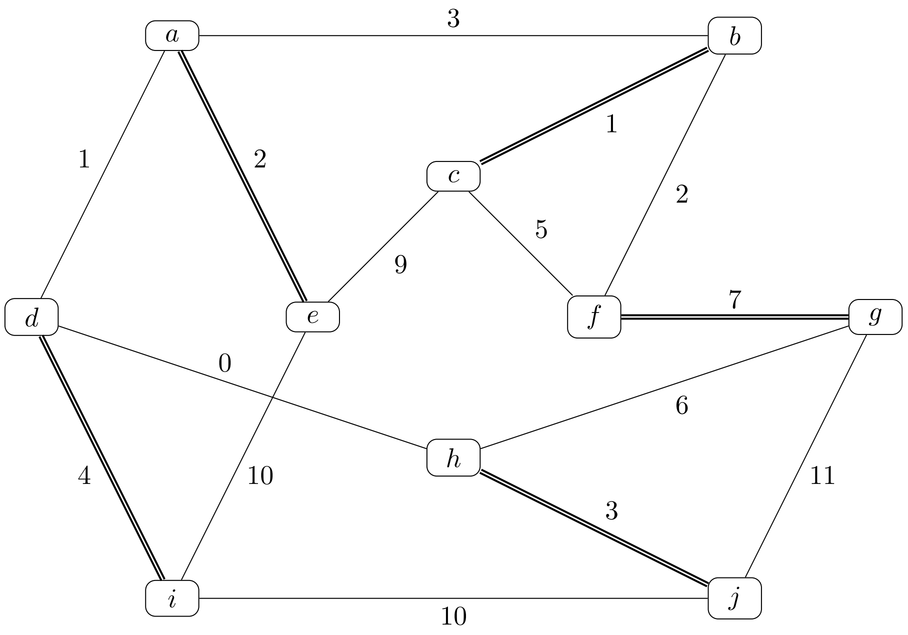


We have all vertices matched and free. This means the algorithm is coming to an end. Only thing left te do is expanding the pseudovertex $bcf$.

FINALISATION: Because edge $(f, g)$ is tight, the blossom will be expanding as follows



RESULT: 'Translating' the matched edges in the dual setting to the actual graph we started with results in the following solution for the problem with $c(M) = 17$:



# Chapter 5

# Phase 2: The Repeated Nearest Neighbor Algorithm

After the matching algorithm in the previous chapter (*Blossom*) the couples have been determined and it is time to determine which couple is sitting where in the classroom. This process is done in two steps of which the first step is described in this chapter and the second (phase 3) in the next chapter.

Remember that the couples have been made by matching according to preferences. So we are interested in which way the couples, once determined, relate to one another.

Every couple has a preference to which couple sits in front of them, behind them and next to them across the aisle. For this chapter we assume:

- Because of the distance to other couples, every couple prioritizes who sits in front of them and behind them over who sits next to them across the aisle.

- Couples in the corners and at the end of a column (the couples that only have one other couple in front or behind them) have a higher priority for the couple that sits next to them on the other side of the aisle.

Based on these assumptions, it makes sense to create a string of couples so the classroom can be filled like shown in figure 5.1.

For couples $ab$ and $cd$ the combined score $c$ is given by:

$$c := c_{ac} + c_{ad} + c_{bc} + c_{bd}$$

Using the combined scores of a couple to any other couple, we can create a complete graph with the couples as the vertices and the combined scores as the edges.

A string as demonstrated in figure 5.1 can now be found by finding the minimal cost Hamiltonian circuit in a complete graph. This problem is known as the 'Traveling Salesman Problem' (TSP) and there is no algorithm that produces an exact solution in polynomial time. Therefore the heuristic (greedy) approximation algorithm known as the 'Nearest Neighbor algorithm' (NNA) will be used to find a suboptimal solution. It is because of this one phase in the entire algorithm that we will find a suboptimal solution for the Classroom Problem.

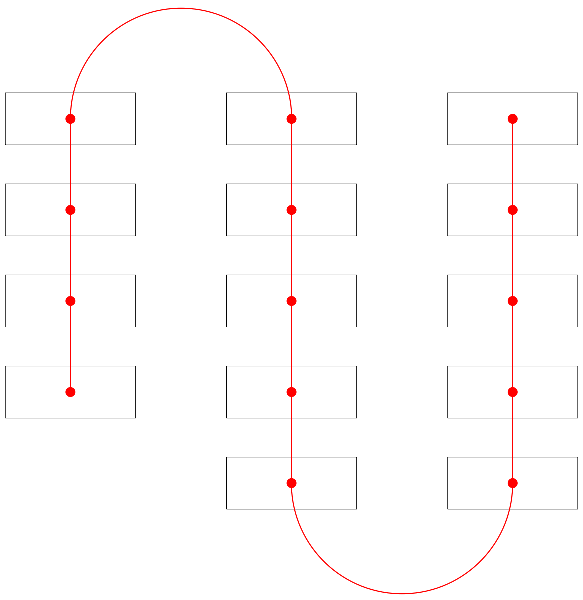


Figure 5.1:

## 5.1 The algorithm

Given a weighted graph with $n$ vertices. The Nearest Neighbor algorithm is given by:

**Step 1.** Start with a vertex $S$, mark $S$ as visited and set $S$ as current vertex.

**Step 2.** Determine the edge with the lowest weight connecting the current vertex and an unvisited vertex $V$ and add this edge to the solution. (If there are more edges with lowest cost, then pick one edge arbitrarily)

**Step 3.** Mark $V$ as visited and set $V$ as the current vertex.

**Step 4.** If all vertices are visited, then go to step 6.

**Step 5.** Go back to step 2.

**Step 6.** Add the edge connecting the current vertex to $S$.

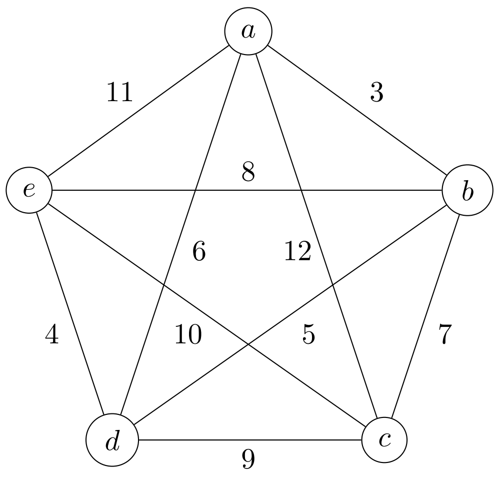
**Step 7.** Terminate algorithm.

Repeatedly applying this algorithm with respect to all vertices as starting point results in the Repeated Nearest Neighbor algorithm (RNNA).

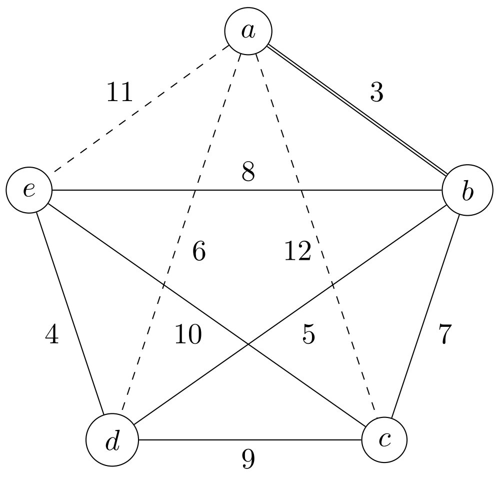
The algorithm needs $n$ calculations per starting point. So the overall complexity of the RNNA is in the order of $\mathcal{O}(n^2)$

## 5.2 Example

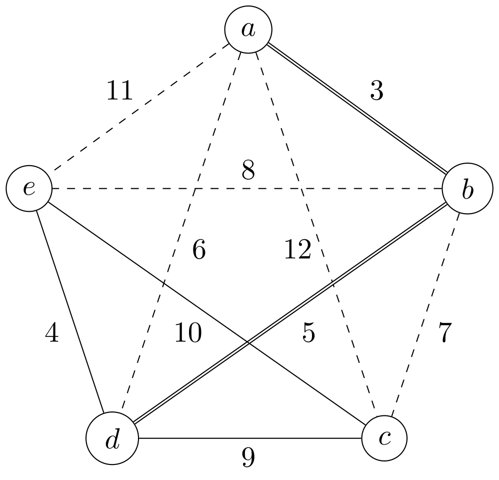
Consider the next weighted complete graph:



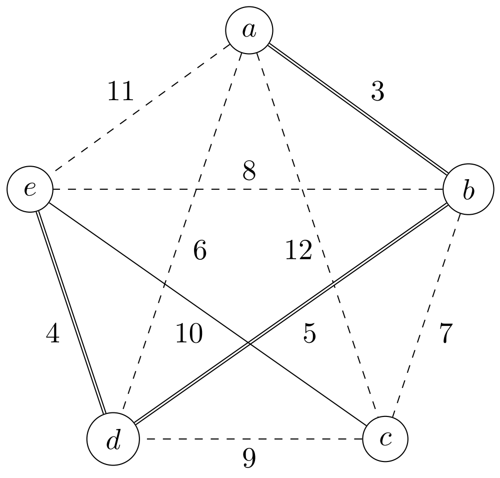
Select $a$ as the starting vertex. The edge with the lowest weight is $ab$. So the next vertex is $b$.



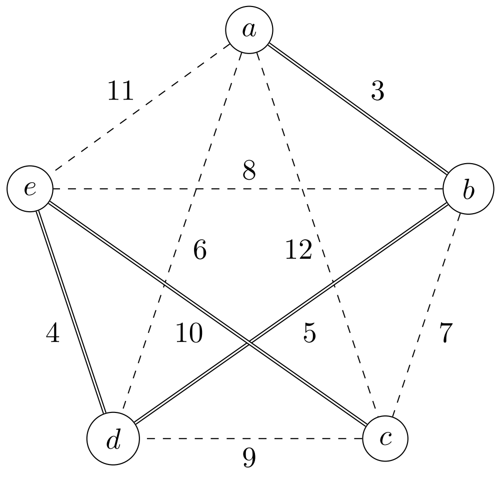
We add the edge $ab$ to the solution (indicated by a double line). Instead of marking vertices as visited we 'remove' the edges connecting to the previous vertex that are not added to the solution (indicated by dashed lines). From vertex $b$ the edge with the lowest weight is $bd$.



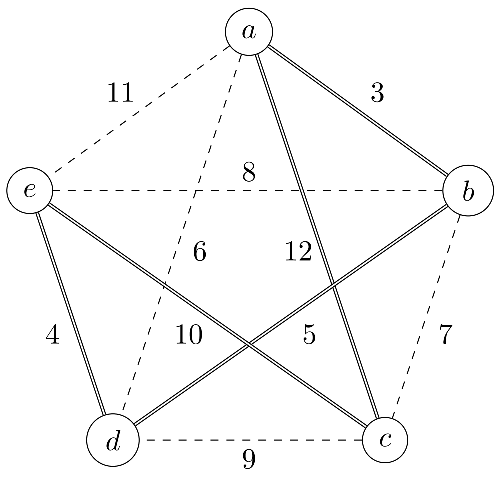
The edge *bd* is added to the solution and all remaining edges connected to *b* are removed. From the remaining edges connecting to *d* the edge *de* has the lowest weight.



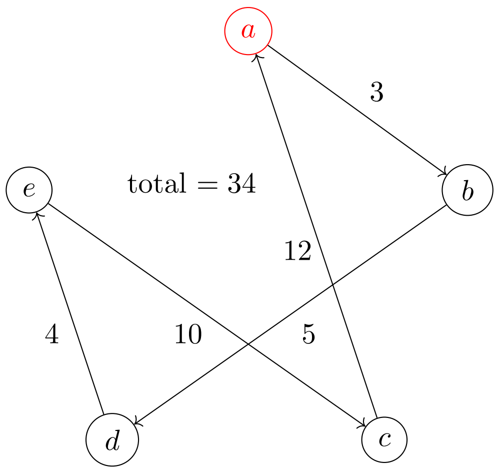
The edge *de* is added to the solution and the remaining edges to *e* have been removed. At this point only vertex *c* is still unvisited so the next edge is *ce*.



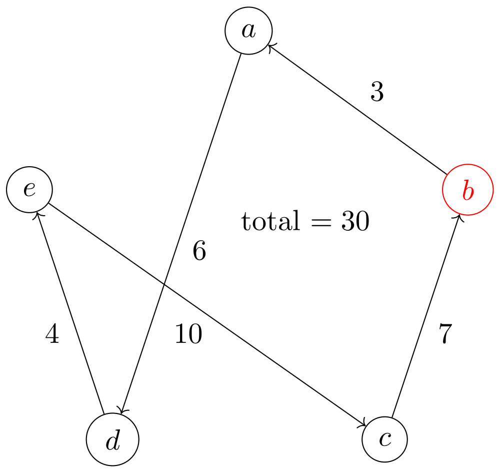
The edge *ce* is added to the solution. No more unvisited vertices are left. So the circuit is finished by adding the edge *ac* (previously removed) to the solution.

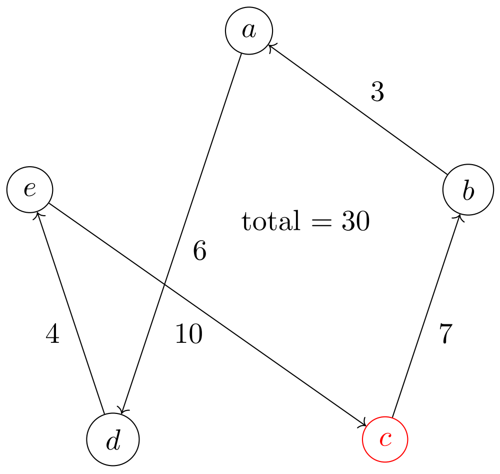


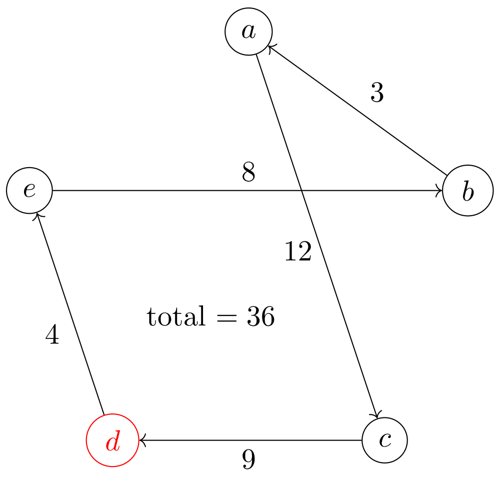
Using vertex $a$ as starting point we have found the following circuit with a total weight of 34.

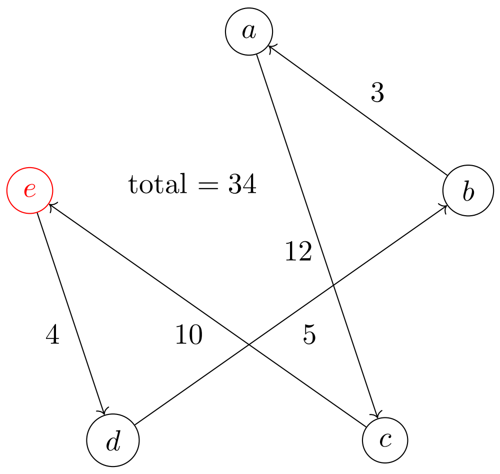


Applying the same algorithm with the other vertices as starting point results in the following circuits.









Notice that starting at $b$ and $c$ result in exactly the same circuit. Also starting at $a$ and $e$ results in the same circuit albeit in the opposite direction. So the Repeated Nearest Neighbor produces the following three unique circuits on this graph

| Circuit | Total weight |
|---|---|
| *abdeca* | 34 |
| *badecb* | 30 |
| *debacd* | 36 |

## 5.3 Application to the classroom problem

As mentioned in the introduction of this chapter, we are searching for a string. So by eliminating the edge with the highest cost in each circuit found by the Repeated Nearest Neighbor algorithm, we create a string as wished. It is important to consider every circuit found by the RNNA because it is not necessarily true that the best circuit will also give the best string. By removing the highest cost edges in the solution found in the example, we find the following results for the strings:

| Circuit | Total weight | Highest cost edge | Cost | String | Weight |
|---|---|---|---|---|---|
| *abdeca* | 34 | *ac* | 12 | *abdec* | 22 |
| *badecb* | 30 | *ce* | 10 | *cbade* | 20 |
| *debacd* | 36 | *ac* | 12 | *cdeba* | 24 |

So applying the RNNA to the example above and removing the highest cost edges we find the string *cbade* with total weight 20 to bring to the next phase of the algorithm for the Classroom Problem.

Notice that it is possible to find multiple strings with the lowest weight. This can happen in two ways:

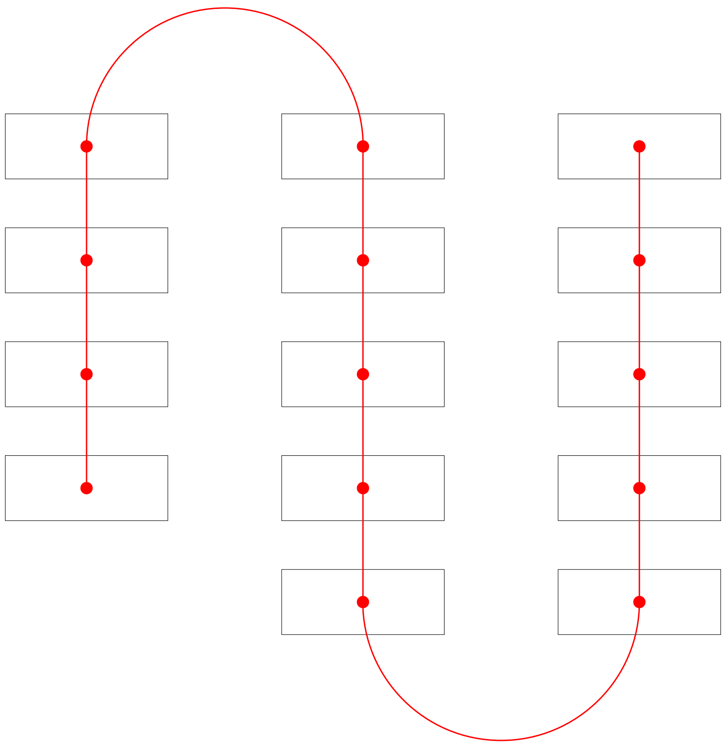
- More than one circuit produces unique strings with the same weight.
- There are multiple instances of highest cost edges within a single circuit. So the resulting string depends on the edge that is removed, without effecting the weight of the string.

All possible strings with the lowest weight resulting from removing edges from the circuits will be considered in the next phase.

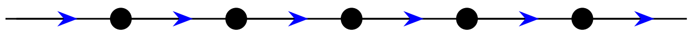
# Chapter 6

# Phase 3: String Orientation and Selection

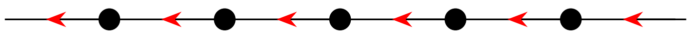
Now that we have found one or more strings of couples it is time to decide the orientation of the string. Remember that we are trying to find a string to fill the classroom like this:



In this phase the orientation of the strings is determined. When starting at the bottom left table, we want to determine weither this orientation



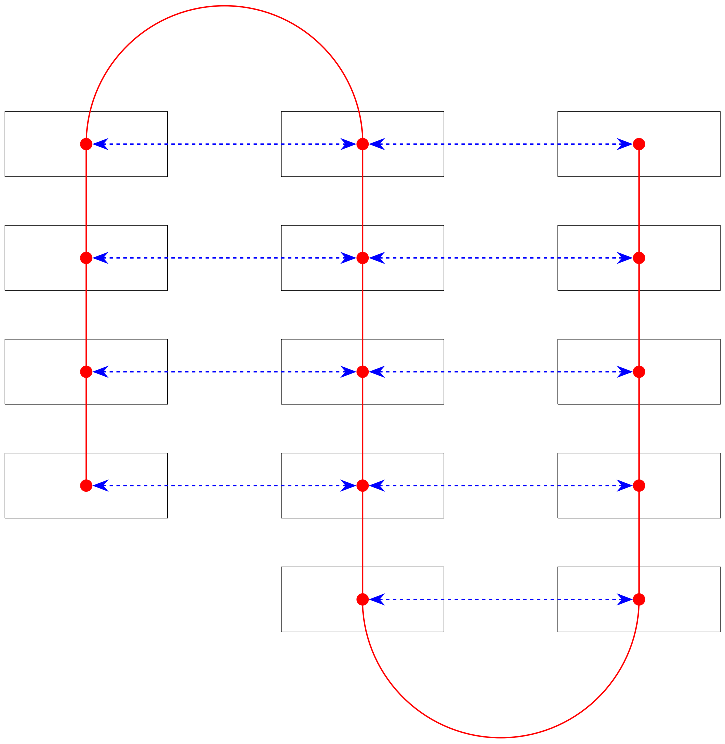
or this orientation



is optimal.

## 6.1 The algorithm

For this phase brute force calculations are used. To determine which of the two possible orientations of the string is optimal the scores of the couples next to each other (across the aisle) are added up for both orientations.

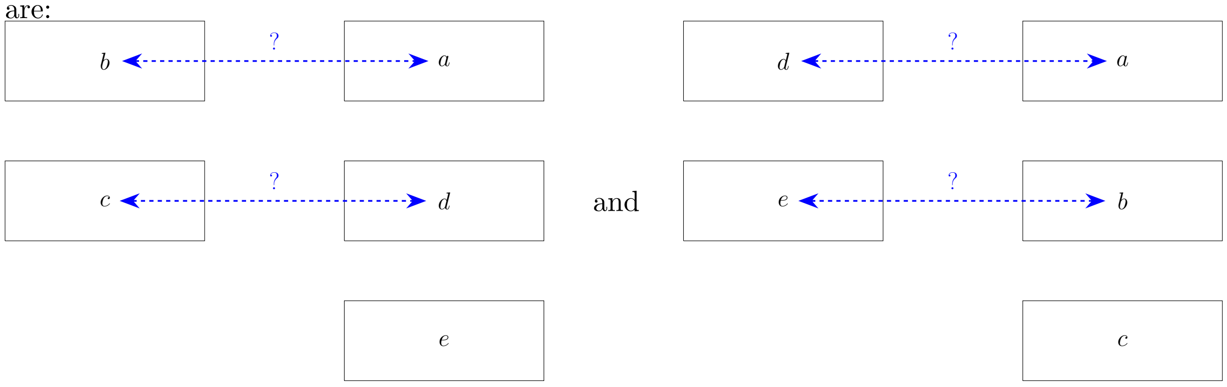


To determine the orientation score the scores between the couples across the aisle (indicated by the blue dashed lines) are added together. The orientation with the lowest score is the orientation that will be brought to phase 4. If, after phase 2, there are more than one (unique) strings with the lowest score, the lowest orientation score of all strings will be used in phase 4. In case there are more than one (unique) orientations of strings with the lowest score, all of these will be used in phase 4.

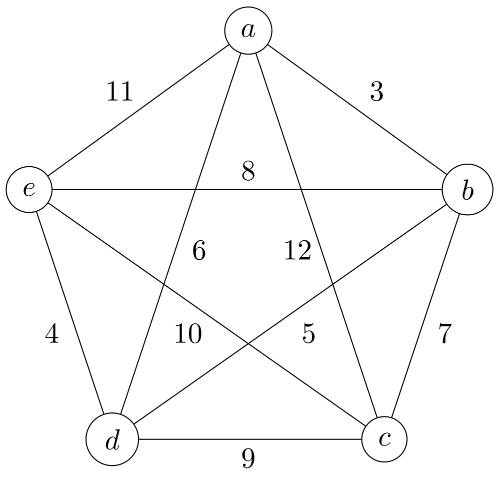
The number of cost calculations necessary per string is 2. At most $\frac{n^2}{4}$ strings are checked for orientation. So this phase takes at most $\frac{n^2}{2}$ calculations and is of the order $\mathcal{O}(n^2)$.

## 6.2 Example

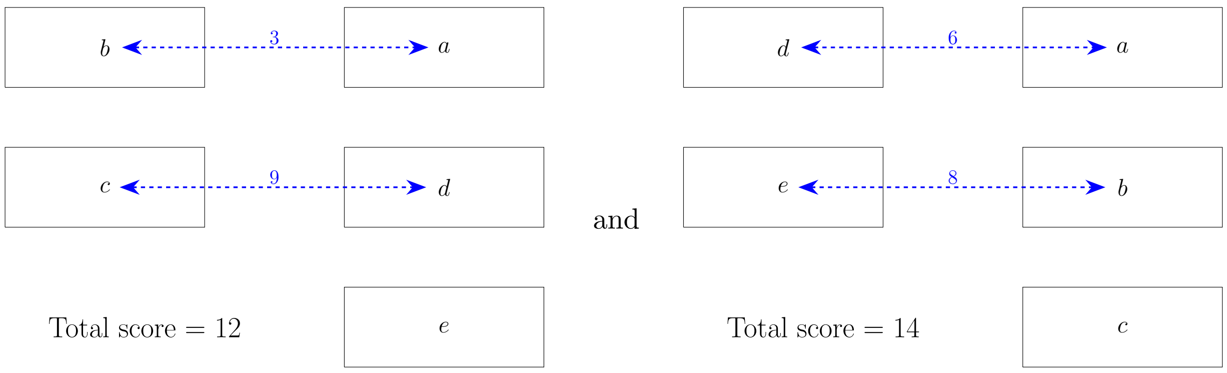
Following from the example used in chapter 5 we have received one string with lowest score. The string we received was *cbade* with a weight of 20. So the two orientation we need to check are:



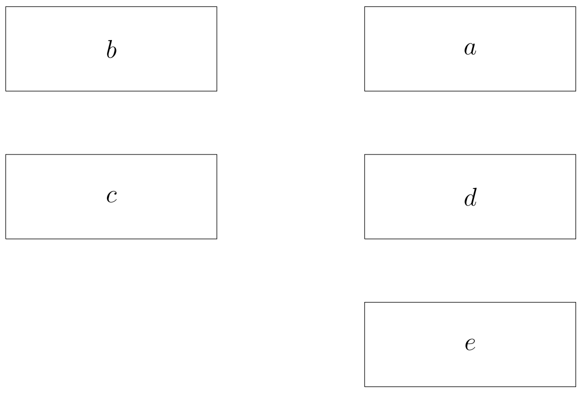
Applying the weigths from the graph:



we find the orientationscores to be



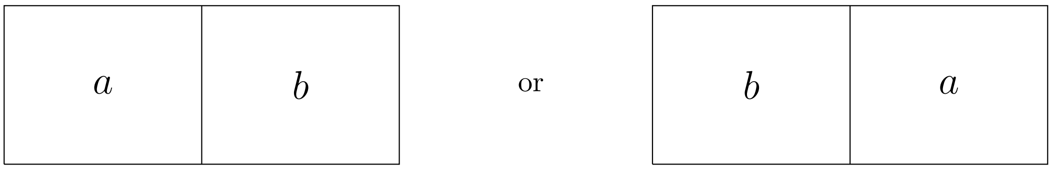
With this example we have determined that the couples should sit like:



Chapter 7

# Phase 4: Couple Orientation

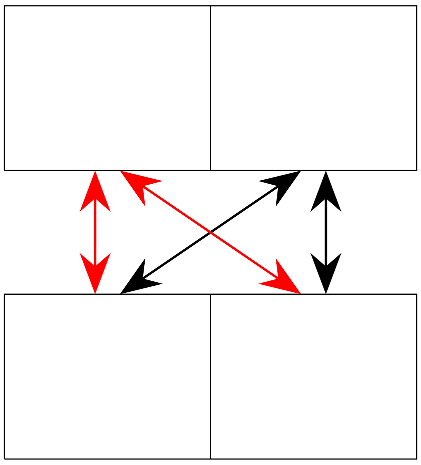
Now that we have found one (or more) optimal string(s) (including orientation) it is time to get to the final phase of the algorithm. After phase 3 the couples are located (sub-)optimally, but the orientation of each couple is still to be determined. For each couple we have to determine weither they are orientated like



We have to ask ourselfs two questions.

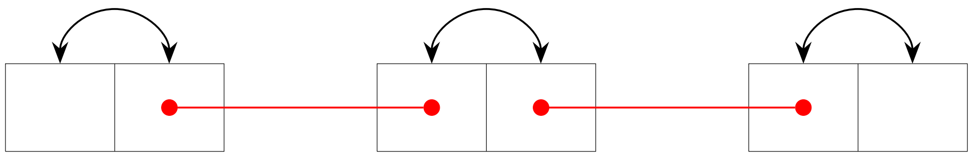
1. What is the influence of the couple(s) in front and/or behind the couple on the optimum?
2. What is the influence of the couple(s) across the aisle of the couple on the optimum?

To answer the first question we take a look at the individual scores between two couples in front/behind each other.



We assume that the two red/black arrows are equally significant for the optimal orientation. So by adding all the individual scores between these students, which equals the score between these two couples, we see that the orientation of these couples have no influence on the optimum. Since all orientations of these two couples results in the same (optimal) score.

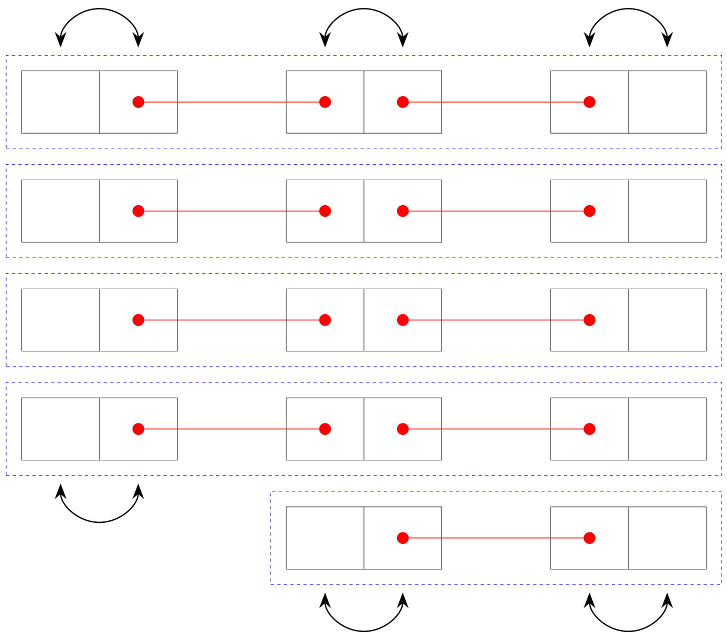
For the second question we have to look at each row in the classroom



When the individual scores differ, the orientation of each couple makes a difference for the optimum when you add the scores indicated by the red lines. So for each row these scores need to be checked for all the possible orientations of the couples in that row.

## 7.1 The algorithm

For this phase brute-force calculations are used. To determine the optimal orientation of each couple in each row we add up the (individual) scores between the students who are sitting across the aisle from each other.



For each row with couples $ab$, $cd$ and $ef$ we need to check $2^3 = 8$ orientations:

| Orientations | | | | | | Scores to check | |
|---|---|---|---|---|---|---|---|
| $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $b \leftrightarrow c$ | $d \leftrightarrow e$ |
| $a$ | $b$ | $c$ | $d$ | $f$ | $e$ | $b \leftrightarrow c$ | $d \leftrightarrow f$ |
| $a$ | $b$ | $d$ | $c$ | $e$ | $f$ | $b \leftrightarrow d$ | $c \leftrightarrow e$ |
| $a$ | $b$ | $d$ | $c$ | $f$ | $e$ | $b \leftrightarrow d$ | $c \leftrightarrow f$ |
| $b$ | $a$ | $c$ | $d$ | $e$ | $f$ | $a \leftrightarrow c$ | $d \leftrightarrow e$ |
| $b$ | $a$ | $c$ | $d$ | $f$ | $e$ | $a \leftrightarrow c$ | $d \leftrightarrow f$ |
| $b$ | $a$ | $d$ | $c$ | $e$ | $f$ | $a \leftrightarrow d$ | $c \leftrightarrow e$ |
| $b$ | $a$ | $d$ | $c$ | $f$ | $e$ | $a \leftrightarrow d$ | $c \leftrightarrow f$ |

For rows with only two couples we need to check for couples $ab$ and $cd$:

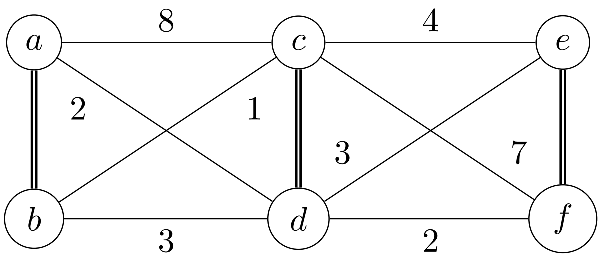
| Orientations | | Score to check |
|---|---|---|
| $a$ $b$ | $c$ $d$ | $b \leftrightarrow c$ |
| $a$ $b$ | $d$ $c$ | $b \leftrightarrow d$ |
| $b$ $a$ | $c$ $d$ | $a \leftrightarrow c$ |
| $b$ $a$ | $d$ $c$ | $a \leftrightarrow d$ |

For a classroom of the design we are using this phase takes $4 \times 8 + 4 = 36$ calculations for each string received from phase 3, which is at most $\frac{n^2}{2}$. So the total number of cost calculations is at most $18n^2$.

More generally, when the exact value of $n$ is not known, the order of this phase differs. For each row of 6 students we need to make 8 calculations. Depending of $n$ we have a number of rows. To determine the upper bound of this phase, we need to determine the number of rows with 6 students (3 couples). For $n$ students the number of rows with 6 students is at most $\lceil \frac{n}{6} \rceil$. To get an upper bound for this we use $\lceil \frac{n}{6} \rceil < \frac{n}{6} + 1$. For each string we now have at most $8(\frac{n}{6} + 1) = \frac{8n}{6} + 8$ calculations. For at most $\frac{n^2}{2}$ strings to calculate we find at most $\frac{2n^3}{3} + 4n^2$ cost calculations for this phase. So the order of this phase is $\mathcal{O}(n^3)$.

## 7.2 Example

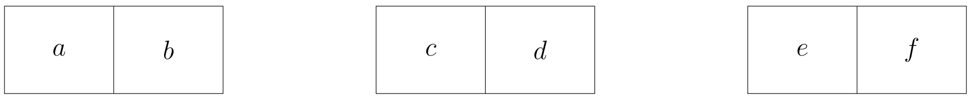
As an example we are going to determine the optimal orientation for one row. Consider the following (reduced) graph of individual scores between students $a$, $b$, $c$, $d$, $e$ and $f$. Notice that $ab$, $cd$ and $ef$ are the couples and are indicated with double lines. The scores between students who are couples are left out.



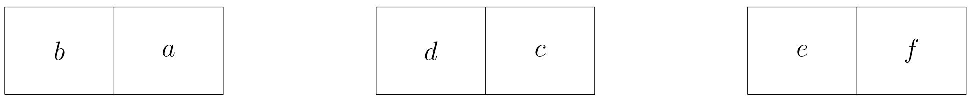
Checking all 8 possible orientations results in:

| Orientation couple | Score | Orientation couple | Score | Orientation couple | | Total |
|---|---|---|---|---|---|---|
| $a$ $b$ | 1 | $c$ $d$ | 3 | $e$ $f$ | | 4 |
| $a$ $b$ | 1 | $c$ $d$ | 4 | $f$ $e$ | | 5 |
| $a$ $b$ | 3 | $d$ $c$ | 2 | $e$ $f$ | | 5 |
| $a$ $b$ | 3 | $d$ $c$ | 7 | $f$ $e$ | | 10 |
| $b$ $a$ | 8 | $c$ $d$ | 3 | $e$ $f$ | | 11 |
| $b$ $a$ | 8 | $c$ $d$ | 4 | $f$ $e$ | | 12 |
| $b$ $a$ | 2 | $d$ $c$ | 2 | $e$ $f$ | | 4 |
| $b$ $a$ | 2 | $d$ $c$ | 7 | $f$ $e$ | | 9 |

For this row we find two possible optimal orientations:



and



With a total score of 4. For the total solution to the classroom problem either of these orientations can be used for this row without affecting the optimum.

# Chapter 8

# Example: A class of 28 students

To apply this algorithm to an actual class a teacher (and mentor), Marko Milosevic, of a 3 Havo class at the Melanchthon school in Rotterdam was found willing to ask his students the following question:

Next to whom of your classmates would you like to sit?
Chose three classmates and divide ten points between them.

The result of this inquiry are represented in figure 8.1: a graph with 28 vertices, representing the 28 students. The edges are weighted with the sum of the points that the students gave each other. If two students awarded each other points, the sum of those points is doubled as a bonus for selecting each other. When just one of the students named the other but not vice versa, this bonus was not rewarded. The function of this bonus is to increase the likelihood of two students that selected each other to be matched as an couple.

Looking at figure 8.1 we notice a few things. The graph consists of three disjunct groups. This might cause an issue in phase 2 and will be adressed in the corresponding section. A group of four students (2, 8, 15 and 22) who only named each other and no one else. And two larger groups with odd numbers of students which is going to cause problems in phase 1. Which will be adressed in the next section. One student is especially noticeable. Student 27 has an ‘autism spectrum disorder’ and was experiencing too much anxiety to answer the above question. Student 27 has been mentioned by student 11 making sure that student 27 has at least one connection.

## 8.1 Phase 1: Blossom

Before applying the Blossom algorithm, it is important to notice that in this case the higher the score the higher the preference. So instead of searching for a minimum cost perfect matching, we want to find a maximum cost perfect matching. For the Blossom algorithm (Chapter 4) to work we need to switch the values of the edges. For each $e \in E$ define the new cost as

$$c_{\text{new}}(e) = \max_{e \in E}(c(e)) - c(e)$$

Another way to solve this problem is to alter the Blossom algorithm so it will solve the maximum variant of the problem. In 1986 Z. Galil [4] documented solving a maximum weight perfect matching problem using the blossom algorithm. This algorithm was used by Joris van Rantwijk

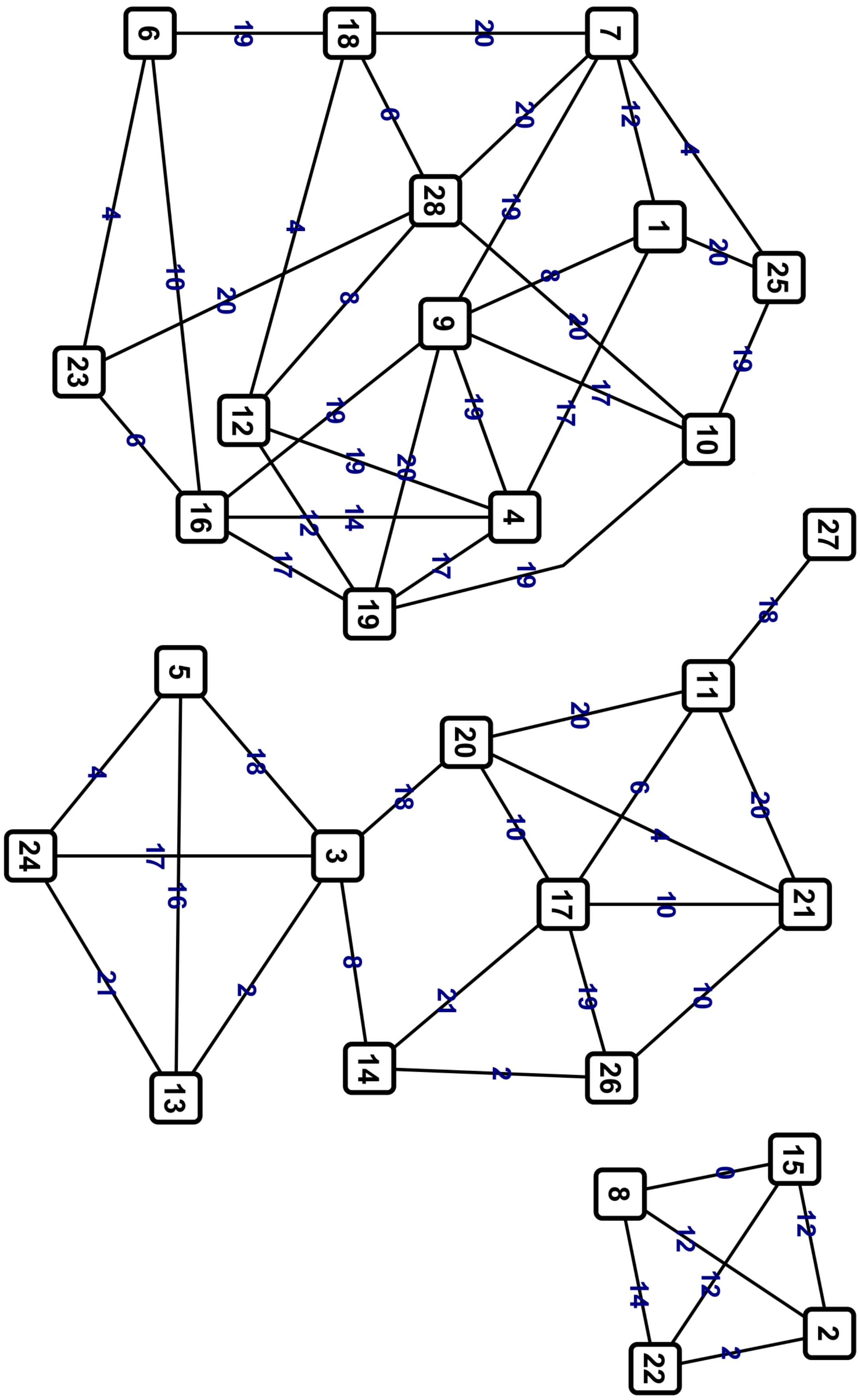


Figure 8.1: Graph to the actual 28 students dataset

to create a working program in Python that was translated to Matlab by Daniel R. Saunders. The latter made the Matlab program free for everyone on the Matlab File Exchange [10]. The Matlabcode has not been included in the appendices, because of its length.

When you let the above program run with the given graph, an error occurs. Although the Matlab program should select for matchings of maximum cardinality, the program does not return such a solution. This is due to the fact that two of the three disjunct groups with an odd number of vertices. In both these groups a student will be left out. These two students are student 10 and 27.

The only way to complete the matching is by making students 10 and 17 a match, although they are not connected in the graph. However, for the algorithm to work properly we want phase 1 to give us a result where all students are matched and we don't have to add matches manually. To ensure a result of maximum cardinality we make the graph of 8.1 a complete graph. We do this by adding edges between all vertices, that don't have an edge yet, of weight 0. Running the program again, but with the complete graph, the results are as follows:

| Couples | | | | | |
|---|---|---|---|---|---|
| 1 | & | 9 | 8 | & | 15 |
| 2 | & | 22 | 10 | & | 27 |
| 3 | & | 13 | 11 | & | 17 |
| 4 | & | 16 | 12 | & | 19 |
| 5 | & | 24 | 14 | & | 26 |
| 6 | & | 23 | 18 | & | 28 |
| 7 | & | 25 | 20 | & | 21 |

Notice that students 10 and 27 are now matched by the program and we don't have to add them manually. Although there was no potential match between these students, adding extra edges made it possible for the program to match them.

REMARKS:

1. The tool is meant to calculate a (sub)optimal classroom lay-out based on data from more than one question. Letting the students answer more (related) questions like "Who do you work well with?", will result in more edges within the graph. This would lower the chances of not getting a result, without adding zero edges in this case.
2. Because it is possible to get a result that is not feasible, the tool should have an integrated check. When the result is not feasible the tool should add edges and recalculate before continuing to the next phase.

## 8.2 Phase 2: RNNA

As with the Blossom algorithm, we talked about finding a minimal cost circuit in chapter 4 about the Repeated Nearest Neighbor algorithm. In this classroom example we are trying to maximize the cost of the circuits. In the same way as described in section 8.1, we can choose to either switch the weights or make a program that calculates the maximum cost circuit. Because we didn't switch the weights with the Blossom part, we are not switching them now. Instead we apply the same principals to use a 'Repeated Furthest Neighbor' algorithm. The Matlab program is in Appendix C.

Before we can run the program, we first have to create a new dataset (CouplesData). The new dataset is filled with the scores between te couples, instead of between the individual students. Therefore the dataset used for RNNA is only half as big as the dataset (Data). In this case we are creating a dataset of size 14. To determine the scores between the couples we use the original dataset and between two couples $ab$ and $cd$ to score is defined as:

$$\text{CouplesData}(ab, cd) = \text{Data}(a, c) + \text{Data}(a, d) + \text{Data}(b, c) + \text{Data}(b, d)$$

For the RNNA to work we need a complete graph. There are two ways to make sure we have a complete graph. We either use the completed graph from phase 1 or we add edges to the CouplesData graph manually. We do this if all four edges between the students are nonexistent in Data and add CouplesData$(ab, cd) = 0$ for those four edges. This way we ensure that the graph with CouplesData is complete. This way we also have resolved the issue that could have arisen from the disjunct groups in the original graph 8.1. The resulting matrix is:

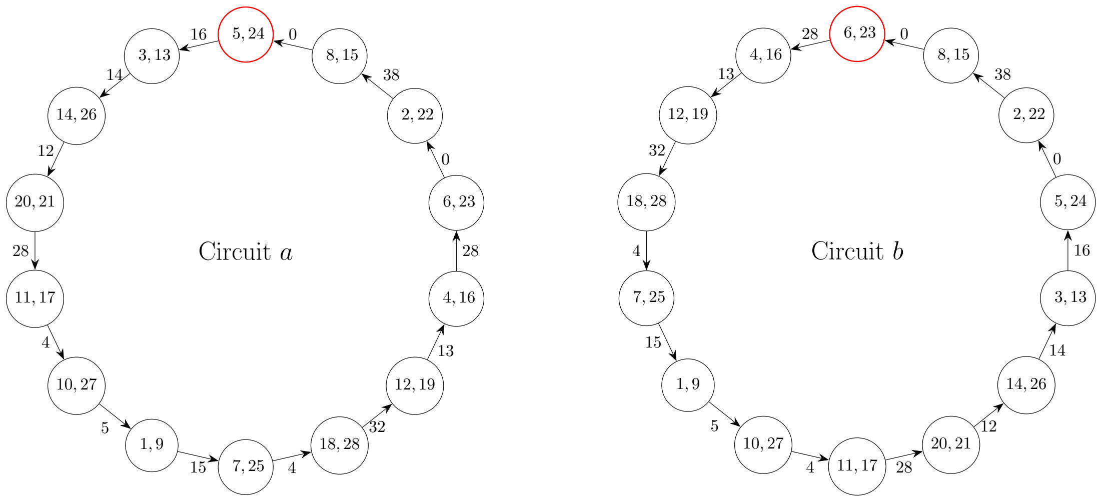
| Scores | | Couples | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $1, 9$ | $2, 22$ | $3, 13$ | $4, 16$ | $5, 24$ | $6, 23$ | $7, 25$ | $8, 15$ | $10, 27$ | $11, 17$ | $12, 19$ | $14, 26$ | $18, 28$ | $20, 21$ |
| Couples | $1, 9$ | x | 0 | 0 | 11 | 0 | 0 | 15 | 0 | 5 | 0 | 2 | 0 | 0 | 0 |
| | $2, 22$ | 0 | x | 0 | 0 | 0 | 0 | 0 | 38 | 0 | 0 | 0 | 0 | 0 | 0 |
| | $3, 13$ | 0 | 0 | x | 0 | 16 | 0 | 0 | 0 | 0 | 0 | 0 | 14 | 0 | 4 |
| | $4, 16$ | 11 | 0 | 0 | x | 0 | 28 | 0 | 0 | 0 | 0 | 13 | 0 | 0 | 0 |
| | $5, 24$ | 0 | 0 | 16 | 0 | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | $6, 23$ | 0 | 0 | 0 | 28 | 0 | x | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 |
| | $7, 25$ | 15 | 0 | 0 | 0 | 0 | 0 | x | 0 | 3 | 0 | 0 | 0 | 4 | 0 |
| | $8, 15$ | 0 | 38 | 0 | 0 | 0 | 0 | 0 | x | 0 | 0 | 0 | 0 | 0 | 0 |
| | $10, 27$ | 5 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | x | 4 | 3 | 0 | 2 | 0 |
| | $11, 17$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | x | 0 | 4 | 0 | 28 |
| | $12, 19$ | 2 | 0 | 0 | 13 | 0 | 0 | 0 | 0 | 3 | 0 | x | 0 | 32 | 0 |
| | $14, 26$ | 0 | 0 | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | x | 0 | 12 |
| | $18, 28$ | 0 | 0 | 0 | 0 | 0 | 5 | 4 | 0 | 2 | 0 | 32 | 0 | x | 0 |
| | $20, 21$ | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 28 | 0 | 12 | 0 | x |

Running the Matlab program from Appendix C, we find the following circuit scores, minimum valued edges and path scores from each starting couple:

| Starting couple | Circuit score | Minimum edge | Path score | Starting couple | Circuit score | Minimum edge | Path score |
|---|---|---|---|---|---|---|---|
| 1 & 9 | 190 | 0 | 190 | 8 & 15 | 190 | 0 | 190 |
| 2 & 22 | 190 | 0 | 190 | 10 & 27 | 191 | 0 | 191 |
| 3 & 13 | 204 | 0 | 204 | 11 & 17 | 204 | 0 | 204 |
| 4 & 16 | 182 | 0 | 182 | 12 & 19 | 206 | 0 | 206 |
| 5 & 24 | 209 | 0 | 209 | 14 & 26 | 204 | 0 | 204 |
| 6 & 23 | 209 | 0 | 209 | 18 & 28 | 203 | 0 | 203 |
| 7 & 25 | 206 | 0 | 206 | 20 & 21 | 195 | 0 | 195 |

With Path score = Circuit score − Minimum edge.

So the Matlab program returns two circuits containing (sub)optimal paths. One starting from couple $5, 24$ and one from couple $6, 23$. The resulting circuits (with scores) are:

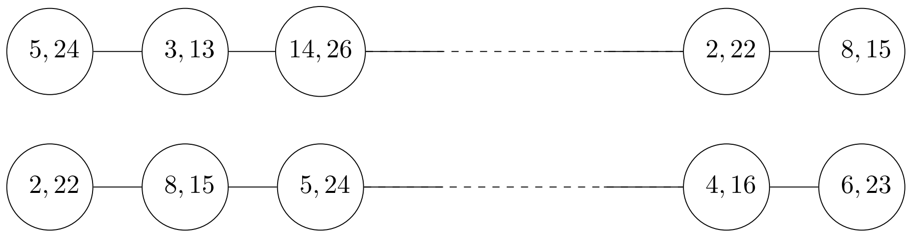


**Remark 9.** *The arrows only indicate in which order the program has made the circuits. Traveling through the circuit the other way will obviously result in the same score. In phase* 3 *the distinction between the directions will be made.*

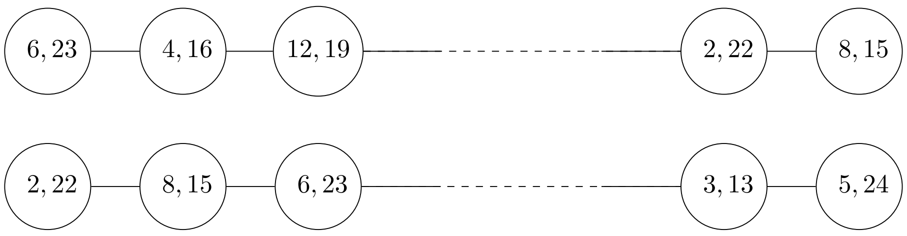
Note that the circuits are almost the same. If you pass trough circuit $a$ the other direction and switch couples $2, 22$ and $8, 15$, the result will be circuit $b$.
Now that we have results from the Repeated Furthest Neighbor algorithm we can take the results into phase 3.

## 8.3 Phase 3: String orientation

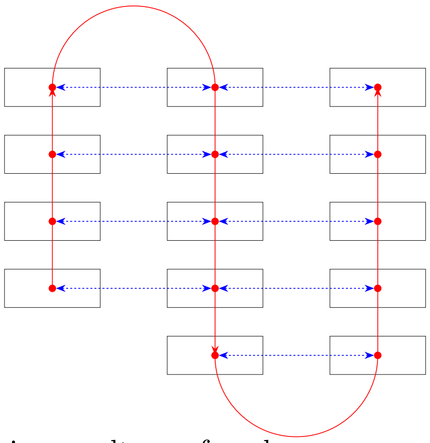
Before we start any calculations we first have to prepare the the circuits for this phase. We are going to make paths from the circuits found in the previous section. To create a path from a circuit we need to eliminate one edge. Because we are trying to maximize the scores it is clear that we eliminate the edge with the lowest weight. Note that in both circuits the lowest weight edge is valued at 0 and that in both circuits there are two edges with this lowest weight. Instead of chosing one of the edges and leaving the other option out we can create both paths from each circuit. The total number of paths is 4 for this example.
From circuit $a$ we obtain a path starting with couple $5, 24$ and ending with couple $8, 15$ and a path starting with couple $2, 22$ and ending with couple $6, 23$.



From circuit $b$ we obtain a path starting with couple $6, 23$ and ending with couple $8, 15$ and a path starting with couple $2, 22$ and ending with couple $5, 24$.



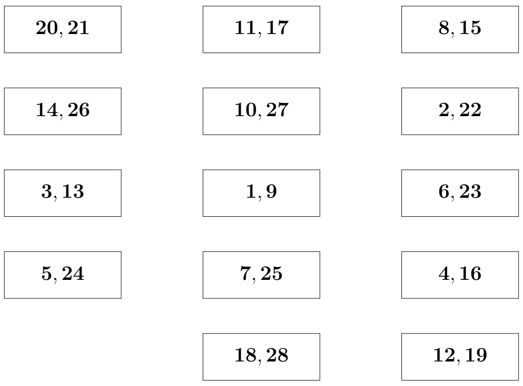
For each path it must be determined which direction (→ or ←) is most optimal. This is done by adding the scores between the aisles (indicated by the blue dashed lines in the illustration to the right). We have a total of four paths, so eight calculations need to be done.



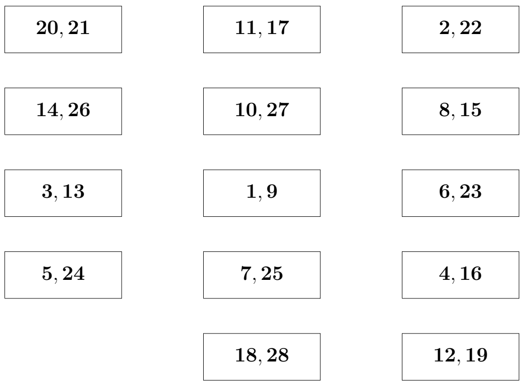
Using the Matlab program from Appendix D the following results are found:

| Circuit | Starting/Ending Couple | Orientation | Score |
|---|---|---|---|
| $a$ | $5, 24/8, 15$ | → | 60 |
| | | ← | 22 |
| | $2, 22/6, 23$ | → | 31 |
| | | ← | 18 |
| $b$ | $6, 23/8, 15$ | → | 18 |
| | | ← | 31 |
| | $2, 22/5, 24$ | → | 22 |
| | | ← | 60 |

The two classrooms we find are:



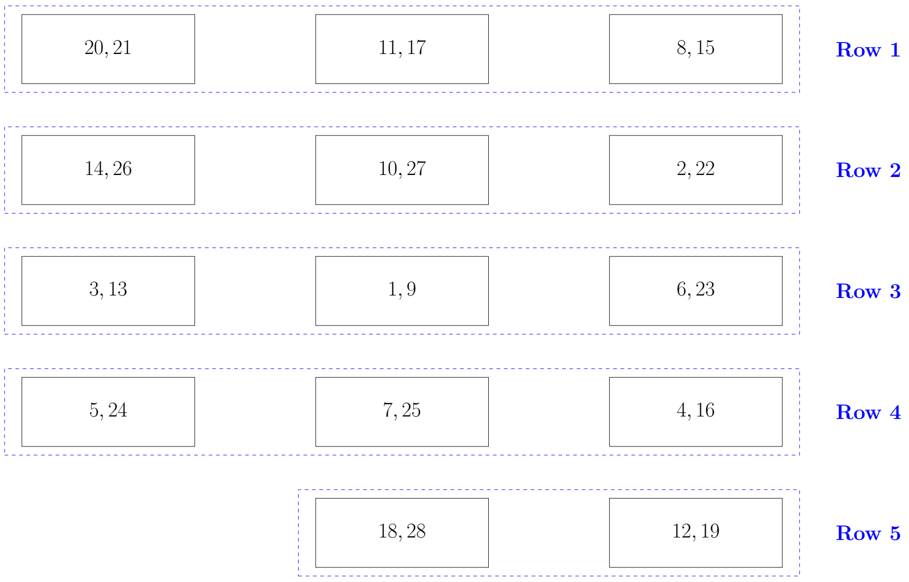
and



Notice that, except for the couples 2, 22 and 8, 15, the classrooms are the same. Recall that the students 2, 8, 15 and 22 are the students who only named each other. The scores from these students to the other students is zero, so for the next phase only one of the classrooms settings above needs to be considered.

## 8.4 Phase 4: Couple orientations

Although the Matlab program (Appendix E) makes the phase 4 calculations for all optimal results from phase 3, we are going to limit ourselves to the first solution. For each row from the first solution we need to calculate what the optimal orientation is for each of the couples in the row. Note that for this phase the original data is being used instead of the CouplesData used in phases 2 and 3.

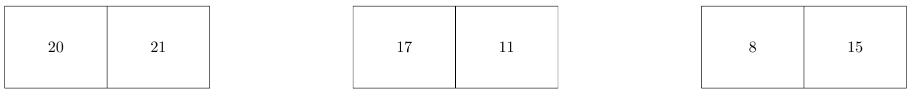


For rows 1 to 4 a total of eight calculations need to be made and only four calculations are required for row 5.

For the first row we find:

| Row 1 | | | | | |
|---|---|---|---|---|---|
| Orientation couple | Score | Orientation couple | Score | Orientation couple | Total |
| 20 21 | **2** | 11 17 | **0** | 8 15 | **2** |
| 20 21 | **2** | 11 17 | **0** | 15 8 | **2** |
| 20 21 | **12** | 17 11 | **0** | 8 15 | **12** |
| 20 21 | **12** | 17 11 | **0** | 15 8 | **12** |
| 21 20 | **2** | 11 17 | **0** | 8 15 | **2** |
| 21 20 | **2** | 11 17 | **0** | 15 8 | **2** |
| 21 20 | **12** | 17 11 | **0** | 8 15 | **12** |
| 21 20 | **12** | 17 11 | **0** | 15 8 | **12** |

There are four possible optimal solutions. For the final result we are free to pick one of the optimal solutions for this row. Choosing the first instance of an optimal solution we find as a result for the first row:



For the second row we find the following results:

| Row 2 | | | | | |
|---|---|---|---|---|---|
| Orientation couple | Score | Orientation couple | Score | Orientation couple | Total |
| 14 26 | **0** | 10 27 | **0** | 2 22 | **0** |
| 14 26 | **0** | 10 27 | **0** | 22 2 | **0** |
| 14 26 | **0** | 27 10 | **0** | 2 22 | **0** |
| 14 26 | **0** | 27 10 | **0** | 22 2 | **0** |
| 26 14 | **0** | 10 27 | **0** | 2 22 | **0** |
| 26 14 | **0** | 10 27 | **0** | 22 2 | **0** |
| 26 14 | **0** | 27 10 | **0** | 2 22 | **0** |
| 26 14 | **0** | 27 10 | **0** | 22 2 | **0** |

None of the orientations results higher than zero. So, again, we are free to choose an orientation. Note that because of the relatively small number of edges in the graph, it is not unimaginable that we only get 'zero' results for a row. Choosing the first orientation we find for row 2:

| | | | | | |
|---|---|---|---|---|---|
| 14 | 26 | 10 | 27 | 2 | 22 |

For the third row we find:

| Row 3 | | | | | |
|---|---|---|---|---|---|
| Orientation couple | Score | Orientation couple | Score | Orientation couple | Total |
| 3 13 | **0** | 1 9 | **0** | 6 23 | **0** |
| 3 13 | **0** | 1 9 | **0** | 23 6 | **0** |
| 3 13 | **0** | 9 1 | **0** | 6 23 | **0** |
| 3 13 | **0** | 9 1 | **0** | 23 6 | **0** |
| 13 3 | **0** | 1 9 | **0** | 6 23 | **0** |
| 13 3 | **0** | 1 9 | **0** | 23 6 | **0** |
| 13 3 | **0** | 9 1 | **0** | 6 23 | **0** |
| 13 3 | **0** | 9 1 | **0** | 23 6 | **0** |

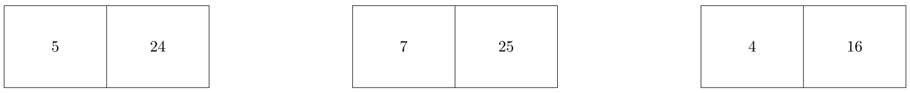
Just as with row 2, we don't find any solutions higher than zero. Choosing the first orientation we find for row 3:

| | | | | | |
|---|---|---|---|---|---|
| 3 | 13 | 1 | 9 | 6 | 23 |

For the fourth row we find:

| Row 4 | | | | | |
|---|---|---|---|---|---|
| Orientation couple | Score | Orientation couple | Score | Orientation couple | Total |
| 5 24 | **0** | 7 25 | **0** | 4 16 | **0** |
| 5 24 | **0** | 7 25 | **0** | 16 4 | **0** |
| 5 24 | **0** | 25 7 | **0** | 4 16 | **0** |
| 5 24 | **0** | 25 7 | **0** | 16 4 | **0** |
| 24 5 | **0** | 7 25 | **0** | 4 16 | **0** |
| 24 5 | **0** | 7 25 | **0** | 16 4 | **0** |
| 24 5 | **0** | 25 7 | **0** | 4 16 | **0** |
| 24 5 | **0** | 25 7 | **0** | 16 4 | **0** |

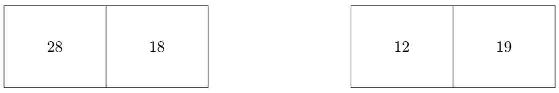
Yet again we can freely choose one of the orientations. Choosing the first one we find for row 4:



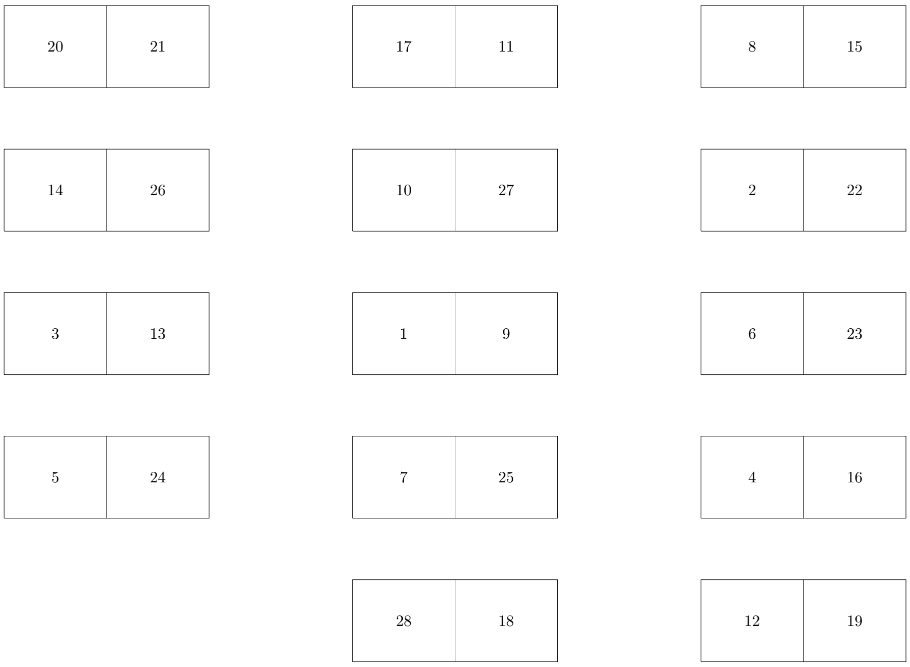
For the fifth row we find:

| Row 5 | | | |
|---|---|---|---|
| Orientation couple | Score | Orientation couple | Total |
| 18 28 | **14** | 12 19 | **14** |
| 18 28 | **0** | 19 12 | **0** |
| 28 18 | **18** | 12 19 | **18** |
| 28 18 | **0** | 19 12 | **0** |

Here we find a single optimal solution for this row. For row 5 we find:



Combining the solutions for all the rows results in the following classroom lay-out:



It is important to note that this example was done with data from just one question. For clearer result it is desirable to use data from more than one question. Other questions that could be combined with the question used here are:

- Next to who of your classmates don't you want to sit?
- With who of your classmates do you work well with?
- With who of your classmates don't you work well with?
- ...

# Chapter 9

# Discussion

## 9.1 Triplets

The first and foremost limitation of this algorithm is that it can only deal with classrooms where the students sit in couples, as in figure 9.1. Classrooms where there is a column of triplets (figure 9.2) is not an uncommon thing at schools. This algorithm is not capable of optimizing a classroom lay-out where triplets are allowed. In phase 1 (Chapter 4) we have created couples. For creating triplets the Blossom algorithm is not suited and should be replaced by an other algorithm, that creates triplets and couples.

Also the chosen strategy in phase 2 and phase 3 is no longer valid. Because creating a string of the couples / triplets, using the RNNA, will not ensure that all the triplets will end up in the column with sets of three seats. So for seating the couples / triplets an other strategy must be chosen to determine where the couples / triplets are seated. And a new algorithm according with the new strategy.

The final phase will stay the same. For each row it now takes 12 brute-force calculations instead of 8. Which is very reasonable for a brute-force method.

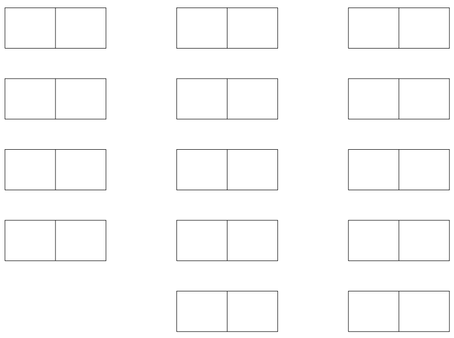


Figure 9.1: A classroom where this algorithm works

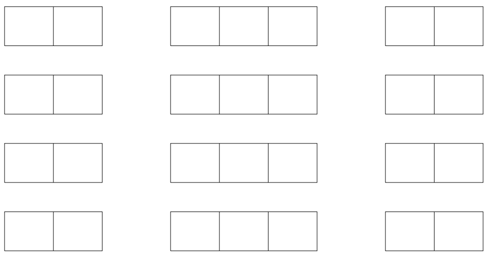


Figure 9.2: A classroom where this algorithm does not work

## 9.2 Phase 2, 3 and the RNNA

The strategy to create strings and fill the classroom from column to column is debatable. The same string could also be used to fill the classroom from row to row, although we assumed that it is more important for a student who sits in front / behind of them than who sits accross the aisle of them.

The strategy is mostly debatable because creating a string might not be the best strategy. In the process of developing the strategy, a second run of the Blossom algorithm was considered, to create quadruples from the couples. And again to create groups of eight. This process will fail when a column consists of a number of couples that is not a power of 2. To create the strings we have searched for (Hamilton) circuits in a complete graph. This problem is better known as the *Traveling Salesman Problem*(TSP). There exists no exact algorithm for the TSP that solves the problem in polynomial time.

There exist exact algorithms that can solve the TSP if the number of “cities” is not to big. In this thesis we have worked with 14 couples (represented by cities in the TSP). A good algorithm that can exaclty solve the TSP of this size is the *Branch and Bound algorithm*(BnB) as described by John D.C. Little et al. in [8]. However the BnB does not run in polynomial time. This is the reason we decided to work with the (Repeated) Nearest Neighbor algorithm. Although the RNNA is a heuristic (greedy) algorithm, and does not ensure an optimal solution, it does run in polynomial time. Just like the other phases of the total algorithm described in this thesis. It is because of the RNNA that the total algorithm will result in a (sub)optimal solution.

## 9.3 Possible improvements

A few improvements to this algorithm can be made.

- The implementation of the Branch and Bound algorithm to replace the Repeated Nearest Neighbor algorithm
- A ‘better’ strategy to determine the positions of the couples in the classroom.

The possibility for the algorithm to deal with triplets in the classroom could be considered an improvement. But because three of the four phases described in this thesis will no longer work, an entirely new algorithm must be developed.

## 9.4 Further research questions

The goal of this algorithm is that it will be implemented in a tool for teachers to optimize the classroom lay-out to their wishes. This can either be to improve social cohesion, to have an optimal learning environment or any other optimum. This thesis only worked with the question: “Who do you want to sit next to?”. But what questions do you need to ask to improve the social cohesion of a class? Or, the questions needed to create an optimal learning environment. And the weights these questions need to have to end up with the desired result. This kind of research needs to be left to a different area of science, most likely sociology.

And, of course, a tool needs to be developed. A tool that by a click on a button prints out the classroom lay-out that matches the desired aim(s) of the teacher for that lesson. This tool will need the data from the answered questions from the students, the aim(s) from the teacher, the translation from the aims to the needed questions and their respective weights, the lay-out of the classroom and the appropiate algorithm.

# Chapter 10

# Conclusion

Now that the algorithm is complete and an example has been worked through, it is time to draw some conclusions. We will be discussing the overal complexity of the algorithm and make conclusions for each phase of the algorithm. But before we continu, we conclude that the algorithm works. Given a graph with a sufficient number of edges the algorithm (and the program) succesfully returns a classroom lay-out.

## 10.1 Complexity

In the chapters 4 through 7 we worked out the different phases of the algorithm. In each chapter an upper limit for the (computational) complexity was given. Phase 1 (the Blossom algorithm) has a complexity of $\mathcal{O}(n^2m)$. Remember that $n$ was the number of students and $m$ the number of edges between them. Phase 2 (the Repeated Nearest Neighbor algorithm) has a complexity of $\mathcal{O}(n^2)$. Between phase 2 and phase 3 some extra calculations need to be made to produce strings from the circuits. For each circuit, $n$ calculations need to be made to produce a string. This brings the total complexity of phase 2 to $\mathcal{O}(n^3)$. Phase 3 takes at most $\frac{n^2}{2}$ calculations, so is of the order $\mathcal{O}(n^2)$. An upper bound of cost calculations for phase 4 is $\frac{2n^3}{3} + 4n^2$, so the order is $\mathcal{O}(n^3)$.

The total complexity thus becomes $\mathcal{O}(n^2m + n^3 + n^2 + n^3) = \mathcal{O}(n^2m + n^3)$. When a class is asked to name three classmates for each question to create the data, the number of edges $m$ in phase 1 will always be bigger or equal than the number of students $n$. So $m \geq n$.

| Phase | Complexity |
|---|---|
| 1 | $\mathcal{O}(n^2m)$ |
| 2 | $\mathcal{O}(n^3)$ |
| 3 | $\mathcal{O}(n^2)$ |
| 4 | $\mathcal{O}(n^3)$ |
| Total | $\mathcal{O}(n^2m)$ |

Running the Matlab program on an 8 year old laptop computer takes 20.528120 seconds. This includes the time needed to import the data from a Microsoft Excel file. If the data is already imported it only takes 0.569090 seconds to complete the program and return a classroom lay-out.

Having an efficient method of importing data into the Matlab program, will result in a runtime fast enough for the program to be used in a tool.

## 10.2 Phase 1

To match the students into couples, the Blossom algorithm works really well. As long as $n$ is an even number.

When a class consists of an odd number of students it becomes necessary to add a dummy vertex to the graph with edges of 'horrible' weights to all other students. Horrible might be either a really high or a really low weigth depending on a minimizing or maximizing optimalisation problem.

If the graph is a complete graph, we can fine-tune the order of the algorithm. A complete graph with $n$ vertices has $m = \frac{n(n-1)}{2}$ edges, so the order of the algorithm becomes $\mathcal{O}(n^3(n-1))$. This is bounded by $\mathcal{O}(n^4)$. So even for a complete graph, this algorithm can still solve the minimum cost perfect matching in polynomial time.

## 10.3 Phase 2 and Phase 3

Although being an approximation algorithm, the RNNA does a good job at creating circuits and paths to use in these phases. Before starting the algorithm of phase 2 a complete graph is produced. One could try to find a string (or circuit) in an incomplete graph, but, as seen in section 8.1, it is possible for a group of students to isolate themselves from the rest of the class. Because of the distinct subgraphs, it is impossible to track a circuit. Thus, the creation of a complete graph is necessary.

## 10.4 Phase 4

Once the couples have been placed in the classroom, this brute-force method is straightforward. With the computational complexity in the order of $\mathcal{O}(n^3)$, this phase is not a bottleneck for the algorithm as a whole.

## 10.5 Data

As mentioned in Section 8.1, if the data used to create the graph is not sufficient to create enough edges in that graph, than the program might get stuck and additional edges must be added to the graph to succesfully compute a solution. But even if the starting graph would be a complete one, the algorithm still produces a result within polynomial time.

# Appendices

# Appendix A: Matlabcode for Gale-Shapley algorithm

```
%Example Marriage Problem
%M = [1 2 3 4;1 4 3 2;2 1 3 4;4 2 3 1]; %Preference matrix Males
%F = [3 4 2 1;3 1 4 2;2 3 4 1;3 2 1 4]; %Preference matrix Females
%Application to Classroom Problem
M = [6 3 5 1 4 2;4 6 2 5 3 1;3 5 6 1 2 4;4 1 5 6 3 2;5 2 3 1 6 4;2 4 5 3 1 6];
    %Preference matrix Males
F = [6 3 5 1 4 2;4 6 2 5 3 1;3 5 6 1 2 4;4 1 5 6 3 2;5 2 3 1 6 4;2 4 5 3 1 6];
    %Preference matrix Females
%In application to the Classroom Problem: M=F

%Initialisation
n=length(M); %number of Males
Am = zeros(n,1); %Availability Males    0:=available 1:=unavailable
Af = zeros(n,1); %Availability Females
Match = zeros(n,n); %Matches
CountChoice = ones(n,1); %counter to the choose a male is at
Mord = zeros(n,n); %Ordening of the males. First entry is first preference
Ford = zeros(n,n); %Ordening of the females

%Ordening of M and F
for i=1:n
    for j=1:n
        Mord(i,j) = find(M(i,:)==j);
        Ford(i,j) = find(F(i,:)==j);
    end;
end;

%Algorithm
while sum(Am)~=n %Every male must be unavailable
    for i=1:n
        if Am(i)==0 %Male is available
            T=Mord(i,CountChoice(i)); %Target
            if Af(T) == 0 %Target is not holding a proposal
                Am(i) = 1; %Male becomes unavailable
                Af(T) = 1; %Target becomes unavailable
                Match(i,T) = 1; %Male and Target are matched
            else %Target is not available
```

```
                Comp = find(Match(:,T)==1); %Who is the competition
                if F(T,i)<F(T,Comp) %Target prefers male over competitor
                    Am(Comp)=0; %Competitor becomes available
                    Am(i)=1; %Male becomes unavailable
                    Match(Comp,T)=0;%Competitor and Target become unmatched
                    Match(i,T)=1; %Male and target become matched
                else %Target does not prefer male over competitor
                    CountChoice(i)=CountChoice(i)+1;
                                   %Male goes to the next target
                end;
            end;
        end;
    end;
end;
Match
```

# Appendix B: Matlabcode for the Roommate Problem

```
%Example
X = [4 6 2 5 3;6 3 5 1 4;4 5 1 6 2;2 6 5 1 3;4 2 3 6 1;5 1 4 2 3];
    %Preference matrix

%Initialisation
n=length(X); %number people
A1 = zeros(n,1); %Availability Askers    0:=available 1:=unavailable
A2 = zeros(n,1); %Availability Askees
Match = zeros(n,n); %Matches
CountChoice = ones(n,1); %counter to the choose a person is at

%Algorithm
%Phase 1: Matching
while sum(A1)~=n %Every person must have asked succesfully
    if sum(CountChoice(:,1)>n-1)>0 %Check for state 1
        break
    end;
    for i=1:n
        if A1(i)==0 %Person is available to ask
            T=X(i,CountChoice(i)); %Target
            if A2(T) == 0 %Target is not holding a proposal
                A1(i) = 1; %Person becomes unavailable
                A2(T) = 1; %Target becomes unavailable
                Match(i,T) = 1; %Person and Target are matched
            else %Target is not available
                Comp = find(Match(:,T)==1); %Who is the competition
                if find(X(T,:)==i)<find(X(T,:)==Comp)
                                    %Target prefers person over competitor
                    A1(Comp)=0; %Competitor becomes available
                    A1(i)=1; %Person becomes unavailable
                    Match(Comp,T)=0;%Competitor and Target become unmatched
                    Match(i,T)=1; %Person and target become matched
                else %Target does not prefer person over competitor
                    CountChoice(i)=CountChoice(i)+1;
                                   %Person goes to the next target
                end;
            end;
        end;
    end;
end;

if sum(sum(Match))<n
    disp('No Stable Match Possible.')
    Match
else
%Phase 1: Reduction
Xred(:,:,1) = X; %Prepare reduction matrix
%Reduction rule i)
for i=1:n
    Prop(i) = find(Match(:,i)==1); %From who (x) is i holding a proposal
    PosProp(i) = find(X(i,:)==Prop(i));
                                  %What is the position of x in i's list
    for j=1:n-1
        if j>PosProp(i) %Everyone worse than x
            Xred(i,j,1)=0; %Remove from list
        end;
    end;
end;

%Reduction rule ii)
for i=1:n
    for j=1:PosProp(i)-1
        a=find(X(X(i,j),:)==i); %Position of i in row of X(i,j)
        if a>PosProp(X(i,j))%Position of i is worse than proposer to X(i,j)
            Xred(i,j,1)=0;
        end;
    end;
end;

%Check is result is a stable matching
tempcounter = 0; %counter for nonzero elements
for i=1:n
    tempcounter = tempcounter + nnz(Xred(i,:,1)); %add nonzero elements
end;
if tempcounter == n
    disp('A stable match have been found')
    Match
else

%Phase 2: Recognition of All-or-nothing cycles
reduction=1; %set reduction performed
while nnz(Xred(:,:,reduction))>n %Check for state 1)

%Search for x(1)
condition = 0; %Create condition to stop searching
j=1; %Person
```

```
while condition==0
    if nnz(Xred(j,:,reduction))>1
        x(1)=j;
        condition=1;
    else
        j=j+1;
    end;
end;

%Create array for x and y
i=1;
while sum(unique(x))==sum(x)
    searchsecond = find(Xred(x(i),:,reduction)~=0,2,'first');
                                %Determine second person on x(i)'s list
    y(i)=Xred(x(i),searchsecond(end),reduction); %Set y(i)
    searchlast = find(Xred(y(i),:,reduction)~=0,1,'last');
                                       %Find last person on y(i)'s list
    x(i+1) = Xred(y(i),searchlast,reduction); %Set x(i+1)
    i=i+1;
end;

%Create arrays for a and b
s=find(x(:)==x(end),1); %Determine index of first recurring number
j=1;
for i=s:length(x)-1
    a(j) = x(s+j-1); %set a(j)
    k=find(Xred(a(j),:,reduction)~=0,2);
                         %Determine first nonzero elements in Xred(a(j),:)
    b(j+1) = Xred(a(j),k(2),reduction); %Set b(j+1)
    j=j+1;
end;
b(1)=b(end); %set b(1)=b(r+1)

%Phase 2: Reduction
Xred(:,:,reduction+1)=Xred(:,:,reduction);
                                     %set new reduced preference matrix
%Reduction rule i)
for i=1:length(a)
    PosProp2(i)=find(Xred(b(i+1),:,reduction)==a(i));
                          %What is the position of a(i) in b(i+1)'s list
    for j=1:n-1
        if j>PosProp2(i) %all successors of a(i)
            Xred(b(i+1),j,reduction+1)=0; %remove successors
        end;
    end;
end;

%Reduction rule ii)
for i=1:length(a)
```

```
        k=1;
        for l=PosProp2(i)+1:n-1
            if Xred(b(i+1),l,reduction)~=0
                RemoveFrom(i,k) = Xred(b(i+1),l,reduction);
                                      %From which lists must b(i+1) be removed
                k=k+1;
            end;
        end;
    end;

    %Remove b(i)'s from lists
    for i=1:length(a)
        for j=1:length(RemoveFrom(i,:))
            for l=1:n-1
                if Xred(RemoveFrom(i,j),l,reduction)==b(i+1)
                    Xred(RemoveFrom(i,j),l,reduction+1)=0; %Remove b from list
                end;
            end;
        end;
    end;
    reduction=reduction+1;
    end;

    %check for state 2)
    if nnz(Xred(:,:,reduction))==0
      disp('No stable matching exists. All reduced preference lists are empty')
    else %Phase 2 ends in state 3)
        Match = zeros(n,n);
        for i=1:n
            Match(i,Xred(i,find(Xred(i,:,end)~=0),end))=1; %Set Match matrix
        end;
    end;

    Xfinal = zeros(n,2);
    Xfinal(:,1) = [1 2 3 4 5 6].';
    A = Xred(:,:,end).';
    %two different represenations of the result
    Xfinal(:,2) = reshape(A(A~=0),6,1)
    Match
    end;
    end;
```

# Appendix C: Matlabcode for the Repeated Furthest Neighbor algorithm

```
function [strings,scores,circuits] = RFN(data) %Repeated Furthest Neighbor

n=length(data); %Determine sample size
strings=zeros(n,n);
scores=zeros(n,n); %Create matrices to fill
circuits=zeros(n+1,n);
for i=1:n
    strings(1,i)=i;
    circuits(1,i)=i;
    TempData = data; %Temporary data copy used for eliminating edges without changin
                      the data
    t=i;
    r=2;
    while sum(TempData(t,:))> -1*n
        m=max(TempData(t,:)); %value maximum edge
        k = find(TempData(t,:)==m); %location maximum edge
        k = k(1,1); %choose first encounter of maximum edge
        strings(r,i)=k; %add next vertix to string and circuit
        circuits(r,i)=k;
        scores(r-1,i)=m; %record score edge in string
        TempData(t,:)=-1; % eliminate edges to t
        TempData(:,t)=-1; % eliminate edges to t
        t=k; %go to next vertix
        r=r+1;
    end;
    scores(n,i)=data(strings(n,i),strings(1,i)); %Add the score between the first an
                                                   last entry
end;
circuits(n+1,:)=circuits(1,:); %complete the circuit
```

# Appendix D: Matlabcode for phase 3

The code for Phase 3 consists of two functions, one filling the classroom with all the strings and a second to isolate the filled classrooms with the best scores.

Matlabcode for the 'Fill Classroom' function:

```
function [FilledClassrooms,Score] = FillClassroom(Strings,classroom,Data)

[z,n]=size(Strings); %determine the amount of strings
[p,q]=size(classroom); %determine the size of the classroom
FilledClassrooms=-1.*ones(p,q,n);%for each string a classroom is prepared
Score=zeros(1,n);
for i=1:n
    counter=1; %counter to progress through each string
    for a=1:p-1 %fill the first column of the classroom
        FilledClassrooms(p-a,1,i)=Strings(counter,i);
        counter=counter+1;
    end;
    for b=1:p %fill the second column of the classroom
        FilledClassrooms(b,2,i)=Strings(counter,i);
        counter=counter+1;
    end;

    for c=1:p %fill the third column of the classroom
        FilledClassrooms(p-c+1,3,i)=Strings(counter,i);
        counter=counter+1;
    end;
    for d=1:p-1
        Score(1,i)=Score(1,i)+Data(FilledClassrooms(d,1,i),FilledClassrooms(d,2,i))+
        Data(FilledClassrooms(d,2,i),FilledClassrooms(d,3,i)); %for each row the
                                                                    scores are added
    end;
    Score(1,i)=Score(1,i)+Data(FilledClassrooms(p,2,i),FilledClassrooms(p,3,i));
    %for the final row (of size 2) the score is added
end;
```

Matlabcode for the 'Isolate Optimal Classrooms' function:

```
function [Classrooms,ClassroomScores] = IsolateOptimalClassrooms(rooms,scores)

k=find(scores==max(scores)); %determine the maximum scores of the strings
[n,m,z]=size(rooms); %size of the classroom
p=length(k); %The number of times the optimal scores occurs
Classrooms=zeros(n,m,p);
ClassroomScores=zeros(1,p);
for i=1:p
    Classrooms(:,:,i)=rooms(:,:,k(i)); %isolate optimal classrooms
    ClassroomScores(1,i)=scores(1,k(i)); %isolate scores of optimal classrooms
end;
```

# Appendix E: Matlabcode for phase 4

```
function [FilledExpClassrooms,FilledExpScores] =
                    FillExpandedClassrooms(rooms,couples,data,ExpClassroom)

[n,m,p]=size(rooms);
for i=1:p
    FilledExpClassrooms(n,1:2,i)=[-1,-1];
    FilledExpScores=0;
    if n<=3
        for j=1:n-1
            score=data(couples(rooms(j,1,i),2),couples(rooms(j,2,i),1));
            FilledExpClassrooms(j,1:2,i)=
                        [couples(rooms(j,1,i),1),couples(rooms(j,1,i),2)];
            FilledExpClassrooms(j,3:4,i)=
                        [couples(rooms(j,2,i),1),couples(rooms(j,2,i),2)];

            if score<data(couples(rooms(j,1,i),1),couples(rooms(j,2,i),1))
               score=data(couples(rooms(j,1,i),1),couples(rooms(j,2,i),1));
                FilledExpClassrooms(j,1:2,i)=
                        [couples(rooms(j,1,i),2),couples(rooms(j,1,i),1)];
                FilledExpClassrooms(j,3:4,i)=
                        [couples(rooms(j,2,i),1),couples(rooms(j,2,i),2)];
            end;
            if score<data(couples(rooms(j,1,i),2),couples(rooms(j,2,i),2))
               score=data(couples(rooms(j,1,i),2),couples(rooms(j,2,i),2));
                FilledExpClassrooms(j,1:2,i)=
                        [couples(rooms(j,1,i),1),couples(rooms(j,1,i),2)];
                FilledExpClassrooms(j,3:4,i)=
                        [couples(rooms(j,2,i),2),couples(rooms(j,2,i),1)];
            end;
            if score<data(couples(rooms(j,1,i),1),couples(rooms(j,2,i),2))
               score=data(couples(rooms(j,1,i),1),couples(rooms(j,2,i),2));
                FilledExpClassrooms(j,1:2,i)=
                        [couples(rooms(j,1,i),2),couples(rooms(j,1,i),1)];
                FilledExpClassrooms(j,3:4,i)=
                        [couples(rooms(j,2,i),2),couples(rooms(j,2,i),1)];
            end;
            FilledExpScores=FilledExpScores+score;
        end;
        FilledExpClassrooms(n,2*m-1:2*m,i)=
                        [couples(rooms(n,2,i),1),couples(rooms(n,2,i),2)];
    else
        for j=1:n-1
            score=data(couples(rooms(j,1,i),2),couples(rooms(j,2,i),1))+
                     data(couples(rooms(j,2,i),2),couples(rooms(j,3,i),1));
            FilledExpClassrooms(j,1:2,i)=
                         [couples(rooms(j,1,i),1),couples(rooms(j,1,i),2)];
            FilledExpClassrooms(j,3:4,i)=
                         [couples(rooms(j,2,i),1),couples(rooms(j,2,i),2)];
            FilledExpClassrooms(j,5:6,i)=
                         [couples(rooms(j,3,i),1),couples(rooms(j,3,i),2)];

            if score<data(couples(rooms(j,1,i),1),couples(rooms(j,2,i),1))+
                      data(couples(rooms(j,2,i),2),couples(rooms(j,3,i),1))
               score=data(couples(rooms(j,1,i),1),couples(rooms(j,2,i),1))+
                     data(couples(rooms(j,2,i),2),couples(rooms(j,3,i),1));
                FilledExpClassrooms(j,1:2,i)=
                         [couples(rooms(j,1,i),2),couples(rooms(j,1,i),1)];
                FilledExpClassrooms(j,3:4,i)=
                         [couples(rooms(j,2,i),1),couples(rooms(j,2,i),2)];
                FilledExpClassrooms(j,5:6,i)=
                         [couples(rooms(j,3,i),1),couples(rooms(j,3,i),2)];
            end;
            if score<data(couples(rooms(j,1,i),2),couples(rooms(j,2,i),2))+
                      data(couples(rooms(j,2,i),1),couples(rooms(j,3,i),1))
               score=data(couples(rooms(j,1,i),2),couples(rooms(j,2,i),2))+
                     data(couples(rooms(j,2,i),1),couples(rooms(j,3,i),1));
                FilledExpClassrooms(j,1:2,i)=
                         [couples(rooms(j,1,i),1),couples(rooms(j,1,i),2)];
                FilledExpClassrooms(j,3:4,i)=
                         [couples(rooms(j,2,i),2),couples(rooms(j,2,i),1)];
                FilledExpClassrooms(j,5:6,i)=
                         [couples(rooms(j,3,i),1),couples(rooms(j,3,i),2)];
            end;
            if score<data(couples(rooms(j,1,i),1),couples(rooms(j,2,i),2))+
                      data(couples(rooms(j,2,i),1),couples(rooms(j,3,i),1))
               score=data(couples(rooms(j,1,i),1),couples(rooms(j,2,i),2))+
                     data(couples(rooms(j,2,i),1),couples(rooms(j,3,i),1));
                FilledExpClassrooms(j,1:2,i)=
                         [couples(rooms(j,1,i),2),couples(rooms(j,1,i),1)];
                FilledExpClassrooms(j,3:4,i)=
                         [couples(rooms(j,2,i),2),couples(rooms(j,2,i),1)];
                FilledExpClassrooms(j,5:6,i)=
                         [couples(rooms(j,3,i),1),couples(rooms(j,3,i),2)];
            end;
            if score<data(couples(rooms(j,1,i),2),couples(rooms(j,2,i),1))+
                      data(couples(rooms(j,2,i),2),couples(rooms(j,3,i),2))
```

```
           score=data(couples(rooms(j,1,i),2),couples(rooms(j,2,i),1))+
                 data(couples(rooms(j,2,i),2),couples(rooms(j,3,i),2));
            FilledExpClassrooms(j,1:2,i)=
                     [couples(rooms(j,1,i),1),couples(rooms(j,1,i),2)];
            FilledExpClassrooms(j,3:4,i)=
                     [couples(rooms(j,2,i),1),couples(rooms(j,2,i),2)];
            FilledExpClassrooms(j,5:6,i)=
                     [couples(rooms(j,3,i),2),couples(rooms(j,3,i),1)];
        end;
        if score<data(couples(rooms(j,1,i),1),couples(rooms(j,2,i),2))+
                  data(couples(rooms(j,2,i),1),couples(rooms(j,3,i),2))
           score=data(couples(rooms(j,1,i),1),couples(rooms(j,2,i),2))+
                 data(couples(rooms(j,2,i),1),couples(rooms(j,3,i),2));
            FilledExpClassrooms(j,1:2,i)=
                     [couples(rooms(j,1,i),2),couples(rooms(j,1,i),1)];
            FilledExpClassrooms(j,3:4,i)=
                     [couples(rooms(j,2,i),2),couples(rooms(j,2,i),1)];
            FilledExpClassrooms(j,5:6,i)=
                     [couples(rooms(j,3,i),2),couples(rooms(j,3,i),1)];
        end;

        if score<data(couples(rooms(j,1,i),2),couples(rooms(j,2,i),2))+
                  data(couples(rooms(j,2,i),1),couples(rooms(j,3,i),2))
           score=data(couples(rooms(j,1,i),1),couples(rooms(j,2,i),2))+
                 data(couples(rooms(j,2,i),1),couples(rooms(j,3,i),2));
            FilledExpClassrooms(j,1:2,i)=
                     [couples(rooms(j,1,i),1),couples(rooms(j,1,i),2)];
            FilledExpClassrooms(j,3:4,i)=
                     [couples(rooms(j,2,i),2),couples(rooms(j,2,i),1)];
            FilledExpClassrooms(j,5:6,i)=
                     [couples(rooms(j,3,i),2),couples(rooms(j,3,i),1)];
        end;

        if score<data(couples(rooms(j,1,i),1),couples(rooms(j,2,i),1))+
                  data(couples(rooms(j,2,i),2),couples(rooms(j,3,i),2))
           score=data(couples(rooms(j,1,i),1),couples(rooms(j,2,i),2))+
                 data(couples(rooms(j,2,i),1),couples(rooms(j,3,i),2));
            FilledExpClassrooms(j,1:2,i)=
                     [couples(rooms(j,1,i),2),couples(rooms(j,1,i),1)];
            FilledExpClassrooms(j,3:4,i)=
                     [couples(rooms(j,2,i),1),couples(rooms(j,2,i),2)];
            FilledExpClassrooms(j,5:6,i)=
                     [couples(rooms(j,3,i),2),couples(rooms(j,3,i),1)];
        end;

        FilledExpScores=FilledExpScores+score;
    end;

    score=data(couples(rooms(n,2,i),2),couples(rooms(n,3,i),1));
```

```
        FilledExpClassrooms(n,3:4,i)=
                         [couples(rooms(n,2,i),1),couples(rooms(n,2,i),2)];
        FilledExpClassrooms(n,5:6,i)=
                         [couples(rooms(n,3,i),1),couples(rooms(n,3,i),2)];

        if score<data(couples(rooms(n,2,i),1),couples(rooms(n,3,i),1))
            score=data(couples(rooms(n,2,i),1),couples(rooms(n,3,i),1));
            FilledExpClassrooms(n,3:4,i)=
                         [couples(rooms(n,2,i),2),couples(rooms(n,2,i),1)];
            FilledExpClassrooms(n,5:6,i)=
                         [couples(rooms(n,3,i),1),couples(rooms(n,3,i),2)];
        end;

        if score<data(couples(rooms(n,2,i),2),couples(rooms(n,3,i),2))
            score=data(couples(rooms(n,2,i),2),couples(rooms(n,3,i),2));
            FilledExpClassrooms(n,3:4,i)=
                         [couples(rooms(n,2,i),1),couples(rooms(n,2,i),2)];
            FilledExpClassrooms(n,5:6,i)=
                         [couples(rooms(n,3,i),2),couples(rooms(n,3,i),1)];
        end;

        if score<data(couples(rooms(n,2,i),1),couples(rooms(n,3,i),2))
            score=data(couples(rooms(n,2,i),1),couples(rooms(n,3,i),2));
            FilledExpClassrooms(n,3:4,i)=
                         [couples(rooms(n,2,i),2),couples(rooms(n,2,i),1)];
            FilledExpClassrooms(n,5:6,i)=
                         [couples(rooms(n,3,i),2),couples(rooms(n,3,i),1)];
        end;
        FilledExpScores=FilledExpScores+score;
    end;
end;
```

# Bibliography

[1] Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of Research of the National Bureau of Standards B*, 69(125-130):55–56, 1965.

[2] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

[3] David Gale and Lloyd S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[4] Zvi Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Computing Surveys (CSUR)*, 18(1):23–38, 1986.

[5] Robert W. Irving. An efficient algorithm for the "stable roommates" problem. *Journal of Algorithms*, 6(4):577–595, 1985.

[6] Robert W. Irving and David F. Manlove. The stable roommates problem with ties. *Journal of Algorithms*, 43(1):85–105, 2002.

[7] Vladimir Kolmogorov. Blossom v: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1(1):43–67, 2009.

[8] John D.C. Little, Katta G. Murty, Dura W. Sweeney, and Caroline Karel. An algorithm for the traveling salesman problem. *Operations research*, 11(6):972–989, 1963.

[9] Glennon Doyle Melton. This brilliant math teacher has a formula to save kids' lives. https://www.huffingtonpost.com/glennon-melton/this-brilliant-math-teacher-has-a-formula-to-save-kids-lives_b_4899349.html, 2014. blog.

[10] Daniel R. Saunders. Mathworks file exchange. https://nl.mathworks.com/matlabcentral/fileexchange/42827-weighted-maximum-matching-in-general-graphs.