BACHELOR

Weighted bipartite matching and the assignment of students to highschools in Amsterdam

Klaren, L.M.

Award date:
2016

Link to publication

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
1 Introduction

In Amsterdam the students from the final year of primary school need to be assigned to high schools. In a lot of other cities in the Netherlands students can apply to the school they like and if they meet the requirements than they are let in. But in Amsterdam this is not the case. In Amsterdam there are some schools that are more popular than others and thus have more students applying there then they can handle. So schools with this problem need to turn possible students down, but which students should that be?

To decide which students should be assigned to which schools they use a centralised system. All the students hand in preference lists of the schools they would like to go to and then an assignment is made based on the preference lists of the students and the capacities of the schools. Over time different systems like Boston and Deferred Acceptance have been used. All of the assignment systems have their own flaws and advantages. Some of these systems, like Deferred Acceptance, assume a preference on the side of the schools, but the schools are not allowed to have a preference for which students they want in their classes on the basis of grades. So the preference lists of the schools required for these systems are made by lotteries and one could question what this does for an assignment. But Deferred Acceptance does have the advantage of being ‘strategy proof’.

Boston maximizes the number of students assigned to a school of their first preference. But this also causes others to be assigned to schools low on their preference list.

In this paper I would like to propose an alternative system: the well-known and well solved combinatorial optimization problem called ‘weighted bipartite matching’. To model the Amsterdam highschool assignment problem as such a matching problem and solve it, by techniques such as the Hungarian Method or the simplex method for linear programming, a weight function has to be constructed. In this paper I investigate the possibility to translate certain goals one has for an assignment to a weight function in such a way that existing methods for finding a maximum weight matching will yield in an optimal assignment with respect to the goals.

The structure of this paper is as follows. I will first give a problem definition where I talk about the main elements of this problem, namely the students and the schools, and their properties. I also talk about some properties an assignment should satisfy. Then I look into assignment systems that already have been used in Amsterdam and their advantages and disadvantages. After this I will set a mathematical basis so that I can apply weighted bipartite matching. For the application I will first set up a bipartite graph that corresponds to the highschool assignment system in Amsterdam. Then I discuss the application of regular weighted bipartite matching, and after that the application of weighted bipartite b-matching to the Amsterdam highschool assignment problem. For this application I investigate what suitable weight functions would be and after these are found I do a simulation study. Finally, I compare with the previously used methods and give my conclusion.
2 Amsterdam highschool assignment problem: problem definition

For this problem the main elements are the students and the schools. A school has the following properties:

- Limited capacity. This is the maximal amount of students that can be admitted to the school.
- No preference for students (preference on the basis of grades is illegal in the Netherlands [8]).
- But some schools do give priority to some students. In Amsterdam it is allowed to give priority to [5]:
  - siblings of students already attending this school.
  - students from affiliated schools.
  - children of the personnel.
  - students who chose this school based on their religion or beliefs.
  - students with serious social or medical conditions at play. (Maximum of 2 percent)
- A school offers one or more types of education. Generally Vmbo/Havo/Vwo or subtypes of this, but there are also some special classes called ‘profielklassen’. ¹

And a student has these properties:

- A preference list
  - Contains a ranked list of schools. The school with number 1 is the school with the highest preference.
  - Has a certain length (not all students have a list of the same length).
- Advice from primary school
  - Every student gets an advice about what track level he can follow in high school.
  - In Amsterdam this advice can be something out of the following [5]:
    * Praktijkonderwijs
    * Vmbo-b (± lwoo)
    * Vmbo-b/k (± lwoo)
    * Vmbo-k (± lwoo)
    * Vmbo-t (± lwoo)
    * Vmbo-t/Havo
    * Havo
    * Havo/Vwo
    * Vwo
    * Kopklas ²

¹Anyone could be accepted into these ‘profielklassen’ if they have the right ‘talent’. This will be checked after the assignment is made. If it turns out that the student can not go to this class, then the school offers them a place at one of their regular classes. [5]
²The ‘kopklas’ is an extra year before a student goes to a highschool. After this year the student can go to one of the regular education tracks (Vmbo/Havo/Vwo). [5]
Actually there is a group of students that do not participate in the central assignment system in Amsterdam. These are the students with the advices ‘praktijkonderwijs’ or ‘Kopklas’. [2]

Based on all these properties an assignment of students to the schools has to be made. An assignment is defined as follows:

**Definition 2.1 (Assignment)** Let \( U \) be the set of students and \( V \) be the set of schools. An assignment \( \mu \) (or assignment of students to schools) is a function \( \mu : U \cup V \rightarrow 2^{U \cup V} \), the set of all subsets of \( U \cup V \), such that:

1. \( \mu(u) \subseteq V \) for all students \( u \in U \), where \( |\mu(u)| = 1 \), and
2. \( \mu(v) \subseteq U \) for all schools \( v \in V \), and
3. \( v \in \mu(u) \) if and only if \( u \in \mu(v) \).

If \( \mu(u) = \emptyset \) then we say that student \( u \) is unassigned in \( \mu \). Similarly if \( \mu(v) = \emptyset \) we say that school \( v \) is unmatched in \( \mu \), meaning that there are no students assigned to school \( v \). For simplicity we will write \( \mu(u) = v \) instead of \( \mu(u) = \{v\} \), we will then say that student \( u \) is assigned to school \( v \).

In such an assignment a student must be assigned to a school that offers education at the students level (according to the primary school advice). Every school has a limited capacity, so a school or a certain track at a school can possibly not facilitate all students that would like to go there. In this case some students will need to be assigned to another school, but which students? And to which school do they need to be assigned then?

There are multiple goals one could have for the assignment. One goal is to assign all students to a school. But besides this, you could maximize the number of students assigned to their first preference; or you could minimize the number of students that get assigned outside of their top-X preference list; or one could optimize other criteria derived from the preference lists.
3 Possible Systems

There are several existing systems to make this kind of assignment. Any system that has an assignment as a result, we call an assignment system. These assignment systems all have their own advantages and disadvantages. I will first introduce some concepts, then make some basic assumptions and after that I will explain some of the existing systems and give some of their advantages and disadvantages.

3.1 Properties of assignment systems

When making an assignment one wants to place every student at the school the student wants to go to the most. It is not always possible to do this. But to at least place any student at the school with his highest possible preference it is desirable that one knows what the true preferences are. To give the students an incentive to submit a true preference list, one wants the system to be so called ‘strategy-proof’.

Definition 3.1 (Strategy-proof) We say that an assignment system $\varphi$ is (individually) manipulable if there exists two sets of preference lists $P$ and $P'$, containing the preference lists of all participating students and such that

(i) $P'$ differs from $P$ only in the preference list of one student, say student $x$, and
(ii) when $\varphi$ uses the set $P'$ instead of the set $P$, student $x$ gets assigned to a school that is higher on his preference list as in set $P$.

An assignment system $\varphi$ is strategy-proof if it is not (individually) manipulable. [7]

Because when the assignment system is strategy-proof an individual cannot manipulate the system by changing his preference list to get assigned to a school of higher (true) preference.

It is also desirable that after the assignment is made there are no groups of two or more students that can switch schools with each other and thereby all get assigned to a school of higher preference. So, one wants an assignment system that produces a Pareto efficient assignment.

Definition 3.2 (Pareto efficient) Assume that any student prefers to be assigned to any school on or not on their preference list over not to be assigned at all.

Then we say that an assignment $\mu'$ Pareto dominates an assignment $\mu$ if

(i) there is at least one student that in $\mu'$ is assigned to a school of higher preference than the one he is assigned to in $\mu$, and
(ii) all students are in $\mu'$ assigned to a school of higher or equal preference as in assignment $\mu$.

An assignment $\mu$ is Pareto efficient if there does not exist another assignment $\mu'$ that Pareto dominates $\mu$. [7]

In the case that the schools also have an ordered preference list of the students there is also the concept of stability.

Definition 3.3 (Stable) In an assignment $\mu$ a student $x$ is justifiably envious when

(i) student $x$ is assigned to a school $A$, but has a higher preference for a school $B$, and
(ii) there is a student $y$, assigned to school $B$, who is lower on school $B$’s preference list than student $x$ is.

An assignment $\mu$ is stable when there are no students that are justifiably envious.

There might also be other (dis)advantages, but I will explain those when they come.
3.2 Some existing systems and their (dis)advantages

The first system I will discuss is Boston, then Deferred Acceptance and finally Random Serial Dictatorship.

Boston: For this system, the students will submit an ordered preference list containing their most preferred schools. The schools make a (partially) ordered list of the students based on priority and they have a lottery to make this ordering strict. We can look at this ordered list as a preference list from the schools. At first the students will all be temporarily assigned to the first school on their preference list. When there are more students assigned to a school than its capacity the lowest ranked students will be rejected. All of these students will then go into a second round, where they are assigned to their second-most-preferred school if it still has room. Note that once students are assigned and not rejected, they will remain assigned to that school. If after the second round students are rejected, they will continue to the next school on their list. This process keeps going until no students are rejected anymore. [6]

Advantages:

• This system maximizes the number of students that get assigned to the first school on their list. Note that this might not be their most preferred school, since students might have made the list strategically instead of in accordance with their true preference.

• When one works with rounds where students give just one preference per round, then students often only have to apply to one school since the largest part of the students gets assigned to their first preference. This makes it easier for the students because only if they are rejected in the first round they have to consider other schools. [1]

Disadvantages:

• Not strategy-proof, since there is a clear strategy. Namely if you want to go to a school that is very popular but suspect that your chances of getting in are slim, you can put a less popular school before this school on your list to at least get into a school you somewhat like. And when switching after the assignment is made is allowed, one might place a popular school first on their list. Because when you get assigned to the popular school the chances are high that there is someone assigned to your true first preference that wants to switch with you.

• Not stable.

• Not Pareto efficient.

• This system uses a preference list on the side of the schools even though the schools do not really have a preference for the students. Moreover, they are not allowed one.

• When a student gets rejected in the first round, he might not be able to apply to the school on his second, third etc. preference in the next round, because they are already full in the first round. [1]

Deferred Acceptance (DA): Again, the students will submit an ordered preference list containing their most preferred schools. The schools each make an ordered list of the students based on priority and they have a lottery to make this ordering strict. This lottery can be done separately at every school (multiple tie-breaking; MTB) or can be one centralized lottery (single tie-breaking; STB). The obtained ordered lists can be seen as preference lists from the schools. In this system the students will first be temporarily assigned to the first school on their preference list, where the lowest-ranked students are then rejected. But in this system the students that are not rejected do not get definitively assigned to that school, just tentatively. The rejected students then get temporarily assigned to the next school on their list, where, if the school does not have enough
3 POSSIBLE SYSTEMS

places, the lowest-ranked students out of ALL the students assigned to that school are rejected. So students that stayed assigned in round 1 might still be rejected in round 2. Again, the rejected students continue to the next school on their list. This process keeps going until no students are rejected anymore. [6]

Advantages:

- Strategy-proof.
- Stable, but pertaining to a fake preference list on the schools’ side.
- If the schools do not give priority to some students, then DA-STB is Pareto efficient. ³ Because if the schools do give priority, a student will (most likely) only have priority at a few schools, which thus causes there to be different preference lists at the different schools. So the DA-STB system then actually turns into DA-MTB.
- In the data from Amsterdam in 2015, DA-STB placed about 82.0% of the student population on their first preference. For DA-MTB this was 73.6%. [4]

Disadvantages:

- DA-MTB is not Pareto efficient. When the schools can give priority to some students DA-STB is also not Pareto efficient.
- In the data from Amsterdam in 2015, DA-STB placed about 1.6% of the student population randomly, meaning that they were randomly assigned to a school that is not on their preference list. For DA-MTB this was 0.9%. [4]
- This system uses a preference list (or multiple in the case of MTB) on the side of the schools even though the schools do not really have a preference for the students.

Random Serial Dictatorship (RSD): For this system the students will submit an ordered preference list containing their most preferred schools. Also there is one lottery, that gives a strict ordering of the students. Then, starting with the student that is on the first in the ordering and then working their way down the list, the students are assigned to the school with at least one place open that is the highest on their preference list. This process ends when the last student on the list is assigned.

Advantages:

- Strategy-proof
- Stable
- Pareto efficient
- This system places the students at the school that are highest on their preference list, given that the school still has room left.

Disadvantages:

- The assignment of an individual student depends a lot on where they are in the lottery-based ordering. If they are high on the list they will very likely get in to the school of their first preference, while this chance is a lot smaller when the student is further down in the ordering. [1]
- The chances that a student gets assigned lower on his preference list are higher with RSD than with DA-MTB. Because of this students (in Amsterdam in 2016) were advised to make longer preference lists than they would have with DA-MTB. [1]

³Actually then we have that DA-STB is equal to Random Serial Dictatorship. [4]
3.3 Systems that have been used in Amsterdam

Since 2005, the schools in Amsterdam have used a centralized application and admission system. The system that was used for this is inspired by Boston. [6] In this assignment system they made use of several rounds, where students only had to apply to one school per round. If a student is rejected in round $k$ and the $k + 1$\textsuperscript{th} school on his list (meaning the list they would have made for Boston) has no places left, then he skips the $k + 1$\textsuperscript{th} round. But this system is, just like Boston, not strategy-proof, not stable nor is it Pareto efficient.

So in 2015 another system was proposed. This was Deferred Acceptance with multiple tie breaking.[3] This system was chosen because the odds that a student is assigned to a school that is low on his preference list are small and also because this system is strategy-proof.[1] But this system is also not Pareto efficient, and indeed there were groups of students that could swap places and be at a school of higher preference without anyone getting of worse than before. Since it was not allowed to swap with other students, this generated a lot of fuss amongst the students and parents and there was even a group of parents that filed a law suit about this (which they lost in the end [9]).

Then in 2016 Amsterdam used another variant of Deferred Acceptance to assign their students to highschools. This time it was Deferred Acceptance with single tie breaking.[2] This system was described to the students in de ‘Keuzegids’ [5] and it shows that this assignment system is actually the same as Random Serial Dictatorship. With DA-STB (and RSD) it is expected that compared to the previous year more students are getting at the school of their first preference, but this will also cause more students to get low on their preference list or even that they are not placed on their preference list. When a student cannot be placed at a school on the list, he gets the opportunity to apply again to one of the schools that still have places left. This system is Pareto efficient, which the parents and students found desirable.
4 Mathematical Background

To be able to apply weighted bipartite matching to the Amsterdam highschool assignment problem I will need a few mathematical definitions and theorems. These are stated in this chapter.

4.1 (Bipartite) graph

Definition 4.1 (Graph) A graph is a pair $G = (V, E)$, where $V$ is a finite set and $E$ is a family of unordered pairs from $V$. The elements of $V$ are called vertices. The elements of $E$ are called edges. When $v \in e \in E$, then we say that $v$ and $e$ are incident. We sometimes write an element $e = \{u, v\} \in E$ for short as $e = uv$. An edge $uv$ is said to connect $u$ and $v$. 

There is also a special graph I will use a lot in this report, namely the bipartite graph.

Definition 4.2 (Bipartite graph) A graph $G = (V, E)$ is bipartite if $V$ can be partitioned into two sets $U$ and $W$ (called colourclasses of $G$) such that each edge of $G$ connects an element of $U$ and an element of $W$.

Note that in a bipartite graph there are no edges between vertices in just one of the two colourclasses. An edge is always incident with one vertex in one colourclass and one vertex in the other colourclass.

4.2 Matching and vertex cover

Definition 4.3 (Matching) Given a graph $G = (V, E)$, a matching $M$ in $G$ is a set of edges where no two edges in $M$ are incident with the same vertex. We sometimes use that the incidence function $x$ of a matching $M$ is a function $x : E \rightarrow \{0, 1\}$ such that: $x(e) = 1$ if $e \in M$ and $x(e) = 0$ otherwise. Also, we call a matching in a bipartite graph a bipartite matching. A maximum matching is a matching containing the largest possible number of edges. The matching number $\nu(G)$ is the size of a maximum matching in $G$. \[11\]

Corollary 4.4 Let $U$ and $V$ be two sets. Let there be a bipartite graph $G = (U \cup V, E)$, with $U$ and $V$ the colourclasses of $G$ and some set of edges $E$. Let $M$ be a matching in $G$. Now let $\mu$ be a function $\mu : U \cup V \rightarrow P(U \cup V)$ defined by

$$\mu(u) = v \quad \text{if and only if} \quad uv \in M.$$ 

Then we have that $\mu$ is an assignment of elements of $U$ to elements of $V$.

Definition 4.5 (Weighted matching) Given a graph $G = (V, E)$ and a weight function $w : E \rightarrow \mathbb{R}$. Then a matching $M$ in $G$ is a weighted matching with total weight

$$\sum_{e \in M} w(e).$$

Definition 4.6 (Vertex cover) Given a graph $G = (V, E)$, a vertex cover $W$ is a subset of $V$ such that each edge in $E$ is incident with at least one vertex in $W$. A minimum vertex cover is a vertex cover containing the smallest possible number of vertices. The vertex cover number $\tau(G)$ is the size of a minimum vertex cover of $G$. \[11\]

It is easy to see that for any graph $G$ it holds that $\nu(G) \leq \tau(G)$. This holds because in any matching two distinct edges are incident with distinct vertices. So for any vertex cover there are at least as many vertices needed for a cover as there are edges in a maximum matching. In particular, this holds for any minimum vertex cover. \[10\]
4.3 Duality

Theorem 4.7 (General duality theorem) Let a matrix, a column vector and a row vector be given:

\[
\begin{pmatrix}
A & B & C \\
D & E & F \\
G & H & K
\end{pmatrix},
\begin{pmatrix}
a \\
b \\
c
\end{pmatrix}
\text{ and } \begin{pmatrix}
d & e & f
\end{pmatrix},
\]

where \(A, B, C, D, E, F, G, H, K\) are matrices, \(a, b, c\) are column vectors and \(d, e, f\) are row vectors, all of the appropriate dimensions. Then

\[
\det \begin{pmatrix}
A & B & C \\
D & E & F \\
G & H & K
\end{pmatrix} = 0.
\]

First note that there can not be any columns in \(B\) without the \(i\)-th column has exactly one 1 in the upper part and vice versa. [11]

4.4 Totally unimodular matrices

Definition 4.8 (Totally Unimodular Matrix) A \(n \times m\)-matrix \(M\) is totally unimodular (TU) if \(\det(M') \in \{-1, 0, 1\}\) for all square submatrices \(M'\) of \(M\).

Definition 4.9 (Incidence matrix) For a graph \(G = (V, E)\) the incidence matrix \(M^G\) is a \(V \times E\)-matrix where \(m_{ve} = 1\) if vertex \(v \in V\) is incident with edge \(e \in E\) and \(m_{ve} = 0\) otherwise.

Proposition 4.10 The incidence matrix \(M^G\) of a bipartite graph \(G = (V \cup U, E)\), with \(V\) and \(U\) the colourclasses of \(G\), is TU.

Proof of 4.10 First recall that \(M^G\) is a \((V \cup U) \times E\)-matrix and consists of only zeros and ones. Furthermore, since an edge is only attached to two vertices, we know that every column has exactly two ones and the rest of the entries are zeros. Moreover, because \(M^G\) is bipartite, one of these ones is in the upper part of the column (corresponding to \(V\)) and one is in the lower part (corresponding to \(U\)).

Now suppose there is a square \(k \times k\) submatrix of \(M^G\) with \(\det\) not in the set \(\{-1, 0, 1\}\). Let \(B\) be the smallest such matrix.

First note that there can not be any columns in \(B\) with zero nonzero entries, because then that would be a column with only zeros and thus \(\det(B) = 0\).

Suppose that the \(i\)-th column has exactly one 1 in the \(j\)-th row. Let \(B'\) be the \((k - 1) \times (k - 1)\) matrix that is \(B\) without the \(i\)th column and \(j\)-th row. Then \(\det(B) = \pm \det(B')\), so that \(B'\) is a smaller counterexample than \(B\), so this is a contradiction.

Therefore all columns in \(B\) have exactly two ones and the rest of the entries are zeros. Since \(B\) is a submatrix of \(M^G\), we also have that in every column of \(B\) there is one 1 in the upper part and one 1 in the lower part. Now create a matrix \(B'\) from the matrix \(B\), but with the bottom rows, corresponding to the vertices from \(U\), multiplied by \(-1\). Then we know that \(\det(B') = \pm \det(B)\) with the sign depending on the number of rows that were multiplied by \(-1\). \(B'\) has one 1 and one \(-1\) in every column, so by using row addition we can transform this matrix into one with a column full of zeros. Row addition does not change the determinant and thus we have \(\det(B') = 0\) and therefore \(\det(B) = 0\), which is in contradiction with the assumption that \(\det(B) \notin \{-1, 0, 1\}\).

Therefore there is no square submatrix of \(M^G\) with \(\det \notin \{-1, 0, 1\}\) and thus \(M^G\) is totally unimodular.
Theorem 4.11 Let $A$ be a totally unimodular $m \times n$ matrix and let $b \in \mathbb{Z}^m$. Then the polyhedron $P = \{x|Ax \leq b\}$ is integer. [10]

Corollary 4.12 Let $A$ be a totally unimodular $m \times n$ matrix, let $b \in \mathbb{Z}^m$ and let $c \in \mathbb{Z}^n$. Then both optima in the LP duality equation
\[
\max\{c^T x | Ax \leq b\} = \min\{y^T b | y \geq 0, y^T A = c^T\}
\]
have integer optimum solutions (if the optima are finite). [10]

Corollary 4.13 Let $G = (V,E)$ be a bipartite graph and let $A$ be its $V \times E$ incidence matrix and let $b \in \mathbb{Z}^V$. Also let there be a weight function $w : E \rightarrow \mathbb{Z}^+$. Consider the LP-duality equation
\[
\max\{w^T x | x \geq 0, Ax \leq b\} = \min\{y^T b | y \geq 0, y^T A \geq w^T\}.
\]
Then both these optima are attained by integer $x^*$ and $y^*$. [10]

Proof of 4.13 Note that one can write the constraint $x \geq 0$ as $Ix \geq 0$ and thus as $-Ix \leq 0$. So we can combine the two constraints in the left hand side of the duality equation to one constraint, namely:
\[
\begin{pmatrix} A \\ -I \end{pmatrix} x \leq \begin{pmatrix} b \\ 0 \end{pmatrix}.
\]
$G$ is bipartite, thus by Theorem 4.10 we have that the incidence matrix $A$ is totally unimodular. It is easy to see that $I$ is also totally unimodular, thus the matrix $\begin{pmatrix} A \\ -I \end{pmatrix}$ is also totally unimodular. Then by Corollary 4.12 we have that the optimum $\max\{w^T x | x \geq 0, Ax \leq b\}$ is attained by an integer $x^*$. And because of the duality equation it follows that the optimum $\min\{y^T b | y \geq 0, y^T A \geq w^T\}$ is attained by an integer solution $y^*$.

4.5 Bipartite b-matchings

In this report I will not only use regular matchings, but I will also use b-matchings.

Definition 4.14 (b-matching) Let $G = (V,E)$ be a graph with $V \times E$ incidence matrix $A$. For $b : V \rightarrow \mathbb{Z}^+$, a b-matching is a function $x : E \rightarrow \{0,1\}$ such that for each vertex $v$ of $G$:
\[
x(\delta(v)) \leq b_v,
\]
where $\delta(v)$ is the set of edges incident with $v$ and $x(\delta(v)) = \sum_{e \in \delta(v)} x(e)$. [10]

Note that when $b_v = 1$ for all $v \in V$, then a b-matching $x$ coincides with the incidence function of a regular matching.

The concept of b-matching models the concept of an assignment in a graph-theoretic setting. From now on when I use ‘an assignment $x$’, I mean that I use a b-matching $x$ for a given capacity function $b$.

Corollary 4.15 Let $G = (V,E)$ be a bipartite graph with $V \times E$ incidence matrix $A$. Let $b : V \rightarrow \mathbb{Z}^+$ and $w : E \rightarrow \mathbb{Z}^+$, then the maximum weight b-matching
\[
\max\{w^T x | x \geq 0, Ax \leq b\}
\]
is attained by an integer vector $x^*$.

Proof of 4.15 This follows directly from Corollary 4.13.
4.6 Lexicographic order

Definition 4.16 (Lexicographic order (or alphabetic order)) Let $A$ and $B$ be two sets with orderings $<_A$ and $<_B$ respectively, these can be either total or partial. Then for the cartesian product $A \times B$ we have the following lexicographic order $<$, which is either total or partial depending on the type of orderings $<_A$ and $<_B$:

For $(a_1, b_1), (a_2, b_2) \in A \times B$ we have $(a_1, b_1) < (a_2, b_2)$ if and only if either

1. $a_1 <_A a_2$, or
2. $a_1 = a_2$ and $b_1 <_B b_2$.

Now note that for $(a, b), (c, d) \in A \times B$ we sometimes write $(a, b) > (c, d)$ instead of $(c, d) < (a, b)$.

Note also that the lexicographic order can be extended to cartesian products of any length by applying the definition repetitively, since $A \times B \times C = A \times (B \times C)$.

In this report I will often use the standard lexicographic order $>$ on $(\mathbb{Z})^n_+$ for $n \in \mathbb{N}$. For this I use the existing complete order on $\mathbb{Z}_+$. Thus for $(x_1, \ldots, x_n), (y_1, \ldots, y_n) \in \mathbb{Z}_+^n$ we have that

$$(x_1, \ldots, x_n) > (y_1, \ldots, y_n)$$

if and only if there is a $j \in \{1, \ldots, n\}$ such that $x_j > y_j$ and for all $i$ with $i < j$ we have $x_i = y_i$. 

5 Application of weighted bipartite matching to the Amsterdam highschool assignment problem

In this chapter I will investigate if it is possible to apply regular weighted bipartite matching to this assignment problem. This would have the advantage that there is no need for a ‘fake’ preference list on the side of the schools. Another advantage is that this is a Pareto efficient system, because the system will optimize the assignment on the basis of a specific weight function based on pre-defined goals.

5.1 Setting up the bipartite graph

To apply this method there needs to be a bipartite graph to apply it on. So we first need to set up a bipartite graph in accordance with the input values in this problem.

There are a finite number of schools in Amsterdam all with one or more levels of education. Since students are only allowed to apply to classes of the education level corresponding to the primary school advice and since schools can offer different classes of different levels, we will look at classes instead of schools. Because every school offers a finite number of classes, there is also a finite number of classes. Now let $V = \{v_1, \ldots, v_m\}$ be the set of all the classes, so for example for some $i \neq j$: $v_i$ could represent the ‘Havo’ class of the school ‘Damstede’ and $v_j$ could also represent the school ‘Damstede’, but the class ‘Vwo’. The ordering of this set does not matter.

There are also a finite number (say $n$) of students that will apply to the schools that are participating in the central system. Let $U = \{u_1, \ldots, u_n\}$ be the set of all students, so $u_i$ represents student $i$. The ordering of this set does not matter either.

Now we want to create a bipartite graph $G = (V \cup U, E)$, but what should the set of edges $E$ be? We want the classes to be on one side of the graph and the students on the other, so all edges must be between a class on one side and a student on the other side. There are several options to do this, for example:

1. Take all edges and thus make $G$ a complete graph.
2. Take only the edges when the student has the class in their preference list.
3. Take all edges that offer the level of education that the student can follow in accordance with the primary school advice.

By doing 1 there will be edges present that could never work, since a student with for example ‘Vmbo-t’ advice can not follow education at ‘Vwo’ level. Thus there would be a lot of unallowed edges, so that one will not work for our purpose.

Doing 2 would in most cases not work either, because then a student could make a preference list with just one school on it. If the goal is to try and assign all students to schools on their preference lists, then this student will have to be matched to the one school and that would not be fair to students that hand in longer lists (and thus have more possibilities to be matched). This would mean that there is a clear strategy to manipulate the system, so then the system would not be strategy-proof. Moreover if all students make their list very short, the problem could then easily become unsolvable. But in the case where all students are obliged to make an equally long list, this one could work.

Taking the edges like 3 seems to be reasonable, since for every student edges are taken to all allowed classes. Note that this also includes the schools from the students preference lists, since a student can only put classes on his preference list which he is allowed to go to. Taking any more edges is not necessary and could lead to an inadmissible solution for the same reason as in 1.

For the graph $G = (V \cup U, E)$ I will take the edges as in 3, because this includes all possible allowed assignments of students to schools. We can even allow ourselves the possibility to leave some of the edges out of the graph when a student can for example not go to a certain school due...
to a medical condition. When applying weighted bipartite matching, I will use a weight function that will place as much students as possible on schools on their preference lists. In that way, in a matching there will be as much of the edges as possible in the set of edges as mentioned in 2.

We know that the students can be assigned to at most one class and also that the classes have a limited capacity. So let’s define a capacity function $c : V \cup U \rightarrow \mathbb{N}$. For any $v_i \in V$ I will take $c(v_i)$ to be the capacity of class $v_i$ and for any $u_i \in U$ I will take $c(u_i) = 1$, since a student can be assigned to at most one class.

On the side of the schools there is also the possibility to give priority to some students. I will not pay attention to this property of the problem as we have little information on these priorities.

### 5.2 Assumptions

There are some basic assumptions about the graph $G$ that I can make.

**Assumption 5.1 (Enough capacity)** We have

$$\sum_{i=1}^{m} c(v_i) \geq n,$$

since one can never assign all students otherwise, because then there are more students than places at the classes.

Even more so, for the assignment to be valid all students need to be assigned to a class of their advised level. So therefore we must have that the next assumption holds.

**Assumption 5.2 (Enough capacity per level)** For each level of education all the classes combined have sufficient capacity to place all the students that will follow that level and have the corresponding primary school advice.

Finally for some of the theorems it is a necessary condition that there are no isolated vertices, so I assume the following:

**Assumption 5.3 (No isolated vertices)** Every student is allowed to go to at least one class and for every class there is at least one student that is allowed to go to it.

### 5.3 Applying capacitated weighted bipartite matching

For now I will assume that all students have to be assigned to a class. Let $M^G = (m_{ve})$ be the incidence matrix of $G$. So, $M^G$ is a $(V \cup U) \times E$-matrix where $m_{ve} = 1$ if vertex $v \in V \cup U$ is incident with edge $e \in E$, and $m_{ve} = 0$ if it is not.

Let $w : E \rightarrow \mathbb{R}$ be a weight function on the edges of $E$. This weight function has to show the difference in preference for every edge. For now there is no set weight function, but it does have to satisfy some properties. Namely that the weight will be 0 if the school $j$ is not on the list of student $i$, but the edge between them is in $E$; and that an edge between a class and a student will have a lower weight than another edge when the corresponding school has a lower preference on the students preference list.

Then let $W = (w(e))$, meaning that it is a vector with the weights of all the edges in $E$. Also let $C = (c(v))$ be a vector with all the capacities of the vertices in $V \cup U$. Define a function $q : V \cup U \rightarrow \mathbb{Z}_+$ with $q(v) = 0$ for every class $v \in V$ and $q(u) = 1$ for every student $u \in U$. Now

---

4When there are lowerbounds for the amount of students that can be assigned to a class (quorums), one could change the values of $q$ for the classes $v \in V$ to these lowerbounds instead of the 0-values which are currently used.
let \( Q = (q(v)) \) be a vector with all the values of \( q \) of the vertices in \( V \cup U \). Then we have some objective function \( W^T x \) that we want to maximize while satisfying the constraints

\[
M^G x \leq C \quad \text{and} \quad M^G x \geq Q,
\]

where \( x \in \{0, 1\}^E \) denotes an assignment of students to classes.

The constraint \( M^G x \leq C \) ensures that there are at most \( c(v_i) \) students assigned to class \( v_i \), and that every student \( u_j \) is assigned to at most one class.

The constraint \( M^G x \geq Q \) ensures that every student is assigned to at least one class and that every class has at least 0 students assigned to it (so no negative assignments, since this does not make sense).

Thus we have the following LP:

\[
\max \{ W^T x : \quad M^G x \leq C, \quad M^G x \geq Q, \quad x \in \{0, 1\}^E \}.
\]

This is a MIP, but we can actually get the same solutions with an LP, as can be seen in the next proposition. This is favorable, because an LP has a faster solution method than a MIP.

**Proposition 5.4** The optimum \( \max \{ W^T x : \quad M^G x \leq C, \quad M^G x \geq Q, \quad x \in [0, 1]^E \} \) is attained by an integer \( x^* \).

**Proof** We can rewrite the constraint \( M^G x \geq Q \) to \( -M^G x \leq -Q \). Similarly we rewrite \( x \in [0, 1]^E \) to \( x \geq 0 \) and \( Ix \leq 1 \). Thus we can combine the constraints and get following LP:

\[
\max \{ W^T x : \quad x \geq 0, \quad \begin{pmatrix} M^G \\ -M^G \\ I \end{pmatrix} x \leq \begin{pmatrix} C \\ -Q \\ 1 \end{pmatrix} \}
\]

\( G \) is bipartite, thus we have that \( M^G \) is TU (by Proposition 4.10). The matrix \( \begin{pmatrix} M^G \\ -M^G \\ I \end{pmatrix} \) is also TU, which is shown as follows:

Suppose that there is a \( k \times k \) square submatrix with \( det \notin \{-1, 0, 1\} \). Let \( B \) be the smallest such matrix.

Then \( B \) cannot have rows from \( I \), since then that row contains either only zeros (which gives \( det(B) = 0 \), thus a contradiction) or one 1 and the rest are zeros. In the second case, we take \( B' \) to be the square matrix that is \( B \) without the row and column of the one 1. We have \( det(B) = \pm 1 \cdot det(B') \), thus \( B' \) is a smaller submatrix with \( det \notin \{-1, 0, 1\} \), thus here is a contradiction.

Furthermore we have that \( B \) cannot lie completely in \( M^G \) or \( -M^G \) because these are TU and only have square submatrices with with \( det \in \{-1, 0, 1\} \). \( B \) has at least two nonzero entries in every row and column, because otherwise, by the same argument as before with \( I \), we reach a contradiction. Thus every column contains either two, three or four nonzeros.

If there are four nonzeros then there are two of them in \( M^G \) and two in \( -M^G \), thus we can change this column into one with only zeros by using row addition in the same way we did this in the proof of 4.10. Then we then \( det(B) = 0 \) and we have reached a contradiction.

If there are three nonzeros in a column, then there are two of them in either \( M^G \) or \( -M^G \), thus doing the same row addition as before on these rows, creates a column with just one 1 in it. Then we again have a smaller submatrix \( B' \) with \( det \notin \{-1, 0, 1\} \), by taking \( B \) without the row and column of the 1, thus again a contradiction. Thus every column must contain exactly two nonzeros, and more specifically, one 1 and one -1. Thus by using the row addition trick one again we reach a contradiction.

Therefore there can not be a square submatrix with \( det \notin \{-1, 0, 1\} \), thus \( \begin{pmatrix} M^G \\ -M^G \\ I \end{pmatrix} \) is TU. Now by Corollary 4.13 we get that the optimum is attained by an integer \( x^* \).
5 APPLICATION OF WEIGHTED BIPARTITE MATCHING TO THE AMSTERDAM HIGHSCHOOL ASSIGNMENT PROBLEM

5.4 Reducing the problem to a regular weighted bipartite matching problem

A regular weighted matching contains edges such that no two distinct edges are incident with the same vertex. In the graph \( G \) that would result in a matching were every \( v_i \in V \) has at most one edge incident with it. So, that way we would only be able to assign \( m \) out of the \( n \) students to the \( m \) classes.

To solve this, I’ll make an auxiliary bipartite graph \( G' = (V' \cup U, E') \) where I’ve split up the classes even further into individual vertices, meaning that a vertex \( v_i \in V \) with capacity \( c(v_i) \) gets split up into vertices \( v_{i,1}, v_{i,2}, \ldots, v_{i,c(v_i)} \in V' \). To go along with this, if there was an edge in \( G \) from \( u_k \in U \) to \( v_i \in V \), then there are also edges in \( G' \) between \( u_k \) and \( v_{i,j} \in V' \). For the weight on the edges of \( E' \) I will use another weight function \( w' : E' \to \mathbb{R} \) for which we have that the weight \( w' \) of an edge between vertex \( u_k \) and \( v_{i,j} \) is for all \( j \) equal to the weight \( w \) of the edge between \( u_k \) and \( v_i \), since for the student \( u_k \) it only matters that he is assigned to class \( v_i \) and the specific spot \( j \) at this school is not of interest.

Let \( M^{G'} \) be the incidence matrix of \( G' \). And similarly to what we had with the problem in the previous section, we have \( W' = (w'(e)) \). Furthermore we define the function \( q' : V' \cup U \to \mathbb{Z}_+ \) with \( q'(v_{i,j}) = q(v_i) \) for every \( v_{i,j} \in V' \) and \( q'(u) = q(u) \) for all \( u \in U \). Let \( Q' = (q'(v)) \) be the vector with all the values of \( q' \) of the vertices in \( V' \cup U \). In the graph \( G' \) all vertices have a capacity of 1. So for the objective function it holds that

\[
\max\{W^T x : M^G x \leq C, \quad M^G x \geq Q, \quad x \in \{0,1\}^E\} = \max\{W'^T x : M^{G'} x \leq 1, \quad M^{G'} x \geq Q', \quad x \in \{0,1\}^{E'}\},
\]

where the right hand side is a regular weighted bipartite matching problem. Note that the constraint \( M^G x \leq 1 \) makes sure that every student \( u_k \) is assigned to at most one class and that there is at most one student assigned to every \( v_{i,j} \) at class \( v_i \). Therefore it also ensures that there are at most \( c(v_i) \) students assigned to class \( v_i \), so this constraint corresponds with the previous constraint \( M^G x \leq C \).

It is clear that the constraints \( M^G x \geq Q \) and \( M^{G'} x \geq Q' \) also correspond.

This way of using regular weighted bipartite matching in the highschool assignment problem is a good start, but it still has lots of constraints and a big set of vertices and edges that might cause complications in practice. Besides this, the system only works when all students can be assigned and is infeasible when this is not possible. So I would like to have an application with fewer constraints, vertices and edges and also which leaves more freedom to choose your own goals and does not necessarily require that all students have to be assigned.
6 Application of weighted bipartite b-matching to the Amsterdam highschool assignment problem

For this application I will use the graph $G = (V \cup U, E)$ from Section 5.1, but without the assumptions about capacity (Assumptions 5.1 and 5.2), with the assumption that there are no isolated vertices (Assumption 5.3) and without the obligation to assign all students to a class.

Recall that there are $n$ students, $m$ classes and a capacity function $c : V \cup U \to \mathbb{N}$. Also define $N = \min\{n, \sum_{v \in V} c(v)\}$. Then the following theorem holds:

**Proposition 6.1** The maximum size of a b-matching on $G$ is less or equal to $N$.

**Proof** For any b-matching $M : E \to \{0, 1\}$ it holds for every $v \in V \cup U$ that

$$M(\delta(v)) \leq c(v),$$

where $\delta(v)$ is the set of edges incident with $v$ and $M(\delta(v)) = \sum_{e \in \delta(v)} M(e)$.

By definition, the size of a matching $M$ is equal to $\sum_{e \in E} M(e)$.

Since every edge $e = uv$ is incident with a vertex $u \in U$ and a vertex $v \in V$, we have that

$$\sum_{e \in E} M(e) = \sum_{u \in U} \sum_{e \in \delta(u)} M(e) = \sum_{v \in V} \sum_{e \in \delta(v)} M(e).$$

And thus that

$$\sum_{e \in E} M(e) = \sum_{u \in U} M(\delta(u)) = \sum_{v \in V} M(\delta(v)).$$

Recall that all the students have a capacity of 1, so for every $u \in U$ we have $c(u) = 1$. Thus for every $u \in U$ we also have that $M(\delta(u)) \leq 1$. So

$$\sum_{u \in U} M(\delta(u)) \leq \sum_{u \in U} 1 = n.$$

Also we have that

$$\sum_{v \in V} M(\delta(v)) \leq \sum_{v \in V} c(v).$$

Thus we have for any matching $M$ that both

$$\sum_{e \in E} M(e) \leq n$$

and

$$\sum_{e \in E} M(e) \leq \sum_{v \in V} c(v).$$

So it holds that

$$\sum_{e \in E} M(e) \leq \min\{n, \sum_{v \in V} c(v)\} = N.$$

Now assume all preference lists have the same length (for example: 8) and take $L$ to be equal to this length. Then I can define a preference function $p : E \to \{0, ..., L\}$, where for every edge $e = u_iv_j$ we have that $p(e)$ equals the place of class $v_j$ on the preference list of student $u_i$. So, if class $v_j$ is first on the preference list we have $p(e) = 1$ and when it is last on the list we have $p(e) = L$. When a class is not on the preference list we set $p(e) = 0$. 


6 APPLICATION OF WEIGHTED BIPARTITE B-MATCHING TO THE AMSTERDAM
HIGHSCHOOL ASSIGNMENT PROBLEM

6.1 Finding a suitable weight function for the Amsterdam LP

As mentioned in the problem definition, one could have several goals for an assignment. For now we will use the following goals:

• Goal 1: Assign the maximum amount of students that can be assigned.
• Goal 2: Assign as many students as possible to schools on their preference lists, so assigning them in the top \( L \) of their preference lists.
• Goal 3: Assign as many students as possible in the top \( L - 1 \) of their preference lists.
• ...
• Goal \( L \): Assign as many students as possible in the top 2 of their preference lists.
• Goal \( L + 1 \): Assign as many students as possible in the top 1 of their preference lists.

Note that in none of these goals it matters which students get assigned, just that the maximum number of students is assigned within some criteria.

If our only goal is goal 1, then we can solve the following LP:

\[
\max \{ K_1 x : \ M^G x \leq C, \ Ix \leq 1, \ x \geq 0 \},
\]

where \( C = (c(v)) \), a vector of length \( |V \cup U| \), \( M^G \) is the incidence matrix of \( G \) and \( K_1 = 1^T \).

While if our only goal is goal 2, we have another LP. Note that in goal 2 it does not matter if the students assigned to classes within their top \( L \) are assigned to their first preference or at their last, just that they are within their top \( L \). This leads to the following LP:

\[
\max \{ K_2 x : \ M^G x \leq C, \ Ix \leq 1, \ x \geq 0 \},
\]

where \( K_2 = (k_2(e))^T \) with

\[
k_2(e) = \begin{cases} 1 & \text{if } p(e) \in \{1, ..., L\}, \\ 0 & \text{otherwise}. \end{cases}
\]

So we have that \( K_2 x = \#\text{students assigned in top } L \).

I assume that there is a strict priority ordering amongst the goals. So if we now want to achieve both goal 1 and 2, where the priority lies with goal 1, we need something more than the previous LP’s. We could attain this by using a weight function \( w \) consisting of multiple components. So as follows, we solve an LP of the form

\[
\max \{ A_1 \cdot K_1 x + A_2 \cdot K_2 x : \ M^G x \leq C, \ Ix \leq 1, \ x \geq 0 \},
\]

with \( A_1, A_2 \in \mathbb{R} \). Thus here we use the weight function \( w(x) = A_1 \cdot K_1 x + A_2 \cdot K_2 x \). In this weight function \( A_1 \cdot K_1 \) has to make sure that goal 1 is achieved and \( A_2 \cdot K_2 \) ensures that, while subject to goal 1, goal 2 is achieved \(^5\).

To make this more clear I will use the standard lexicographic order on \( \mathbb{Z}_+ \times \mathbb{Z}_+ \), since for any assignment \( x \) we only look at the following properties to see how good the assignment is:

\[
K_1 x = \#\text{students assigned} \in \mathbb{Z}_+ \quad \text{and} \quad K_2 x = \#\text{students assigned in top } L \in \mathbb{Z}_+,
\]

\(^5\)With this I mean that the maximum amount of students is assigned in accordance with goal 1 and that, while subject to this condition, the maximal number of students is assigned in their top \( L \) in accordance with goal 2. So among all assignments that maximize goal 1, we pick the one that is (closest to) maximizing goal 2.
thus we say that the value of an assignment \(\text{val}(x) := (K_1x, K_2x) =: (x_1, x_2)\), where the most right part is just for easier writing. Now for two assignments \(x, y\) with \(\text{val}(x) = (x_1, x_2)\) and \(\text{val}(y) = (y_1, y_2)\), we say that \(\text{val}(x) > \text{val}(y)\) \((x\) is a better assignment than \(y)\) if and only if either
1. \(x_1 > y_1\), or
2. \(x_1 = y_1\) and \(x_2 > y_2\).

Here an assignment is (one of) the best when goal 1 is achieved and, while subject to goal 1, goal 2 is achieved.

In an assignment \(x\), goal 1 is achieved when there is no assignment \(y\) such that \(y_1 > x_1\). Goal 2, while subject to goal 1, is achieved in \(x\) when there is no assignment \(y\) such that \(y_1 = x_1\) and \(y_2 > x_2\). Thus goal 2, while subject to goal 1, is achieved in \(x\) when there is no assignment \(y\) such that \(\text{val}(y) > \text{val}(x)\).

Example 6.2 Suppose we have four students and three classes, to which all students are allowed to go according to the primary school advice. Also suppose we have the following preference lists of length 2 and capacities:

<table>
<thead>
<tr>
<th>Student</th>
<th>Pref. List</th>
<th>School</th>
<th>Cap.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1 3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2 3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2 1</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Let the assignment \(x\) be

<table>
<thead>
<tr>
<th>Student</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>School assigned</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

and assignment \(y\) be

<table>
<thead>
<tr>
<th>Student</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>School assigned</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Then we have that \(\text{val}(x) = (4, 4)\) and \(\text{val}(y) = (4, 3)\). Thus by the lexicographic order we have that \(\text{val}(x) > \text{val}(y)\). Furthermore we have that \(N = \min\{4, 2+1+2\} = 4\), thus we know that for any assignment \(z\) with \(\text{val}(z) = (z_1, z_2)\) we have \(z_1, z_2 \leq 4\), since \(N = 4\) is an upper bound for the size of a maximal matching (or assignment). So an assignment with value \((4, 4)\) is an assignment of maximum value, thus in this example \(x\) is (one of) the best assignment(s).

Because of the upper bound \(N\) for the size of a maximal matching, I can use the weight function as in the following proposition.

Proposition 6.3 Suppose we have the LP

\[
\max \{w(x) : \text{ } M^G x \leq C, \text{ } Ix \leq 1, \text{ } x \geq 0\}.
\]

Where we have \(w(x) = A_1 \cdot K_1 x + A_2 \cdot K_2 x\). Let \(A_2 = 1\) and \(A_1 = N+1\), then for two assignments \(x\) and \(y\) we have that \(\text{val}(x) > \text{val}(y) \iff w(x) > w(y)\).
Proof Note that for any assignment \( x \) with \( \text{val}(x) = (x_1, x_2) \), we have that \( x_1, x_2 \geq 0 \), since \( K_1, K_2, x \geq 0 \). Note also that \( x_1, x_2 \leq N \), since \( N \) is an upper bound for the size of a maximal matching.

\[ \Rightarrow \text{ : Assume that } \text{val}(x) > \text{val}(y), \text{ then we have either} \]
1. \( x_1 > y_1 \), or
2. \( x_1 = y_1 \) and \( x_2 > y_2 \).

If we have the second case, we have that
\[
(N + 1)x_1 + x_2 = (N + 1)y_1 + x_2 > (N + 1)y_1 + y_2.
\]

So \( w(x) > w(y) \).

In the first case, we have \( x_1 > y_1 \), thus \( x_1 \geq y_1 + 1 \). Then we get that
\[
(N + 1)x_1 + x_2 \geq (N + 1)x_1 \geq (N + 1)(y_1 + 1) = (N + 1)y_1 + N + 1 > (N + 1)y_1 + y_2.
\]

since \( y_2 \leq N \). Thus we have \( w(x) > w(y) \).

\[ \Leftarrow \text{ : Now assume that } w(x) > w(y). \text{ Thus } (N + 1)x_1 + x_2 > (N + 1)y_1 + y_2. \]

If \( x_1 = y_1 \), then we must have that \( x_2 > y_2 \).

Now suppose we have \( x_1 \neq y_1 \). Then we have
\[
(N + 1)x_1 + N \geq (N + 1)x_1 + x_2 > (N + 1)y_1 + y_2 \geq (N + 1)y_1,
\]
so \( (N + 1)x_1 + N > (N + 1)y_1 \). If \( y_1 > x_1 \), then \( y_1 \geq x_1 + 1 \), thus
\[
(N + 1)y_1 \geq (N + 1)(x_1 + 1) = (N + 1)x_1 + N + 1 > (N + 1)x_1 + N,
\]
which is in contradiction with \( (N + 1)x_1 + N > (N + 1)y_1 \). Thus we must have that \( x_1 > y_1 \).

In conclusion, we have that either
1. \( x_1 > y_1 \), or
2. \( x_1 = y_1 \) and \( x_2 > y_2 \).

Which means that we have that \( \text{val}(x) > \text{val}(y) \).

Now I’d like to extend this to the argument to when we want to achieve goals 1, 2 and 3 where the priorities lie in that order. We would then get this LP:

\[
\max \{ A_1 \cdot K_1 x + A_2 \cdot K_2 x + A_3 \cdot K_3 x : M^G x \leq C, \ Ix \leq 1, \ x \geq 0 \},
\]

with \( A_1, A_2, A_3 \in \mathbb{R} \) and where \( K_3 = (k_3(e))^T \) with

\[
k_3(e) = \begin{cases} 
1 & \text{if } p(e) \in \{1,...,L-1\}, \\
0 & \text{otherwise.}
\end{cases}
\]

Thus now we have weight function \( w = A_1 \cdot K_1 x + A_2 \cdot K_2 x + A_3 \cdot K_3 x \). Similarly to before we have that \( A_1 \cdot K_1 \) in the weight function makes sure that goal 1 is achieved, and \( A_2 \cdot K_2 \) achieves goal 2, while subject to goal 1, and finally \( A_3 \cdot K_3 \) ensures that goal 3 is achieved, while subject to both goal 1 and 2.

Now to see how good an assignment \( x \) is, we look at:

\[
K_1 x = \# \text{students assigned} \in \mathbb{Z}_+,
\]

\[
K_2 x = \# \text{students assigned in top L} \in \mathbb{Z}_+ \text{ and}
\]

\[
K_3 x = \# \text{students assigned in top } L - 1 \in \mathbb{Z}_+.
\]
So we can use the standard lexicographic order on $\mathbb{Z}^3_+$, thus we say that the value of an assignment $x$ is $\text{val}(x) := (K_1x, K_2x, K_3x) =: (x_1, x_2, x_3)$. Here we again have that an assignment $x$ is (one of) the best assignment(s) when it has maximum value according to the standard lexicographic order on $\mathbb{Z}^3_+$, so when there is no assignment $y$ with $\text{val}(y) > \text{val}(x)$.

It still holds that $N$ is an upper bound for the size of a maximal matching, thus we can use the weight function as in the next proposition.

**Proposition 6.4** Suppose we have the LP

$$\max\{w(x) : \ M^G x \leq C, \ Ix \leq 1, \ x \geq 0\}. $$

Where we have $w(x) = A_1K_1x + A_2K_2x + A_3K_3x$. Let $A_3 = 1$, $A_2 = N+1$ and $A_1 = (N+1)^2$, then for two assignments $x$ and $y$, we have that

$$\text{val}(x) > \text{val}(y) \iff w(x) > w(y).$$

**Proof** Note that for any assignment $x$ with $\text{val}(x) = (x_1, x_2, x_3)$ we have that $x_1, x_2, x_3 \geq 0$, since $K_1, K_2, K_3, x \geq 0$. Note also that $x_1, x_2, x_3 \leq N$, since $N$ is an upper bound for the size of a maximal matching.

$\Rightarrow$ : Assume that $\text{val}(x) > \text{val}(y)$, then we have either

1. $x_1 > y_1$, or
2. $x_1 = y_1$ and $x_2 > y_2$, or
3. $x_1 = y_1$, $x_2 = y_2$ and $x_3 > y_3$.

If we have the third case, we have that

$$(N+1)^2x_1 + (N+1)x_2 + x_3 = (N+1)^2y_1 + (N+1)y_2 + x_3 > (N+1)^2y_1 + (N+1)y_2 + y_3.$$  

So $w(x) > w(y)$.

In the second case, we have $x_2 > y_2$, thus $x_2 \geq y_2 + 1$. Then we get that

$$w(x) = (N+1)^2x_1 + (N+1)x_2 + x_3$$

$$\geq (N+1)^2x_1 + (N+1)x_2$$

$$\geq (N+1)^2y_1 + (N+1)(y_2 + 1)$$

$$= (N+1)^2y_1 + (N+1)y_2 + N + 1$$

$$> (N+1)^2y_1 + (N+1)y_2 + y_3$$

$$= w(y).$$

In the first case, we have $x_1 > y_1$, thus $x_1 \geq y_1 + 1$. Then we get that

$$w(x) = (N+1)^2x_1 + (N+1)x_2 + x_3$$

$$\geq (N+1)^2x_1$$

$$\geq (N+1)^2(y_1 + 1)$$

$$= (N+1)^2y_1 + (N+1)^2$$

$$= (N+1)^2y_1 + (N+1)y_2 + N + 1$$

$$\geq (N+1)^2y_1 + (N+1)y_2 + y_3$$

$$= w(y).$$

$\Leftarrow$ : Now assume $w(x) > w(y)$. Thus

$$(N+1)^2x_1 + (N+1)x_2 + x_3 = (N+1)^2y_1 + (N+1)y_2 + y_3.$$
6 APPLICATION OF WEIGHTED BIPARTITE B-MATCHING TO THE AMSTERDAM HIGHSCHOOL ASSIGNMENT PROBLEM

If \( x_1 = y_1 \) and \( x_2 = y_2 \), then we must have \( x_3 > y_3 \).
If we have \( x_1 = y_1 \) and \( x_2 \neq y_2 \). From \( w(x) > w(y) \) we get that \( (N+1)x_2 + x_3 > (N+1)y_2 + y_3 \).
Suppose that \( y_2 > x_2 \), then \( y_2 \geq x_2 + 1 \), so:

\[
\begin{align*}
(N+1)y_2 + y_3 & \geq (N+1)y_2 \\
& \geq (N+1)(x_2 + 1) \\
& = (N+1)x_2 + N + 1 \\
& > (N+1)x_2 + x_1,
\end{align*}
\]

thus this gives a contradiction. Therefore it must hold that \( x_2 > y_2 \), since \( x_2 \neq y_2 \).

Now look at the case where \( x_1 \neq y_1 \). Suppose we have \( y_1 > x_1 \), then \( y_1 \geq x_1 + 1 \), so:

\[
\begin{align*}
w(y) &= (N+1)^2y_1 + (N+1)y_2 + y_3 \\
& \geq (N+1)^2y_1 \\
& \geq (N+1)^2(x_1 + 1) \\
& = (N+1)^2x_1 + (N+1)^2 \\
& \geq (N+1)^2x_1 + (N+1)N + 1 \\
& \geq (N+1)^2x_1 + (N+1)x_2 + N + 1 \\
& > (N+1)^2x_1 + (N+1)x_2 + x_3 \\
& = w(x),
\end{align*}
\]

thus this gives a contradiction. Therefore it must hold that \( x_1 > y_1 \), since \( x_1 \neq y_1 \).

In conclusion, we have that either
\( x_1 > y_1 \), or
\( x_1 = y_1 \) and \( x_2 > y_2 \), or
\( x_1 = y_1 \), \( x_2 = y_2 \) and \( x_3 > y_3 \).

Which means that we have \( \text{val}(x) > \text{val}(y) \).

\[\blacksquare\]

6.2 Taking any subset of goals and changing the order of priorities

Now that we have an idea of how we can design a good weight function, we would like also know what weight function to use when we have any subset of goals with any order of priorities.

This could be useful, because a student has to give a strict ordering of classes on their preference list, but he might not really have a strict preference. For example, it might be so that students generally are indifferent about if they get assigned to school \( i - 1 \), \( i \) or \( i + 1 \) of their preference list. With this in mind it might be useful not to have all the goals, but to be able to use only a selection of them. One can easily find a good weight function by using the logic of the previous section.

**Example 6.5** The schoolboard in Amsterdam might find that they want to assign as many students as possible (goal 1), and of those students assign as many as possible to schools on their preference lists (goal 2) and then they want as many as possible of them assigned to their top 3 (goal \( L - 1 \)). To do this, with the logic of the previous section, this leads to the following LP:

\[
\text{max}\{(N+1)^2 \cdot K_1 x + (N+1) \cdot K_2 x + K_{L-1} x : \ M^G x \leq C, \ Ix \leq 1, \ x \geq 0\}.
\]

Moreover when choosing goals one is not limited to the list of goals mentioned previously. With the kind of weight functions we use, one could have any goal with ‘individual weight function’ \( K \) such that \( 0 \leq Kx \leq N \) and \( Kx \in \mathbb{Z}_+ \) holds and then the weight function would still work. With individual weight function I mean that if you were only trying to achieve that individual goal, you
would use weight function \( K \).

So in general we have an ordered set of desired goals

\[ B = \{ b_1, b_2, ..., b_r : b_i \neq b_j \forall i \neq j \}, \]

with priority order \( b_1, b_2, ..., b_r \), meaning that goal \( b_1 \) has the highest priority and goal \( b_r \) has the lowest priority. Also we have that the goals have individual weight functions \( K_1, K_2, ..., K_r \) such that for any assignment \( x \) it holds that \( 0 \leq K_i x \leq N \) and \( K_i x \in \mathbb{Z}_+ \) for all \( i \).

To see how good an assignment is when using the goals in \( B \), we look at

\[ x_i := K_i x \in \mathbb{Z}_+ \quad \text{for} \quad i = 1, ..., r. \]

Thus we can look at the value \( \text{val}(x) = (x_1, x_2, ..., x_r) \) of an assignment \( x \) and judge its value according to the standard lexicographic order on \( \mathbb{Z}_+^r \). Just like in the small examples in the previous section we now have that an assignment \( x \) is (one of) the best assignment(s) when it has maximum value according to the standard lexicographic order on \( \mathbb{Z}_+^r \). So when there does not exist an assignment \( y \) with \( \text{val}(y) > \text{val}(x) \).

**Proposition 6.6** Suppose we have the LP

\[ \text{max} \{ w(x) : M^G x \leq C, \ Ix \leq 1, \ x \geq 0 \}. \]

Where we have \( w(x) = \sum_{i=1}^{r} A_i \cdot K_i x \) with \( K_1, ..., K_r \) the individual weight functions of the goals \( b_1, ..., b_r \) with the property that for all \( i \) we have that \( 0 \leq K_i x \leq N \) and \( K_i x \in \mathbb{Z}_+ \). Let \( A_i = (N+1)^{k-i} \) for all \( i \), then for two assignments \( x \) and \( y \), we have that

\[ \text{val}(x) > \text{val}(y) \iff w(x) > w(y), \]

where \( \text{val}(x) \) is as defined above.

To prove this we use the following lemma.

**Lemma 6.7** For any \( k \in \mathbb{N}_{\geq 2} \) we have

\[ (N+1)^k = \left( \sum_{i=1}^{k-1} (N+1)^{k-i} \cdot N \right) + (N+1). \]

**Proof of 6.7** I will give a proof by induction.

Let \( k = 2 \), then

\[ (N+1)^2 = (N+1) \cdot (N+1) = (N+1) \cdot N + (N+1). \]

This is the induction basis.

Now assume that for a certain \( k \in \mathbb{N}_{\geq 2} \) we have

\[ (N+1)^k = \left( \sum_{i=1}^{k-1} (N+1)^{k-i} \cdot N \right) + (N+1). \] (IH)

Then

\[ (N+1)^{k+1} = (N+1)^k \cdot (N+1) = (N+1)^k \cdot N + (N+1)^k \]
6 APPLICATION OF WEIGHTED BIPARTITE B-MATCHING TO THE AMSTERDAM HIGHSCHOOL ASSIGNMENT PROBLEM

\[ N \cdot \left( \sum_{i=1}^{k-1} (N+1)^{k-i} \cdot N \right) + (N+1) \cdot \left( \sum_{i=1}^{k} (N+1)^{k-i} \cdot N \right) + (N+1) \quad \text{by (IH)} \]

\[ = (N+1) \cdot \left( \sum_{i=1}^{k-1} (N+1)^{k-i} \cdot N \right) + (N+1) \cdot (N+1) \]

\[ = \left( \sum_{i=1}^{k-1} (N+1)^{k+1-i} \cdot N \right) + (N+1) \cdot N + (N+1) \]

\[ = \left( \sum_{i=1}^{k} (N+1)^{k+1-i} \cdot N \right) + (N+1). \]

Thus

\[ (N+1)^{k+1} = \left( \sum_{i=1}^{k} (N+1)^{k+1-i} \cdot N \right) + (N+1). \]

Which finishes our proof by induction.

\[ \square \]

Proof of 6.6 Note that for any assignment \( x \) with \( val(x) = (x_1, x_2, ..., x_r) \) we have that \( x_i \geq 0 \) for all \( i \), since all \( K_i \) and \( x \) are \( \geq 0 \).

\[ \Rightarrow \] Assume that \( val(x) > val(y) \), then there is a \( j \in \{1, ..., r\} \) such that \( x_j > y_j \) and such that for all \( l < j \) we have \( x_l = y_l \). Thus we have that

\[ \sum_{i=1}^{j-1} A_i \cdot x_i = \sum_{i=1}^{j-1} A_i \cdot y_i. \]

Also because \( x_j > y_j \) we have that \( x_j \geq y_j + 1 \), thus:

\[ \sum_{i=j}^{r} A_i \cdot x_i = \sum_{i=j}^{r} (N+1)^{r-i} \cdot x_i \]

\[ \geq (N+1)^{r-j} x_j \]

\[ \geq (N+1)^{r-j} (y_j + 1) \quad \text{since} \ x_j \geq y_j + 1 \]

\[ = (N+1)^{r-j} y_j + (N+1)^{r-j} \]

\[ = (N+1)^{r-j} y_j + \left( \sum_{i=j+1}^{r-1} (N+1)^{r-i} \cdot N \right) + (N+1) \quad \text{by Lemma 6.7} \]

\[ \geq (N+1)^{r-j} y_j + \left( \sum_{i=j+1}^{r-1} (N+1)^{r-i} y_i \right) + (N+1) \quad \text{since} \ y_i \leq N \text{ for all} \ i \]

\[ > (N+1)^{r-j} y_j + \left( \sum_{i=j+1}^{r-1} (N+1)^{r-i} y_i \right) + y_r \quad \text{since} \ y_r \leq N, \text{ thus} \ y_r < N+1 \]

\[ = \sum_{i=j}^{r} (N+1)^{r-i} \cdot y_i \]

\[ = \sum_{i=j}^{r} A_i \cdot y_i. \]
Thus we have that
\[ \sum_{i=j}^{r} A_i \cdot x_i > \sum_{i=j}^{r} A_i \cdot y_i \]
and
\[ \sum_{i=1}^{j-1} A_i \cdot x_i = \sum_{i=1}^{j-1} A_i \cdot y_i. \]
These two equations gives that \( w(x) > w(y). \)

“\( \preceq \)”: Now assume that \( w(x) > w(y). \) Then
\[ \sum_{i=1}^{r} A_i \cdot x_i > \sum_{i=1}^{r} A_i \cdot y_i. \]
So there must be at least one \( j \in \{1,\ldots,r\} \) such that \( x_j > y_j. \) Choose \( j \) the smallest such index.

Now what is left to prove is that for all \( l < j \) it holds that \( x_l = y_l. \) Note that there are no \( l < j \) with \( x_l > y_l, \) since \( j \) already was the smallest index with this property. Now suppose that there is an \( i < j \) with \( x_i < y_i. \) Thus \( y_i \geq x_i + 1 \). Choose \( i \) to be the smallest such index. Then the following holds:
\[
\sum_{l=i}^{r} A_l \cdot y_l = \sum_{l=i}^{r} (N + 1)^{r-l} \cdot y_l \\
\geq (N + 1)^{r-i} \cdot y_i \\
\geq (N + 1)^{r-i} \cdot (x_i + 1) \quad \text{since } y_i \geq x_i + 1 \\
= (N + 1)^{r-i}x_i + (N + 1)^{r-i} \\
= (N + 1)^{r-i}x_i + \left( \sum_{l=i+1}^{r-1} (N + 1)^{r-l} \cdot N \right) + (N + 1) \quad \text{by Lemma 6.7} \\
\geq (N + 1)^{r-i}x_i + \left( \sum_{l=i+1}^{r-1} (N + 1)^{r-l}x_l \right) + (N + 1) \quad \text{since } x_l \leq N \text{ for all } l \\
> (N + 1)^{r-i}x_i + \left( \sum_{l=i+1}^{r-1} (N + 1)^{r-l}x_l \right) + x_r \quad \text{since } x_r \leq N, \text{ thus } x_r < N + 1 \\
= \sum_{l=i}^{r} (N + 1)^{r-l} \cdot x_l \\
= \sum_{l=i}^{r} A_l \cdot x_l.
\]
And since we chose \( i \) to be the smallest index such that \( x_i < y_i \) and \( i < j \), we have \( x_l = y_l \) for all \( l < i. \) So we get that \( w(y) > w(x) \) which is in contradiction with the assumed \( w(x) > w(y). \) Thus there does not exist an \( i < j \) with \( x_i < y_i \) and therefore we have that for all \( l < j \) it holds that \( x_l = y_l. \) So by the standard lexicographic order on \( \mathbb{Z}_r \), we have \( \text{val}(x) > \text{val}(y) \).

Thus this system can, by using a suitable weight function, optimize many criteria based on the students preference lists, but it does have the disadvantage of not being strategy proof.

**Example 6.8** Suppose we have four schools \( \{1,2,3, \text{ and } 4\} \), all with a capacity of 1. Also suppose that we have four students with the following preference lists:

Thus we have that
\[ \sum_{i=j}^{r} A_i \cdot x_i > \sum_{i=j}^{r} A_i \cdot y_i \]
and
\[ \sum_{i=1}^{j-1} A_i \cdot x_i = \sum_{i=1}^{j-1} A_i \cdot y_i. \]
And finally suppose we have the following goals in the following priority order:

1. Assign as many students as possible to their top 1.
2. Assign as many students as possible to their top 3.

In this example we have that the maximum number of students that can be assigned to their top 1 is just one student. If we then want to assign as many students as we can to their top 3, we have to assign student 4 to school 4 to get all four students in their top 3. If actually student 4 changes his list to: 3 1 2, then we can assign him to his new, fake, first preference and thus get two students in the top 1. The system would say that that result is a better assignment (and for the new preference lists even the best) than keeping the assignment as before this manipulation. Thus student 4 has individually manipulated the system into placing him at his true second preference instead of his third. Thus this system is not strategy proof.

However, to make this manipulation, student 4 has to be aware of many details: preference lists of all other students and the goals with their priority order. So in real life it would be hard for a student to manipulate the system.
7 AIMMS model

7.1 Data

I used ten slightly different data sets, that I all created by using information from a technical report about the 2016 allocation by Gautier et al. [2]. The created data is presented in Excel-files. These Excel-files consists of 3 worksheets. The first and second worksheet always contain the same information and the third one is different in each data set. The first worksheet called ‘Schools’ is partially shown in Figure 1. It contains a list of the names of the schools with their classes (130 classes) and an shorter identifier on the vertical axis and the advice types on the horizontal axis. The 12 possible advices are as mentioned in section 2, but without ‘praktijkonderwijs’ and ‘kopklas’. For every class the capacity is given and also there are columns with either a ‘1’ or nothing, which denotes whether a student with the advice in that column is accepted to the class in that row or not. The information on which classes accept which advices came mostly from the technical report of Gautier et al. and was completed by using the ‘Keuzegids’ from the municipality of Amsterdam [5].

<table>
<thead>
<tr>
<th>Name School</th>
<th>Class ID</th>
<th>Capacity</th>
<th>VmboB</th>
<th>VmboBv</th>
<th>VmboBoB</th>
<th>VmboBoBk</th>
<th>VmboBr</th>
<th>VmboBrk</th>
<th>VmboBrk</th>
<th>VmboBtv</th>
<th>VmboBtvK</th>
<th>Havo</th>
<th>HavoVanVwvo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Berlage Lyceum BLT 1</td>
<td>140</td>
<td></td>
<td>1  1 1</td>
<td></td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Berlage Lyceum BLT 2</td>
<td>75</td>
<td>1  1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Berlage Lyceum BLT 3</td>
<td>84</td>
<td></td>
<td>1  1 1</td>
<td></td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bernard Nieuwe BNCS</td>
<td>29</td>
<td></td>
<td>1  1 1</td>
<td></td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bernard Nieuwe BNCS</td>
<td>23</td>
<td></td>
<td>1  1 1</td>
<td></td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bindtmeier Coll IBC</td>
<td>50</td>
<td></td>
<td>1  1 1</td>
<td></td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bindtmeier Coll IBC</td>
<td>40</td>
<td>1  1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bindtmeier Coll IBC</td>
<td>180</td>
<td>1  1 1 1</td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bredero Beroep BB</td>
<td>170</td>
<td>1  1 1 1</td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bredero Maavo - BMS</td>
<td>48</td>
<td>1  1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bredero Maavo - BMT</td>
<td>66</td>
<td></td>
<td>1  1 1</td>
<td></td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bredero Maavo - IAB</td>
<td>122</td>
<td></td>
<td>1  1 1</td>
<td></td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Caland 2</td>
<td>C21</td>
<td>140</td>
<td>1  1 1</td>
<td>1  1</td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Caland 2</td>
<td>C22</td>
<td>100</td>
<td>1  1 1</td>
<td>1  1</td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Caland 2</td>
<td>C23</td>
<td>150</td>
<td>1  1 1</td>
<td>1  1</td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calandlyceum - CLT</td>
<td>56</td>
<td></td>
<td>1  1 1</td>
<td></td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calandlyceum - CLT 1</td>
<td>112</td>
<td></td>
<td>1  1 1</td>
<td></td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calandlyceum - CLT 2</td>
<td>133</td>
<td></td>
<td>1  1 1</td>
<td></td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calandlyceum - CLT 3</td>
<td>38</td>
<td></td>
<td>1  1 1</td>
<td></td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cathrij College</td>
<td>CC1</td>
<td>89</td>
<td>1  1 1 1</td>
<td>1  1</td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cathrij College</td>
<td>CC2</td>
<td>51</td>
<td>1  1 1 1</td>
<td>1  1</td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cartesius 2</td>
<td>CT2</td>
<td>56</td>
<td>1  1 1 1</td>
<td>1  1</td>
<td></td>
<td></td>
<td>1  1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Worksheet ‘Schools’

The second worksheet called ‘StudentsAdvices’ is partially shown in Figure 2. This worksheet contains a list of 7453 anonymous students (no names, just a list of numbers) on the vertical axis and again the advice types on the horizontal axis. The table contains entries ‘1’ or empty, where an entry ‘1’ depicts that the student in that row has the advice in that column and an empty entry means that the student does not have that advice.

The third and final worksheet is called ‘StudentPreferences’ and an example of this worksheet is partially shown in Figure 3. This worksheet contains the list of students on the vertical axis and the list of classes on the horizontal axis. From the data from Gautier et al. [2] I found that the average length of a preference list is 7.72, so I chose to use a fixed length of 8 classes for the preference lists. The table holds the preferences of all the students for the classes. In every row (so for every student) each preference from 1 to 8 is given exactly once. The rest of the table is

---

6Because the students that received a ‘praktijkonderwijs’ or ‘kopklas’ advice did not participate in the central matching. [2].

7I use a fixed length, because, for example when optimizing the goal ‘number of students assigned on their preference list, having a shorter list than other students could be beneficial for the student with a short list and the opposite for the students with longer lists. Also see Section 5.1.
filled up with zeros.

<table>
<thead>
<tr>
<th>Student ID nr.</th>
<th>B6</th>
<th>BLT1</th>
<th>BLT2</th>
<th>BLT3</th>
<th>BNC1</th>
<th>BNC2</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 3: Example of worksheet ‘StudentPreferences’

I generated the preference lists by first calculating (with AIMMS), based on the information in [2], the probability that a student with advice ‘a’ would put a school ‘c’ on rank ‘r’. For rank r = 1, 2, 3 this means preference 1,2, or 3. For rank r = 4 this means that the student chooses school ‘c’ on a preference that is 4 or higher (so on preference 4, 5, 6, 7 or 8). The AIMMS-model calculating these probabilities is provided in the Appendix A.1. From these probabilities I then generated (using Java) ten different instances of preference lists. I created several instances to make sure that any conclusions I draw will be based on several data sets. With just one data set any drawn conclusion could be just a coincidence.

### 7.2 The models

I have implemented the LP from Proposition 6.6 to make assignments using weighted bipartite b-matching. Note that I can indeed use an LP and do not need a MIP, because we have bipartite graph and thus we will find integer solutions when the simplex method is used to find optimal solutions, since this method only looks at corner point optimal solutions. AIMMS uses the solver CPLEX 12.6.2, which is based on the simplex method.

To test out how different goals have an effect on the assignment I used three sets of goals with each there own priority order.
AIMMS MODEL

Priorities of Goals

<table>
<thead>
<tr>
<th>Goals</th>
<th>Set 1</th>
<th>Set 2</th>
<th>Set 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Maximum total nr. of students assigned</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2. Maximum nr. of students assigned to preference list (top 8)</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3. Maximum nr. of students assigned to top 5</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>4. Maximum nr. of students assigned to top 3</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5. Maximum nr. of students assigned to top 2</td>
<td>4</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>6. Maximum nr. of students assigned to top 1</td>
<td>3</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

Where a goal with priority 1 in the table is of the highest priority. When a cell in the table is empty it means that that goal was not used for that set.

For these goals we have the individual weight functions $K_i : E \rightarrow \{0, 1\}$ as follows:

For $i = 2, 3, 4, 5$ and $6$: $K_i = (k_i(e))^\top$ with

$$k_i(e) = \begin{cases} 
1 & \text{if } p(e) \in \text{top } X, \\
0 & \text{otherwise,}
\end{cases}$$

where $p(e)$ (for $e = uv$) denotes the preference of student $u$ for class $v$. Furthermore where for $i = 2, 3, 4, 5, 6$ that $X = 8, 5, 3, 2, 1$ respectively. For $i = 1$ we have $K_1 = 1^\top$. For our data sets we have that min{ total nr. of students, total sum of capacities } = 7453 and since there are 7453 students we know that for any assignment $x$ (of students to classes) it holds that $K_1 \cdot x = 1^\top \cdot x \leq 7453$. Furthermore, since it is obvious that for all $i \neq 1$ we have that $(k_i(e))^\top \cdot x \leq 1^\top \cdot x$, we also have that $K_i \cdot x \leq 7453$. Combined with the fact that all $K_i \cdot x \in \mathbb{Z}^+$ we have that these individual weight functions are allowed when using Proposition 6.6, thus the chosen goals are allowed to be used in this LP.

I implemented this LP in AIMMS as can be seen in Appendix B.1. I also implemented three other assignment systems, which were already described in Section 3.2. These systems are Boston, Deferred Acceptance MTB and Random Serial Dictatorship. I implemented those three systems in Java.

7.3 Results of the AIMMS-model

For every one of the ten data sets I ran all six models (three versions of weighted bipartite b-matching and the three old systems). Then I took the average of the results of all ten data sets and combined these in six tables, tables 1 to 6.

One thing that immediately shows is that no student is placed in a class at random in any of the versions of weighted bipartite b-matching (WBB), nor in DA-MTB. Moreover, for WBB with set 1 and 3 we have that 99.8% of the students is assigned within their top-3 classes. Note that the only students not assigned to a class in their top 3 are students with a ‘Vwo’-advice. For the version with set 2, there is a stunning 100% of the students assigned within their top-3. While for Boston, DA-MTB and RSD this is 96.58%, 98.34% and 96.97%, so none of these can match this result with WBB. This is expected, since WBB optimizes the assignment based on the students preferences, while the previously used systems were used for other reasons (like strategy proofness and Pareto efficiency).

Another thing that comes to eye is that in all three versions of WBB and Boston 86,42% of the students are assigned to the class of their first preference. We know that this is actually the maximum percentage because this is the first priority in set 3 and because this is a known fact from the Boston system. But for set 1 and 2 this was not the first priority, but it has still reached the maximum percentage.

Thus in these examples of WBB we have the advantage of Boston that the maximum amount of students is assigned to their first preference, but not the disadvantage of the many students assigned to schools lower on their preference lists.
### Table 1: Weighted biparte b-matching, set 1: Placement to preference per advice, percentages

<table>
<thead>
<tr>
<th>Advice</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>#random</th>
<th>#not random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vmbo b</td>
<td>1.05</td>
<td>0.25</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>97</td>
</tr>
<tr>
<td>Vmbo b + lwoo</td>
<td>6.17</td>
<td>0.23</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>477</td>
</tr>
<tr>
<td>Vmbo b/k</td>
<td>1.23</td>
<td>0.28</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>113</td>
</tr>
<tr>
<td>Vmbo b/k + lwoo</td>
<td>2.39</td>
<td>0.03</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>181</td>
</tr>
<tr>
<td>Vmbo k</td>
<td>6.84</td>
<td>1.15</td>
<td>0.06</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>600</td>
</tr>
<tr>
<td>Vmbo k + lwoo</td>
<td>3.01</td>
<td>0.06</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>229</td>
</tr>
<tr>
<td>Vmbo t</td>
<td>12.12</td>
<td>1.25</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>997</td>
</tr>
<tr>
<td>Vmbo t + lwoo</td>
<td>0.81</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>60</td>
</tr>
<tr>
<td>Vmbo t/havo</td>
<td>7.48</td>
<td>0.81</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>618</td>
</tr>
<tr>
<td>Havo</td>
<td>11.53</td>
<td>2.14</td>
<td>0.74</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>1074</td>
</tr>
<tr>
<td>Havo/vwo</td>
<td>10.78</td>
<td>2.06</td>
<td>0.64</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>1005</td>
</tr>
<tr>
<td>Vwo</td>
<td>23.01</td>
<td>3.47</td>
<td>0.37</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
<td>0</td>
<td>2002</td>
</tr>
<tr>
<td>Total</td>
<td>86.42</td>
<td>11.75</td>
<td>1.81</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
<td>0</td>
<td>7453</td>
</tr>
</tbody>
</table>

### Table 2: Weighted biparte b-matching, set 2: Placement to preference per advice, percentages

<table>
<thead>
<tr>
<th>Advice</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>#random</th>
<th>#not random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vmbo b</td>
<td>1.03</td>
<td>0.25</td>
<td>0.03</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>97</td>
</tr>
<tr>
<td>Vmbo b + lwoo</td>
<td>6.17</td>
<td>0.20</td>
<td>0.03</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>477</td>
</tr>
<tr>
<td>Vmbo b/k</td>
<td>1.25</td>
<td>0.22</td>
<td>0.05</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>113</td>
</tr>
<tr>
<td>Vmbo b/k + lwoo</td>
<td>2.39</td>
<td>0.03</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>181</td>
</tr>
<tr>
<td>Vmbo k</td>
<td>6.85</td>
<td>0.94</td>
<td>0.26</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>600</td>
</tr>
<tr>
<td>Vmbo k + lwoo</td>
<td>3.00</td>
<td>0.06</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>229</td>
</tr>
<tr>
<td>Vmbo t</td>
<td>12.08</td>
<td>0.97</td>
<td>0.33</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>997</td>
</tr>
<tr>
<td>Vmbo t + lwoo</td>
<td>0.81</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>60</td>
</tr>
<tr>
<td>Vmbo t/havo</td>
<td>7.51</td>
<td>0.54</td>
<td>0.24</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>618</td>
</tr>
<tr>
<td>Havo</td>
<td>11.53</td>
<td>1.67</td>
<td>1.22</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>1074</td>
</tr>
<tr>
<td>Havo/vwo</td>
<td>10.78</td>
<td>1.58</td>
<td>1.13</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>1005</td>
</tr>
<tr>
<td>Vwo</td>
<td>23.02</td>
<td>2.33</td>
<td>1.51</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>2002</td>
</tr>
<tr>
<td>Total</td>
<td>86.42</td>
<td>8.77</td>
<td>4.81</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>7453</td>
</tr>
</tbody>
</table>

### Table 3: Weighted biparte b-matching, set 3: Placement to preference per advice, percentages

<table>
<thead>
<tr>
<th>Advice</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>#random</th>
<th>#not random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vmbo b</td>
<td>1.03</td>
<td>0.27</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>97</td>
</tr>
<tr>
<td>Vmbo b + lwoo</td>
<td>6.18</td>
<td>0.22</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>477</td>
</tr>
<tr>
<td>Vmbo b/k</td>
<td>1.24</td>
<td>0.27</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>113</td>
</tr>
<tr>
<td>Vmbo b/k + lwoo</td>
<td>2.40</td>
<td>0.02</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>181</td>
</tr>
<tr>
<td>Vmbo k</td>
<td>6.80</td>
<td>1.19</td>
<td>0.06</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>600</td>
</tr>
<tr>
<td>Vmbo k + lwoo</td>
<td>3.02</td>
<td>0.05</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>229</td>
</tr>
<tr>
<td>Vmbo t</td>
<td>12.04</td>
<td>1.34</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>997</td>
</tr>
<tr>
<td>Vmbo t + lwoo</td>
<td>0.81</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>60</td>
</tr>
<tr>
<td>Vmbo t/havo</td>
<td>7.57</td>
<td>0.72</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>618</td>
</tr>
<tr>
<td>Havo</td>
<td>11.58</td>
<td>2.13</td>
<td>0.70</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>1074</td>
</tr>
<tr>
<td>Havo/vwo</td>
<td>10.74</td>
<td>2.08</td>
<td>0.67</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>1005</td>
</tr>
<tr>
<td>Vwo</td>
<td>23.01</td>
<td>3.47</td>
<td>0.37</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
<td>0</td>
<td>2002</td>
</tr>
<tr>
<td>Total</td>
<td>86.42</td>
<td>11.75</td>
<td>1.81</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
<td>0</td>
<td>7453</td>
</tr>
</tbody>
</table>
### Table 4: Boston: Placement to preference per advice, percentages

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>#random</th>
<th>#not random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vmbo b</td>
<td>1.06</td>
<td>0.19</td>
<td>0.04</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>97</td>
</tr>
<tr>
<td>Vmbo b + lwoo</td>
<td>6.20</td>
<td>0.17</td>
<td>0.02</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>477</td>
</tr>
<tr>
<td>Vmbo b/k</td>
<td>1.25</td>
<td>0.20</td>
<td>0.05</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>113</td>
</tr>
<tr>
<td>Vmbo b/k + lwoo</td>
<td>2.34</td>
<td>0.08</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>181</td>
</tr>
<tr>
<td>Vmbo k</td>
<td>6.83</td>
<td>0.96</td>
<td>0.21</td>
<td>0.04</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>600</td>
</tr>
<tr>
<td>Vmbo k + lwoo</td>
<td>3.01</td>
<td>0.05</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>229</td>
</tr>
<tr>
<td>Vmbo t</td>
<td>12.14</td>
<td>0.96</td>
<td>0.21</td>
<td>0.06</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>997</td>
</tr>
<tr>
<td>Vmbo t + lwoo</td>
<td>0.81</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>60</td>
</tr>
<tr>
<td>Vmbo t/havo</td>
<td>7.48</td>
<td>0.58</td>
<td>0.15</td>
<td>0.05</td>
<td>0.02</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>618</td>
</tr>
<tr>
<td>Havo</td>
<td>11.52</td>
<td>1.38</td>
<td>0.66</td>
<td>0.39</td>
<td>0.20</td>
<td>0.13</td>
<td>0.07</td>
<td>0.03</td>
<td>2</td>
<td>1072</td>
</tr>
<tr>
<td>Havo/vwo</td>
<td>10.79</td>
<td>1.31</td>
<td>0.63</td>
<td>0.40</td>
<td>0.19</td>
<td>0.08</td>
<td>0.04</td>
<td>0.03</td>
<td>0</td>
<td>1005</td>
</tr>
<tr>
<td>Vwo</td>
<td>22.97</td>
<td>1.46</td>
<td>0.81</td>
<td>0.70</td>
<td>0.36</td>
<td>0.23</td>
<td>0.14</td>
<td>0.09</td>
<td>8</td>
<td>1994</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>86.42</td>
<td>7.35</td>
<td>2.81</td>
<td>1.66</td>
<td>0.78</td>
<td>0.44</td>
<td>0.25</td>
<td>0.16</td>
<td>10</td>
<td>7443</td>
</tr>
</tbody>
</table>

### Table 5: Deferred Acceptance MTB: Placement to preference per advice, percentages

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>#random</th>
<th>#not random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vmbo b</td>
<td>1.05</td>
<td>0.23</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>97</td>
</tr>
<tr>
<td>Vmbo b + lwoo</td>
<td>6.15</td>
<td>0.24</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>477</td>
</tr>
<tr>
<td>Vmbo b/k</td>
<td>1.19</td>
<td>0.30</td>
<td>0.03</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>113</td>
</tr>
<tr>
<td>Vmbo b/k + lwoo</td>
<td>2.34</td>
<td>0.08</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>181</td>
</tr>
<tr>
<td>Vmbo k</td>
<td>6.68</td>
<td>1.28</td>
<td>0.09</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>600</td>
</tr>
<tr>
<td>Vmbo k + lwoo</td>
<td>2.97</td>
<td>0.10</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>229</td>
</tr>
<tr>
<td>Vmbo t</td>
<td>11.74</td>
<td>1.51</td>
<td>0.12</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>997</td>
</tr>
<tr>
<td>Vmbo t + lwoo</td>
<td>0.81</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>60</td>
</tr>
<tr>
<td>Vmbo t/havo</td>
<td>7.18</td>
<td>1.01</td>
<td>0.09</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>618</td>
</tr>
<tr>
<td>Havo</td>
<td>9.68</td>
<td>3.16</td>
<td>1.12</td>
<td>0.37</td>
<td>0.07</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>1074</td>
</tr>
<tr>
<td>Havo/vwo</td>
<td>9.04</td>
<td>3.07</td>
<td>1.00</td>
<td>0.29</td>
<td>0.07</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>1005</td>
</tr>
<tr>
<td>Vwo</td>
<td>18.20</td>
<td>5.98</td>
<td>1.89</td>
<td>0.61</td>
<td>0.14</td>
<td>0.03</td>
<td>0.01</td>
<td>0.00</td>
<td>0</td>
<td>2002</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>77.03</td>
<td>16.95</td>
<td>4.36</td>
<td>1.30</td>
<td>0.28</td>
<td>0.06</td>
<td>0.01</td>
<td>0.00</td>
<td>8</td>
<td>7453</td>
</tr>
</tbody>
</table>

### Table 6: Random Serial Dictatorship: Placement to preference per advice, percentages

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>#random</th>
<th>#not random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vmbo b</td>
<td>1.08</td>
<td>0.20</td>
<td>0.02</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>97</td>
</tr>
<tr>
<td>Vmbo b + lwoo</td>
<td>6.15</td>
<td>0.22</td>
<td>0.02</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>477</td>
</tr>
<tr>
<td>Vmbo b/k</td>
<td>1.21</td>
<td>0.27</td>
<td>0.03</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>113</td>
</tr>
<tr>
<td>Vmbo b/k + lwoo</td>
<td>2.35</td>
<td>0.07</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>181</td>
</tr>
<tr>
<td>Vmbo k</td>
<td>6.74</td>
<td>1.17</td>
<td>0.12</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>600</td>
</tr>
<tr>
<td>Vmbo k + lwoo</td>
<td>2.98</td>
<td>0.09</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>229</td>
</tr>
<tr>
<td>Vmbo t</td>
<td>11.89</td>
<td>1.25</td>
<td>0.19</td>
<td>0.03</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>997</td>
</tr>
<tr>
<td>Vmbo t + lwoo</td>
<td>0.81</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>60</td>
</tr>
<tr>
<td>Vmbo t/havo</td>
<td>7.35</td>
<td>0.78</td>
<td>0.12</td>
<td>0.04</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>618</td>
</tr>
<tr>
<td>Havo</td>
<td>10.73</td>
<td>2.12</td>
<td>0.84</td>
<td>0.39</td>
<td>0.17</td>
<td>0.08</td>
<td>0.04</td>
<td>0.01</td>
<td>2</td>
<td>1073</td>
</tr>
<tr>
<td>Havo/vwo</td>
<td>10.03</td>
<td>2.06</td>
<td>0.78</td>
<td>0.36</td>
<td>0.15</td>
<td>0.06</td>
<td>0.02</td>
<td>0.02</td>
<td>1</td>
<td>1004</td>
</tr>
<tr>
<td>Vwo</td>
<td>21.72</td>
<td>2.49</td>
<td>1.07</td>
<td>0.76</td>
<td>0.40</td>
<td>0.20</td>
<td>0.10</td>
<td>0.06</td>
<td>5</td>
<td>1997</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>83.03</td>
<td>10.73</td>
<td>3.21</td>
<td>1.60</td>
<td>0.74</td>
<td>0.34</td>
<td>0.16</td>
<td>0.09</td>
<td>8</td>
<td>7445</td>
</tr>
</tbody>
</table>
One can see that the versions of WBB score the best or very close to the best (within 0.1% of the best) in every place of the table. Especially in the total scores one can see that WBB seems to perform the best, with which I mean that it places the most students in the top 1, 2 and 3 and has just very few outliers.  

For this simulation study I have chosen merely three sets of goals, but one can imagine that other sets with other goals have a different result. I saw this happening when playing around with the chosen goals and their priority order.

I actually implemented in AIMMS that I could choose as a goal: the maximum nr. of students assigned to their top $X$ for all $X \in \{1, 2, \ldots, 8\}$. I could not apply all these goals at once right away, because this gave a problem too large to solve in AIMMS. But when I started with one of the previously mentioned three sets (or another one for that matter) and then after AIMMS solved the LP I added another goal (that was not used yet) of the next priority and then solved again. When iterating this I could eventually use all goals. Furthermore, because of the overlap in these goals the outliers that were visible with set 1 and 3 could be pushed up to the top 4 or 5. So, this causes me to recommended that when using this system for an assignment you do not stop at your pre-defined set of goals, but look further to see if your assignment can get even better.

---

8With outliers I here mean that there are only a few single students that are assigned to schools with a preference that is not 1, 2 or 3.
8 Conclusion

In this final section I will summarize the advantages and disadvantages of the system with solving a case of weighted bipartite b-matching (WBB) and compare this system with the previously used methods. Then I will give some recommendations for using this system and for possible further research into this system.

A nice property of an optimal solution of a maximum weight bipartite b-matching combined with any weight function that has the property that it increases with increasing preference, is that the matching (or assignment) is Pareto efficient. Thus in this system with such goals there are no groups of two or more students that can by switching classes all get assigned to a class of higher preference. A disadvantage of this system is that it is not strategy proof, but I believe that there is no obvious strategy (like with Boston) to manipulate the system if you do not know all the preferences of all other students and not know if other students will try to manipulate the system as well. The stability property that is applicable to some of the old systems is actually not applicable to WBB, because WBB does not assume a preference from both the students and the classes (or schools).

In the simulation study we found that for the chosen sets of goals we could assign almost all students to a class within their top 3 preferences (at least 99.8%). We also saw that even though a goal might not have the first priority it can still reach its maximum. Overall the simulation results showed that the system with weighted bipartite b-matching assigned students to schools higher on their preference lists than any of the other three systems, which is expected since this is what WBB optimizes using the chosen goals.

Before using this system I recommend that you have a clear image of what you want to achieve in your assignment. You can choose your goals accordingly and create an assignment. If the assignment created is not entirely what you desire, you can add other goals with the next priorities to get the assignment closer to your wishes. But one has to watch out with adding too many goals at once, because AIMMS cannot handle it anymore when the numbers get too big. An alternative is to not integrate the goals into the weight function, but to integrate them as constraints, which is something I have not done in this paper.

So, in conclusion, I think that a system using weighted bipartite b-matching is good alternative system to solve the Amsterdam highschool assignment problem, which seems to not have been considered before. It can optimize an assignment based on the preferences of students and goals that can be set by the user. Unfortunately this assignment system is not strategy proof. But when the goals are chosen appropriately the optimal assignment is Pareto efficient.

There are still a few things that I have not paid attention to. For example the priorities given by schools to some of the students. One could also still look into the preference list length, because I chose to use a fixed length to prevent an obvious manipulation strategy, but maybe it is possible to use variational lengths with a minimum length? It could also be worthwhile to research what an optimal preference list length would be in both the fixed and variational length case. Furthermore the WBB system looks at the number of students that satisfy a certain condition, but not which students. This is important when for example two students have the same preference list and a decision between them has to be made. I have not made any conscious decisions as to which students were chosen, but this was implicitly done by the AIMMS solver. Maybe a lottery would be more fair to make any choices explicit and controllable.
REFERENCES

References


Appendix A  Data generation

A.1 Probability generator

This is the AIMMS-code I used for generating the probabilities that a student with a certain advice would put a certain class on a certain place on their preference list. A big thank you to dr.ir. C.A.J. Hurkens for helping with this.

```aimms
Model Main_ProbabilityGenerator {
    DeclarationSection PreviouslyMadeDeclaration {
        StringParameter Workbookname {
            Definition: "SimulatieData_v3.xlsx";
        }
        Set Students {
            Index: s, s2;
        }
        Set Preferences {
            SubsetOf: Integers;
            Index: p;
            Definition: {
                {1 .. PrefListLength}
            }
        }
        ElementParameter PrefListLength {
            Range: Integers;
            Definition: 8;
        }
        Parameter StudentHasAdvice {
            IndexDomain: (s,a);
        }
        Parameter EdgeAllowed {
            IndexDomain: (s,c);
        }
        //
        DeclarationSection Givens {
            Set Classes {
                Index: c, cc;
            }
            Set Advices {
                Index: a;
            }
            Set Ranks {
                SubsetOf: Integers;
                Index: r;
                Definition: {
                    {1 .. 4}
                }
            }
            Parameter ClassOffersAdvice {
                IndexDomain: (c,a);
                Range: binary;
            }
            Parameter PrefSumWanted {
                IndexDomain: (r,c);
                Range: integer;
                Comment: "Contains the number of students who put class c on rank r";
            }
            Parameter NrStudentsWithAdvice {
                IndexDomain: a;
                Range: integer;
                Definition: sum[s, StudentHasAdvice(s,a)];
            }
            //
            DeclarationSection Helpers {
                Parameter TotalNrStudents {
                    Range: integer;
                }
            }
    }
}
```
A DATA GENERATION

Parameter TotalNrClasses {
  Range: integer;
  Definition: card(Classes);
}

Parameter TotalNrAdvices {
  Range: integer;
  Definition: card(Advices);
}

Parameter Popularity {
  IndexDomain: c;
  Range: integer;
  Definition: sum[r, PrefSumWanted(r,c)];
  Comment: "Contains the total number of students who put class c on their preference list."
}

Parameter NrStudentsThatChoseRank {
  IndexDomain: r;
  Range: integer;
  Definition: sum[c, PrefSumWanted(r,c)];
}

Parameter TotalNrRanksChosen {
  Range: integer;
  Definition: sum[r, NrStudentsThatChoseRank(r)];
}

Parameter AverageListLength {
  Range: nonnegative;
  Definition: TotalNrRanksChosen*1.0/TotalNrStudents;
}

Parameter NrCompatibleClasses {
  IndexDomain: a;
  Range: integer;
  Definition: sum[c, ClassOffersAdvice(c,a)];
}

Parameter NrCompatibleAdvices {
  IndexDomain: c;
  Range: integer;
  Definition: sum[a, ClassOffersAdvice(c,a)];
}

Parameter NrCompatibleACPairs {
  Range: integer;
  Definition: sum[(a,c), ClassOffersAdvice(c,a)];
}

ElementParameter ProfileRepresentative {
  IndexDomain: c;
  Range: Classes;
  Definition: { first [cc | cc <= c AND NrCompatibleAdvices(c) = NrCompatibleAdvices(cc) AND 
  sum(a, abs[ClassOffersAdvice(c,a) - ClassOffersAdvice(cc,a)]=0 ] }
  Comment: "Choose a representative for all classes that offer the exact same advice-profile, 
  meaning all the exact same advice types are offered."
}

Set ProfReps {
  SubsetOf: Classes;
  Index: rp;
  Definition: union(c, {ProfileRepresentative(c)});
}

Parameter NrProfReps {
  Range: integer;
  Definition: card(ProfReps);
}

Parameter NrRepSelected {
  IndexDomain: (r,rp);
  Range: integer;
  Definition: sum[c | ProfileRepresentative(c)=rp, PrefSumWanted(r,c)];
}
A DATA GENERATION

Parameter TotalNrRepSelected {
  IndexDomain: rp;
  Range: integer;
  Definition: sum[r, NrRepSelected(r,rp)];
}

Parameter NrCandidates {
  IndexDomain: rp;
  Range: integer;
  Definition: sum[a | ClassOffersAdvice(rp,a), NrStudentsWithAdvice(a)];
}

Parameter NrFlexCandidates {
  IndexDomain: rp;
  Range: integer;
  Definition: sum[a | ClassOffersAdvice(rp,a) AND NrCompatibleProfileReps(a)>1,
                   NrStudentsWithAdvice(a)];
  Comment: "Number of candidates that are compatible, but also have other options";
}

Parameter NrFlexAdvices {
  IndexDomain: rp;
  Range: integer;
  Definition: sum[a | ClassOffersAdvice(rp,a) AND NrCompatibleProfileReps(a)>1, 1];
}

Parameter NrNonFlexCandidates {
  IndexDomain: rp;
  Range: integer;
  Definition: sum[a | ClassOffersAdvice(rp,a) AND NrCompatibleProfileReps(a)=1,
                   NrStudentsWithAdvice(a)];
}

Parameter NrNonFlexAdvices {
  IndexDomain: rp;
  Range: integer;
  Definition: sum[a | ClassOffersAdvice(rp,a) AND NrCompatibleProfileReps(a)=1, 1];
}

Parameter NrCompatibleProfileReps {
  IndexDomain: a;
  Range: integer;
  Definition: sum[rp, ClassOffersAdvice(rp,a)];
}

Parameter NrCompatibleARPairs {
  Range: integer;
  Definition: sum[(a,rp), ClassOffersAdvice(rp,a)];
}

DeclarationSection PropModel {
  Variable ProbACR {
    IndexDomain: (a,c,r) | ClassOffersAdvice(c,a) AND r <= NrCompatibleClasses(a);
    Range: [0, 1];
    Comment: "Probability that a student with advice a puts class c on rank r on their preference list."
  }

  Variable ProbAC {
    IndexDomain: (a,c) | ClassOffersAdvice(c,a);
    Range: [0, 1];
    Definition: sum[r, ProbACR(a,c,r)];
    Comment: "Probability that a student with advice a puts class c somewhere on their preference list."
  }

  Constraint InitialProb {
    IndexDomain: a;
    Definition: sum[c, ProbACR(a,c,1)]=1;
    Comment: "Everyone chooses a first preference";
  }

  Constraint DecreasingProb {
    IndexDomain: (a,r) | r=2 OR r=3;
    Definition: sum[c, ProbACR(a,c,r) - ProbACR(a,c,r-1)] <= 0;
  }
}
A DATA GENERATION

Constraint ExpectedRankSelection {
  IndexDomain: (c,r);
  Definition : sum[a, NrStudentsWithAdvice(a)*ProbACR(a,c,r)] = PrefSumWanted(r,c);
  Comment: "The expected number of times a rank is chosen for a class is equal to the prefsumwanted from the data.";
}

Constraint ProfileRepDistribution {
  IndexDomain: (a,c) | ClassOffersAdvice(c,a) AND c<>ProfileRepresentative(c);
  Definition : ProbAC(a,c)*Popularity(ProfileRepresentative(c)) = ProbAC(a,ProfileRepresentative(c))*Popularity(c);
  Comment: "ProbAC(a,c)/ProbAC(a,ProfRep(c)) = Pop(c)/Pop(ProfRep(c)), the proportions of the probabilities match with the proportions of the popularity given in the data within the same profile−group.";
}

Variable PositiveDeviation {
  IndexDomain: (a,c,r) | ClassOffersAdvice(c,a) AND NrCompatibleAdvices(c) > 1;
  Range: nonnegative;
}

Variable NegativeDeviation {
  IndexDomain: (a,c,r) | ClassOffersAdvice(c,a) AND NrCompatibleAdvices(c) > 1;
  Range: nonnegative;
}

Constraint DeviationDefinition {
  IndexDomain: (a,c,r) | ClassOffersAdvice(c,a) AND NrCompatibleAdvices(c) > 1;
  Definition : {
    ProbACR(a,c,r) − ProbAC(a,c)*PrefSumWanted(r,c)/Popularity(c) = PositiveDeviation(a,c,r) − NegativeDeviation(a,c,r)
  }
  Comment: "We try to get both these deviation−quantities as close to 0 as possible. Thus getting as close as possible to the following proportional equality: ProbACR(a,c,r)/ProbAC(a,c) = PrefSumwanted(r,c)/Popularity(c).";
}

Variable TotalNominalAbsoluteDeviation {
  Range: free;
  Definition : sum[(a,c,r), NrStudentsWithAdvice(a)*(PositiveDeviation(a,c,r) + NegativeDeviation(a,c,r))];
}

MathematicalProgram CalculateProbsLowestNominalShift {
  Objective: TotalNominalAbsoluteDeviation;
  Direction: minimize;
  Constraints: AllConstraints;
  Variables: AllVariables;
  Type: LP;
}

DeclarationSection TerminationThings {
  StringParameter WritingWorkbookname {
    Definition : "SimulatieDataToWriteOn.xlsx";
  }
  Parameter ProbAC1 {
    IndexDomain: (a,c);
  }
  Parameter ProbAC2 {
    IndexDomain: (a,c);
  }
  Parameter ProbAC3 {
    IndexDomain: (a,c);
  }
  Parameter ProbAC4 {
    IndexDomain: (a,c);
  }
  Procedure MainInitialization {
    Body: {
      ! Read in data:
      Spreadsheet:: SetVisibility (Workbookname, 'off');
    }
  }
}
A DATA GENERATION

Spreadsheet::SetActiveSheet(Workbookname, "Schools");
Spreadsheet::RetrieveSet(Workbookname, Classes, "SchoolRangeV1");
Spreadsheet::RetrieveSet(Workbookname, Advices, "AdviceRangeH1");
Spreadsheet::RetrieveTable(Workbookname, ClassOffersAdvice, "ClassOffersRange", ", "SchoolRangeV1", ", AdviceRangeH1");

Spreadsheet::SetActiveSheet(Workbookname, "StudentsAdvices");
Spreadsheet::RetrieveSet(Workbookname, Students, "StudentRangeV1");
Spreadsheet::RetrieveTable(Workbookname, StudentHasAdvice, "StudentAdviceRange", ", "StudentRangeV1", ", AdviceRangeH2");

Spreadsheet::SetActiveSheet(Workbookname, "StudentPreferences");
Spreadsheet::RetrieveTable(Workbookname, PrefSumWanted, "PrefSumRange", "PrefSumRangeVA ", "ClassRangeH1");

Spreadsheet::CloseWorkbook(Workbookname, 1);

! Initialize EdgeAllowed
for (s in Students, a in Advices) do
  if StudentHasAdvice(s,a) = 1 then
    for (c in Classes | ClassOffersAdvice(c,a) = 1) do
      if ClassOffersAdvice(c,a) = 1 then
        EdgeAllowed(s,c) := 1;
      else
        EdgeAllowed(s,c) := 0;
      endif;
    endfor;
  endif;
endfor;

Procedure MainExecution {
  Body: {
    solve CalculateProbsLowestNominalShift;
  }
}

Procedure MainTermination {
  Body: {
    for (a in Advices, c in Classes) do
      if (ProbACR(a,c,1) > 0) then ProbAC1(a,c) := ProbACR(a,c,1); else ProbAC1(a,c) := −1; endif;
      !The −1 is just as a default, because a 0 does not get printed and this distorts the table.
      if (ProbACR(a,c,2) > 0) then ProbAC2(a,c) := ProbACR(a,c,2); else ProbAC2(a,c) := −1; endif;
      if (ProbACR(a,c,3) > 0) then ProbAC3(a,c) := ProbACR(a,c,3); else ProbAC3(a,c) := −1; endif;
      if (ProbACR(a,c,4) > 0) then ProbAC4(a,c) := ProbACR(a,c,4); else ProbAC4(a,c) := −1; endif;
    endfor;

    Spreadsheet:: SetVisibility (WritingWorkbookname, 'off');
    Spreadsheet::SetActiveSheet(WritingWorkbookname, "Probabilities");
    Spreadsheet::AssignTable(WritingWorkbookname, ProbAC1, "ProbRange1", "AdviceRangeV1", "ClassRangeH2");
    Spreadsheet::AssignTable(WritingWorkbookname, ProbAC2, "ProbRange2", "AdviceRangeV2", "ClassRangeH2");
    Spreadsheet::AssignTable(WritingWorkbookname, ProbAC3, "ProbRange3", "AdviceRangeV3", "ClassRangeH2");
    Spreadsheet::AssignTable(WritingWorkbookname, ProbAC4, "ProbRange4", "AdviceRangeV4", "ClassRangeH2");

    Spreadsheet::CloseWorkbook(WritingWorkbookname, 1);

    return 1;
  }
}
A.2 Preference list generator

The following Java-code does not stand on its own. It uses the same initialisation and helper functions as the code from Appendix B.2. So when the code below is pasted in the Java-code in Appendix B.2 it does work.

```java
// Reads the probabilities for all ranks generated by the Probability generator
float[][] initProbabilities(int nrAdvices, int nrClasses) throws FileNotFoundException {
    Scanner sc = new Scanner(new File("Probabilities.csv"));
    float[][] probabilities = new float[8][nrClasses];
    sc.nextLine(); // headers
    probabilities[0] = scanProbabilities(sc, nrAdvices, nrClasses);
    sc.nextLine(); // empty line
    probabilities[1] = scanProbabilities(sc, nrAdvices, nrClasses);
    sc.nextLine(); // empty line
    probabilities[2] = scanProbabilities(sc, nrAdvices, nrClasses);
    sc.nextLine(); // empty line
        scanProbabilities(sc, nrAdvices, nrClasses);
    return probabilities;
}

// Helps the above function by reading the probabilities for a single rank
float[] scanProbabilities(Scanner sc, int nrAdvices, int nrClasses) {
    float[] probabilities = new float[nrAdvices][nrClasses];
    String line = sc.nextLine();
    String[] chars = line.split(";");
    for (int i = 0; i < nrAdvices; i++) {
        for (int j = 0; j < nrClasses; j++) {
            probabilities[i][j] = Float.parseFloat(chars[j + 2]);
        }
    }
    return probabilities;
}

// Prints the resulting PrefList to a file
static void printPrefLists(int[][] prefLists, String filename) throws FileNotFoundException {
    PrintWriter pr = new PrintWriter(filename);
    for (int i = 0; i < prefLists.length; i++) {
        for (int j = 0; j < prefLists[i].length; j++) {
            pr.print(prefLists[i][j] + ";");
        }
        pr.println();
    }
    pr.close();
}

// Generate prefLists with the probabilities
int[][] calculatePrefListsFromPreferences(int nrStudents, int nrClasses, int nrAdvices) throws FileNotFoundException {
    int[][] prefLists = new int[nrStudents][nrClasses];
    Boolean[][] stHasAdv = initStHasAdv(nrStudents, nrAdvices);
    float[][] probabilities = initProbabilities(nrAdvices, nrClasses);
    // Get advices
    int[] advices = new int[nrStudents];
    for (int i = 0; i < nrStudents; i++) {
        for (int j = 0; j < nrAdvices; j++) {
            if (stHasAdv[i][j]) {
                advices[i] = j;
                break;
            }
        }
    }
    return prefLists;
}
```
A DATA GENERATION

// Sum the probabilities
float [[[ ]] ] probabilitySums = new float[probabilities.length][nrAdvices];
for (int i = 0; i < probabilities.length; i++) {
    for (int j = 0; j < nrAdvices; j++) {
        float sum = 0;
        for (int k = 0; k < nrClasses; k++) {
            sum += probabilities[i][j][k];
        }
        probabilitySums[i][j] = sum;
    }
}

Random rand = new Random();
for (int i = 0; i < nrStudents; i++) {
    for (int j = 0; j < 8; j++) {
        int pref;
do {
            // Create uniform random variable over [0, sum)
            float u = rand.nextFloat() * probabilitySums[j][advices[i]];
            float cumulative = 0;
pref = 0;
            for (int k = 0; k < nrClasses; k++) {
                cumulative += probabilities[j][advices[i]][k];
                if (u < cumulative) {
                    pref = k;
break;
                }
            }
        } while (prefLists[i][pref] != 0);
prefLists[i][pref] = j + 1; // loop is zero-based
    }
}
return prefLists ;

public static void main(String[] args) throws FileNotFoundException {
    int nrStudents = 7453;
    int nrClasses = 130;
    int nrAdvices = 12;

    int [[ ]] ] prefLists = new OldMethods().calculatePrefListsFromPreferences(nrStudents, nrClasses, nrAdvices);
    printPrefLists ( prefLists , "GeneratedPrefLists-10.csv");
}
Appendix B  Simulation codes

B.1 Weighted bipartite b-matching

This is the AIMMS-code I used for finding an assignment by solving an LP model for weighted bipartite b-matching.

```aimms
Model Main_BipartiteBMatching {
  DeclarationSection InputDeclaration {
    StringParameter Workbookname {
      Definition: "SimulatieData−10.xlsx";
      Comment: "Contains the name of the excel−file with the data";
    }
  }
  Set Classes {
    Index: c;
    Comment: "Contains all the classes";
  }
  Set Students {
    Index: s;
    Comment: "Contains all students";
  }
  Set Advices {
    Index: a;
    Comment: "Contains all possible advices";
  }
  Parameter Capacity {
    IndexDomain: c;
    Range: nonnegative;
    Comment: "Contains the capacity of each class c";
  }
  ElementParameter NUpperboundSizeMatching {
    Range: Integers;
    Definition: { Min(card(students), sum[c, Capacity(c)]); }
    Comment: "This is N, a upperbound for the size of a (maximal) b−matching";
  }
  Parameter ClassOffersAdvice {
    IndexDomain: (c,a);
    Range: binary;
    Comment: "Equals 1 when class c offers advice a, and 0 otherwise";
  }
  Parameter StudentHasAdvice {
    IndexDomain: (s,a);
    Range: binary;
    Comment: "Equals 1 if student s has advice a, and 0 otherwise";
  }
  Parameter EdgeAllowed {
    IndexDomain: (s,c);
    Range: binary;
    Comment: "Is 1 when the student s has an advice that is offered in class c, 0 otherwise";
  }
  Set PossiblePreferences {
    SubsetOf: Integers;
    Index: p;
    Definition: { 1 .. LLengthLongestPrefList }
    Comment: "These are the (possible) preferences that all students have given";
  }
  Parameter PreferenceList {
    IndexDomain: (s,c);
    Range: integer;
    Comment: "Contains a the preferencelists as in the excelfile of all students";
  }
  ElementParameter LLengthLongestPrefList {
    Range: Integers;
  }
}
```
B SIMULATION CODES

Definition: 8;
Comment: "All preference lists have length 8";

} }  
DeclarationSection GoalsDeclaration {
Set Goals {
SubsetOf: Integers;
Index: g;
Definition: {
\{1 .. LLengthLongestPrefList+1\}
}
Comment: "I used the goals as described in section 6.1 of my report. Here there is one goal for each top p (where p is a preference) and also the 1st goal is to assign as many students as possible."
}
Parameter PriorityOfGoals {
IndexDomain: g;
Range: integer;
Comment: "This is a user-defined parameter. Contains the priority of each goal g."
}
ElementParameter SizeSetUsedGoals {
Range: Integers;
Definition: sum[ g | PriorityOfGoals(g)>0, 1];
Comment: "Not all declared goals have to be used, in the user interface one can give a priority order. Only when the priority>0 the goal is used. This parameter equals the number of goals used."
}
Parameter KIndividualWeightFunction {
IndexDomain: (g,s,c);
Range: binary;
Comment: "These are the individual weight functions K in the report."
}
Parameter AWeightfunctionPart {
IndexDomain: g;
Definition: Power((NUpperboundSizeMatching + 1),(SizeSetUsedGoals − PriorityOfGoals(g)));  
Comment: "This is the part of the combined weight function. It is A_{g} in the report."
}
}
DeclarationSection ModelDeclaration {
Variable StudentAssignedToClass {
IndexDomain: (s,c) | EdgeAllowed(s,c);
Range: [0, 1];
Comment: "This is the assignment of student s to school c. Note that it is not binary, because this LP has integer solutions due to unimodularity."
}
Variable TotalWeight {
Range: free;
Definition: sum[ (g,s,c) | PriorityOfGoals(g)>0, AWeightfunctionPart(g)*KIndividualWeightFunction(g,s,c)*StudentAssignedToClass(s,c)];
Comment: "This is the total weight of the objective function / weight function."
}
Constraint CapacityConstraint {
IndexDomain: c;
Definition: sum[s, StudentAssignedToClass(s,c)] <= Capacity(c);
Comment: "This says that there can not be more students assigned to class c than there is capacity at class c."
}
Constraint UpperboundConstraint {
IndexDomain: s;
Definition: sum[c, StudentAssignedToClass(s,c)] <=1;
Comment: "This says that a student can be assigned to at most one class."
}
MathematicalProgram MaximizeTotalWeight {
}
Objective: TotalWeight;
Direction: maximize;
Constraints: AllConstraints;
Variables: AllVariables;
Type: LP;

Declaration Section

Output Declaration

Variable NrStudentsAssignedToXthPreference
- IndexDomain: p;
- Range: free;
- Definition: \( \sum [(s,c) \mid \text{Preferencelist}(s,c)=p, \text{StudentAssignedToClass}(s,c)] \);

Variable NrStudentsAssigned
- Range: free;
- Definition: \( \sum [(s,c), \text{StudentAssignedToClass}(s,c)] \);

Variable NrStudentsAssignedOutsidePrefList
- Range: free;
- Definition: NrStudentsAssigned \( - \sum [p, \text{NrStudentsAssignedToXthPreference}(p)] \);

Variable NrStudentsOfAdviceAssignedToXthPreference
- IndexDomain: (a,p);
- Range: free;
- Definition: \( \sum [(s,c) \mid \text{StudentHasAdvice}(s,a) = 1 \text{ AND } \text{Preferencelist}(s,c)=p, \text{StudentAssignedToClass}(s,c)] \);

String Parameter WritingWorkbookname
- Definition: "Result3\_BipartiteBMatching\_10.xlsx";
- Comment: "Contains the name of the excel-file where the results will be written in";

Procedure InitializeFromExcel

Body:
- ! Read in data:
  - Spreadsheet:: SetVisibility (Workbookname, 'off');
  - Spreadsheet:: SetActiveSheet(Workbookname, "Schools");
  - Spreadsheet::RetrieveSet(Workbookname, Classes, "SchoolRangeV1");
  - Spreadsheet::RetrieveParameter(Workbookname, Capacity, "CapacityRange");
  - Spreadsheet::RetrieveSet(Workbookname, Advices, "AdviceRangeH1");
  - Spreadsheet::RetrieveTable(Workbookname, ClassOffersAdvice, "ClassOffersRange", "SchoolRangeV1", "AdviceRangeH1");
  - Spreadsheet:: SetActiveSheet(Workbookname, "StudentsAdvices");
  - Spreadsheet::RetrieveSet(Workbookname, Students, "StudentRangeV1");
  - Spreadsheet::RetrieveTable(Workbookname, StudentHasAdvice, "StudentAdviceRange", "StudentRangeV1", "AdviceRangeH2");
  - Spreadsheet:: SetActiveSheet(Workbookname, "StudentPreferences");
  - Spreadsheet::RetrieveTable(Workbookname, Preferencelist, "PreferenceListRange", "StudentRangeV2", "ClassRangeH1");

Comment: "This reads the data from the excel-file.";

Procedure InitializeKIndividualWeightFunction

Body:
- ! Initialize IndicatorTop
  for (g in Goals, s in Students, c in Classes) do
    if g>1 then
      if Preferencelist(s,c) \( >= 1 \) and Preferencelist(s,c) \( <= \) LLengthLongestPrefList+2-g
        then KIndividualWeightFunction(g,s,c) := 1;
    else

\[44\]
\text{KIndividualWeightFunction}(\text{g},s,c) := 0;  
\text{endif};  
\text{else if } \text{g}=1 \text{ then}  
\text{KIndividualWeightFunction}(\text{g},s,c) := 1  
\text{endif};  
\text{endif};  
\text{endfor};  
\}  
\text{Comment: "This initializes the individual weightfunctions K"};  
\}  
\text{Procedure InitializeEdgeAllowed}  
\{  
\text{Body:}  
\{  
! \text{Initialize EdgeAllowed}  
\text{for } (s \text{ in Students}, a \text{ in Advices}) \text{ do}  
\text{if } \text{StudentHasAdvice}(s,a) = 1 \text{ then}  
\text{for } (c \text{ in Classes } \text{ ClassOffersAdvice}(c,a) = 1 \text{ do}  
\text{if } \text{ClassOffersAdvice}(c,a) = 1 \text{ then}  
\text{EdgeAllowed}(s,c) := 1;  
\text{else } \text{EdgeAllowed}(s,c) := 0;  
\text{endif};  
\text{endif};  
\text{endfor};  
\text{endif};  
\text{endfor};  
\}  
\text{Comment: "This initializes EdgeAllowed"};  
\}  
\text{Procedure MainExecution}  
\{  
\text{Body:}  
\{  
\text{solve MaximizeTotalWeight};  
\}  
\text{Comment: "This solves the LP MaximizeTotalWeight"};  
\}  
\text{Procedure EmptyVariables}  
\{  
\text{Body:}  
\{  
\text{StudentAssignedToClass}(s,c) := 0;  
\text{NrStudentsAssignedToXthPreference}(p) := 0;  
\text{NrStudentsAssigned} := 0;  
\text{NrStudentsAssignedOutsidePrefList} := 0;  
\text{NrStudentsOfAdviceAssignedToXthPreference}(a,p) := 0;  
\}  
\text{Comment: "This is sometimes helpful, to start with a clean sheet."};  
\}  
\text{Procedure MainTermination}  
\{  
\text{Body:}  
\{  
\text{Spreadsheet::SetVisibility (WritingWorkbookname, 'off')};  
\text{Spreadsheet::SetActiveSheet(WritingWorkbookname, "Sheet1")};  
\text{Spreadsheet::AssignValue(WritingWorkbookname, NrStudentsAssigned, "NrStAsRange");  
\text{Spreadsheet::AssignParameter(WritingWorkbookname, NrStudentsAssignedToXthPreference, " NrStAsXPrtRange");  
\text{Spreadsheet::AssignTable(WritingWorkbookname, NrStudentsOfAdviceAssignedToXthPreference, " NrStOfAdAsXPrtRange", "AdviceRangeV1", "PrefRangeH1");  
\text{Spreadsheet::AssignParameter(WritingWorkbookname, PriorityOfGoals, "PrOfGoRange");  
\text{Spreadsheet::CloseWorkbook(WritingWorkbookname, 1 ) ;  
\text{return 1; !To terminate}  
\}  
\text{Comment: "Writes the key results to an excel file"];  
\}  
\}
B.2 Boston, Deferred Acceptance MTB and Random Serial Dictatorship

This is the Java-code I used for finding assignments with Boston, Deferred Acceptance MTB and Random Serial Dictatorship.

```java
import java.util.*;
import java.io.*;

public class OldMethods {

  //INITIALISATION
  //---

  //Setting up the students
  int[] initStudents(int nrStudents) {
    int[] students = new int[nrStudents];
    for (int i = 0; i < students.length; i++) {
      students[i] = i + 1;
    }
    return students;
  }

  //Setting up the advices
  String[] initAdvices(int nrAdvices) {
    String[] advices = new String[nrAdvices];
    try {
      Scanner sc = new Scanner(new File("Advices.txt"));
      int temp = 0;
      while (sc.hasNext()) {
        advices[temp] = sc.next();
        temp++;
      }
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    }
    return advices;
  }

  //Setting up the classes
  String[] initClasses (int nrClasses) {
    String[] classes = new String[nrClasses];
    try {
      Scanner sc = new Scanner(new File("Classes.txt"));
      int temp = 0;
      while (sc.hasNext()) {
        classes[temp] = sc.next();
        temp++;
      }
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    }
    return classes;
  }

  //Reading in the levels offered in every class
  Boolean[][] initOffersLvl (int nrClasses, int nrAdvices) {
    Boolean[][] offersLvl = new Boolean[nrClasses][nrAdvices];
    //---
  }
}
```
try {
    Scanner sc = new Scanner(new File("ClassOffersLevels.txt"));

    for (int i = 0; i < nrClasses; i++) {
        int temp = 0;
        while (sc.hasNext() && temp < nrAdvices) {
            if (sc.nextInt() == 1) {
                offersLvl[i][temp] = true;
            } else {
                offersLvl[i][temp] = false;
            }
            temp++;
        }
        sc.nextLine();
    }
}

// Reading in capacities of every class
int[][] initCap(int nrClasses) {
    int[] cap = new int[nrClasses];

    try {
        Scanner sc = new Scanner(new File("Capacities.txt"));
        int temp = 0;
        while (sc.hasNext()) {
            cap[temp] = sc.nextInt();
            temp++;
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    return cap;
}

// Reading in the advices of every student
Boolean[][] initStHasAdv(int nrStudents, int nrAdvices) {
    Boolean[][] stHasAdv = new Boolean[nrStudents][nrAdvices];

    try {
        Scanner sc = new Scanner(new File("StudentHasAdvice.txt"));
        for (int i = 0; i < nrStudents; i++) {
            int temp = 0;
            while (sc.hasNext() && temp < nrAdvices) {
                if (sc.nextInt() == 1) {
                    stHasAdv[i][temp] = true;
                } else {
                    stHasAdv[i][temp] = false;
                }
                temp++;
            }
            sc.nextLine();
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    return stHasAdv;
}

// Reading in the preferencelists of all students
int [][] initStPrefLists (int nrStudents, int nrClasses, String PrefListFileName) {
    int [][] stPrefLists = new int[nrStudents][nrClasses];
    try {
        Scanner sc = new Scanner(new File(PrefListFileName));
        for (int i = 0; i < nrStudents; i++) {
            String line = sc.nextLine();
            String [] chars = line.split(";");
            for (int j = 0; j < nrClasses; j++) {
                if (j >= chars.length || chars[j].equals("")) {
                    stPrefLists[i][j] = 0;
                } else {
                    stPrefLists[i][j] = Integer.parseInt(chars[j]);
                }
            }
        }
        catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    return stPrefLists ;
}

void randomSerialDictatorship(int nrStudents, int nrClasses, int nrAdvices, String PrefListFileName, String ResultFileName) {
    int [] students = initStudents(nrStudents);
    //String [] advices = initAdvices(nrAdvices);
    //String [] classes = initClasses(nrClasses);
    //Boolean [][] offersLvl = initOffersLvl(nrClasses, nrAdvices);
    int [] cap = initCap(nrClasses);
    Boolean [][] stHasAdv = initStHasAdv(nrStudents, nrAdvices);
    int [][] stPrefLists = initStPrefLists(nrStudents, nrClasses, PrefListFileName);

    //Make a random list of students
    List<Integer> studentList = new ArrayList<Integer>();
    for (int i = 0; i < nrStudents; i++) {
        studentList.add(i, students[i]);
    }
    Collections.shuffle(studentList);
    int [] rdStList = new int[nrStudents];
    for (int i = 0; i < nrStudents; i++) {
        rdStList[i] = studentList.get(i);
    }

    //Find all the preferences
    int [][] ordPrefLists = getOrdPrefLists(stPrefLists, nrStudents, nrClasses);

    //To keep 'score'
    int [] nrStudentsAssignedToXthPreference = new int[8];
    int [] nrStudentsAssignedToClass = new int[nrClasses];
    int [] assignment = new int[nrStudents];

    //Do the assignment
    for (int i = 0; i < rdStList.length; i++) {
        int tempSt = rdStList[i]-1;
    }
if (nrStudentsAssignedToClass[ordPrefLists[tempSt][0]] < cap[ordPrefLists[tempSt][0]]) {
    assignment[tempSt] = ordPrefLists[tempSt][0];
    nrStudentsAssignedToXthPreference[0] += 1;
    nrStudentsAssignedToClass[ordPrefLists[tempSt][0]] += 1;
    //System.out.println("Assigned to first pref: " + tempSt);
} else if (nrStudentsAssignedToClass[ordPrefLists[tempSt][1]] < cap[ordPrefLists[tempSt][1]]) {
    assignment[tempSt] = ordPrefLists[tempSt][1];
    nrStudentsAssignedToXthPreference[1] += 1;
    nrStudentsAssignedToClass[ordPrefLists[tempSt][1]] += 1;
    nrStudentsAssignedToClass[ordPrefLists[tempSt][0]] += 1;
    //System.out.println("Assigned to second pref: " + tempSt);
} else if (nrStudentsAssignedToClass[ordPrefLists[tempSt][2]] < cap[ordPrefLists[tempSt][2]]) {
    assignment[tempSt] = ordPrefLists[tempSt][2];
    nrStudentsAssignedToXthPreference[2] += 1;
    nrStudentsAssignedToClass[ordPrefLists[tempSt][2]] += 1;
    nrStudentsAssignedToClass[ordPrefLists[tempSt][1]] += 1;
    //System.out.println("Assigned to third pref: " + tempSt);
} else if (nrStudentsAssignedToClass[ordPrefLists[tempSt][3]] < cap[ordPrefLists[tempSt][3]]) {
    assignment[tempSt] = ordPrefLists[tempSt][3];
    nrStudentsAssignedToXthPreference[3] += 1;
    nrStudentsAssignedToClass[ordPrefLists[tempSt][3]] += 1;
    nrStudentsAssignedToClass[ordPrefLists[tempSt][2]] += 1;
    //System.out.println("Assigned to fourth pref: " + tempSt);
} else if (nrStudentsAssignedToClass[ordPrefLists[tempSt][4]] < cap[ordPrefLists[tempSt][4]]) {
    assignment[tempSt] = ordPrefLists[tempSt][4];
    nrStudentsAssignedToXthPreference[4] += 1;
    nrStudentsAssignedToClass[ordPrefLists[tempSt][4]] += 1;
    nrStudentsAssignedToClass[ordPrefLists[tempSt][3]] += 1;
    //System.out.println("Assigned to fifth pref: " + tempSt);
} else if (nrStudentsAssignedToClass[ordPrefLists[tempSt][5]] < cap[ordPrefLists[tempSt][5]]) {
    assignment[tempSt] = ordPrefLists[tempSt][5];
    nrStudentsAssignedToXthPreference[5] += 1;
    nrStudentsAssignedToClass[ordPrefLists[tempSt][5]] += 1;
    nrStudentsAssignedToClass[ordPrefLists[tempSt][4]] += 1;
    //System.out.println("Assigned to sixth pref: " + tempSt);
} else if (nrStudentsAssignedToClass[ordPrefLists[tempSt][6]] < cap[ordPrefLists[tempSt][6]]) {
    assignment[tempSt] = ordPrefLists[tempSt][6];
    nrStudentsAssignedToXthPreference[6] += 1;
    nrStudentsAssignedToClass[ordPrefLists[tempSt][6]] += 1;
    nrStudentsAssignedToClass[ordPrefLists[tempSt][5]] += 1;
    //System.out.println("Assigned to seventh pref: " + tempSt);
} else if (nrStudentsAssignedToClass[ordPrefLists[tempSt][7]] < cap[ordPrefLists[tempSt][7]]) {
    assignment[tempSt] = ordPrefLists[tempSt][7];
    nrStudentsAssignedToXthPreference[7] += 1;
    nrStudentsAssignedToClass[ordPrefLists[tempSt][7]] += 1;
    nrStudentsAssignedToClass[ordPrefLists[tempSt][6]] += 1;
    //System.out.println("Assigned to eighth pref: " + tempSt);
} else {
    //System.out.println("Student " + tempSt + " cannot be assigned on its prefList.");
}

//Final scores
int nrStudentsAssigned = 0;
for (int i = 0; i < nrStudentsAssignedToXthPreference.length; i++) {
    nrStudentsAssigned += nrStudentsAssignedToXthPreference[i];
}
int [][] nrStudentsOfAdviceAssignedToXthPreference =
calculateNrStudentsOfAdviceAssignedToXthPreference(stHasAdv, ordPrefLists, nrStudents, nrAdvices, assignment);

//Print scores
printResults(nrStudentsAssignedToXthPreference, nrStudentsAssigned, rdStList,
            nrStudentsOfAdviceAssignedToXthPreference, ResultFileName);

void boston (int nrStudents, int nrClasses, int nrAdvices, String PrefListFileName, String ResultFileName) {
    int [] students = initStudents(nrStudents);
    //String [] advices = initAdvices(nrAdvices);
    //String [] classes = initClasses(nrClasses);
    //Boolean [][] offersLvl = initOffersLvl(nrClasses, nrAdvices);
    int [] cap = initCap(nrClasses);
    Boolean [][] stHasAdv = initStHasAdv(nrStudents, nrAdvices);
    int [] [] stPrefLists = initStPrefLists(nrStudents, nrClasses, PrefListFileName);
}
//Find all the preferences
int [][] ordPrefLists = getOrdPrefLists(stPrefLists, nrStudents, nrClasses);

//To keep 'score'
int [] nrStudentsAssignedToXthPreference = new int[8];
int [] nrStudentsAssignedToClass = new int[nrClasses];
int [] assignment = new int[nrStudents];

//School's lottery/preference list made at random
List<Integer> studentList = new ArrayList<Integer>();
for (int i = 0; i < nrStudents; i++) {
    studentList.add(i, students[i]);
}
Collections.shuffle(studentList);
int [] rdStList = new int[nrStudents];
for (int i = 0; i < nrStudents; i++) {
    rdStList[i] = studentList.get(i);
}
List<Integer> origShuffledStudentList = new ArrayList<Integer>();
origShuffledStudentList.addAll(studentList);

//Do the assignment
int j = 0;
do {
    List<Integer> tempStudentList = new ArrayList<Integer>();
tempStudentList.addAll(studentList);
    for (int i = 0; i < tempStudentList.size(); i++) {
        int tempSt = tempStudentList.get(i) - 1;
        //System.out.print(tempSt + " ");
        if (isCapLeft(nrStudentsAssignedToClass, cap, ordPrefLists[tempSt][j])) {
            assignment[tempSt] = ordPrefLists[tempSt][j];
            nrStudentsAssignedToXthPreference[j] += 1;
            nrStudentsAssignedToClass[ordPrefLists[tempSt][j]] += 1;
            int k = studentList.indexOf(tempStudentList.get(i));
            studentList.remove(k);
        }
    }
    j++;
    //System.out.println();
} while (j < 8 && !studentList.isEmpty());

//Final 'scores'
int nrStudentsAssigned = 0;
for (int i = 0; i < nrStudentsAssignedToXthPreference.length; i++) {
    nrStudentsAssigned += nrStudentsAssignedToXthPreference[i];
}
int [][] nrStudentsOfAdviceAssignedToXthPreference =
calculateNrStudentsOfAdviceAssignedToXthPreference(stHasAdv, ordPrefLists, nrStudents, nrAdvices, assignment);

//Print 'scores'
printResults(nrStudentsAssignedToXthPreference, nrStudentsAssigned, rdStList,
nrStudentsOfAdviceAssignedToXthPreference, ResultFileName);

//Deferred Acceptance MTB

void deferredAcceptanceMTB (int nrStudents, int nrClasses, int nrAdvices, String PrefListFileName, String ResultFileName) {
    //For this method I use an extra class (with capacity nrStudents, which is on every students 9th pref)
after the real last class to which I will assign students when they can't be assigned to their prefList.

```java
int [] students = initStudents(nrStudents);
//String[] advices = initAdvices(nrAdvices);
//String[] classes = initClasses(nrClasses);
//Boolean[][] offersLvl = initOffersLvl(nrClasses, nrAdvices);
int [] tempCap = initCap(nrClasses);
int [] cap = new int[nrClasses + 1];
for (int i = 0; i < cap.length; i++) {
    if (i < tempCap.length) {
        cap[i] = tempCap[i];
    } else {
        cap[i] = nrStudents;
    }
}
Boolean[][] stHasAdv = initStHasAdv(nrStudents, nrAdvices);
int [][] stPrefLists = initStPrefLists(nrStudents, nrClasses, PrefListFileName);
//Find all the preferences
int [][] tempOrdPrefLists = getOrdPrefLists(stPrefLists, nrStudents, nrClasses);
int [][] ordPrefLists = new int[nrStudents][8+1];
for (int i = 0; i < nrStudents; i++) {
    for (int j = 0; j < ordPrefLists[i].length; j++) {
        if (j < tempOrdPrefLists[i].length) {
            ordPrefLists[i][j] = tempOrdPrefLists[i][j];
        } else {
            ordPrefLists[i][j] = nrClasses;
        }
    }
}
//To keep 'score'
int [] nrStudentsAssignedToXthPreference = new int[8+1];
int [] nrStudentsAssignedToClass = new int[nrClasses+1];
int [] assignment = new int[nrStudents];
//School’s lotteries /preference lists made at random
//todo change to MTB
int [][] lotteryLists = new int[nrClasses+1][nrStudents];
for (int c = 0; c < nrClasses+1; c++) {
    List<Integer> studentList = new ArrayList<Integer>();
    for (int i = 0; i < nrStudents; i++) {
        studentList.add(i, students[i]);
    }
    Collections.shuffle(studentList);
    //Assigning it back into lotteryLists
    for (int i = 0; i < nrStudents; i++) {
        lotteryLists[c][i] = studentList.get(i);
    }
} //printMatrix(lotteryLists);
//To keep track of which preference every student is assigned to currently
int [] currentPrefAssigned = new int[nrStudents];
//Do the assignment
//First assign all students to first preference
for (int i = 0; i < nrStudents; i++) {
    assignment[i] = ordPrefLists[i][0];
    currentPrefAssigned[i] = 0;
}
//Then reassign students until the capacities are no longer exceeded
```
List<Integer> tempUnassign = assignStudentsToTempUnassign(cap, lotteryLists, assignment);
for (int i = 0; i < tempUnassign.size(); i++) {
    int tempSt = tempUnassign.get(i);
    currentPrefAssigned[tempSt] += 1;
    //System.out.println("currentPrefAssigned[");
    assignment[tempSt] = ordPrefLists[tempSt][currentPrefAssigned[tempSt]];
    if (currentPrefAssigned[tempSt] == 8) {
        //System.out.println(tempSt + " is assigned in default class");
    }
}
tempUnassign.clear();
}
while (capIsExceededForSomeClass(assignment, cap));

//Final 'scores'
int[] nrStudentsAssignedToXthPreference = calculateNrStudentsAssignedToXthPreference(currentPrefAssigned, 
    nrStudentsAssignedToXthPreference);
int[] nrStudentsAssignedToClass = calculateNrStudentsAssignedToClass(assignment, 
    nrStudentsAssignedToClass);
int[] nrStudentsOfAdviceAssignedToXthPreference = 
    calculateNrStudentsOfAdviceAssignedToXthPreference(stHasAdv, ordPrefLists, 
    nrStudents, nrAdvices, assignment);

int nrStudentsAssigned = 0;
for (int i = 0; i < nrStudentsAssignedToXthPreference.length; i++) {
    nrStudentsAssigned += nrStudentsAssignedToXthPreference[i];
}

//Print 'scores'
printResultsTwo(nrStudentsAssignedToXthPreference, nrStudentsAssigned, 
    nrStudentsOfAdviceAssignedToXthPreference, ResultFileName);

//PRINTERS, CALCULATORS AND OTHER HELPING FUNCTIONS
//---

//Checks if there is still a space left in a class i
Boolean isCapLeft(int[] nrStudentsAssignedToClass, int[] cap, int i) {
    if (nrStudentsAssignedToClass[i] < cap[i]) {
        return true;
    } else {
        return false;
    }
}

//Checks if the capacity is exceeded for some class
Boolean capIsExceededForSomeClass(int[] assignment, int[] cap) {
    int[] tempNrStudentsAssignedToClass = new int[cap.length];
    int[] nrStudentsAssignedToClass = calculateNrStudentsAssignedToClass(assignment, 
        tempNrStudentsAssignedToClass);
    for (int c = 0; c < cap.length; c++) {
        if (nrStudentsAssignedToClass[c] > cap[c]) {
            return true;
        }
    }
    return false;
}

//Calculates nrStudentsAssignedToXthPreference
int[] calculateNrStudentsAssignedToXthPreference(int[] currentPrefAssigned, int[]
    nrStudentsAssignedToXthPreference) {
    for (int i = 0; i < nrStudentsAssignedToXthPreference.length; i++) {
        nrStudentsAssignedToXthPreference[i] = 0;
        for (int j = 0; j < currentPrefAssigned.length; j++) {
            if (currentPrefAssigned[j] == i) {
                nrStudentsAssignedToXthPreference[i] += 1;
            }
        }
    }
    return nrStudentsAssignedToXthPreference;
}
B SIMULATION CODES

//Calculates nrStudentsAssignedToXthPreference (Basically the same as the above calculator.)
int [] calculateNrStudentsAssignedToXthPreference (int [] assignment, int [] nrStudentsAssignedToXthPreference) {
    for (int i = 0; i < nrStudentsAssignedToXthPreference.length; i++) {
        nrStudentsAssignedToXthPreference[i] = 0;
        for (int j = 0; j < assignment.length; j++) {
            if (assignment[j] == i) {
                nrStudentsAssignedToXthPreference[i] += 1;
            }
        }
    }
    return nrStudentsAssignedToXthPreference;
}

//Calculates nrStudentsOfAdviceAssignedToXthPreference
int [][] calculateNrStudentsOfAdviceAssignedToXthPreference(Boolean [][] stHasAdv, int [][] ordPrefLists, int nrStudents, int nrAdvices, int [] assignment) {
    int [][] nrStudentsOfAdviceAssignedToXthPreference = new int[nrAdvices][8];
    for (int i = 0; i < nrAdvices; i++) {
        for (int j = 0; j < 8; j++) {
            for (int s = 0; s < nrStudents; s++) {
                if (stHasAdv[s][i]) {
                    if (ordPrefLists[s][j] == assignment[s]) {
                        nrStudentsOfAdviceAssignedToXthPreference[i][j] += 1;
                    }
                }
            }
        }
    }
    return nrStudentsOfAdviceAssignedToXthPreference;
}

//Finds which students should be assigned to tempUnassign
List<Integer> assignStudentsToTempUnassign (int [] cap, int [][] lotteryLists , int [] assignment) {
    List<Integer> tempUnassign = new ArrayList<Integer>();
    for (int c = 0; c < lotteryLists.length; c++) {
        int tempNr = 0;
        for (int i = 0; i < lotteryLists[c].length; i++) {
            int tempSt = lotteryLists[c][i] - 1;
            if (assignment[tempSt] == c) {
                tempNr += 1;
                if (tempNr > cap[c]) {
                    tempUnassign.add(tempSt);
                }
            }
        }
    }
    return tempUnassign;
}

//To get the index of a certain value in an array
public int getArrayIndex(int [] arr, int value) {
    int k=0;
    for (int i=0;i<arr.length;i++){
        if (arr[i]==value){
            k=i;
            break;
        }
    }
    return k;
}
return k;
}

// To reduce StPrefLists to an ordered PrefList with only the classes that a preference is given to in it
public int[][] getOrdPrefLists(int[][] stPrefLists, int nrStudents, int nrClasses) {
    int[][] ordPrefLists = new int[nrStudents][8];
    for (int i = 0; i < nrStudents; i++) {
        int[] tempArr = getRow(stPrefLists, i, nrClasses);
        for (int j = 0; j < 8; j++) {
            ordPrefLists[i][j] = getArrayIndex(tempArr, j+1);
        }
    }
    return ordPrefLists;
}

// Returns row i of stPrefLists
public int[] getRow(int[][] stPrefLists, int i, int nrClasses) {
    int[] tempArr = new int[nrClasses];
    for (int k = 0; k < nrClasses; k++) {
        tempArr[k] = stPrefLists[i][k];
    }
    return tempArr;
}

// Displays the results in a txt. file
public void printResults(int[] nrStudentsAssignedToXthPreference, int nrStudentsAssigned, int[] rdStList, int[][] nrStudentsOfAdviceAssignedToXthPreference, String ResultFileName) {
    try {
        PrintWriter pr = new PrintWriter(ResultFileName);
        pr.println("This is the rdStList used");
        for (int i=0; i < rdStList.length; i++) {
            pr.print(rdStList[i] + " ");
        }
        for (int i = 0; i < nrStudentsAssignedToXthPreference.length; i++) {
            pr.println("nrStudentsAssignedToXthPreference (" + (i+1) + ") : " + nrStudentsAssignedToXthPreference[i]);
        }
        pr.println("nrStudentsAssigned : " + nrStudentsAssigned);
        for (int r = 0; r < nrStudentsOfAdviceAssignedToXthPreference.length; r++) {
            pr.println("nrStudentsOfAdviceAssignedToXthPreference : a=" + r + ");
            for (int c = 0; c < nrStudentsOfAdviceAssignedToXthPreference[r].length; c++) {
                pr.println("nrStudentsOfAdviceAssignedToXthPreference[" + r + "] : " + nrStudentsOfAdviceAssignedToXthPreference[r][c]);
            }
        }
        pr.println();
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("No such file exists.");
    }
}

// Displays the results without rdStList in a txt. file
public void printResultsTwo(int[] nrStudentsAssignedToXthPreference, int nrStudentsAssigned, int[][] nrStudentsOfAdviceAssignedToXthPreference, String ResultFileName) {
    try {
        PrintWriter pr = new PrintWriter(ResultFileName);
        for (int i = 0; i < nrStudentsAssignedToXthPreference.length; i++) {
            pr.println("nrStudentsAssignedToXthPreference : " + (i+1) + " : " + nrStudentsAssignedToXthPreference[i]);
        }
        pr.println();
    }
}

54
pr. println("nrStudentsAssigned :" + nrStudentsAssigned);
for (int r = 0; r < nrStudentsOfAdviceAssignedToXthPreference.length; r++) {
    pr. println("nrStudentsOfAdviceAssignedToXthPreference: a=" + r + " :" +
    for (int c = 0; c < nrStudentsOfAdviceAssignedToXthPreference[r].length; c++) {
        pr. println(nrStudentsOfAdviceAssignedToXthPreference[r][c] + ";");
    }
    pr. println();
}
pr. close();
}
catch (Exception e) {
    e.printStackTrace();
    System.out.println("No such file exists.");
}

//Displays a 2d array in the console, one line per row.
static void printMatrix(int [][] grid) {
    for (int r=0; r<10; r++) {
        for (int c=0; c<grid[r].length; c++)
            System.out.print(grid[r][c] + " ");
        System.out.println();
    }
}

//Displays things to check if assignment system is working.
static void printArray(int [] array, int [][] ordPrefLists, int [] rdStList, List<Integer> studentList) {
    for (int r=0; r<10;r++) {
        System.out.println("Student " + rdStList[r] + "/" + studentList.get(r) + " to school " + array[r] + ",
        first pref: " + ordPrefLists[r][0] + ");
    }
}

public static void main(String[] args) {
    int nrStudents = 7453;
    int nrClasses = 130;
    int nrAdvices = 12;
    String PrefListFileName = "GeneratedPrefLists−10.csv";
    String ResultFileName1 = "Result(RandomSerialDictatorship−10).txt";
    String ResultFileName2 = "Result(Boston−10).txt";
    String ResultFileName3 = "Result(DeferredAcceptanceMTB−10).txt";
    new OldMethods().randomSerialDictatorship(nrStudents, nrClasses, nrAdvices, PrefListFileName, ResultFileName1);
    new OldMethods().boston(nrStudents, nrClasses, nrAdvices, PrefListFileName, ResultFileName2);
    new OldMethods().deferredAcceptanceMTB(nrStudents, nrClasses, nrAdvices, PrefListFileName, ResultFileName3);
}