Institutionen för datavetenskap
Department of Computer and Information Science

Final thesis

Improved algorithm for weighted matching of employees

by

Emil Olofsson

LIU-IDA/LITH-EX-A--15/063--SE

2015-11-20
Improved algorithm for weighted matching of employees
Final Thesis

Improved algorithm for weighted matching of employees

by

Emil Olofsson

LIU-IDA/LITH-EX-A--15/063--SE

2015-11-20

Supervisor: Roger Ström (Netlight Consulting AB)
Frej Naimi-Akbar (Netlight Consulting AB)
Dag Sonntag (Linköping University)

Examiner: Jose M. Peña (Linköping University)
Improved algorithm for weighted matching of employees
# Table of Contents

Table of Contents ........................................................................................................... 5  
Abstract ......................................................................................................................... 9  
Acknowledgements ....................................................................................................... 9  

## Chapter 1  Introduction .......................................................................................... 11  
1.1  Background ........................................................................................................... 11  
1.1.1  Netlight ........................................................................................................ 12  
1.2  Problem description ............................................................................................ 12  
1.3  Purpose and goal ................................................................................................ 13  
1.4  Methodology ....................................................................................................... 13  
1.4.1  Before-phase .................................................................................................. 13  
1.4.2  During-phase .................................................................................................. 14  
1.4.3  After-phase .................................................................................................... 14  
1.5  Scope .................................................................................................................... 15  
1.6  Contribution ........................................................................................................ 15  
1.7  Outline .................................................................................................................. 15  

## Chapter 2  Theory ................................................................................................. 17  
2.1  Graph theory ....................................................................................................... 17  
2.2  Linear Programming .......................................................................................... 18  
2.2.1  The Simplex method ..................................................................................... 19  
2.2.1.1  Geometric interpretation ......................................................................... 19  
2.2.1.2  Restrictions .............................................................................................. 20  
2.2.2  Integer Programming .................................................................................... 20  
2.3  Recommender Systems ....................................................................................... 20  
2.3.1  Collaborative filtering ................................................................................. 21  
2.3.2  Content-based filtering ............................................................................... 21  
2.3.3  Hybrid Recommender Systems .................................................................. 21  
2.3.4  Problems ........................................................................................................ 21  
2.3.4.1  Cold Start ................................................................................................. 21  
2.3.4.2  Gray Sheep ............................................................................................... 22  
2.3.4.3  Scalability ................................................................................................ 22  
2.4  Data mining ........................................................................................................ 22  
2.4.1  SONAR .......................................................................................................... 23  
2.5  Linear regression ............................................................................................... 23  

## Chapter 3  CareerLight ......................................................................................... 27  
3.1  Roles ..................................................................................................................... 27  
3.2  Administrator workflow ..................................................................................... 27  
3.2.1  Overview ........................................................................................................ 28  
3.2.2  General configuration ................................................................................. 28  
3.2.2.1  Participants ............................................................................................... 29  
3.2.2.2  Wishes ...................................................................................................... 29  
3.2.2.3  Matching .................................................................................................. 30  
3.2.2.4  Evaluation ................................................................................................. 31  
3.2.2.5  Mentor sessions ....................................................................................... 31  
3.2.3  Participants ...................................................................................................... 32  
3.2.4  Wishes ............................................................................................................ 32  
3.2.5  Relationships ................................................................................................. 33  
3.2.6  Matching ....................................................................................................... 34  
3.2.7  Evaluation ..................................................................................................... 34
Chapter 4  Requirements .................................................................................................................. 41

4.1 Problems with the existing system .............................................................................................. 41
  4.1.1 Insufficient input data ............................................................................................................. 41
  4.1.2 No possibility to reiterate the match result ............................................................................. 41
  4.1.3 Questionable relationships ................................................................................................... 42
  4.1.4 Little variation between evaluation rounds ............................................................................ 42
  4.1.5 Platform dependent black-box algorithm ............................................................................ 42
  4.1.6 No support for CCB-matrix feature ..................................................................................... 42

4.2 System requirements .................................................................................................................... 43

4.3 Guiding questions ........................................................................................................................ 43

Chapter 5  Solution alternatives ....................................................................................................... 45

5.1 Discarded solution alternatives .................................................................................................. 45
  5.1.1 Improving quality of input to the matching algorithm ............................................................ 45
  5.1.1.1 SONAR data sources ......................................................................................................... 46
  5.1.1.2 Neural Network based weight aggregator ...................................................................... 47
  5.1.2 Improving the matching algorithm ....................................................................................... 48
  5.1.2.1 Constraint satisfaction framework OptaPlanner .............................................................. 48

5.2 Implemented solution alternatives ............................................................................................. 49
  5.2.1 Improving quality of input to the matching algorithm ............................................................ 49
  5.2.1.1 Reverse wish provider ...................................................................................................... 49
  5.2.1.2 Proof of concept: Yammer provider ............................................................................... 50
  5.2.2 Improving the matching algorithm ........................................................................................ 52
  5.2.2.1 Choco constraint satisfaction solver .............................................................................. 52
  5.2.2.2 CCB-matrix support ........................................................................................................ 55

Chapter 6  Results ............................................................................................................................... 57

6.1 Reverse wish provider .................................................................................................................. 57
  6.2 Yammer provider ........................................................................................................................ 57

6.3 Choco constraint satisfaction solver .......................................................................................... 60
  6.3.1 Administrator evaluation ....................................................................................................... 60
  6.3.2 Participant evaluation .............................................................................................................. 62

6.4 CCB-matrix support ...................................................................................................................... 65

Chapter 7  Discussion and conclusions ............................................................................................... 67

7.1 Discussion ..................................................................................................................................... 67
  7.1.1 Reverse wish provider ............................................................................................................ 67
  7.1.2 Yammer provider .................................................................................................................... 67
  7.1.3 Choco constraint satisfaction solver ..................................................................................... 68
  7.1.4 CCB-matrix support .............................................................................................................. 69

7.2 Answering the guiding questions ................................................................................................ 70
  7.2.1 Has any previous work been done regarding matching of incomplete graphs that could
       provide useful hints on how to address the problem at hand? .................................................. 70
  7.2.2 Is there a Recommender System technique in existing research that can be used to
       complete the input graph? .......................................................................................................... 70
7.2.3 Are there any additional data sources that can be mined to add information to the input graph, and how do different data sources relate to each other in terms of estimating the quality of a matching pair? ................................................................. 70

7.2.4 How can the transparency of the system be increased to satisfy the users’ needs? ........... 71

7.2.5 Are there any available optimization frameworks that are completely platform independent that can be used to replace the current platform dependent solver, and would they introduce any increase in computation time? .................................................................................. 71

7.2.6 Are there any existing scientific methods to measure the performance of a new implementation of a matching system compared to the current matching system? ............. 71

7.3 Conclusions ......................................................................................................................... 72

7.4 Recommendations for future work .................................................................................... 72

7.4.1 Consideration of previous evaluation rounds ................................................................. 72

7.4.2 Increasing transparency ................................................................................................. 73

7.4.3 Fully implement OptaPlanner ....................................................................................... 73

7.4.4 Improve CCB-matrix size assignment .......................................................................... 73

7.4.4.1 Restrict CCB-matrix levels to certain thresholds ......................................................... 73

7.4.4.2 Possibility to specify CCB-matrix size for individual wishes .................................... 73

7.4.5 Implement additional relationship providers ................................................................. 74

References .............................................................................................................................. 75

Index ........................................................................................................................................ 79
Improved algorithm for weighted matching of employees
Abstract

This report gives the reader a detailed description of a computer engineering master thesis work done at the company Netlight Consulting AB. Netlight Consulting AB is a growing IT consulting company based in Stockholm with offices in major cities across Europe. One of their key success factors is their focus on personal and professional development amongst all employees. An essential part of this development program consist of reoccurring evaluation periods, where every employee receives written constructive feedback from some of their co-workers. This thesis’ focus lies in improving the algorithm that organizes which employee should evaluate who. The original algorithm turned out to harbor a number of flaws, e.g. it was not always able to deliver a satisfactory matching where every participant received the minimum number of evaluations.

In this thesis a new matching algorithm has been implemented that is platform independent and that facilitates future modifications with accessible source code written in Java. The input data for the matching algorithm, i.e. the set of all potential evaluation pairs, is of importance to obtain satisfactory matching results. The number of potential evaluation pairs determines the number of possible matching combinations, which in turn increases the probability to find a satisfactory matching. In this thesis the input data has been extended by utilizing a data mining technique known as SONAR. Two different data mining sources were evaluated, and one of them is shown to extend the number of potential evaluation pairs in the matching input by 20%. Finally, a new feature to support assignment of different evaluation sizes was added to the matching algorithm.

Acknowledgements

During the course of this thesis I received a lot of help that saved hours of work and that has been invaluable for the completion of this thesis.

Dag Sonntag – LiU supervisor
Frej Naimi-Akbar – Netlight supervisor
Roger Ström – Netlight supervisor
Jose M. Peña – LiU examiner
Mårten Sjöstrand, Viktor Pirard, Daniel Lundmark, Peter Rosberg – Architects and developers of the existing system, Netlight
Victor Robertsson, Hakim Rudels – System administrators, Netlight

Linköping, May 2015

Emil Olofsson
Improved algorithm for weighted matching of employees
Chapter 1

Introduction

This chapter starts with an introduction to the background of the field Recommender Systems and its use within the IT consultant industry. Next section gives a description of the IT consultant company Netlight Consulting AB, at which this thesis was implemented, followed by a description of the problem that needed to be solved. Then the purpose of this thesis report is covered, followed by a description of the project methodology used. The next sections gives a description of what delimitations have been made and a discussion on how this thesis project contributes to the field of study. Finally the outline of this report is provided.

1.1 Background

Since the emergence of the first web-based social network services in the 90’s web developers, retailers and researchers have made attempts to benefit from the information they provide. Being able to deduce what products a certain user might be interested in by studying the preferences of their closest peers has a great marketing potential, and this has led to the rise of a field of study called Recommender Systems. Such systems are in essence algorithms that utilizes available user data to figure out a number of additional entities that might fit a user’s preference profile and could therefore be recommended. In recent years, some companies have taken up an interest in reversing the recommendation process, attempting to recreate a network of social connections from looking at users habits and common interests. For example when it comes to team building, knowledge of the social network within the company is a big advantage.

A major trend in companies engaged in the IT consultant industry is to let employees evaluate each other to obtain material for professional development programs and salary reviews. It is often the case that in many consulting firms a handful of employees within the human resources department are responsible for hundreds or thousands of consultants. In these cases it is essential that consultants who work together evaluate each other since the human resources personnel cannot possibly acquire enough data about every single consultant. Implementing an evaluation process in which consultants evaluate each other require some knowledge about each individuals social network within the company in order to assign relevant evaluation pairings. For this reason, the reversed Recommender System becomes an interesting topic as it has the potential to find personal relationships that are suitable for evaluation pairing. Records of managers, previous teams and assignments of the consultants are often stored in databases within the company and can be used as data sources for a reversed Recommender System.

In this thesis, the input to a reversed Recommender System has been improved by evaluating additional data sources that were assumed to contain relevant social information. The purpose of the reversed Recommender System is to create a social network to be used as input for an algorithm that produces a set of evaluation pairs. In this report, the utility that acts as a reversed Recommender System is referred to as the relationship method, and the pairing algorithm as the matching algorithm.
1.1.1 Netlight

The company behind this thesis project is an IT consultant company based in Stockholm called Netlight Consulting AB. With over 500 employees and offices in five countries and a yearly turnover of 400M SEK, Netlight is considered a large business. The company was founded in 1999 and has grown fast into what it is today, mainly during the past decade.

Netlight has a routine to perform annual evaluation rounds in which all employees give feedback to each other in the form of written evaluations. These evaluations are an important basis for personal development and setting salaries within the company, and they are therefore carried out twice every year. In order to keep all employees as happy as possible, it is important that the evaluation workload is evenly distributed among all participants and that each participant gets matched with someone who actually knows them. Currently there are about 400 participants in each evaluation round, each of them having their own personal contacts and preferences. To match which employees should evaluate which manually would be a uninspiring and time consuming job, and thus a system has been designed to perform the task automatically. This system is called CareerLight, and has been used since 2013. CareerLight hosts the relationship method and the matching algorithm that lie in focus of this thesis.

1.2 Problem description

The initial description of this thesis project presented by Netlight was vague in terms of what tasks needed to be done. The description presented two general tasks, the first was to evaluate the quality and performance of their existing matching algorithm and to look for and evaluate alternative matching algorithm suggestions. The second task was to aggregate data from different systems in order to provide better information for the algorithm to work with. The thesis description also provided a set of keywords that indicated what areas of expertise that were assessed as relevant; algorithm, development, machine-learning, database and back-end.

An initial study of the problem reveal ed that since the launch of CareerLight in 2013, many consultants had complained about the poor quality of the matching of their evaluation partners. Many argued that there were much better alternatives for evaluation partners available, and that they should have been matched with them instead. To reduce these complaints, the administrators of CareerLight had to put many hours of work into each evaluation round, trying to improve the matching quality manually. However, hours of manual work meant extra expenses for Netlight that could potentially be reduced if the core of the problem was addressed properly, i.e. by improving the quality of the matchings produced by CareerLights matching algorithm.

In addition to the problem background described above, a decision to add a new feature to CareerLight had been made at Netlight. Previously there had been one standard evaluation template that all assigned evaluators should follow. The new feature, on the other hand, should allow templates of different evaluation sizes to be used, based on how well the employees in the evaluation pair knew each other. The templates of the different evaluation sizes were called CCB-matrices because they reflect different evaluation depths on one axis and the evaluation areas Competence, Creativity and Business sense on the other axis. Assigning a large CCB-matrix evaluation to a matched evaluation pair would require that the evaluator writes a full, in-depth evaluation, while a small sized evaluation would require just a sentence or two commenting the overall impression of the evaluatee. Adapting the matching algorithm to assign CCB-matrix sizes to evaluation pairs also became part of this thesis.
An initial study of the problem produced the following list of problems associated with the existing matching algorithm:

- Insufficient input data
- Time consuming manual work
- No possibility to reiterate the match result
- Questionable relationships
- Little variation between evaluation rounds
- Platform dependent black-box algorithm
- No support for CCB-matrix feature

An elaboration of each of these points can be found in Section 4.1, after a deeper explanation of the inner workings of CareerLight has been given in Chapter 3.

1.3 Purpose and goal

The purpose of this thesis was to investigate which parts of the matching algorithm needed improvement and explore which systems could be aggregated to improve the quality of the input data for the algorithm. The goal was to implement at least one potential solution supported by existing research to each task in the problem description, and evaluate its performance.

1.4 Methodology

The project model used for this project was of waterfall type with many elements borrowed from the LIPS project model [31]. The LIPS-model describes a process that consist of three phases: Before, During and After.

In the Before-phase the problem is analyzed so that individual key sub-problems can be identified. These are then used as basis for the formulation of a requirement specification that contain the expected requirements for the final product. The Before-phase ends with a plan being made, describing how the problem should be solved and how the requirements should be fulfilled.

The During-phase is where the actual implementation takes place. First a brain storming session should take place to produce a list that contain as many solution alternatives as possible. The solution alternatives can be anything that has the potential to fulfill one or several of the requirements in the specification. The different solution alternatives should then be evaluated in detail, to make decisions whether to keep them as part of the final product or if they should be discarded. The During-phase ends when a set of solution alternatives has been implemented that fulfills all the requirements.

Finally, in the After-phase the final product is evaluated, documented and delivered.

1.4.1 Before-phase

In the Before-phase an initial study of the problem was conducted in order to specify requirements and plan the implementation of the project. This resulted in two artefacts, a time-plan and a requirement specification.

The initial study consisted mainly of briefing-meetings with personnel at Netlight that were involved with the CareerLight system in different ways. From a meeting with one of the system architects behind CareerLight I learned more about the underlying architecture. CareerLight is a web-based tool designed to assist administrators in implementing an evaluation round all the way from configuration to the delivery of written evaluations to every employee. A detailed description of the CareerLight system is provided in Chapter 3.

Implementing an evaluation round in CareerLight is a process in several steps that begins with configuring the evaluation round by setting its parameters and stating which employees should be
included. In the next step the administrator triggers a fully automated relationship method that gathers information from Netlight’s internal databases about all the included employees, and produces relationship connections between them as potential evaluators. Each connection also receives a weight based on the relationship types discovered by the relationship method. These connections are then translated into a weighted and directed graph according to the model described in Section 3.6.2, which is later used as input for the matching algorithm that produces the final evaluation pair assignments.

Meeting with one of the developers of CareerLight revealed in detail how the existing matching algorithm was implemented. It was based on an optimizer framework that attempted to optimize a target function while still adhering to a set of constraints. The target function in this case was the total sum of the weights of the relationships included in the solution. The constraints that limited the optimizer were formulated to ensure that the evaluation workload was distributed equally amongst all employees and that every employee received an acceptable amount of evaluations.

I also met with an administrator that had experience of using the administrative tools of the system that could shed light on the issues experienced from a user perspective. From this meeting I learned that the main issue with CareerLight was that it produced incomplete matching results that were too time consuming for the administrators to correct by hand.

1.4.2 During-phase

The first half of the During-phase consisted of searching for and reading through scientific articles on relevant subjects to find as many solution alternatives as possible to the previously specified requirements. Many alternatives could be discarded early in the process based on various grounds. For example, some articles could seem relevant at first glance but proved to focus on too unrelated issues when reading further. Others required access to data that was not available to me, either because of privacy issues or because they simply did not exist. Finally, some alternatives were deemed too complex to be implemented within the time frame of this thesis. The first half of the During-phase resulted in an implementation plan consisting of two alternatives to improve the matching algorithm itself, and two alternatives to improve the input data for the matching algorithm. The implementation plan marked a half-time milestone and a reconciliation was made with the examiner and the supervisor at Linköping university.

The implementation of the solution alternatives started shortly before the half-time milestone, and in the second part of the During-phase implementation took of properly. Problems associated with the implementation forced additional cuts in the selection of solution alternatives to be made, to make sure that the project could be completed on time. Debugging and cleaning up the code marked the end of the During-phase.

1.4.3 After-phase

The After-phase was initiated with an evaluation of the implemented solutions. A meeting was held with a CareerLight administrator and one back-end developer. The new features were demonstrated and a discussion was held about the improvements from an administrator perspective. The matching quality was verified by sending a questionnaire to employees that volunteered to say what they thought about matchings produced for them individually by the new algorithm. The results from these surveys were then analyzed to obtain a metric for the performance of the implemented algorithm and to identify topics relevant for future work.
1.5 Scope

Due to the size of the task assigned by Netlight, the scope of this thesis report has been limited to focus on improving the input data to the matching algorithm and on implementing a new, platform independent matching algorithm. The entire CareerLight system is very voluminous, consisting of both back-end features such as databases, data-access-objects and API functions, and front-end parts like the administrator GUI. Although most new features implemented in the back-end requires changes to be made in the front-end user interface before final integration, the front-end parts have been left mainly untouched.

1.6 Contribution

This thesis presents a method to solve a matching problem of a non bi-partite graph. A data mining method is used to extend and thus improve the quality of the input graph, and the impact on the final match result is evaluated. This work has created value for Netlight in the form of a new implementation of their matching algorithm that performs within reasonable time, supports their new CCB-matrix feature, is platform independent and is easier to modify. It produces match results with higher completeness because of an extension to the relationship method, and the matches have approximately the same quality as those produced by its predecessor.

Much research has been done in the fields of Recommender Systems and data mining. However, the issue of understanding the social relationships between individuals by looking at implicit data has only been studied by a few teams of researchers. Most solutions presented in this area starts with some basic knowledge of the targets social network construction, which is then used as input, sometimes along with additional data, to an algorithm that attempts to find missing connections in the network. A good example of one such algorithm is the feature “People you may know” used on the social media platform Facebook. These solutions resembles what was done in one part of this thesis, as a known but incomplete social network was used as input to an algorithm that attempted to bring the social network closer to completion. The other part of this thesis differs from many scientific papers on Recommender Systems however, as they mostly assume that the matching entities are divided into different sets (e.g. users and items) instead of being part of the same set, which is the case with the employees at Netlight.

1.7 Outline

The next chapter introduces the theoretical aspects of important techniques and concepts used throughout the thesis. It can be used as an encyclopedia to be referenced while reading this thesis. The chapter starts with an introduction of graph theory, followed by an introduction of linear programming and a method to solve these types of problems called the Simplex method. This is followed by an introduction to Recommender Systems and the concept of data mining. Finally we take a look at linear regression.

A thorough description of the system CareerLight, that contains the matching algorithm is given in Chapter 3. In particular we examine the workflow to initiate an evaluation round from an administrator perspective. This is then followed by the architecture of the back-end part of CareerLight, and a description of how the employee matching is modeled and implemented.

In Chapter 4, the problems listed in Section 1.2 will be elaborated further, and a set of requirements for the final implementation is provided. Chapter 4 is then concluded by listing a number of guiding questions used as guidelines for the literature study.

In Chapter 5 the results from the literature study is presented. It emanated into a set of considered solution alternatives, covering various parts of the requirement specification from Chapter 4. The first part of Chapter 5 presents the solution alternatives that were considered but discarded. The second part describes the solution alternatives that proved to be feasible, and we will look closer at how they were implemented. In total, two different data sources that could potentially improve the input to the matching algorithm, and one implementation of a constraint satisfaction framework that could potentially improve the matching algorithm itself were kept for the final evaluation.
In Chapter 6 the methods used to evaluate the implemented solution alternatives are given, and the results from the evaluation are presented. One of the input data sources was evaluated using linear regression, comparing it to existing data from a previous evaluation round. The second input data source was evaluated together with the implementation of the new constraint satisfaction framework and its CCB-matrix integration by conducting a survey among the employees of Netlight, and by holding an interview with CareerLight administrators.

Chapter 7 concludes this thesis with a discussion of the obtained results and the conclusions made. The chapter discuss to what degree the implemented solution alternatives satisfies the specified requirements, and the guiding questions from Chapter 4 are answered. Finally, a list of recommended future work is presented.
Chapter 2

Theory

In this chapter some of the key concepts used when reasoning about the functionality and solution alternatives of CareerLight are presented. This chapter can be used as an encyclopedia, and can be referenced for explanations.

The first section of this chapter gives an introduction to graph theory. In this thesis, graphs are used to model both the input data to the matching algorithm and the resulting matching output. The next section introduces linear programming and constraint satisfaction, which are the techniques used by the matching algorithm for solving the matching problem.

In Section 2.3 the field of Recommender Systems is introduced. This thesis belongs to that field, and some common types of Recommender Systems are introduced together with some common problems associated with them. The following section gives an introduction to the concept of data mining, and how it can be used to extend the matching input. One method called SONAR is examined in particular.

Finally, this chapter is concluded by introducing linear regression, which is an algebraic method to find coefficients for a linear function that best aligns with some measurement data. Linear regression is used in this thesis to evaluate one of the data sources that was hypothesized to improve the quality of the matching input.

2.1 Graph theory

One of the main tasks at hand for this thesis is to construct a network of employees based on their social and professional relationships. This can be modeled using a graph, since graphs are capable of holding useful information about networks. A graph, \( G(N,E) \), is a set of nodes, \( N \), that are connected by a set of edges, \( E \). A node can represent any type of entity, and the edges can represent any type of relationship between the nodes. If an edge depicts a relation that holds from node \( v_i \) to node \( v_j \), but not from node \( v_j \) to node \( v_i \) it is called an arc from \( v_i \) to \( v_j \). We can say that a graph is directed if it consist of arcs instead of edges. A graph can also store weights that in some regard indicate the strength of the edges. If there, for every edge \( e_{ij} \in E \), exists a weight \( w_{ij} \in W \) we say that a graph is weighted.

In this particular problem we let nodes represent employees at Netlight, and arcs represent the social connections between them. We are interested in modeling whether one employee is suitable to evaluate another employee, which implies a directed graph should be used. Finally we want to store a measure of the expected quality of evaluations at different edges, so weights are introduced as well. The weights are obtained from a data mining method (see Section 2.4) used within CareerLight, and each weight reflect how strong a social relationship between two employees is. It is assumed that a strong social relationship between two employees implies that they can write evaluations of high quality to each other.
The purpose of creating a graph like this is to be able to run it through a matching algorithm, which will be covered in detail in Section 3.6. Most previous research on matching [16, 33] and Recommender Systems [2, 5, 6, 17, 18, 23, 25, 27, 35, 36] assume that there are two or more separate sets of entities, $S_1, S_2, \ldots, S_n$ such that $S_1 \cap S_2 \cap \ldots \cap S_n = \emptyset$, and that a matching is performed by selecting edges that connect entities from different sets with each other. A graph that is divided into different entity sets is said to be bi-partite, and a typical example of this set-up is a Recommender System that suggests cooking recipes from one set of recipes to different users in another set of users, as described in [13]. In the problem at hand however, all entities to be matched are part of the same set of employees, and thus we have a non bi-partite graph.

### 2.2 Linear Programming

Linear programming is a method to find an optimal solution to a given problem. There is no universal definition of what an optimal solution for any given problem is, as it depends on the problem. In most cases though, the optimal solution to a problem can be found by either maximizing or minimizing some value, $z$. For example, an optimal business plan can be said to be found when profits are maximized. To solve a linear programming problem, we must first specify the characteristics of the optimum. We define a target function so that, $z = f(X)$, that calculates the value to be either maximized or minimized from a vector of problem variables, $X = (x_1, x_2, \ldots, x_n)$.

In almost all real-life problems $X$ is limited in some way. If $X$ is a vector of the number of different units produced in a factory, the maximum number of units that can be produced could be limited by production cost and availability of raw materials. These constraints puts a limit to how many units can be produced, which in turn puts a limit on the profit that is possible to produce by the factory. In order to maximize the profit generated such constraints must be taken into consideration. If the vector $A$ contains the production costs for the different units, then we can say that the optimal solution is constrained by the constraint $A^T X \leq b$, meaning that the total production costs of all units must be less than or equal to some production budget $b$. Finally, we must specify the constraint that it is not possible to produce less than 0 of any entity with the constraint $x_1, x_2, \ldots, x_n \geq 0$. A linear programming problem is formally written on the following form:

\[
\text{Maximize} \quad z = c^T X \\
\text{Subject to} \quad A^T X = b \quad \text{(Eq. 2-1)} \\
\quad x_1, x_2, \ldots, x_n \geq 0
\]

In this example we have $f(X) = c^T X$, where $c$ is a vector of estimated profits from individual units produced. In Equation 2.1 the constraints are provided with an equality sign. The reason for this is that the algebraic implementation of the most common algorithm used for solving LP-problems, the Simplex
algorithm, only allows equalities in the constraints. It is however possible to get around this restriction by introducing something called slack variables. Consider the constraint:

\[ a_1x_1 \leq b \]
\[ x_1 \geq 0 \]  
\text{(Eq. 2-2)}

This constraint can be written on equality form by introducing a slack variable \( x_2 \):

\[ a_1x_1 + x_2 = b \]
\[ x_1, x_2 \geq 0 \]  
\text{(Eq. 2-3)}

Because the slack variable \( x_2 \) must be equal or greater to 0, we get a constraint with the same properties as in Equation 2-2. A sub-category of linear programming optimization is called constraint satisfaction, where the focus lies in finding any solution that satisfies the specified constraints, without regards to any maximization or minimization objectives. In the following sub-section a method to solve a linear programming problem is given.

### 2.2.1 The Simplex method

Regarding the matter of maximizing \( f(X) \), and thus solving a linear programming problem, there are a number of algorithms to do this. One common method is called the Simplex method [14]. It has been shown that for certain problems the Simplex method has an exponential worst case time complexity, but for most common problems the time complexity is polynomial. A geometric interpretation of how the algorithm works is presented in this section.

#### 2.2.1.1 Geometric interpretation

In order to explain how the Simplex method works we must first define the space \( V \in \mathbb{R}^{\mid X\mid} \), \( |X| \) being the total number of unknown variables in our problem. Next, we let all of the constraints form hyper planes in \( V \), each cutting \( V \) into two parts with all valid points on one side and all invalid points on the other. Together they form a convex polytope due to the linear nature of the constraints. This polytope contains all of the feasible points for our solution. In order to find the point that maximizes the target function we let \( Y \) be the function value of \( f(X) \). Then \( Y \) will also be the distance from the origin to the hyper plane formed by \( f(X) \). Every point that lies on the area that is the intersection of the polytope and \( f(X) \) is a valid solution to \( f(X) \). So \( f(X) \) will be maximized when the distance \( Y \) is as big as possible while \( f(X) \) still intersects the polytope formed by the constraints in at least one point. This will result in a solution that is a surface of points if \( f(X) \) aligns with the surface of one of the active constraints, or else a single point that is one of the corners of the polytope.
The Simplex algorithm makes use of this fact by trying to find the corner of the polytope that is the furthest away from the origin along the normal of the target function. This is done by first selecting any corner of the polytope. Then it searches along the outgoing edges for the next corner that gives the greatest increase in $Y$, and selects it. This step is then repeated until no more corners can be found that increases $Y$. At this point, we know that the selected corner is a maxima. If the polytope is convex it can be shown that any maxima is also a global maxima, which in turn means that whenever a maxima has been found we can be certain that no better solutions exists.

### 2.2.1.2 Restrictions

There are some circumstances that renders the Simplex algorithm unable to find any optimal solutions. If the traversed polytope is unbounded in the direction of $Y$, an outer-most corner will never be found. Also, if there is no overlap of the feasible spaces defined by the constraints, there will be no valid points that provide an optimal solution. The latter restriction is a problem tackled in this thesis.

### 2.2.2 Integer Programming

Recall from Section 2.2 that $X$ is a vector of integer values. Since the regular Simplex algorithm searches for the optimal solution in $V$ it is likely that a solution of real numbers will be provided. Enforcing integer-only solutions turns out to be a bit more complicated than simply rounding the elements of $X$ to the nearest integer value. It turns out that the optimal integer solution sometimes appears quite far from the optimal real solution, so a couple of modifications are required. One solution is to use a method called Gomory’s mixed integer cuts [9]. The method proposes an algorithm that initially ignores the integer constraint and finds the regular solution. If the solution $X$ contains any real values, then a cut is made in the polytope of valid points by introducing a new constraint. This constraint forms a hyper plane that intersects the closest valid integer points so that the current, non-integer solution is excluded while all valid integer solutions remain. Next, the algorithm searches for a new optimal solution, and as long as the solution contains non-integer elements new cuts are made.

### 2.3 Recommender Systems

In Section 3.6.2 a method is presented that finds an optimal matching, $G_M$, once the input graph, $G_{PE}$, is known. However, constructing $G_{PE}$ is no easy task. There is no single resource at Netlight that explicitly contains all information about the social and professional relationships that exist between all of the employees.
employees. There exist however a couple of resources that contain information that has the potential to implicitly provide the wanted information. Regarding this matter, there is a popular field of study called Recommender Systems [19] that deals with problems similar to this. A Recommender System usually takes the form of an algorithm that gathers data on a user’s preferences in order to be able to suggest new items that the user also might like. In our case, we let the set of considered items be the rest of the employees in our social network. Three major approaches to achieve this are commonly discussed in research today; Collaborative filtering, content-based filtering or a hybrid combination of the two. An introduction to each approach is given in the following sub-sections. Finally, some common problems associated with Recommender Systems are introduced in Section 2.3.4.

2.3.1 Collaborative filtering

In the collaborative filtering approach, the Recommender System creates a profile of every user’s preferences. For every user profile, it then searches among the other users to find similar preference profiles that may contain popular items that are not yet part of the user’s list of consumed items. This approach has turned out to perform well [30], provided that there is enough data to make any decent conclusions. A user needs to have at least a few items in their consumption list for the Recommender System to be able to make a somewhat accurate preference profile. This makes the collaborative filtering approach somewhat prone to the cold start problem, which will be covered in Section 2.3.4.

2.3.2 Content-based filtering

Instead of searching for correlation between different users’ profiles, the content-based approach puts focus on the individual user and their relation to the contents of the available items. Every item is categorized into a set of properties. Based on what properties the user seems to like, new items with similar properties can be recommended. This approach does not require as much information as collaborative filtering to produce decent recommendations. The downsides of this technique however are that the Recommender System will be limited to only make recommendations of items in the same genres that the user has already discovered, and items in some domains, like music and video, are hard to categorize automatically [1].

2.3.3 Hybrid Recommender Systems

In order to overcome the shortcomings of the individual filtering techniques, attempts have been made to combine collaborative- and content-based filtering in different ways. A Recommender System could benefit from combining the ability to recommend new categories of items from collaborative filtering with the ability to recommend similar items in cold start scenarios (see Section 2.3.4.1) from content-based filtering. Combined together, these two filtering techniques can verify each other’s recommendations, enhancing the probability of producing relevant recommendations to the user. An examples of one such system is given in [3].

2.3.4 Problems related to Recommender Systems

Regardless of the type of Recommender System being used, there exist a number of well-known problems that must be tackled. This section describes three problems associated with Recommender Systems that have been identified to be relevant for this thesis.

2.3.4.1 Cold Start

Many Recommender Systems suffers from problems with producing relevant recommendations when data is sparse. These situations often occur with new users and new items being added into the system. This is
known as the cold start problem and it is an area that has attracted a lot of attention from researchers trying to mitigate its impact. Many approaches involve either prompting new users for demographic information [17], their personal preferences [35], or using existing social networks to derive a measure of trust between users that could indicate they have similar preferences [10, 36].

From the problem description in Section 1.2 we know that CareerLight’s existing matching algorithm is unable to produce satisfactory match results. This problem is believed to be partially caused by the scarcity of the input data provided to the matching algorithm and can be argued to be an instance of the cold start problem.

### 2.3.4.2 Gray Sheep

The concept of Recommender Systems is built upon the assumption that there exist some correlation between some similar user data and similar user preferences. There exists users however, whose preference profiles consequently deviates from the norm. These users are called Gray Sheep, and it has turned out to pose a great challenge to produce relevant recommendations for them. One proposed method to mitigate this problem [8] utilizes clustering techniques to identify the gray sheep users from the rest, so that they can be treated separately.

The gray sheep problem could potentially be a problem for this thesis. It could turn out that some employees deviate from the norm regarding their digital fingerprint, which may cause any data mining implementation to make faulty assumptions about their social relationships, which in turn are used as input for the matching algorithm.

### 2.3.4.3 Scalability

Many systems that utilize recommendation techniques tend to grow rapidly in terms of users and recommendable items. The sheer amount of combinations between entities can therefore cause enormous computation complexity which in turn leads to unacceptable running times. For the most part, users expect immediate responses from Recommender Systems, so scalability is a matter that certainly needs attention.

Netlight is a growing company, and it is expected that the number of participants of an evaluation round increases with each new evaluation round. It is therefore important to consider scalability when working with the matching algorithm.

### 2.4 Data mining

In this thesis, one of the main problems to solve is the matter of extending the input graph, $G_{PE}$, so that the probability of the existence of a matching solution is increased. In the existing solution, the most obvious social connections such as number of recorded working hours on the same project, mentor-student relationships and other recorded organizational relationships have already been used. In addition to this, users are also prompted to provide wishes who they would like to be their evaluators. This alone however was deemed insufficient to be used as input to the existing matching algorithm, and hence more information about the social relationships within Netlight needs to be extracted to achieve acceptable results. To extend the matching input, a data mining technique can be used. The concept of data mining is to extract potentially relevant information from various external data sources. The information is then used to make statistical conclusions about the model at hand. In essence this means that external data sources may hold information that cannot be found anywhere else, and a data mining technique is the tool to extract this information. One such data mining technique called SONAR was selected as a template for the implementation in this thesis, and it is described in the following sub-section.
2.4.1 SONAR

At the company International Business Machines (IBM) some research has been made on the creation of a social network using implicit information from data mining. In 2008 they created an API called Social Network Architecture (SONAR) [12] that could be installed on various internal applications at IBM that held social information. The API could be configured to package relevant social information from each application, and respond in a uniform way to queries. An aggregator was used to merge the results into a measure of social connection strength by calculating the scalar product between the result vector and a vector of manually defined weights. In most aspects SONAR aligns with the relationship method described in Section 3.5.1, that was already present in the existing CareerLight system.

The same year IBM also conducted a study [11] where the SONAR API was utilized to assess the differences between social connection conclusions based on publicly available data and privately available data. The private source used was the email accounts of the participants, and they used several public sources, including blog comments and group memberships. Their conclusion was that the information embedded in public sources better reflected the social networks of the participants, finding important ties that email sources could not.

In 2009, researchers at IBM conducted a study [4] regarding the matter of extending a person’s existing social network by suggesting additional potential contacts. Four contact suggestion algorithms were evaluated, two content matching techniques, SONAR and the friend-of-a-friend algorithm. The content based methods were based on the assumption that if two people operates within a field of similar topics, they may like the same things and therefore may like each other. The friend-of-a-friend algorithm creates clusters of interconnected people from an existing social network and then searches for missing links between socially entangled people. The assumption is that if they have similar friends, the possibility exists that they two are friends as well. Their results showed that while content matching was best at suggesting new but relevant connections, SONAR was best at finding already known friends.

2.5 Linear regression

Using data mining to extract information from different sources raises the need for some method to make conclusions from the data collected. If a vector $D = (d_1, d_2 \ldots d_n)$ contains the results from mining $n$ different sources using some data mining method, each producing a binary value $d_i \in [0,1]$ indicating whether a particular relationship holds or not, we can easily calculate the mean of the elements in $D$ using:

$$R = \frac{\sum_{i=1}^{n} d_i}{n}$$  \hspace{1cm} (Eq. 2-4)

This equation produces a real value, $R$, holding the percentage of positive relations returned from the data mining method. This solution might suffice for the simple set-up provided above, but in most contexts this is not enough. The data vector is in most cases not confined to contain binary elements only. Assume we let $D$ consist of the results mined from two data sources, $d_1 \in [0,1]$ being whether person $A$ has befriended person $B$ or not on a certain social network platform, and $d_2 \in [0,\infty[$ being the number of comments that person $A$ have written to person $B$ on another social network platform. In this scenario it is no longer a matter of calculating the mean between the elements in $D$, since we do not know how many comments in $d_2$ are required to match a positive friend-connection in $d_1$. By introducing a weight vector, $W = (w_1, w_2 \ldots w_n)$ mapped to the different elements in $D$ we have a way to handle this problem. Using Equation 2.6 we obtain a weighted result, $R_{w}$. By adjusting the weights in $W$, it is possible to find a weight configuration that produces satisfactory results.

$$R_{w} = \frac{\sum_{i=1}^{n} d_i w_i}{\sum_{i=1}^{n} w_i}$$  \hspace{1cm} (Eq. 2-5)

In mathematics, there is a field called Linear regression that tackles the issue of finding the optimal weights for a certain training set where the ideal results are known. Suppose we want to find the linear function that best describes the growth rate of a potted plant. A series of measurements have been made during 5 days, producing a vector $X$ of the measured heights, see Table 2-1.
Improved algorithm for weighted matching of employees

Eldén et al. [7] provides a method to calculate the coefficient $b$ that produces the line that best approximates the function passing through all the points in the data set. We obtain $b$ by using the relationship from Equation 2-6:

$$X^TXb = X^TY$$

(Eq. 2-6)

Inserting the values from Table 2-1 in the equation we get:

$$
\begin{pmatrix}
1 & 2 & 3 & 4 & 5 \\
2 & 3 & 4 & 5 & 11,1 \\
3 & 4 & 5 & 11,9 \\
4 & 5 & 11,2 \\
5 & 13,2 \\
\end{pmatrix}
\leftrightarrow

\begin{pmatrix}
9 \\
9,8 \\
11,1 \\
11,9 \\
13,2 \\
\end{pmatrix}
$$

$$55b = 175,5 \leftrightarrow b \approx 3,2$$

In this example we get a function that approximates $Y$ at different values of $X$ rather poorly, as can be seen in Diagram 2-1.

The results suggests that a bias exists in the data. This can for example be due to that the plant must have been growing for a short while before the measurements started. Instead of searching for a line on the form $Y' = bx$ a much better result would be obtained using a bias value that adds a constant to the line function. This line can be described using $Y' = \beta_0 + \beta_1 x$. This means that we now have to find the bias value $\beta_0$ in addition to the coefficient before $x$. This can however easily be done by turning the X vector into a column matrix, adding one column of ones:

<table>
<thead>
<tr>
<th>$Y$: Height, cm</th>
<th>9</th>
<th>9,8</th>
<th>11,1</th>
<th>11,9</th>
<th>13,2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$: Day</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 2-1. Fictional measurement data used for a simple linear regression example.

![Diagram 2-1. A diagram showing the measured values $Y$ and the line $Y'$ that approximates $Y$ best, using no bias.](image-url)
Using the \( X \)-matrix in relationship from Equation 2-6 just as before we get:

\[
X^TX\beta = X^TY \quad \leftrightarrow \\
\beta = (X^TX)^{-1}X^TY \quad \leftrightarrow \\
\beta = \begin{pmatrix} 5 & 15 \\ 15 & 55 \end{pmatrix}^{-1} \begin{pmatrix} 55 \\ 175.5 \end{pmatrix} \quad \leftrightarrow \\
\beta = \begin{pmatrix} 1.1 & -0.3 \\ -0.3 & 0.1 \end{pmatrix} \begin{pmatrix} 55 \\ 175.5 \end{pmatrix} = \begin{pmatrix} 7.85 \\ 1.05 \end{pmatrix} = \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}
\]

Inserting these values into our new formula produces a much better result, the measured values and the approximated function are almost indistinguishable:

\[
\begin{align*}
\text{Diagram 2-2.} & \quad \text{A diagram showing the measured values } Y & \text{ and the line } Y' \text{ that approximates } Y \text{ best, using bias.}
\end{align*}
\]
Chapter 3

CareerLight

The matching algorithm that is the core of this thesis is a vital part of a large internal system called CareerLight. In this chapter, a thorough description of the CareerLight system is provided. First the different user roles are defined followed by a tour through the process of creating a matching from start to finish from an administrator perspective. We will then have a look at the system architecture and its implementation. Finally we will look closer at two methods of interest; relationship generation and matching generation.

3.1 Roles

The user roles of CareerLight can be divided into two main groups, administrators and participants. Both groups are employees at Netlight and both are potential participants of an evaluation round. Administrators are employees that are responsible for implementing an evaluation round, and CareerLight has a special administrator GUI to assist them in this process. Participants are employees that have been selected to participate in a certain evaluation round. In the evaluation process, the participants can take on one or both of two sub-roles. If they write one or more evaluations for another participant, they are said to be an evaluator for that participant. On the other hand, a participant that receives an evaluation from an evaluator is said to be an evaluatee for that evaluator. A complete list of participant sub-roles is provided in Section 3.2.3.

3.2 Administrator workflow

CareerLight is a system that provides tools for administrators to create, configure, populate and match an evaluation round. The workflow is divided into a several step process, and to provide a clear picture of the system, each step is described in this section. To configure an evaluation round, the administrator has to work through the following steps:

1. Overview
2. General configuration
3. Participants
4. Wishes
5. Relationships
6. Matching

Each of these steps are represented by a tab in the administrator GUI, and each of them are described in the following sub-sections.
3.2.1 Overview

Initially, the administrator logs in to the administrator GUI of CareerLight, where he or she can choose to create a new evaluation round or continue working with an existing evaluation round. Creating a new evaluation round prompts the administrator for a name of the new evaluation round.

3.2.2 General configuration

Under general configuration, the administrator can set detailed configuration parameters in five sub-steps; Participants, Wishes, Matching, Evaluation and Mentor sessions. Each of these sub-steps are described in the following sub-sections.
3.2.2.1 Participants

In the first configuration step the administrator can set the dates when the evaluation round starts and ends, and when participants can start writing their evaluations. They can also configure a maximum limit of absence in percent so that employees with too much absence can be excluded from the evaluation. Finally they can select which employee levels that should participate in the evaluation, both as evaluatees and as evaluators.

![Figure 3-2. The participants configuration step where the administrator can select which participants should be included in the evaluation round.](image)

3.2.2.2 Wishes

In the second configuration step it is possible to make configurations for wishes. Wishes are relationships between employees that they create themselves. If wishes are enabled, a wishing period starts where all participants of the evaluation round have to log in and wish for other employees that they consider to be suitable evaluators for them. The administrator has the possibility of limiting the maximum and minimum number of wishes made by each employee.

![Figure 3-3. Under the wishes configuration step the administrator can choose whether to allow wishes or not.](image)
3.2.2.3 Matching

In the third configuration step configurations for the matching algorithm are made. The administrator can set the maximum amount of evaluations any participant has to write, and the minimum number of evaluations that every participant should receive. The priority of the different relationship types between employees can also be configured in this step. The initial relationship types are:

- Wishes (as described in the previous step).
- Solution manager (SM) - Has been manager for a project that the employee has been working on during the last six months.
- Joint Project – Also known as the colleague relationship. Has been working on the same project for some time during the last six months.
- Engagement search (ES) - Has been involved with finding a new project/client for the employee during the last six months.

**Figure 3-4.** The matching configuration step. Here the administrator can set the priority of different relationship types and the maximum and minimum amounts of evaluations.
3.2.2.4 Evaluation

The fourth evaluation step enables configuration of different CCB-matrix sizes. CCB is short for Competence, Creativity and Business sense, which are the three main topics that evaluators should consider when writing evaluations. These topics are analyzed into different depths depending on the assigned CCB-matrix size. For example, a matching assigned with a small CCB-matrix size only requires that the evaluator writes a one line comment regarding each topic. A large CCB-matrix size on the other hand requires a full depth analysis of the employees strengths and improvement areas regarding their competence, creativity and their business sense. To ensure a high quality of the evaluations it is important to assign evaluations with large CCB-matrix size to matched pairs with strong relationships, and smaller matrices for weaker relationships. Adapting the algorithm to assign the correct CCB-matrix sizes to the matched pairs is one of the requirements for this thesis, see Section 4.1.6.

3.2.2.5 Mentor sessions

The mentor sessions have not been handled in this thesis and play no role within the scope of this report, and will therefore not be discussed.
3.2.3 Participants

Under Participants it is possible to press a button that initiates a function that includes all participants that meet the requirements set in the configuration step in 3.2.2.1. After generating the participant list, all participants are displayed in a table with their assigned evaluation roles. The possible roles are:

- Full participant - Both writes and receives evaluations.
- Write for others - Writes but does not receive evaluations.
- Self-evaluator - Only evaluates themselves. Recently employed employees often evaluates themselves during their first evaluation round.
- Excluded - Is not a participant in this evaluation round.

Figure 3-6. The Participants step, where an overview of all the participants of the evaluation round is provided.

3.2.4 Wishes

Under the Wishes-tab it is possible for the administrator to overview the progress of the wishing period, provided that wishes were configured to be allowed. A table view lists all participants and how many
wishes they have currently made. If the wishing period is coming to an end, the administrator can choose to send emails to all participants that have not yet made their wishes.

### 3.2.5 Relationships

This step is where the administrator triggers the method that generates the relationships that becomes the input for the matching algorithm. A relationship method searches for each of the relationships listed in 3.2.2.3. These are presented in a table with the evaluatee in the left-most column and all their potential evaluators listed in columns horizontally. The name of each evaluatee is accompanied by a letter indicating their role. Each evaluator name is accompanied with a list of all relationship types that connect them with the evaluatee. One remark is that the relationship type called ReverseWish that occurs several times in Figure 3-8 is not one of the initial relationship types, but added later as a result of this thesis. This relationship type will be covered more thoroughly in Section 5.2.1.1.
3.2.6 Matching

The Matching-step is where the administrator initiates the matching algorithm that assigns evaluators to the evaluatees. When the algorithm is finished the result is shown in a table view as in Figure 3-9. Clicking on a row brings up a yellow box (seen to the right in the figure), where the administrator can add additional evaluators, change CCB-matrix sizes for individual evaluators, and overview the matching score that was estimated by the relationships algorithm in the previous step. Finally when manual changes are complete the administrator can save the matching.

3.2.7 Evaluation

The final two steps, Evaluation and Edit evaluation handle the written evaluations. This part of the evaluation process is not relevant for this thesis, and hence it will not be covered in this report.
3.3 Architecture

CareerLight consist of three GUIs and a server back-end part. The administrator GUI has been described in Section 3.2 and it is the most relevant GUI for this thesis. There is also the Participant GUI where participants of an evaluation round can log in and make their wishes during the wishing period, and later write evaluations for the evaluatees assigned to them. The third GUI is the Mentor GUI which is used for the mentor sessions which are not covered in this thesis.

The server back-end part is called CareerBackend and it provides endpoints that responds to requests made by the GUIs. CareerBackend communicates with two databases:

- Agresso – Contains personal information of all employees such as name, contact information, reported time, absence and assignments.
- DB_CareerLight – Contains everything associated with evaluation rounds, such as evaluation round configurations, participants, relationships, matchings, CCB-matrix definitions and the written evaluations.

The existing endpoints are quite extensive, covering creation, modification, retrieval and deletion of entities used within the system. Some relevant entities are evaluation rounds, participants, wish-period configurations, wishes, relationships, match configurations and matchings. A UML diagram of these entities can be seen in Figure 3-12. CareerBackend also provide endpoints for starting methods that populates the list of participants, generates relationships between participants and that selects a subset of the relationships into a matching. In Figure 3-11 some important endpoints of CareerBackend are illustrated.
Figure 3.12. A UML diagram of the Java implementation of the entities used by CareerBackend. A Matching object has a MatchingConfig and a collection of Evaluatees. Each Evaluatee has in turn a collection of Evaluators. Relationships between Evaluatees and Evaluators are stored within the Evaluator object, which keeps a set of Strings for each relationship type, and a double value for the accumulated score.

### 3.4 Implementation

CareerLight is a web-based system, and its GUIs are implemented in HTML and the JavaScript framework AngularJS. The GUIs communicate with the server back-end using the REST architectural style.

CareerBackend is a Java-based backend implemented using the Spring application framework, and it runs on a Tomcat server.

It uses MSSQL databases for storing data. The DB_CareerLight database is hosted by Netlight, and CareerLight has both read and write permissions. The Agresso database comes from a third party vendor, and due to warranty reasons, CareerLight is restricted to read-only operations.
3.5 Methods of interest

CareerBackend has two back-end methods that are of special interest. They are the Relationships and Matching endpoints found in Figure 3-11. The relationship generation method iterates over all participating evaluatees and calls for a number of pre-defined providers that access different data sources to find potential evaluators. It can be extended by adding additional providers.

The matching generation method takes the full set of relationships and initiates an algorithm that reduces it to fit within the maximum write and minimum receive parameters while attempting to keep the relationships with the highest matching score as long as possible.

3.5.1 Relationship generation method

The method that generates the relationships between the participants starts by creating a new Matching object to store all possible relationships. It calls the get-methods of the matching configuration and the participants endpoints to obtain a list of all participants and the matching parameters for the current evaluation round. The Matching-object containing all participants is then passed into two relationship providers, which searches for specific relationships and appends them to the Matching.

When finished, the relationship method stores the matching object with its relationships in the DB_CareerLight database.

![Figure 3-13. A block diagram of the relationship method. A call to the generate endpoint creates a new Matching object which is then filled with relations found by two relationship providers.](image)

3.5.1.1 Colleague/SM/ES provider

This provider fetches employee data from the Agresso database. It searches for three different relationship types within the last six months. An employee is considered a colleague if they have been co-workers on a project. The matching score for colleagues is set by calculating the percentage of the total work hours the pair have been co-workers. This percentage is then multiplied by the colleague weight found in the matching configuration before it is added to the total matching score of the pair.

The same process is done for solution managers (SM) and Engagement search (ES) relations.

3.5.1.2 Wish provider

The wish provider fetches all wishes made during a wish period from the DB_CareerLight database. If the employee has wished for a certain evaluator, then this relationship is added to the matching. Whenever a
wish relationship is found, its matching score is set to the weight of wishes found in the matching configuration.

### 3.5.2 Matching generation method

The matching generation method takes a Matching object with all possible relationships and reduces the number of evaluatee-evaluator relationships until the maximum evaluations write and minimum evaluations receive constraints are fulfilled. To keep the matching quality as high as possible, it attempts to maximize the total sum of the evaluation score associated with those relationships still in the solution.

![Diagram](image.png)

**Figure 3-14.** A block diagram of the matching method. A call to the generate endpoint creates Matching object containing all possible relationships and the current matching configuration. It is then used as input for a matching algorithm that applies the constraints from the configuration and stores the result in the DB_CareerLight database.

The matching algorithm used is called *EvalMatch*, and it implements a linear programming library to optimize the solution given a set of constraints. A thorough description of its implementation is provided in Section 3.6.

### 3.6 EvalMatch algorithm

The matching algorithm EvalMatch is the core of the entire evaluation round process. This algorithm assigns all evaluation pairs for an evaluation round by selecting high score evaluatee-evaluator pairs from a set of potential evaluation pairs until the constraints “maximum evaluations to write” and “minimum evaluations to receive” are fulfilled.

#### 3.6.1 Implementation

EvalMatch implements a callable library called GNU Linear Programming Kit (GLPK), which can be used for solving Linear programming (LP) and Integer programming problems (IP), see Sections 2.2 and 2.2.2 in the theory chapter. GLPK is written in C++ and compiled into a set of platform dependent, callable binary files. These are in turn wrapped in Java code and packaged into a java *.jar file.

The algorithm itself is written in the programming language Scala, imports the GLPK jar-file and packaged into a *.jar package itself.
3.6.2 Model

The input to EvalMatch is a Matching object (see Figure 3-12) that contains a reference to a MatchingConfig object and a list of Evaluatees. Each Evaluatee contain a list of all possible Evaluators, meaning that there exists at least one relationship between the Evaluatee and the Evaluator. In order to solve the matching as an LP-problem it is necessary to transform the Matching data structure and the constraints into the form:

Maximize \[ z = c^T x \]
Subject to \[ Ax = b \] (Eq. 3.1)
\[ x \in \{0,1\} \]

We will see that this is possible if the Matching object is translated into a weighted and directed graph, see Section 2.1. To create the graph, we let every unique employee, both evaluatees and evaluators be represented by graph nodes. A directed arc is created from every potential evaluator to their corresponding evaluatee, and its weight is set to the matching score of the pair. An example is illustrated in Figure 3-15.

![Figure 3-15](Left) A fictive Matching object containing Evaluatees and their potential Evaluators and associated match-scores. (Right) The Matching object interpreted as a directed, weighted graph.

We call this graph of potential evaluators \( G_{PE} \). Let \( x \) be a vector of integer values on the range \([0, 1]\) representing whether each arc in \( G_{PE} \) is included in the solution or not, \( x_{ij} \in x \) representing the directed arc from node \( i \) to node \( j \). Further, we let \( c \) be a vector containing all weights in \( G_{PE} \), where \( c_{ij} \in c \) represents the weight of the directed arc from node \( i \) to node \( j \). Introducing \( z = c^T x \) produces the sought target function where \( z \) holds the weight sum of all arcs included in the solution. By maximizing \( z \), the weight sum of all included arcs becomes as big as possible, and thus the matching score of the matched relations becomes as high as possible.

The attached MatchingConfig object contain two parameters specifying the desired characteristics of the final matching. One parameter, maximum evaluations to write (\( N_{WRITE} \)) specifies a maximum number of evaluations every employee must write. This means that every node in the matched solution graph \( G_M \) can have at most \( N_{WRITE} \) outgoing arcs. The other, minimum evaluations to receive (\( N_{RECEIVE} \)), specifies a minimum number of evaluations that every employee must receive, meaning that every node in \( G_M \) must have at least \( N_{RECEIVE} \) incoming arcs. These two characteristics can be translated into the following constraints:

\[ \forall i \sum_j x_{ij} \leq N_{WRITE} \] (Eq. 3.2)
\[ \forall j \sum_i x_{ij} \geq N_{RECEIVE} \] (Eq. 3.3)
Translating Eq. 3.2 and Eq. 3.3 to the form $Ax = b$ from Eq. 3.1 requires an $A$-matrix constructed such that:

$$Ax = \begin{pmatrix} x_{0,0} + x_{0,1} + \cdots + x_{0,m} \\ x_{1,0} + x_{1,1} + \cdots + x_{1,m} \\ \vdots \\ x_{n,0} + x_{n,1} + \cdots + x_{n,m} \\ x_{0,0} + x_{1,0} + \cdots + x_{n,0} \\ x_{0,1} + x_{1,1} + \cdots + x_{n,1} \\ \vdots \\ x_{0,m} + x_{1,m} + \cdots + x_{n,m} \end{pmatrix} = b = \begin{pmatrix} N_{\text{WRITE}} \\ N_{\text{WRITE}} \\ \vdots \\ N_{\text{WRITE}} \\ N_{\text{RECEIVE}} \\ N_{\text{RECEIVE}} \\ \vdots \\ N_{\text{RECEIVE}} \end{pmatrix}$$

(Eq. 3.4)

Next, slack variables must be included in order to specify the $\leq$ and $\geq$ conditions (see Section 2.2). The GLPK library conveniently handles this automatically however, and allows constraints to be specified with strict and non-strict inequalities.
Chapter 4

Requirements

This chapter describes the initial problems of CareerLight in detail, and how these were translated into a list of requirements that formed the foundation for the following literature study. The list of requirements for this thesis are presented in Section 4.2, and in Section 4.3 this chapter is concluded by posing a set of guiding questions.

4.1 Problems with the existing system

Besides providing an overview of the CareerLight system, the initial study produced a list of problems associated with CareerLight that gave rise to this thesis. The identified problems are listed in the following sub-sections.

4.1.1 Insufficient input data

The relationships provided from the relationship method turned out to be insufficient to create an input graph for the matching algorithm. The created graph did not contain enough arcs required for the solver to find an optimal solution. The solver produced an error if not all constraints could be fulfilled, which in turn halted the entire evaluation round.

A workaround to the problem was implemented in the matching algorithm. When the algorithm builds the input graph $G_{PR}$ changes were made so that it would include an additional “dummy” node. From this node new arcs could be added to nodes with too few incoming arcs, making sure that the minimum evaluations receive constraint could be fulfilled. All created dummy arcs were given zero weight, so that the LP-solver would favor genuine arcs before the dummy arcs. After the LP-solver found a solution, the dummy node and all arcs connected to it was removed. This would ensure that the matching algorithm would always deliver a solution, but there would be no guarantee that all constraints were met. Some employees could end up receiving too few evaluations, so administrators had to manually assign evaluators, a task that turned out to be too time consuming and that contradicts the purpose of the matching algorithm.

4.1.2 No possibility to reiterate the match result

When manually assigning evaluators to evaluatees it often occurred that an evaluator had to write more evaluations that was specified in the max evaluations write constraint. Fixing this manually would imply that one of the evaluators assigned evaluatees had to be removed, which in turn would mean that this evaluatee would have too few evaluators. Moreover, making sure that all constraints were still satisfied after the manual changes was almost an impossible task. One administrator suggested that the possibility to lock certain evaluation pairs, and then run the matching algorithm again to arrange the rest of the evaluation pairs would save lots of time.
4.1.3 Questionable relationships

Some participants have complained that they have been matched with employees that they have no relationship to at all. They expressed the desire to at least learn why they have been matched with a certain employee.

4.1.4 Little variation between evaluation rounds

Many relationships found by the relationship method used as input for the matching algorithm are based on information stored in the DB_CareerLight database. These relationships (SM, ES and Colleagues) may have been relevant six months ago in a previous evaluation round. However, employees might have changed assignments since then, but the old relationships remain in the database with a possibility to become selected for the next evaluation round as well. If the two employees have not worked together in six months, they may not be able to provide much new feedback to each other since the last evaluation round.

4.1.5 Platform dependent black-box algorithm

There were also some issues with the maintainability of the matching algorithm. The library used to implement it, GLPK, consist of pre-compiled callable binary files. This made the matching algorithm, and thus the entire CareerLight system platform dependent, since different server operating systems require different binary library files. Another issue with GLPK of even more importance is that there is no possibility to control what happens between passing the input to GLPK and receiving the optimized result at the other end.

4.1.6 No support for CCB-matrix feature

The new CCB-matrix feature should enable the matching algorithm to assign different evaluation depths to evaluation pairs, depending on the strength of their relationship. The existing algorithm was implemented without consideration to this feature, and its black-box behavior makes it difficult to adapt to this new feature.
4.2 System requirements

Below is a list of requirements that were concluded from the list of problems presented above.

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Iterative runs</td>
<td>Administrators should be able to lock certain pairs in a matching and then re-run the solver again to match the rest of the pairs to make sure that the constraints still hold.</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Relevancy</td>
<td>The input graph should be extended with more arcs to increase the completeness of the matching result. These arcs shall be relevant, using some heuristic rather than being completely random.</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Speed</td>
<td>The existing matching algorithm has an acceptable running time of approximately 1 minute when the input has about 400 employees. This time should not be exceeded too much. Up to 10 minutes may be accepted.</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Transparency</td>
<td>The system should be able to give a report on how it came to its result.</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Matching score</td>
<td>Matching results should be delivered together with a matching score that gives a hint of the quality of each matching pair.</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>Variability</td>
<td>It should be possible to add variety to matchings between evaluation rounds. The main goal is to make sure that employees won’t get evaluations from the same authors every time.</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>Portability</td>
<td>The matching algorithm should be platform independent, so that it can easily be installed on new hardware.</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>CCB-matrix support</td>
<td>The matching algorithm should have built-in support for assigning different CCB-matrix sizes to matches arranged by their match score.</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4-1: The identified requirements of the system.

All requirements provided in Table 4-1 except requirement 3 have been assigned priority level 1. Priority level 1 means that this requirement must be fulfilled in the final solution. Requirement number 3, Speed, has been assigned priority level 2, which means that this requirement is not of great importance, but it should be kept in mind during the implementation.

4.3 Guiding questions

Given the above requirements, the purpose of this thesis project can be formalized to answer the following questions:

- Has any previous work been done regarding matching of incomplete graphs that could provide useful hints on how to address the problem at hand?
- Is there a Recommender System technique in existing research that can be used to complete the input graph?
- Are there any additional data sources that can be mined to add information to the input graph, and how do different data sources relate to each other in terms of estimating the quality of a matching pair?
- How can the transparency of the system be increased to satisfy the users’ needs?
- Are there any available optimization frameworks that are completely platform independent that can be used to replace the current platform dependent solver, and would they introduce any increase in computation time?
- Are there any existing scientific methods to measure the performance of a new implementation of a matching system compared to the current matching system?

These questions have been formulated in such a way that answering them will automatically satisfy the specified requirements of the system.
Improved algorithm for weighted matching of employees
Chapter 5

Solution alternatives

Looking at the system requirements table presented in Section 4.2 it becomes clear that there are many areas within the CareerLight system that can be improved. From the list of problems in Section 4.1 it can be concluded that 4.1.1 Insufficient input data and 4.1.2 No possibility to reiterate the match result are consequences of one base problem, that the relationship information is too scarce. Solving this problem, by adding additional relationship information, is expected to reduce the negative impacts of the two sub-problems as well. Looking at additional data sources for ways to improve the matching input was one of the two main tasks that were given in the thesis description and hence, much effort has been put to this task.

The second task given in the thesis description was to look for ways to improve the matching algorithm itself, which relates to solving the sub-problem 4.1.5 Platform dependent black-box algorithm. Categorizing the identified problems from Section 4.1 into the two main tasks we get:

- Improving quality of input to the matching algorithm
  - Insufficient input data
  - Time consuming manual work
  - Little variation between evaluation rounds
- Improving the matching algorithm
  - No possibility to reiterate the match result
  - Questionable relationships
  - Platform dependent black-box algorithm
  - No support for CCB-matrix feature

The solution alternatives that were considered for implementation are presented in this chapter. In the first part I present the alternatives that were discarded and the reason why. Then I move on to describe the solution alternatives that were implemented, and why they were chosen.

5.1 Discarded solution alternatives

There was not enough time to implement and test all of the considered solution alternatives within the scope of this thesis, so a selection of the most promising alternatives was made. Some of the alternatives that were considered but later discarded for some reason are presented below, categorized by which type of problem they would solve.

5.1.1 Improving quality of input to the matching algorithm

The discarded solution alternatives listed in this section address the task of improving the matching input.
5.1.1.1 SONAR data sources

The SONAR architecture for extracting social relationships (see Section 2.4.1) was selected for implementation because of its similarities with the already existing system. It proposes a method to extract data from a dynamic number of social data sources, and IBM's results from using SONAR indicates that this architecture has the potential to solve the problems presented in this thesis. An initial brainstorming session produced a list of several potential data sources that could be included to produce relevant SONAR relationship results. It turned out later however that most of these data sources could not be used for various reasons. The data sources that were discarded are listed below.

Facebook

Facebook is a popular and well known social media platform. It was expected that a high percentage of the employees at Netlight would be registered users and at least fairly active on Facebook. Accessing the Facebook API could turn out to be a gold mine in terms of finding additional personal relationships within the company.

This data source was discarded however because of privacy issues. Accessing an individual’s personal social network through the Facebook API requires asking them for their personal Facebook credentials. A discussion with an employee with a higher position at Netlight revealed that this might increase suspicion against CareerLight amongst its users, and might lead to a decline in the number employees using the system.

LinkedIn

LinkedIn is a widespread social media platform that specializes in professional relationships. Many of the employees recruited by Netlight were found by the recruitment team through their profiles on LinkedIn. This could, just like Facebook, turn out to be a valuable source of relationship data.

This source was discarded because of the same reasons as with Facebook. Including LinkedIn would require the users to log in with their personal account credentials, which could potentially harm CareerLight.

Outlook365

Outlook365 is a service provided by Microsoft. It offers both e-mail and calendar functionality, and is used within Netlight as the main e-mail and calendar provider. Accessing Outlook365 could provide useful information from mailing-lists, booked meetings and sender and recipient information on individual e-mails.

Due to the same privacy issues as described above, this data source also had to be discarded. Requesting access to someone’s e-mail is a sensitive matter, and it would be near impossible to convince every user that CareerLight has no interest in the contents of their e-mails.

SocialLight

SocialLight is an application for smartphones developed by Netlight to make it easier for employees to find contact information to other employees. The application allows the user to mark certain employees as favorites, and the initial idea was that these favorite lists could be used as relationships for the matching algorithm.

An initial investigation revealed that the favorite lists are not stored in a central database that CareerLight could access, but instead cached locally on the users smartphone. To be able to make use of the favorite lists of SocialLight, it is necessary to re-design SocialLight so that CareerLight can access the information.

Slack

Slack is a web based social application for written communication in real time. Users can create chat groups that focus on special topics. The message logs of these chat groups are limited to a maximum
number of messages, and when the limit is reached, the oldest messages are deleted to make room for new messages. This makes the message logs unreliable as a source of social interaction data. However, it is possible that users that are registered to many common chat groups have common interests, which may in turn imply that there exists a social relationship between them. Information on chat group memberships is public information for anyone connected to the Netlight network on Slack, so it would not be necessary to ask for credentials.

This data source was discarded because it was introduced at Netlight too recently, and there are still too few users to deem this data source lucrative. Its popularity among Netlight employees is steadily increasing however, and it may be relevant for investigation in the future.

### 5.1.1.2 Neural Network based weight aggregator

Implementing a data mining technique like SONAR to obtain data from several different data sources requires a way to combine the output into a result that makes sense for the given context. The objective for this thesis is to extract social interaction data from several social media platforms that can be used to estimate a matching score between two individuals, i.e. estimate the strength of the social relationship between them. The team from IBM that implemented a SONAR API [12] calculated a weighted average of the outputs from their different data sources in order to obtain an estimation of the relationship quality between two individuals. The weights were configured manually through a setting-page that every individual user had access to. This method is similar to the aggregation technique already used within CareerLight, which uses weights that are manually configured by an administrator, and calculates a linear combination of the social interaction data and their configured weights.

One of the ideas behind implementing the SONAR solution for CareerLight included adding the possibility to extend the pool of data sources by adding new data providers without needing to re-deploy the entire CareerLight system. These providers would implement a common interface and their task would be to connect to a certain source of social interaction data. This would allow Netlight to improve the matching input continually whenever a new source of social interaction data was discovered.

Implementing a SONAR system with modular data providers would require a way to dynamically adjust the weights. An administrator could of course manually adjust the weights from a dynamic list of all the included modules, but that would lead to an additional step of manual work. Another idea was to implement a method that would calculate the optimal weights automatically using training data from previous evaluation rounds. An article by a group of ecological modelers [22] points out that a neural network can be used to quantify the importance of variables (an introduction to neural networks can be found in [28]). The main idea is that a neural network with a single input layer, no hidden layers and a single neuron in the output layer can be trained to estimate the value of a linear combination using a data set where the input variables and linear combinations are known. While training, the neural network adjusts the weights used in the linear combination to match the training data. In CareerLight, it would be possible to use the different relationship types as training variables, and wishes from previous evaluation rounds as desired linear combination outputs, as they reflect the “true” matching preferences of the employees. The network would then be trained until the error of the estimated wishes converges within a satisfactory threshold. The resulting vector of weights would indicate the importance of the different relationship types used as input.

A Java-based neural network framework was found that provided all required functionality to implement the neural network. The idea was that the data from the different data providers would serve as input to the neural network. The neural network would then be trained to output whether two employees would make wishes for each other or not, using data from previous evaluation rounds.

In the end, this idea had to be discarded. Discussions with supervisors both from LiU and from Netlight made it clear that too much risk was involved. Their experience with neural networks was that they are hard to get to work the way intended, increasing the risk of losing time on implementing something that might not work. It is also difficult to backtrack the calculations from the input to output, so a neural network would impose yet another “black-box” element to CareerLight.
5.1.2 Improving the matching algorithm

Considered solution alternatives that would improve the matching algorithm but were discarded are listed in this section.

5.1.2.1 Constraint satisfaction framework OptaPlanner

During the search for alternative optimizer frameworks, a completely Java-based constraint satisfaction solver framework called OptaPlanner was found. The framework promised a wide application area because it allows the programmer to implement their own problem model, planning entities, score calculation methods and difficulty comparators for the entity selection process. An initial implementation showed some promising results on a small test input graph with five nodes and fifteen arcs between them. The theory behind OptaPlanner’s workflow is described below, followed by a description of the initial implementation.

Workflow

OptaPlanner is a planner that attempts to maximize the score of a planning solution during a fixed amount of time. A planning solution contains a set of planning entities that can be any primitive data types or objects, which must be selected from a pre-defined list of planning entities. The entire set of planning entities must be instantiated, i.e. have defined values, in order for the planning solution to be complete. To find a complete solution, the planner attempts to instantiate one planning entity at a time by selecting entities from the pre-defined set that produces the highest score.

OptaPlanner requires a configuration file where additional parameters can be specified. Parameters of interest are:

- SecondsSpentLimit – The number of seconds the planner is allowed to run before terminating.
- ConstructionHeuristicType – The algorithm used to select which planning entity to instantiate.
- Entity/solution tabu size/ratio – Allows to specify a size of entities and solutions to mark as tabu. This means that once tried, these entities and solutions will not be attempted again. Setting a tabu size can be useful as it can help the planner to get out of local maxima.
- Forager acceptedCountLimit – A forager can be used to set a maximum number of entities to evaluate in each iteration. This helps to speed up the planning of large problems with many planning entities.

Implementation

To solve the matching problem of this thesis, a matching algorithm was implemented using the OptaPlanner framework. A planning solution was implemented modeling the desired output graph $G_M$ (see Section 3.6.2) consisting of a fixed set of nodes and a set of planning entities modeling directed arcs between nodes. The set of nodes was mapped to the list of all participants of the evaluation round, and the set of possible planning entities (arcs) was mapped to all the possible relationships found by the relationship method.

A score calculation function was implemented so that if the number of outgoing arcs was less than or equal to the $N_{WRITE}$ constraint the weight of the outgoing arcs would be added to the total score. If the number of outgoing arcs exceeded $N_{WRITE}$ or the number of incoming arcs was less than $N_{RECEIVE}$ the total score would be reduced by the magnitude of the constraint violation.
In the configuration file, the following parameters were set:

- SecondsSpentLimit: 2
- ConstructionHeuristicType: FIRST_FIT_DECREASING
- EntityTabuRatio: 0.2
- SolutionTabuSize: 1000
- AcceptedCountLimit: 1000

The first fit decreasing heuristic utilizes a greedy method to select which planning entity to instantiate. It organizes all planning entities in decreasing difficulty, and then evaluates each one in turn for the best score until the AcceptedCountLimit is reached. In this context, difficulty is a measure of how hard the constraints for a certain entity are to fulfill. In order to make use of the first fit decreasing algorithm, which always instantiates the hardest entities first, a comparator class had to be implemented that could compare the difficulty of two arcs. In my implementation the comparator calculated the sum of the number of incoming arcs to the destination node and the number of outgoing arcs from the start node. A high sum would imply low difficulty, and a low sum would imply high difficulty. Consider an arc going from a node with only one outgoing arc, to a node with only one incoming arc. The number of ways to satisfy the \( N_{\text{WRITE}} \) constraint for the start node and \( N_{\text{RECEIVE}} \) constraints for the end node would in this case be very limited, if not impossible, marking this arc as difficult.

The reason to why this implementation was discarded was because of lack of time. The implementation described above managed to find the optimal matching of an input graph with 6 nodes and 18 arcs in 0.4 seconds. For the test both \( N_{\text{WRITE}} \) and \( N_{\text{RECEIVE}} \) (see Section 3.6.2) were set to 2. When the test graph was exchanged with a copy of the real input graph from the most recent evaluation round with 374 nodes and 3773 arcs problems started to emerge. After about 25 seconds the solver seemed to get stuck in a local maxima that did not even satisfy all constraints. First, the algorithm was put on the shelf for later debugging but when the time for implementation started running out a decision was made to discard this solution entirely to assure that at least one constraint satisfaction framework could be implemented completely.

## 5.2 Implemented solution alternatives

The solution alternatives that were implemented and validated are presented in this section. The solutions are divided into sub-categories based on which of the two main tasks they address.

### 5.2.1 Improving quality of input to the matching algorithm

This section describes the solution alternatives that were fully implemented that aim to improve the quality of the input to the matching algorithm.

#### 5.2.1.1 Reverse wish provider

After discussing the matter of improving the quality of the input to the matching algorithm with a couple of employees at Netlight, the idea to implement a reverse wish provider emerged. The assumption is that if an employee \( A \) makes a wish for employee \( B \) to be their evaluator then obviously employee \( A \) must think that employee \( B \) knows him or her enough to write a good evaluation. This implies that employee \( A \) must possess some knowledge about employee \( B \), making employee \( A \) suitable to write an evaluation for employee \( B \) as well.

**Implementation**

The reverse wish provider was implemented according to the relationship provider standard already present in CareerLight. Figure 5-1 shows an UML diagram of this structure. All relationship provider
classes implement an interface called PossibleMatchProvider, making it possible for the relationship method to use them in a uniform manner. In addition to the ColleagueSmEsProvider and WishProvider a new provider was created named ReverseWishProvider. It fetches the wishes from the DB_CareerLight database via a WishListRepository the same way as the original WishProvider.

Let us assume we have the same employees, A and B as above, where A wished for B to be his or her evaluator. For every wish retrieved, the original wish provider creates a relationship where B is added to A’s list of potential evaluators with the weight of wishes specified in the matching configuration. The reverse wish provider takes the same wishes, but instead it adds A to B’s list of potential evaluators. The relationship weight in this case is set to the constant 1, which is approximately 2% of the original wish weight, making it the lowest prioritized relationship of all present. Since this provider is based on assumptions there is no guarantee that a reverse wish relationship is valid, so this relationship should only be used by the matching algorithm when no other options exist.

5.2.1.2 Proof of concept: Yammer provider
From the original list of possible SONAR data sources all but one had to be discarded. See Section 5.1.1.1 for a list of discarded alternatives and the reason why they did not make it into the final implementation. The solution that remained was the idea to connect to the API of a social media site called Yammer. Netlight has its own network on Yammer, meaning that the network is closed to anyone but those users connected to the Netlight network. Yammer is considered to be the official social media platform for all employees at Netlight, and it currently has 682 users registered to the Netlight network.

Yammer allows its users to create and join different discussion groups, usually based on common interests. Some topics are .Net, User experience, Equality and Humor. There is also a default group called All company, which every user automatically joins when they first register. A user can join any number of groups within the network, where they can post a text on the group wall making it visible to anyone registered to that group. Other group members have the ability to reply to wall posts, making it possible to have conversations. In addition to this, users have the possibility to like a wall post or one of its comments. This puts a small flag on the liked object, indicating that it is “liked” by that user. Users can
also choose to notify other users about their posts and comments. Some additional features include the possibility to follow other users, share files and send private messages.

Yammer comes with an API that allows developers to create external applications that allows users to interact with Yammer without using the official Yammer client.

Hypothesis

The hypothesis of this solution alternative is based on the assumption that the Yammer API can be used to extract social interaction data from Yammer that correlates with the wishes made during a wish period. The assumption is that if two employees interact much on Yammer, there may exist a real-life relationship between them as well. A real-life relationship can, in turn, imply a bigger chance of them making wishes for each other. If such a connection can be proven to exist, it will be possible to extract data from Yammer and use it to predict additional relationships for the matching algorithm input.

Netlight have explicitly expressed that the solutions emerging from this thesis should not ask users for their personal credentials for different sites. This could potentially harm CareerLight by reducing the level of trust [29] among its users. Because of this limitation, it became necessary to narrow the scope of this proof of concept to utilize only publicly available data from Yammer. A study comparing the use of SONAR with public vs. private data sources [11] conclude that public data sources more often reflect a person’s strong relationships, which is exactly what is sought after in this case.

Implementation

In order to prove or disprove whether there exists a connection between social interactions on Yammer and wishes made in CareerLight a separate program was written for this purpose. This program was written in the programming language C#, using the .Net framework to connect to the Yammer API. A complete list of the Yammer API endpoints can be found on their developer pages [34]. For the scope of this proof of concept permission was granted by Netlight to utilize an already existing Yammer application called SearchLight to extract the data that was needed. SearchLight was already connected to all discussion groups within the Netlight Yammer network, providing access to all group discussions. The program was able to extract the following information:

- A complete list of all users connected to the Netlight network. From this list it was possible to extract the name, email address, and Yammer id number of every user.
- A list of group memberships of every user.
- A list of all public messages on the Netlight network. This list contained information on the message author, whether it is a reply to another message or not (and the id of the parent message), a list of users who “liked” the message, and a list of users mentioned in the message.

The extracted data could then be summarized into a data structure that contained the following set of parameters for every possible combination of user pairs:

- Number of common group memberships
- Number of messages written in reply to the other user
- Number of replies received from the other user
- Number of liked messages written by the other user
- Number of messages liked by the other user
- Number of mentions of the other user made in messages
- Number of times mentioned in messages by the other user

This data could finally be saved in a text file for evaluation, which is covered in Section 6.2.
5.2.2 Improving the matching algorithm

This section describes the solution alternatives that were fully implemented that aim to improve the matching algorithm.

5.2.2.1 Choco constraint satisfaction solver

In addition to the discarded framework OptaPlanner described in Section 5.1.2.1, another completely Java based constraint satisfaction framework called Choco [15] was evaluated. Just like OptaPlanner, Choco promised a lot of freedom for the programmer, allowing problem specific implementations of separate parts of the problem solving process. Choco also comes with a plugin called Choco-Graph that address the issue of solving graph-based problems.

The following sub-sections describes the general workflow of the framework, how the framework parameters were configured in the implementation in this thesis and finally what the early test results showed that led to the decision to keep this solution alternative.

Choco workflow

Choco comes with a set of pre-defined data types for modeling a problem and solving it. First, a solver object must be created, and then problem variables and constraints can be attached to it in order to model an optimization problem. A call to one of the solver object’s solve methods initiates the search for either the first solution found, a list of all possible solutions or the optimal solution that satisfies all constraints attached. Individual solutions can be recreated after the search finishes so that the values of the solution variables can be read. The solver searches for a solution by assigning values to all problem variables in such a way that all constraints are satisfied.

To be able to search for an optimal solution a resolution policy and a target variable must be selected. The available resolution policies are to either maximize or minimize the target variable or just make sure that the target variable has a value that satisfies all constraints. When searching for an optimal solution, the solver iterates the search process several times. During the first iteration, the Solver searches for the first solution it can find. It then reads the value of the target variable and adds a new constraint to the problem. If the selected resolution policy is to maximize the target variable, then a constraint is added, saying that the target variable must exceed its current value. The solver then restarts the search, adding new constraints every time a higher target value is found. When no more solutions exists it stops, returning the last valid solution found.

Possible problem variable types to choose from include integers, booleans and real. Each problem variable can be limited to a certain domain of a minimum and maximum allowed value. Boolean problem variables are simply integer variables limited to the domain \([0, 1]\). Some pre-defined constraints to choose from include restricting a problem variable to take on the value produced by a mathematical operation applied to another set of problem variables or constants. For example, it is possible to include a constraint that forces a variable to take on the value of the scalar product of a vector of boolean problem variables and a vector of constant weights. The Choco-Graph plugin provides an extra problem variable type called graph variable. This variable can be configured to model either a directed or undirected graph. The graph variable consist of a number of nodes and a number of either directed arcs or undirected edges connecting the nodes. Choco-Graph also provides a set of graph constraints including setting a maximum and minimum number of predecessor and successor arcs of a given node.
Implementation: Test version

The original matching algorithm that utilizes the GLPK optimizer framework is encapsulated in a class called OptimizerFrameworkImplGLPK that implements an interface called OptimizerRepository. This made it easy to implement a new OptimizerRepository utilizing the Choco framework and switch between them. In Figure 5-2 an UML diagram of the two optimizer classes and their interface is shown.

![UML diagram of the matching algorithm implementations and their common interface.](image)

Initially a test implementation was made to make an early comparison with another planning framework called OptaPlanner in terms of performance, steepness of the learning curve and estimated workload required to implement fully. For this purpose a directed graph solver was implemented on the most canonical form possible:

- A vector of boolean problem variables was created to represent every directed arc between two nodes of the input graph.
- A vector of real values was created to hold the weights of all arcs.
- A real problem variable was created to be the target variable.
- A constraint was created that the target variable should always hold the value of the scalar product between the vector of included arcs and the vector of weights.

To implement the $N_{WRITE}$ and $N_{RECEIVE}$ constraints (see Section 3.6.2) a matrix $M_{MATCH}$ of size $m \times m$ was created, where $m$ is the total number of nodes in the input graph. Every position $(i,j)$ in the matrix was set to reference the boolean vector element corresponding to the directed arc from node $i$ to node $j$ if it existed, or zero otherwise. This resulted in a matrix where every row represented an evaluator. And the elements represented whether the evaluator $i$ was matched with evaluatee $j$ (1) or not (0). Every column would in turn indicate whether evaluatee $j$ was matched with evaluator $i$ or not. For every row in $M_{MATCH}$ a constraint was added such that:

$$\forall i \in M_{MATCH}: \sum_{j \in M_{MATCH}} M_{ij} \leq N_{WRITE}$$  
(Eq. 5-1)

For every column a constraint was added in the same way:

$$\forall j \in M_{MATCH}: \sum_{i \in M_{MATCH}} M_{ij} \geq N_{RECEIVE}$$  
(Eq. 5-2)

These two formulas states that every evaluator should be matched with no more than $N_{WRITE}$ evaluatees, and that every evaluatee should be matched with at least $N_{RECEIVE}$ evaluators.

Finally, Choco’s default search strategy was used to find the optimal solution. Using the resolution policy to maximize the target variable and with $N_{WRITE}$ and $N_{RECEIVE}$ both set to 2, it turned out that the algorithm could find the optimal solution of an input graph with 5 nodes and 15 arcs in a matter of milliseconds. However, increasing the complexity of the problem rapidly increased the solving time. 10 nodes and 30 arcs required over 3 minutes to find the first solution, and 12 nodes and about 30 arcs took more than 15 minutes. Using simple quadratic interpolation on these values it turned out that it would take approximately 83 full days to find a solution for a full set of approximately 400 participants. This time frame could not be accepted, since requirement number 3 in Table 4-1 states that the matching algorithm...
should not take more than 10 minutes. Although the documentation for the default search strategy used was vague in terms of how the strategy worked, the search strategy was identified to be the culprit. Since time complexity issues had been experienced with both OptaPlanner and Choco it was decided that only one of the frameworks should be selected for full implementation. Choco was selected because it was deemed easier to use. Its model using a solver, search strategy and constraints aligned best with the model used for the matching problem. In addition, Choco solved the test graph slightly faster than OptaPlanner, something that could potentially propagate into a huge difference when the full data set is used.

Choco-Graph workflow

A plugin for the Choco framework called Choco-Graph was used in the full implementation of the matching algorithm, and is therefore presented briefly in this section. Choco-Graph provides tools to define constraint satisfaction problems using graphs as problem variables and to enforce graph constraints such as maximum and minimum number of connected arcs to each node. A graph problem variable consists of a set of nodes and arcs that represents a sub-graph between a minimum domain graph and a maximum domain graph that must be specified explicitly. When searching for a solution, the Choco-Graph solver starts from the minimum domain graph and then includes one arc at a time until either all constraints have been satisfied or the maximum domain has been reached.

The execution time of the Choco-Graph solver depends heavily on the search strategy used to select which arcs to include. A search strategy is a function that uses a problem specific heuristic provided by the programmer to select the next arc to include based on the set of currently included arcs and the set of possible arcs. To select which arc to include next, every search strategy must specify a starting node and an end-node that are connected with an arc from the set of possible arcs. The configuration of the Choco-Graph plugin and the implementation of the search strategy used in this thesis are described in the next section.

Implementation: Full algorithm

For the full implementation of the matching algorithm, a graph problem variable was created to hold the final result graph $G_M$ according to the model described in Section 3.6.2. Its minimum domain was specified to be the set of arcs representing all locked evaluation pairs according to requirement 1 in Table 4-1, and the maximum domain graph was set to be the graph of all possible evaluation pairs, $G_{PE}$ (see Section 3.6.2).

The search strategy was implemented to start searching for the end node that had the fewest included incoming arcs. If there were more than one such node, the node with the fewest possible incoming arcs was selected. After selecting the end node, the strategy moved to search among its possible incoming arcs for the start node that would lead to the highest weight.

Finally Choco was configured to maximize a target variable that was constrained to always hold the weight sum of the included arcs of any valid graph solution. A number of test runs were made to verify the implementation using the full input graph of approximately 400 participants. The tests revealed that the Choco-Graph solver started to find the first solutions that satisfied the constraints in a couple of seconds. It then gradually found better solutions with higher weight sums for approximately 25 seconds before it stagnated and had to be terminated. A thorough evaluation of the performance of the matching algorithm is described in Section 6.1.
5.2.2.2 CCB-matrix support

The CCB-matrix feature should allow different evaluation sizes to be assigned to evaluators, depending how strong their relationship with the evaluatee is according to requirement 8 in Table 4-1. The different evaluation sizes are called CCB-matrices (see Section 1.2) and consists of three sizes; large, medium and small. According to the requirement specification however, the matching algorithm should be able to handle any number of sizes.

Implementation

The first step taken in the implementation of the CCB-matrix feature was to divide the matching process into several iterative runs, one for every CCB-matrix size, and instead of requiring a maximum and minimum number of evaluations written and received in total, each CCB-matrix size was configured to have its own minimum and maximum constraints, e.g. so that every employee would receive at least one large evaluation and at least two medium sized evaluations. Support for locking pairs and re-running the algorithm had already been implemented (see Section 5.2.2.1) and could be utilized to match each CCB-matrix size separately. The administrator backend of CareerLight was slightly altered so that it would first run the algorithm using the constraints of the large CCB-matrix configuration, since this evaluation size required the strongest relationships. When a solution was found by the matching algorithm, all the evaluation pairs included in the solution were assigned with the large CCB-matrix size. These pairs were then locked, and the minimum and maximum constraints of the next CCB-matrix size were added to the previous constraints. Then the matching algorithm was initiated again to search for a new solution. This time the result would contain both the pairs with large evaluations assigned to them and a set of new pairs that could then be assigned with the medium CCB-matrix size. This process was repeated until all CCB-matrix sizes had been assigned.
Improved algorithm for weighted matching of employees
Chapter 6

Results

After the implementation of the solution alternatives presented in Section 5.2, their individual performance was evaluated in order to verify that they accomplished the tasks they were intended for. In this chapter, the results of each implemented solution alternative are presented along with the method used to evaluate their performance.

6.1 Reverse wish provider

The configuration from the live evaluation round, performed during the spring 2015, was used to evaluate the reverse wish provider. This configuration contained a total of 496 participants which translated into 496 nodes in the input graph. Using the reverse wish provider to generate the input graph of employee relationships showed that the number of possible arcs could be increased by approximately 20%. The original relationship method produced an input graph containing 3731 potential arcs between nodes. With the reverse wish provider turned on, that number increased to 4548 arcs.

Two test matchings were generated for comparison using the original algorithm implementing the GLPK framework. For the tests both the minimum receive and maximum write parameters were set to 4. Using an input graph created with the original relationship method revealed that not all evaluatees received all 4 evaluator matches. Three of them only received 3 evaluations each, and two evaluatees only received 2 evaluations. In total, 7 arcs were missing. On the other hand, running the same algorithm with reverse wishes turned on gave a complete match result, where all evaluatees had been assigned 4 evaluators.

6.2 Yammer provider

To test if correlation existed between social interactions on Yammer and the probability of the same persons wishing each other in CareerLight a separate program was written. The program first made an API call to extract 657 Yammer users that were connected to the Netlight network. The extracted Yammer user ids were then mapped to Netlight ids using a list of Netlight employees that were participants in the spring 2015 evaluation round.

There exists advanced methods for this type of generic mapping of entities from different data sources that model the same thing, but that are represented differently. One method [20] utilizes text likeness of field names, and data types of separate fields to identify and map the entities. In this case however the structure of the user objects was known beforehand, so a simple method could be used to map Yammer users to Netlight users if either their first and last names were equal or if their email addresses matched.

Out of the original 657 Yammer users, 330 could be mapped to Netlight employees participating in the evaluation round used for verification. The Yammer users that could not be mapped were then discarded. For each of the remaining users, the number of common group memberships with every other user was computed. A series of new API calls were made to extract all publicly available messages that were written between September 2014 and March 2015, which is the time period relevant for the spring 2015 evaluation.
A total of 2569 messages were extracted, but only those that were written by an identified Netlight employee in reply to the message of another identified Netlight employee could be used for analysis. Filtering the messages on this criterion left only 107 useful messages. The rest were either authored by an unidentified Yammer user, a reply to an unidentified Yammer user, or a standalone message. For every user, statistics were calculated from the information provided by the remaining useful messages:

- Number of messages written in reply to another user
- Number of replies received from another user
- Number of liked messages written by another user
- Number of messages liked by another user
- Number of mentions of another user made in messages
- Number of times mentioned in messages by another user

At this point, it became obvious that the useful information that could be extracted from Yammer was scarce, and might not be sufficient to make any good conclusions. 107 messages distributed over 330 identified users means that each user on average only authored 1/3 message. In an attempt to counter this potential problem, a decision was made to make the social interactions bi-directional. This meant that for each pair of users the following properties were computed:

- Number of exchanged messages
- Number of exchanged likes
- Number of exchanged mentions

The number of common group memberships were already a bi-directional property, so it was left untouched. The data scarcity was reduced further by only including users that both had made at least 5 wishes and that displayed at least some activity on Yammer.

Since the Yammer interactions had been made bi-directional, the training data was made bi-directional as well, by also including the reverse wish relationships. To reduce scarcity even more, the argument that also SM/ES/Colleague relationships (see Section 3.5.1.1) hold for “true” relationships finally resulted in including the full matching score between two employees as training data. The data set was randomly divided into two equally large chunks, one used to find weights and the other used for verification. The correlation between the extracted interactions from Yammer and the “true” relationships in the form of wishes was computed using multiple linear regression, see Section 2.5. Using Eq. 6-1 a vector $\beta$ was obtained, containing a bias value, $\beta_0$, and the weights of each Yammer property in $X$ that best approximates their corresponding match value in $Y$.

$$\beta = (X^TX)^{-1}X^TY$$ (Eq. 6-1)

The bias and weights were then applied to the Yammer properties in the verification set to obtain a vector, $Y'$, of estimated match values. To measure the accuracy of the estimated weights the root mean squared error was computed using Eq. 6-2:

$$E_{RMS} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - Y'_i)^2}$$ (Eq. 6-2)

In this equation $n$ is the number of user relations in the data set, $Y$ is the vector of match values from the verification data set, and $Y'$ is the vector of the estimated match values. The match values in the training data set $Y$ are the same values as the estimated match quality values of the relationships used as weights for the arcs in the graph model. The configured relationship weights used when retrieving the match values were as following:

- Solution manager: 50
- Engagement search: 50
- Colleague: 30
- Wish: 50
- Reverse wish: 25
The match value between every user pair was either 0, if no relation existed at all, or on the range \([0, 205]\) depending on the strength of the relationship. To ensure that the data was not split into particularly favorable or unfavorable chunks by chance, a series of different test computations were made, randomly distributing the data into different chunks each time. The computed \(\beta\) vector and the \(E_{RMS}\) of each computation are shown in Table 6-1.

<table>
<thead>
<tr>
<th>(\beta_0) (bias)</th>
<th>(\beta_1) (common group)</th>
<th>(\beta_2) (likes)</th>
<th>(\beta_3) (notifies)</th>
<th>(\beta_4) (comment)</th>
<th>(E_{RMS})</th>
</tr>
</thead>
<tbody>
<tr>
<td>13,28143</td>
<td>-2.42808</td>
<td>1.263508</td>
<td>15,7257</td>
<td>-4.97105</td>
<td>33,03623</td>
</tr>
<tr>
<td>13,1720417</td>
<td>-2.212805985</td>
<td>2.089430107</td>
<td>22,44268242</td>
<td>-5.908753101</td>
<td>31,8770077</td>
</tr>
<tr>
<td>13,35464796</td>
<td>-2.182317702</td>
<td>-0.71694546</td>
<td>18,47922962</td>
<td>-6.060369603</td>
<td>31,12320595</td>
</tr>
<tr>
<td>14,40728796</td>
<td>-3.050553863</td>
<td>-0.19341335</td>
<td>19,26739735</td>
<td>-3.736920179</td>
<td>33,47815876</td>
</tr>
<tr>
<td>12,9469667</td>
<td>-1.934927004</td>
<td>-0.548312568</td>
<td>8,437406683</td>
<td>-2.812854292</td>
<td>32,95224409</td>
</tr>
</tbody>
</table>

Table 6-1. The resulting weights and the computed root mean square error of 5 computations. Between each computation, the data set was split differently using a random selection of user pairs.

Plotting the data from the top-most weight computation produces Diagram 6-1. Each user pair is represented along the x-axis, and the y-axis represents the retrieved or estimated match score.

![Yammer match estimation results](image)

Diagram 6-1. A diagram showing the data from one weight computation. The data is sorted on the blue match score curve. The orange curve shows the estimated match score, and the thick yellow curve marks the root mean squared error of the entire data set. The gray curve shows the raw data used, i.e. the accumulated Yammer activity registered.

It is evident from Diagram 6-1 that the estimated match score does not show any tendency to follow the retrieved match scores, and the conclusions that can be made from this diagram are discussed further in Section 7.1.2.
6.3 Choco constraint satisfaction solver

To evaluate the new implementation of the matching algorithm two surveys were performed. The first survey was directed at the administrators of CareerLight and focused on evaluating the usability of the new functions and estimating if the new implementation would lead to less manual work. The second survey was directed at the matching participants, focusing on evaluating the match quality of the new algorithm.

6.3.1 Administrator evaluation

For the administrator evaluation a meeting was held with one administrator and the current lead developer of CareerLight, who had been working with the CCB-matrix extension. A couple of small changes had been made to the administrator GUI parts of CareerLight so that it would be relatively easy to demonstrate the new functionalities. A field was added in the matching configuration step to configure the timeout of the matching algorithm, as shown in Figure 6-1:

![Figure 6-1.](image)

In the matching configuration step of the administrator GUI a field was added to make it easy to change the timeout limit of the matching algorithm.

In the matching step, a small button was added to initiate a re-generation of the match result with respect to locked pairs, and for every included relationship in the matching, a small checkbox was added to lock that relationship. If a new relationship was added manually it would automatically be locked to ensure that the relationship would still be present after a re-generation. The changes to the matching step can be seen in Figure 6-2.
During the meeting a small presentation of the features of the new implementation was held followed by a demonstration. Then the new implementation was discussed, focusing on the topics completion time, workload and usability. The conclusions from this discussion are presented below:

- Both agreed that the reverse wish relationship was very useful. They liked the idea that if two employees had wished for each other their wishes would be more likely to be chosen than regular one-way wishes.
- The running time of the algorithm was acceptable. Letting each of the three CCB-matrix optimizations run for 30 seconds, resulting in a total running time of 1.5 minutes was considered well within the requirements.
- For the demonstration, the large CCB-matrix configuration was to assign one evaluator to every evaluatee, and to allow a maximum of two evaluations written by each evaluator. Looking at the match results, it turned out that almost half of the evaluators had to write two large evaluations, while the other half did not have to write any at all. Here it was agreed that balancing the workload as much as possible was important, and it would not be possible to use the match result produced during the demonstration in its current state. This topic is discussed further in Section 7.1.4.
- Both attendants agreed that the completeness of the match result seemed better than before and that every improvement of the match result would lead to reduced workload for the administrators.
- The administrator attending the meeting had not been present during the forming of the requirement specification. He had an opinion about the workflow of the matching process that instead of manually adding new pairs, locking them and then regenerating a new matching, he...
wanted to see a result that started with too many relations, then manually reducing them instead. This topic will be discussed further in Section 7.1.3.

6.3.2 Participant evaluation

The participant evaluation was performed by first producing a matching using the same prerequisites as the live matching used during the evaluation round of spring 2015. The constraints were configured as displayed in Table 6-2.

<table>
<thead>
<tr>
<th>CCB-matrix: Large</th>
<th>CCB-matrix: Medium</th>
<th>CCB-matrix: Small</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max evaluations to write</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Max evaluations to receive</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 6-2. A table showing the constraint configuration of the matching produced for the participant evaluation.

A small program was written that would produce a questionnaire for every participant of the matching showing an individual list of their matched evaluatees that they would have to evaluate, and their evaluators that would write evaluations for them. In Figure 6-3 an example of a generated questionnaire is shown.

Figure 6-3. A sample questionnaire used for the participant evaluation.

An e-mail was sent to all employees at the Stockholm office asking for volunteers to participate in the survey. 48 employees replied that they wanted to take part and their personal questionnaire was sent to them. Half of them, 24, replied with answers to the questions asked in the questionnaire. All responses
were written in free text, so to summarize them, the opinions expressed in answer to each question were categorized and counted. Table 6-3 to Table 6-8 shows the opinions expressed in reply to each question.

**Table 6-3.** Replies to the question “Are there any names that you don't recognize or that you think should not be on the list?”

<table>
<thead>
<tr>
<th>Opinion</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>20</td>
</tr>
<tr>
<td>One name not recognized</td>
<td>15</td>
</tr>
<tr>
<td>One maybe</td>
<td>10</td>
</tr>
<tr>
<td>A few names not recognized</td>
<td>5</td>
</tr>
</tbody>
</table>

**Table 6-4.** Replies to the question “Are there any names that you miss on this list that you think should be there?”

<table>
<thead>
<tr>
<th>Opinion</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>18</td>
</tr>
<tr>
<td>Missing a few names (list not specified)</td>
<td>15</td>
</tr>
<tr>
<td>Missing a few evaluatees</td>
<td>10</td>
</tr>
<tr>
<td>Missing a few evaluators</td>
<td>5</td>
</tr>
<tr>
<td>Missing one evaluator</td>
<td>2</td>
</tr>
</tbody>
</table>

**Table 6-5.** Replies to the question “Do you think the individuals in the left list would be able to provide good evaluations with the specified CCB-Matrix size?”

<table>
<thead>
<tr>
<th>Opinion</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>16</td>
</tr>
<tr>
<td>A few should have smaller CCB-matrix size</td>
<td>10</td>
</tr>
<tr>
<td>A few could have bigger CCB-matrix size</td>
<td>5</td>
</tr>
<tr>
<td>One should have smaller CCB-matrix size</td>
<td>2</td>
</tr>
<tr>
<td>One could have bigger CCB-matrix size</td>
<td>2</td>
</tr>
</tbody>
</table>
Would you be able to write an evaluation with the specified CCB-Matrix size for each individual in the right list?

- Yes
- A few should be smaller CCB-matrix size
- A few could be bigger CCB-matrix size
- One could be bigger CCB-matrix size
- One should be smaller CCB-matrix size

Table 6-6. Replies to the question “Would you be able to write an evaluation with the specified CCB-matrix size for each individual in the right list?”

Do you feel that this matching is accurate?

- Yes
- The assigned CCB-Matrix sizes were not good
- The names were relevant
- Better than the existing algorithm

Table 6-7. Replies to the question “Do you feel that this matching is accurate?”
6.4 CCB-matrix support

The implemented CCB-matrix matching was evaluated by making a couple of test runs. The algorithm distributed the matchings according to the specified max write and max receive parameters, and by looking closer at every evaluatee, there was a trend that the matches with the highest match score were assigned the largest CCB-matrix sizes.
Improved algorithm for weighted matching of employees
Chapter 7

Discussion and conclusions

In this chapter, the results that were presented in Chapter 6 are analyzed and discussed. We will see how the results satisfy the requirements from Section 4.2, and answers to the guiding questions from Section 4.3 is provided. Finally the results from this thesis are concluded along with a recommendation for future work.

7.1 Discussion

In this section the results from each of the implemented solution alternatives are discussed and related to their corresponding requirements.

7.1.1 Reverse wish provider

The evaluation of the reverse wish provider showed that the number of arcs were increased by 20%, and that the completeness of the match result was increased. Both the attendants of the administrator survey and individual Netlight employees have expressed that the reverse wish provider was a very good idea. The reverse wish provider could potentially introduce some uncertainty, since the reverse wishes have not been made by the employees explicitly. Looking at Table 6-3 we see that there exist a few occurrences where the participants does not recognize one or a few names on their match result. A closer look reveals that in none of these cases their relation is based on reverse wishes. Some of them are solution manager and colleague relationships, and in one case the participant did not recognize a name that he himself had wished for. Since no complaints were raised regarding the reverse wishes it seems like the hypothesis holds, that wishes can be reversed to increase the quality of the input graph.

Relating to the specified requirements, it can be argued that the reverse wish provider fulfills the following requirement from Table 4-1:

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Relevancy</td>
<td>The input graph should be extended with more arcs to increase the completeness of the matching result. These arcs shall be relevant, using some heuristic rather than being completely random.</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 7-1. The requirement discussed in conjunction with the implementation of the reverse wish provider. See Table 4-1 for the complete list of requirements.

7.1.2 Yammer provider

The proof-of-concept study of using the social media platform Yammer revealed that the amount of useful data that could be extracted was too scarce. The results of the study shows that we receive a bias weight
of about 12 that is slightly larger than the average of the training values, which is approximately 11. This implies that the best approximation of the match values is obtained by guessing a value close to the total average of the training data, reducing it slightly whenever activity from Yammer exists. Looking at Diagram 6-1, it becomes clear that the estimated match values does not correlate well with the real match values. The raw data seems to be scattered evenly across all employee pairs, and the fact that some of the pairs have high match values is not reflected in the Yammer activity data.

The hypothesis for the Yammer provider was that there might exist a connection between social interactions on Yammer and the match score produced by CareerLight’s relationship method. This hypothesis was based on the assumption that employees with high match scores has some sort of professional relationship e.g. colleagues or solution manager. Working together requires some form of communication, and since Yammer is just a platform for communication within the company, it was assumed that this might have been reflected in the data from Yammer. The results from this thesis indicate that this hypothesis has been proven wrong, since Yammer seems to better reflect the employees’ more general social relationships within the company rather than the more specific, professional ones suitable for CareerLight.

### 7.1.3 Choco constraint satisfaction solver

The main reason behind implementing a new constraint satisfaction solver for the matching algorithm was to remove platform dependency and increase maintainability by making the source code more accessible. The implemented algorithm satisfies both of these criteria as it is completely Java based, and its source code is integrated with the rest of the CareerLight source code.

An opinion emerged from the administrator survey that the feature of locking pairs for re-generation of the matching might be redundant. The administrator thought that the best way to work with the matching would be to remove excessive matched pairs from a too big matching instead of adding new pairs to a matching that is lacking connections. This may be the result of individual preferences between administrators. The locking feature was part of the initial requirement specification and is also part of the solution for the CCB-matrix addition, where it still serves a purpose. Aside from a small bug found during the demonstration, the locking feature works as intended and can be argued to fulfill requirement 1 – Iterative runs.

The participant study produced many interesting results. Overall, the participants seemed satisfied with their matchings. In a few cases they reported that they were unsure or did not recognize one or some of the names on their list. A closer look revealed that in none of these cases the new reverse wish relationship was involved. Instead it turned out that these relations usually were old colleague- or SM-relationships. A solution to the problem might come with fulfilling the requirement 6 – Variability requirement, as the relationships might have been more relevant during an evaluation round six months prior to this one. Due to time limitations however, this requirement could not be fulfilled.

Some participants reported that they were missing one or a few names in their matching. This is understandable, since the max write constraint causes restrictions that makes it impossible to assign everyone’s first choices. 18 of 24 participants thought that the matching was accurate and there was even one participant that explicitly wrote that the new matching was better than before.

The final, open question provided many interesting thoughts and ideas. Many participants thought that the matching is bigger than before, i.e. that the list of names is longer. This can be argued to be the result of the new CCB-matrix system, where everyone can write up to 10 small evaluations. Others thought that the matching was similar to the matching they received in the evaluation round of spring 2015, which can be considered a positive result. The new implementation of the algorithm reduces many other problems like platform dependency while producing matchings that are at least similar to the previous algorithm. Many expressed a desire for the algorithm to take previous evaluation rounds into consideration, and since it could not be implemented within the time frame set up for this thesis, this implementation is left as a recommendation for future work.
The running time of the algorithm, 90 seconds for a full matching, is well within the acceptable range. Although the running time has increased by 50%, from 60 to 90 seconds, requirement 3 – Speed can be considered fulfilled.

Regarding requirements 4 and 5 no improvements have been made in the algorithm per se. CareerLight already contains the required functionality to fulfill these requirements. The relation property of the Evaluator class (see Figure 3-12) already contains the reasons for every evaluator-evaluatee relationship. This can be used to fulfill the requirement 4 – Transparency, and is left as a recommendation for future work. The matching score has not been removed from the existing solution, so requirement 5 – Matching score can be considered fulfilled.

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Iterative runs</td>
<td>Administrators should be able to lock certain pairs in a matching and then re-run the solver again to match the rest of the pairs to make sure that the constraints still hold.</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Speed</td>
<td>The existing matching algorithm has an acceptable running time of approximately 1 minute when the input has about 400 employees. This time should not be exceeded too much. Up to 10 minutes may be accepted.</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Transparency</td>
<td>The system should be able to give a report on how it came to its result.</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Matching score</td>
<td>Matching results should be delivered together with a matching score that gives a hint of the quality of each matching pair.</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>Variability</td>
<td>It should be possible to add variety to matchings between evaluation rounds. The main goal is to make sure that employees won’t get evaluations from the same authors every time.</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>Portability</td>
<td>The new system should be platform independent, so that it can easily be installed on new hardware.</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 7-2. The requirements discussed in conjunction with the implementation of the Java based matching algorithm. See Table 4-1 for the complete list of requirements.

7.1.4 CCB-matrix support

A few questions were raised in the participant survey aimed at evaluating the CCB-matrix implementation. 75% of the participants thought that they would be able to write evaluations of the specified sizes provided in the survey. 25% of the participants reported that there was at least one or a few names that should be assigned a smaller size, which is the most critical issue. If an employee is considered to be able to write a large evaluation for someone, they should be able to write a small evaluation for the same person as well. On the other hand, if someone is considered to be able to write a small evaluation for someone, they may not know enough about that person to write a large evaluation. A little less, 62.5% thought that their assigned evaluators would be able to write accurate evaluations of their assigned size.

The fact that a few participants thought that some of their assigned CCB-matrix sizes could be bigger, is not a direct problem. However, this might indicate that the assignment process can be done differently to reduce the number of assigned evaluations that should have been smaller. Some participants even replied with the question “Shouldn’t I write more medium/large evaluations?”.

Regarding the distribution of large evaluations among the evaluators, where almost half of them had to write two large evaluations and the other half none, it can be argued that this was a perfectly valid result given the constraints used for the demonstration. The algorithm was configured to assign at most one evaluator to every evaluatee, and at most two evaluatees to every evaluator. After the survey a test run was made with the max write constraint lowered from 2 to 1. The algorithm actually managed to assign one evaluator to every evaluatee, and the distribution was much better. Some evaluators still did not write any large evaluations since there are more evaluators than evaluatees, but no one had to write more than one large evaluation.
7.2 Answering the guiding questions

In Section 4.3 a number of guiding questions were presented that would work as a guideline for the literature study and implementation of this thesis. The answers that were found during the course of this work are provided in this section.

7.2.1 Has any previous work been done regarding matching of incomplete graphs that could provide useful hints on how to address the problem at hand?

The answer to this question is yes, but the problem must be divided into two sub-problems. The first sub-problem is to find a way to approximate the missing arcs of the incomplete input graph. The next sub-problem is to find a good matching algorithm. A Recommender System technique can be used to approximate the missing links, see the answer to the next question.

There exist a lot of research that address different aspects of graph matching. One article [33] describes a method to approximate a perfect matching with a total cost that is under a certain threshold. Their method is limited to pairing 2n nodes of a non bi-partite graph into n pairs. Another article [21] describes how a social network can be modeled as a graph to find patterns that can be used to conclude additional information of the social network. Most literature on matching that was found seemed to address some version of the stable roommates problem [32], matching an even number of nodes into a set of pairs. This was not sufficient for the purpose of this thesis, where each node must be matched with several others. The already present idea to utilize a constraint satisfaction technique seemed to be the best solution.

7.2.2 Is there a Recommender System technique in existing research that can be used to complete the input graph?

At the company IBM this has been a hot topic for the last couple of years. They implemented a data mining framework called SONAR [12] that satisfies much the same needs as those presented by Netlight. In another article [4] they evaluated different techniques to extend a social network and reached the conclusion that the SONAR technique was best at finding missing connections to persons that were already known to the subject in real life but missing in the network. Since a similar technique was already present in CareerLight, using different relationship providers to conclude a total strength of a relationship, the SONAR technique was selected as basis for the work done in this thesis. In the study they evaluated another method called the friend-of-a-friend algorithm. It did not perform as well as the SONAR technique in regards of finding known relationships, but it performed well enough to be interesting for future work.

7.2.3 Are there any additional data sources that can be mined to add information to the input graph, and how do different data sources relate to each other in terms of estimating the quality of a matching pair?

A brainstorming session produced a list of potential data sources that could contain additional information for the matching algorithm. The following list shows the initial data sources:

- Facebook
- LinkedIn
- Outlook365
- SocialLight
- Slack
- CareerLight’s wish provider
- Yammer
Mainly because of privacy issues most of them had to be discarded. Asking every user to log in to their personal accounts on Facebook, LinkedIn and Outlook365 might decrease the level of trust [29] for the CareerLight system, which in turn could lead to a reduction in active users.

The only remaining data sources were Yammer and the wish provider. Yammer was shown to be a weak source of data, and the only remaining data source that led to an improvement was reversing the wishes from CareerLight’s wish provider.

In the implementations of the SONAR technique found in the literature the different data sources were combined using a weighted average or a weighted linear combination of the data. The weights were set manually in all cases, and the best ones were found via trial and error. One article [22] hinted that a neural network could be used to find the importance of different variables. The option of using a neural network for finding the weights was briefly considered, but was discarded at an early stage because it was deemed too risky.

### 7.2.4 How can the transparency of the system be increased to satisfy the users’ needs?

Working with the CareerLight system revealed that the various relationship types used to match employees were already stored within the classes used in the matching process. Making these relationship values visible to the user of the system may be enough to answer the frequent question “Why was I matched with person X?”.

### 7.2.5 Are there any available optimization frameworks that are completely platform independent that can be used to replace the current platform dependent solver, and would they introduce any increase in computation time?

Two optimization frameworks were found that were completely platform independent. One of them, the Choco constraint satisfaction framework [15], was fully implemented and evaluated. The matching results produced with Choco can be argued to be at least as good as the original matching algorithm. The running time was increased by 50%, but was still well within the acceptable range. The new algorithm compensates this with a few other advantages over the original algorithm; aside from being completely platform independent, its source code is available for any of the developers at Netlight to modify from within CareerLight’s development environment.

The other framework that was found, OptaPlanner, showed some potential during an initial test. There was not time available for a full implementation however, so it remains uncertain how well it compares with the Choco framework.

### 7.2.6 Are there any existing scientific methods to measure the performance of a new implementation of a matching system compared to the current matching system?

Many different implementations of social Recommender Systems were found during the literature study. In some articles [26, 4] their implementation was evaluated by observing the user behavior over a time period. User activities such as connecting to new friends, writing messages and usage rates was monitored and could be used as metrics for the overall performance. In other studies [11, 24, 26] a more subjective approach was used, either combined with the data collection method or on its own, asking the users what they thought of the system and then using the topics reported by most users as a metric for the overall performance.
The latter method of evaluation was selected for this thesis. Monitoring usage statistics would have been too hard considering the fact that evaluation rounds are only held twice a year. Instead it seemed much more lucrative to ask relevant people about their opinions, both administrators and participating employees. While the data might be harder to analyze, the possibility to find unexpected issues experienced by individual users is much higher, as they might not be reflected in measured data.

7.3 Conclusions

From the evaluation of the implemented matching algorithm it seems like it is possible to use the Choco constraint satisfaction framework as a replacement of the old GNU Linear Programming Kit implementation. It removes the platform dependency imposed by GLPK, and it opens the code for easy modifications in the future. Instead of raising an error when no solution can be found, it produces a solution containing as many evaluator-evaluatee relationships as possible, which in turn ensures that the administrators always get something to work with when input data is scarce. It has a slightly longer running time, but it is still far below the acceptable threshold. The participant survey revealed that most users were satisfied with the matching results, but that something still needs to be done to increase the accuracy of the CCB-matrix assignments.

From the results of this thesis it is evident that the reverse wish provider can be used to improve the quality of the input data to the matching algorithm. It proved to increase the number of relationships between employees for the input graph, which in turn resulted in better completeness of the final match result. The concept of mining the social network Yammer for additional data was evaluated, but it turned out to be unusable. The main reason could be that the data collected was too scarce. A couple of hundred social interactions on Yammer distributed over several thousands of possible employee relationships resulted in a data set of mostly zeroes. The few cases where interactions occurred seemed to be evenly distributed over the entire data set, regardless of the matching score retrieved from CareerLight, as can be seen in Diagram 6-1. The possibility remains however, that the extracted data from Yammer combined with a couple of additional data sets from other sources might produce a result that better correlates with the match values.

7.4 Recommendations for future work

This section concludes this thesis by providing a list of topics that might be of interest to investigate in the future:

7.4.1 Consideration of previous evaluation rounds

This topic was part of the initial requirement specification, but it was never implemented due to lack of time. The participant study revealed that this topic might have been of more interest to Netlight than initially thought, and that it should have been prioritized higher than it was.

The idea is that varying evaluators between evaluation rounds would produce evaluations with more substance. There seems to be a tendency that if an evaluator writes many evaluations to the same evaluatee, they come out very similar.

Implementing this would involve finding a way to find the id of the previous evaluation round of the same type. Based on the matching used in that evaluation round, the match values of the relationships present in both evaluation rounds could be lowered a certain amount before computing the new matching. This way, previously matched pairs would have a lower chance of being selected, while they remain available if no other options exists.
7.4.2 Increasing transparency

Another topic present in the requirement specification and also reflected in the participant survey was the issue of increasing the transparency of the matching process. Some employees expressed that they would like to know why they have been matched with someone. Backend-wise this functionality is already available as a set of strings representing the different relationship types that exists between each potential pair. The administrator GUI already makes use of these explanations, as can be seen in Figure 6-1 after each name in the column labelled “Evaluator 1”.

An implementation of this feature would involve a small change in the user GUI to show the relationship types after each of the matched names, the same way it was done for the administrator GUI.

7.4.3 Fully implement OptaPlanner

The OptaPlanner framework could prove to be a faster alternative to Choco. Being fully implemented in Java, OptaPlanner comes with the same features as Choco when it comes to platform independency and configurability. Regarding the quality of the match result, I believe that is just a question of configuration independent of the framework used. The potential benefits with OptaPlanner are that it may prove to run faster than Choco.

The recommendation for implementation is to create a class that inherits the OptimizerRepository interface (see Figure 5-2) that implements the OptaPlanner framework to solve the matching problem. Comparisons with the Choco implementation regarding running time should then be made to find out which solver is the fastest.

7.4.4 Improve CCB-matrix size assignment

The participant study revealed that some employees were not satisfied with the CCB-matrix sizes assigned by the new matching algorithm. Some reported that either themselves or someone on their matched list would be able to write larger evaluations, and some reported that either themselves or someone on their list had been assigned a CCB-matrix size that was too big. The latter problem is the most serious and should be reduced as much as possible.

The last, open question of the survey produced some useful tips how to possibly solve this problem:

7.4.4.1 Restrict CCB-matrix levels to certain thresholds

One survey participant suggested that solution managers should only write small evaluations. Another suggested that large evaluations should be written only by colleagues or wishes. This would make the algorithm dependent on knowing something about the individual relationship types however, which might turn out to cause problems in the future if new relationship types from new data sources are introduced. Instead the match score could be used to assign CCB-matrix sizes. Using a CCB-matrix with 3 sizes for example, it could be possible to use a rule that allows only the top 33% of the possible matches for large evaluations, and then the top 66% for medium and finally all possible matches for small.

7.4.4.2 Possibility to specify CCB-matrix size for individual wishes

The option to specify a preferred evaluation size when making wishes could be added to allow employees to influence their match result. Restricting wishes to certain CCB-matrix sizes could prove to put too much restriction on the input graph, so the matching algorithm may have difficulties finding good solutions. Instead, the employees should be given the option to prioritize wishes differently, indirect giving them different match weights. The default “Medium priority” option could produce the same weight that was configured for wishes by the administrator. The “Low priority” option could assign e.g. 80% of the configured wish weight, and the “High priority” option could assign e.g. 120% of the configured wish weight.
7.4.5 Implement additional relationship providers

The Yammer provider alone proved to be unusable as a data source for CareerLight’s relationship method. Combining social interaction data from Yammer with data from other social media used within Netlight could still prove bear fruit however. The possibility exists that if two users consistently interact with each other across several different social media they may have an increased chance of being suitable to evaluate each other. To test this hypothesis, I suggest using the same method as described in Section 6.2 with a small difference. The different social interaction types should be categorized and clumped together, e.g. so that all messages exchanged across all social media are added together. The variable holding the amount of exchanged messages would then have bigger values for those pairs that interact across several social media.

The social media platform Slack is suitable to use as an additional relationship provider. It was excluded from this thesis because it was believed to have too few users to provide useful data. Combined with Yammer however, it might still prove to be a useful data source.

Another possible relationship provider that could prove to be useful is one that implements the friend-of-a-friend algorithm. The concept of this algorithm is that the known social network graph is analyzed to find clusters of people that form social circles within the company. These social circles can then be used to guess social connections between individuals within the circle. A simple example is could be the following:

If person $A$ has a social connection with the persons $B$ and $C$, and person $D$ has also a social connection with the persons $B$ and $C$. Then the probability exists that there exist a social connection between $A$ and $D$ as well. The more social connections $A$ and $D$ have in common, the bigger the probability is that they also have a connection. The friend-of-a-friend algorithm can be implemented in many different ways, one example is provided in [4].
References


Improved algorithm for weighted matching of employees


Index

administrator, 27
administrator GUI, 28
Agresso, 35
CareerBackend, 35
CareerLight, 12, 27
CCB-matrix, 12, 55
colleague, 30
constraint, 18
  maximum evaluations to write, 39
  minimum evaluations to receive, 39
constraint satisfaction, 19
Data mining, 22
DB_CareerLight, 35
engagement search, 30
EvalMatch, 38
evaluatee, 27
evaluation, 12
evaluation round, 12
evaluator, 27
GLPK, 38
graph, 17
  arc, 17
  bi-partite, 18
directed, 17
edge, 17
node, 17
non bi-partite, 18
weighted, 17
Integer Programming, 20
Linear programming, 18
Linear regression, 23
Netlight, 12
participant, 27
Recommender System, 21
relationship, 33
Relationship method, 37
Relationship provider, 37
  Colleague/Sm/Es Provider, 37
  Reverse wish provider, 49
  Wish Provider, 37
  Yammer provider, 50
slack variable, 19
solution manager, 30
SONAR, 23, 46
wish, 29
wish period, 29
Yammer, 50
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Copyright Emil Olofsson 2015
Presentation Date
2015-11-13

Publishing Date (Electronic version)
2015-12-10

Department and Division
Department of Computer and Information Science

Language
X English
___ Other (specify below)

Number of Pages
80

Type of Publication
X Licentiate thesis
___ Degree thesis
___ Thesis C-level
___ Thesis D-level
___ Report
___ Other (specify below)

ISBN (Licentiate thesis)
ISRN:
LIU-IDA/LITH-EX-A--15/063--SE

Title of series (Licentiate thesis)

Series number/ISSN (Licentiate thesis)

URL, Electronic Version
http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-122763

Publication Title
Improved algorithm for weighted matching of employees

Author(s)
Emil Olofsson

Abstract
This report gives the reader a detailed description of a computer engineering master thesis work done at the company Netlight Consulting AB. Netlight Consulting AB is a growing IT consulting company based in Stockholm with offices in major cities across Europe. One of their key success factors is their focus on personal and professional development amongst all employees. An essential part of this development program consists of recurring evaluation periods, where every employee receives constructive feedback from some of their co-workers. This thesis’ focus lies in improving the algorithm that organizes which employee should evaluate who. The original algorithm turned out to harbor a number of flaws, e.g. it was not always able to deliver a satisfactory matching where every participant received the minimum number of evaluations.

In this thesis a new matching algorithm has been implemented that is platform independent and that facilitates future modifications with accessible source code written in Java. The input data for the matching algorithm, i.e. the set of all potential evaluation pairs, is of importance to obtain satisfactory matching results. The number of potential evaluation pairs determines the number of possible matching combinations, which in turn increases the probability to find a satisfactory matching. In this thesis the input data has been extended by utilizing a data mining technique known as SONAR. Two different data mining sources were evaluated, and one of them is shown to extend the number of potential evaluation pairs in the matching input by 20%. Finally, a new feature to support assignment of different evaluation sizes was added to the matching algorithm.

Keywords
SONAR, Graph, Matching, Data mining, Linear programming, Linear regression, Constraint satisfaction