UPPSALA
UNIVERSITET

UPTEC IT 21020
Examensarbete 30 hp
Juni 2021

Recommendation system for job coaches

Nils Söderkvist

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Recommendation system for job coaches

Nils Söderkvist

For any unemployed person in Sweden that is looking for a job, the most common place they can turn to is the Swedish Public Employment Service, also known as Arbetsförmedlingen, where they can register to get help with the job search process. Occasionally, in order to land an employment, the person might require extra guidance and education, Arbetsförmedlingen outsource this education to external companies called providers where each person gets assigned a coach that can assist them in achieving an employment quicker. Given the current labour market data, can the data be used to help optimize and speed up the job search process?

To try and help optimize the process, the labour market data was inserted into a graph database, using the database, a recommendation system was built which uses different methods to perform each recommendation. The recommendations can be used by a provider to assist them in assigning coaches to newly registered participants as well as recommending activities.

The performance of each recommendation method was evaluated using a statistic measure. While the user-created methods had acceptable performance, the overall best performing recommendation method was collaborative filtering. However, there are definitely some potential for the user-created method, and given some additional testing and tuning, the methods can surely outperform the collaborative filtering method. In addition, expanding the database by adding more data would positively affect the recommendations as well.

Handledare: Andreas Samuelsson
Ämnesgranskare: Georgios Fakas
Examinator: Lars-Åke Nordén
UPTEC IT 21020
Tryckt av: Reprocentralen ITC

Sammanfattning

För en arbetslös person i Sverige som söker jobb är nog det mest vanliga att de vänder sig till Arbetsförmedlingen, där de kan registrera sig för att få extra hjälp med att söka jobb. För att kunna hitta ett jobb så behövre personen ibland lite extra coachning eller utbildning. Då använder sig Arbetsförmedlingen av externa leverantörer som sköter denna utbildning, där varje deltagare får en egen coach tilldelad till sig som hjälper dem med denna utbildning.

Om man ser på den nuvarande arbetsmarknadsdatan, kan denna data användas för att optimera samt göra jobbsökandet snabbare?

För optimera jobbsökandet så lades all arbetsmarknadsdata in i en grafdatabas, och med hjälp av den databasen byggdes ett rekommendationsystem som använder olika metoder för att utföra varje rekommendation. Rekommendationerna kan användas av en leverantör för att hjälpa dem att tilldela coacher till nyregistrerade deltagare samt rekommendera aktiviteter. Prestandan av rekommendationssytemet evaluerades av en statistik metod.

Medan de egenskapade rekommendationsmetoderna hade accepterbar prestanda var det den populära metoden kollaborativ filtering som hade överlägset bäst prestanda. Det finns utan tvekan potential när det kommer till de egenskapade metoderna, om lite mer tid hade lagts ner på testning utav metoderna så skulle de till slut troligtvis prestera bättre än metoden kollaborativ filtering. Utöver detta så skulle en expandering av databasen med att lägga till mer data positivt påverka rekommendationerna och troligtvis öka prestandan.

Contents

1 Introduction **1**

2 Background **3**

2.1 The participant registration process 3

2.2 Graph databases 5

2.2.1 Neo4j 7

2.3 Recommendation systems 9

2.3.1 Collaborative filtering 10

2.3.2 Content-based filtering 11

2.3.3 Hybrid recommendation systems 12

2.4 Similarity methods 13

2.4.1 Jaccard similarity 13

2.5 Mean reciprocal rank 14

2.6 Related Work 15

3 Method **17**

3.1 Gathering and pre-processing the data 17

3.2 Designing the database 21

3.2.1 Python integration 27

3.3 Populating the database 27

3.4 Building the recommendation algorithm 28

3.4.1 Node similarity 28

3.4.2 Recommending coaches 31

3.4.2.1 Naive approach 31

3.4.2.2 Penalty system 31
3.4.3 Front end 33
3.5 Evaluation 37

4 Results and Discussion **39**
4.1 Experiments 39

5 Conclusion **44**

6 Future work **45**

A Document structures **49**
A.1 Participant 49
A.2 Invoice 50
A.3 Order 51
A.4 Delivery 51

1 Introduction

There are currently over half a million people registered at the Swedish Public Employment Service, also known as Arbetsförmedlingen, whereas 40% of these registrants have been unemployed for approximately 0-6 months [3]. In its current form, the process for an unemployed individual from being registered at Arbetsförmedlingen to actually getting an employment could be seen as both monotonous and primitive.

If Arbetsförmedlingen deems that a registrant needs to sharpen up on certain practical skills, require extra coaching or guidance during their job search process, they outsource this education and coaching to external companies, also known as *providers* in the form of different tailored programs. The goal of these tailored programs is to create an opportunity for the registrant to be better prepared for the working life, and at the same time assist them in landing an employment faster [6].

Assedon is a small recently formed company located in Uppsala. Assedon is currently working on a digital tool in collaboration with one of the providers called AlphaCE, that both job seekers and job coaches can use that effectively digitizes the coaching and education process [7][2]. They believe that digitizing this process is the future, they also believe that by using this tool it could increase both the possibility and the accessibility of job coaching remotely which could then lead to increased effectiveness and efficiency, both for the coach and the registrant.

Furthermore, Assedon is also interested in investigating if it is possible to optimize and speed up the job search process given the current labour market data that they receive from the provider, AlphaCE. In order to try and speed up the process, the main goal for this thesis will be to build a functioning recommendation system. Then, in order to determine the performance of the recommendation system, use an appropriate statistic measurement that can be applied to the system. By creating a recommendation system, it could serve as a complement to the coach's current tools in helping the participant achieve an employment. The first real use case of this system could potentially be when the participant is initially registered at the provider, more specifically, it could be used when assigning a coach to a participant. In order to maintain a high quality of the education, and since every participant is different, it is important each participant receive a coach that can understand them correctly and tend to their individual needs. By using the recommendation system, it could be possible to find a coach which has had success with a specific type of participant in the past and therefore could be a great coach for similar future participants.

The tool could also play a huge role to the coach when making the decision to what specific activities the participant should practice during their time at the provider. By

having the participant's characteristics and by using the tool, the coach also have instant access to similar participants who share the same characteristics. There, the coach could search among the similar participants and see common occurring activities for them, and whether they were effective or not.

2 Background

2.1 The participant registration process

In order to be able to create a properly functioning recommendation system, necessary data regarding the participants is needed, the more the better.

The following section will go through the whole registration process, starting from when a registrant is registered to receive extra education and coaching from one of the providers, which in this case is AlphaCE, to actually receiving an employment. The process is independent which means that it is the same, regardless of which provider the registrant chooses.

The following documents mentioned in this section are also where the majority of the data used during this project originates from. To further help clarify as to what is a document and not, for each occurring document in the following section, the name will be written in *italics*.

Each participant that is enrolled in coaching is assigned their own unique *participant* document. This document serves as a base for the participant, since it points to the other various documents that the participant has. The document also contains more general information about the participant, such as age and which track they have been assigned to. Track is a rough determination to how far away a participant is from obtaining an employment. There are 4 different tracks, where track 1 represents that they are close to getting a job but might need some minor assistance to get there, while track 4 is that they are far from a job, for instance, they might need to be more proficient in the Swedish language before they are completely ready for a job.

The third information that the *participant* document includes is the specific program the participant is following. At the time of writing, AlphaCE has two programs that a participant can pursue, Support and matching (Stöd och matchning) and Introduction to work (Introduktion till arbete) [5][4]. Support and matching is more aimed at those who need intensive and individually adapted support when they are searching for a job. Whereas the program Introduction to work is aimed more towards those who have been absent from the labour market for quite some time, and are not quite sure of their practical capabilities and what type of work may be suitable for them [5][4]. The program which they pursue is determined and assigned by Arbetsförmedlingen.

With the help of the participant's assigned coach, they form a strategy in how to proceed with their education and training which also contains the various activities they will be practicing. The document describing the strategy is called *joint planning* and this

is only done once for each participant. Once the *joint planning* document has been created, a document called purchase order, also called *order* for short, is then created. The *order* document contains the total time in days that the participant will be enrolled at the provider, which is approximately three months. The enrollment time will be the total amount of days that AlphaCE can possibly invoice Arbetsförmedlingen. The *order* also functions as a base to each of the other economic documents, since each of these documents contain a reference to the *order*.

The documents previously mentioned above will only be generated once at the start of a participant's time at the provider. A graph that explains the first steps of the cycle can be seen below in Figure 1.

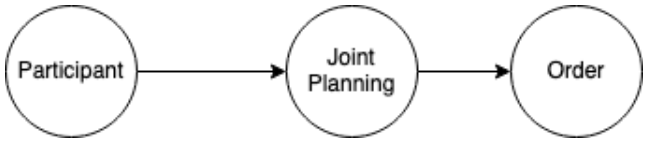

Figure 1 Diagram of the first steps in being registered at an external company

At the end of each month, the coach summarizes the previous month and creates a document called *periodic report* for the participant that includes the various activities which were practiced that month, and whether or not they participated. With each *periodic report*, a *delivery* is automatically generated which is an estimate for how much AlphaCE will need to be reimbursed by Arbetsförmedlingen for that specific month of work. This document will then need to be approved by the coach. When the *delivery* is approved an *invoice* is created that contains activities, articles and various payment info, that is then sent to Arbetsförmedlingen to be paid. If, for some reason, the participant is absent at a meeting, activity or a meeting does not go as planned i.e., a meeting deviates from the original structure or plan, the coach writes an *absence report*, which as the name describes, confirms that something during their enrollment did not go as planned.

If the participant receives an employment or enrolls in education a document called *result 1* is created, which contains further details about their future situation. Later, when four months have passed since the first result, the coach registers a second result, namely *result 2*, which contains information whether the participant is still employed or in education. If the participant is still unemployed or not in education, the participant must decide whether they want to continue and extend the coaching or to terminate and therefore end the coaching completely. An example of a participant and their corresponding documents can be seen in Figure 2 below.

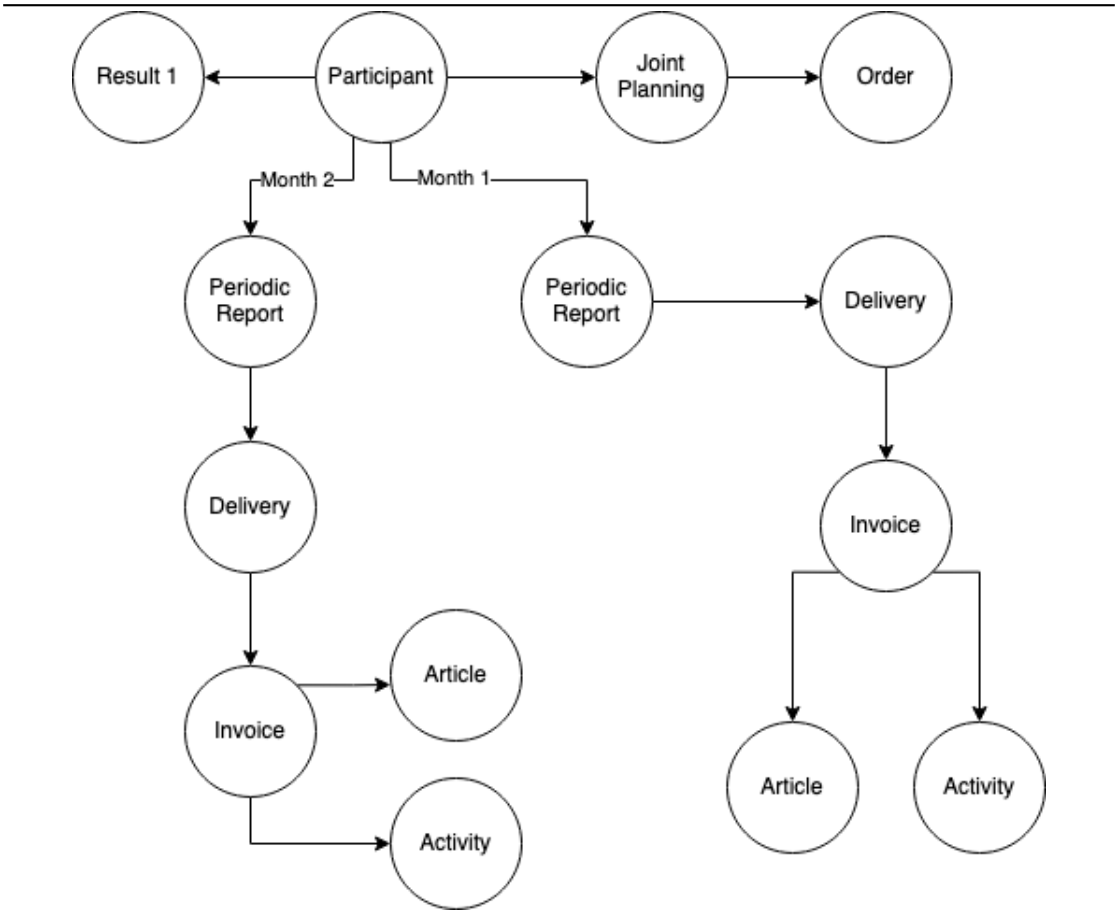

Figure 2 Example diagram of a participant and their corresponding documents.

While observing the figure above, it can be assumed that the participant has spent a total of two months at the provider, due to the occurrence of two *periodic reports*. Furthermore, the participant has also recently achieved an employment or enrolled in education because of the existence of a *result 1*.

2.2 Graph databases

According to graph theory, a graph consists of vertices, also called nodes, which are then connected by edges. A graph database (GDB), is a type of NoSQL database, NoSQL databases are created to address the various limitations and issues of relational databases. Unlike a traditional relational database, where the data is stored using tables, columns and rows it is instead stored using either a node or an edge. The nodes represent the entity of the data while the edges symbolize relationships, these nodes and

relationships can then be retrieved by an user using various queries. In GDBs there is an heavy emphasis placed on the connections between nodes. Instead of using another table to look up specific connections between data the connection is stored directly in the node.

The Resource Description Framework (RDF) is a graph database model which had its first specification published by the World Wide Web Consortium, also known as W3C, all the way back in February 1999 [24]. The data structure in RDF can be summarized as a collection of triples, each consisting of a subject, a predicate and an object. With this set of triples a RDF graph can be conceived [25]. In the graph, the set of triples represents a statement, a directed relationship and a property, an example graph can be seen below in Figure 3. This also works great, but as the properties for an entity increases, so will the total amount of nodes in the graph.

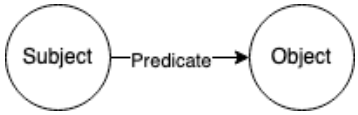

Figure 3 Representation of the RDF graph model.

Placing an example customer with an attribute and an invoice into a simplified RDF graph would roughly look like something in Figure 4.

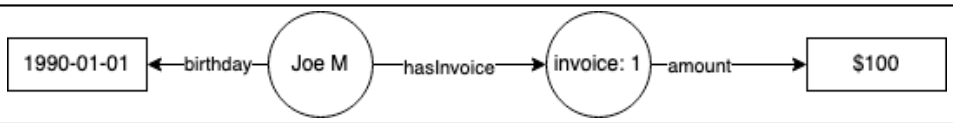

Figure 4 An example RDF database with customer "Joe M" and their invoice.

In the RDF model, properties of entities are also nodes with their own distinct connection. To make it much clearer of what is an entity and what is a property, the properties of an entity is represented as rectangles instead of circles. In a real use case, the nodes and edges of a RDF graph only consist of their own individual Uniform Resource Identifier (URI), which acts as a unique identifier. This is the only label that the nodes and edges have, apart from the literal values from node properties [8].

The labeled property graph (LPG) is a graph database model that is very similar to the RDF model. The main difference is that in an LPG, both the nodes and edges have an internal structure. More specifically, the node and edges both have a unique identifier and a set of key-value pairs. This means that it is possible to store properties directly within both the node and the relationship [11]. This also makes a LPG graph more compact than a RDF graph.

Now, lets place the same data used previously into a graph database using the LPG model. The following graph would look something like in Figure 5 below.

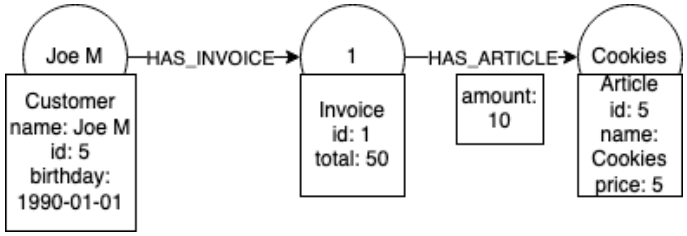

Figure 5 LPG graph of the customer and their invoice with an added article.

In this new graph, we can see that we have the customer from before, *Joe M*, as a node with the unique label *Customer*. This node also has additional properties, such as the name of the customer, a unique identifier and the customers birthday. It is also possible to see the customer has a directed relationship towards an *Invoice* node. Likewise, this node also has additional properties such as a total and a unique identifier. Furthermore, it can be seen that the invoice node has an article attached to it. More precisely, it also has a directed relationship, *HAS_ARTICLE* towards an *Article* node with the additional property amount. Instead of storing the amount of an article as a property within the node, it is clearer and less convoluted to store it in the relationship itself.

Generally, by storing related properties within either the node or the relationship makes for an overall more clean and compact graph.

2.2.1 Neo4j

Neo4j is a LPG graph database management system developed by Neo4j, Inc that had its initial launch in February 2010 [20]. Neo4j is the most popular graph database according to DB-Engines and the 19th most popular database overall [12][13].

Neo4j utilizes its own graph query language called Cypher. While designing Cypher they wanted to make it easy to learn, understand and use for everyone while still maintaining the same functionality of other query languages. At first glance when looking at Cypher queries, the syntax of Cypher visually resembles ASCII art in some way. It is designed with this in mind so that Cypher naturally follows the way we draw and visualize graphs on a whiteboard [10].

Consider the following queries to retrieve all of the employees who work in the IT Department in Figure 6 below. The former is the SQL query required to retrieve data while the latter is the Cypher query.

SQL Statement

```
SELECT name FROM Person
LEFT JOIN Person_Department
  ON Person.Id = Person_Department.PersonId
LEFT JOIN Department
  ON Department.Id = Person_Department.DepartmentId
WHERE Department.name = "IT Department"
```

Cypher Statement

```
MATCH (p:Person)-[:WORKS_AT]->(d:Dept)
WHERE d.name = "IT Department"
RETURN p.name
```

Figure 6 The SQL and Cypher query required to retrieve all of the employees who work in the IT Department.

By observing these two queries it is quite apparent and easy to see that the most compact and self-explanatory query is the one made in Cypher. This also makes it relatively easy for a beginner to understand these queries without having any or minor prior knowledge. Instead of just listing the employee names in a table, modifying the Cypher query to return both the department and the people working there could look like something in Figure 7.

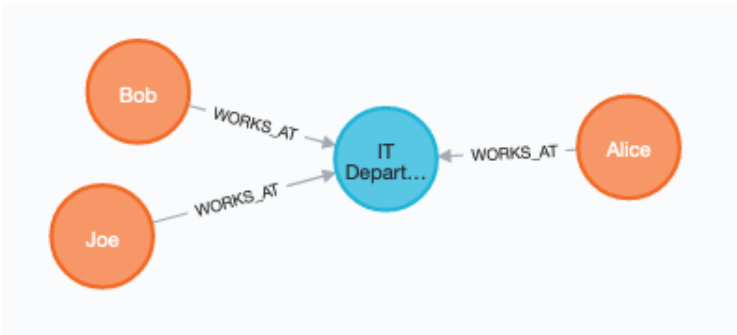

Figure 7 How the Cypher query in Figure 6 could be visualized in a Neo4j graph.

To further pin the two databases against each other, an experiment was conducted where the goal were to find the friends-of-friends in a social network. The data amassed to a total of 1,000,000 people, where each person had approximately 50 friends. The following results strongly suggest that graph databases are one of, if not, the best choice when it comes to highly connected data, which can be seen in Figure 8 [14].

Depth	RDBMS execution time(s)	Neo4j execution time(s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

Figure 8 Comparison of finding friends-of-friends with increasing depths in a relational database versus Neo4j [14].

At a low depth of two, it is possible to see that the run times are practically the same. However, when the depth is increased just by one, the relational database is already falling behind, and not by a little. Going all the way down to a depth of five, the relational database is still running its query. Meanwhile, comparing the Neo4j runtimes, there is an increase of approximately two seconds from depth two to depth five. The reason for this speed is due to Neo4j utilizing something called index-free adjacency. Which means exactly that, relational databases needs to go through separate tables to find the links between data. While in Neo4j, each node has their own physical RAM address in memory and a pointer to all of their neighbour nodes. Furthermore, similar or related nodes are sometimes stored closely together in memory, this is to further increase the chance that they will be in the CPU cache together, which makes the retrieves even faster [17].

2.3 Recommendation systems

A recommendation system is a type of information filtering system that filters and tune data based on various parameters [21]. It is widely used over all parts of the internet, ranging all the way from e-commerce websites to social media websites. Currently, it is very rare to not get exposed to recommendation systems online. When shopping for a specific item online, it is regularly used to recommend similar items based on that specific item or the user's previously purchased items. Likewise, in social media it is frequently used to recommend contacts the user may know based on their current contacts, their current contacts' contacts, and so on.

A fundamental part of any recommendation system is the method, the main algorithm behind the system. While there exists many different types of methods, there are two

types that has seen more wide use than any of the others: collaborative filtering and content-based filtering.

2.3.1 Collaborative filtering

Collaborative filtering places an emphasis on both user interaction and their similarities. It filters content based on previous user interactions, which could be ratings given to items [16]. Next, building on the user's set of items and their ratings, the algorithm matches these ratings with a different user who gave the items a similar rating, these two matching users are then deemed to be similar. Now, if the second user gives a different set of items a good rating which the first user has not yet rated, there is a high chance that the first user is also interested in the set of items and will therefore also give these items a good rating, since they have both share the same taste, so the algorithm recommends the items to the first user.

Collaborative filtering is based on the simple assumption that the people who have agreed in the past, will continue to agree in the future, and they also will continue to prefer similar type of items in the future that they have preferred in the past [9]. Refer to Figure 9 below for an illustration of this specific filtering method.

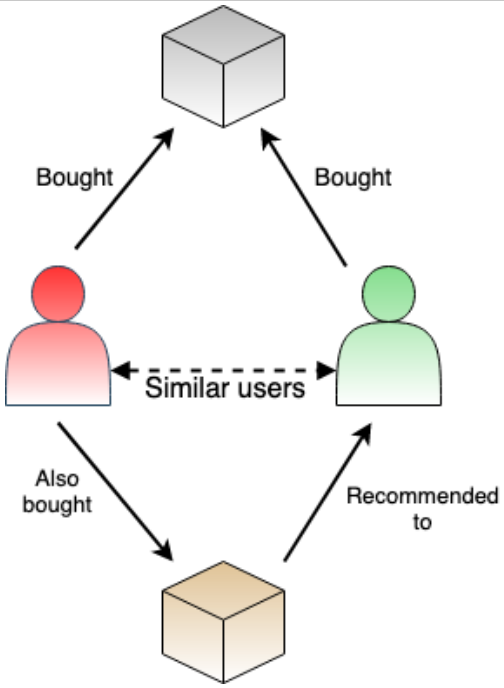

Figure 9 Visual example of collaborative filtering in action. In this method the focus is placed on the users similarities.

While collaborative filtering works great when there exist a plethora of information about the user, their items and their ratings, it falls relatively short when the user data is severely lacking. For instance, it can not give an accurate recommendation for newly registered users, since it has no data on what to base the recommendation on, this is more widely known as the cold start problem. Scalability is also a tricky obstacle to overcome, while the total amount of users and items increase, so too will the necessary computing resources required to perform the recommendations.

2.3.2 Content-based filtering

Unlike collaborative filtering, which places an emphasis on the user interactions and similarities, content-based filtering instead shifts the focus to the items themselves, the features associated with them and their similarities [21]. A visualization to how this algorithm works in practice can be seen in Figure 10 below.

An example of this could be when a user gives an item a good rating, the algorithm will then collect all of the items associated characteristics and features, i.e., the category that the item belongs to, and retrieve other items which share the same characteristics, which in this case is items from the same category and recommend these items to the user simply as *similar items you may like*.

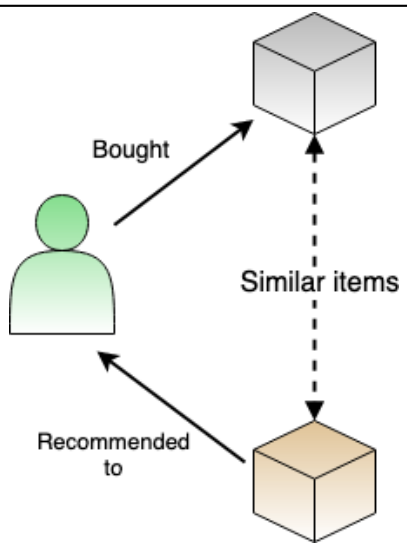

Figure 10 Visual example of content-based filtering, here the focus is placed on the items similarities.

One of the key advantages that content-based filtering has compared to collaborative filtering is that it does not actually need to know a lot about the user themselves, since

the key data that the algorithm uses is located within the items. This could, however, also be seen as a major disadvantage, because in order for the recommendation system to work properly, rich and full descriptions of items are needed.

2.3.3 Hybrid recommendation systems

If a website employs a recommendation system which neither uses the collaborative filtering, content-based filtering method or any other method, there is a high possibility that they instead are utilizing a hybrid recommendation system. By combining the two previous methods a hybrid system can be formed, which enables the usage of techniques from both of the methods. It is generally used to overcome the limitations that each of the individual methods suffer from [16].

Several different strategies to combine collaborative filtering and content-based filtering into a hybrid system could be:

1. to have regular individual implementations of the methods and combining both of their results,
2. implementing some of the characteristics from a content-based method into a collaborative filtering method,
3. implementing some of the characteristics from a collaborative filtering method into a content-based method, and
4. developing a new, more general, unifying model that features implementations of characteristics from both methods.

All of the various strategies mentioned above are actual real-life examples of different implementations of a hybrid system, used by recommendation systems researchers [1].

The popular online streaming service Netflix put a lot of effort in developing and perfecting their own hybrid recommendation system. According to them, about 80% of all the hours spent watching movies and tv-series are influenced by their recommendation system alone. To further improve upon their recommendation system, in 2006 they organized an open competition with a prize pool of $1,000,000, where the objective was to develop a recommendation system that were more accurate than their own [15].

2.4 Similarity methods

In order to properly give recommendations to a specific user, it is first required to find similar users which the recommendations can be based upon. To determine whether certain users are similar or not, a form of computational comparison between the pair of users needs to be done. The need to measure similarity is a fundamental concept that is regularly used in statistics, but it also sees use in related fields. By using a similarity measurement function, it is possible to find the similarity between a pair of objects.

2.4.1 Jaccard similarity

The Jaccard similarity coefficient, also known as the Jaccard index, is a term that was formulated by French professor Paul Jaccard and published in 1901. The term signifies a method that is used to measure both diversity and similarity between two item sets.

Given two item sets, in order to obtain the similarity between them, the intersection of both sets is divided by the union of the sets. The formula to calculate the similarity can be written into Equation 1 below.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{1}$$

The formula yields a value, $0 \leq J(A,B) \leq 1$, where the closer a value is to one, the higher the similarity is, and a value of exactly one means either both two sets are empty, or the sets are completely identical, while a value of zero implies no correlation between the sets at all.

Consider the following two example sets, A and B, given that $A = \{0,5,2,3,6\}$ and $B = \{0,4,5,6,3\}$, the Jaccard index between these two sets can be calculated, which in turn, will yield the sets pairs similarity scores as well. The Jaccard equation featuring the two sets as input and the accompanying solution can be seen below in Equation 2.

$$\begin{aligned} J(A,B) &= \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \\ &= \frac{|\{0,5,3,6\}|}{|\{0,5,2,3,6\}| + |\{0,4,5,6,3\}| - |\{0,5,3,6\}|} \\ &= \frac{|\{0,5,3,6\}|}{|\{0,5,2,3,6,4\}|} = \frac{2}{3} \approx 0.67 \end{aligned} \tag{2}$$

The Jaccard index for the calculated two sets above is 0.67, this means that the sets share more than half of the items in each set. This can further be confirmed by comparing the two sets manually, it is possible to see out of a total of five items in each set, the sets share four of these items.

2.5 Mean reciprocal rank

In order to properly evaluate and rank the list of recommendations the system produces, a metric for rankings must be utilized.

The mean reciprocal rank is a statistic measurement used to evaluate a system that returns a list of items in response to a query, with an emphasis placed on the order of items. Given a single query, the reciprocal rank is $\frac{1}{rank}$, where rank corresponds to the correct items position in the returned list, for example, 1 if the item was first, $\frac{1}{2}$, if the item was second, to $\frac{1}{N}$ where N is the length of the list. If the correct answer is not present in the returned list, the reciprocal rank is 0. The mean reciprocal rank for multiple queries Q, is given by calculating the average of the reciprocal ranks for Q, the formula can be seen below in Equation 3.

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i} \tag{3}$$

Consider a participant that has been recommended potential coaches by a recommendation system using the same method three times, the coaches are stored in three separate lists, A, B and C, where A = {Mary Sue, John Doe, Jan Eriksson}, B = {Anna Andersson, Mary Sue, Jane Doe} and C = {John Doe, Jane Doe, Jan Eriksson}. Given that the participant's assigned coach is *John Doe*, by calculating the mean reciprocal rank of the recommendations, the accuracy can be measured. The reciprocal ranks of each list is $A = \frac{1}{2}$, $B = 0$ and $C = 1$. The mean reciprocal rank can be measured by calculating the average of all the results, seen below in Equation 4.

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i} = \frac{1}{3}(\frac{1}{2} + 1) = \frac{1}{2} \tag{4}$$

The mean reciprocal rank for the three lists of recommended coaches are 0.5, which means that the recommendation method predicts the participant's actual coach most of

the time. In terms of most desirable value, a rank of 1 is the most ideal, since it means that in the list of predicted coaches, the actual coach is the first coach in the list every single time.

2.6 Related Work

In 2017, researchers from universities in the United States developed, in collaboration with popular employment website CareerBuilder, a Graph-based recommendation system (GBR) for online job recommendations. In their designed model, jobs are represented as nodes and the edges connecting the nodes represent the different similarity scores between each pair of jobs. The specific similarity scores are computed using numerous different data sources that seeks to capture user behaviour, resumes and job descriptions. To determine the similarity between each pair of jobs, they first extract relevant information from the job using its description, which is then embedded into vectors. After the embedded vectors for two individual jobs has been generated, the similarity score between them are computed using the cosine similarity on the two vectors. For the users, they built a user preference vector, based on information in the user's resume and their interaction history. For their recommendation method, they utilize a hybrid one, this was done by first implementing a collaborative method as a base and then incorporating some content-based characteristics into it. The results from their system and model proved to be both interesting and practical since it managed to achieve, on average, an accuracy of 90%, which essentially means that 90% of the recommendations that the system yielded, proved to be relevant and satisfactory to the users [22].

While their project is similar to this project to a large extent, since the primary focus of both recommendation systems is to increase the likelihood of getting an employment, there are some key differences. Their system is more direct, since it seeks out to recommend actual, existing, jobs to apply for, while this projects system is more oriented towards recommending activities and coaches, which in the end, indirectly will lead to a job. Also for their project, when it comes to determine the similarity scores between each pair of nodes, they do this by computing the cosine similarity. This project opts to instead use the Jaccard similarity while calculating the similarity scores between different nodes.

A mutual trait that both recommendation systems share, is that both are built using a graph database, albeit utilizing different software. Their model is built using Apache Spark's GraphX framework and their data is stored in Hive databases, which is processed using SparkSQL. In comparison, this project seeks to use Neo4j for both modelling the database and storing the data. To summarize, both recommendation systems share the same motivation, albeit the two systems take two relatively different paths to

reach their shared end goal.

3 Method

The project incorporates three different main parts. Since the data was already scraped and gathered from AlphaCE, the first part was to properly pre-process the data. The second was to model and design the database. The third and last part was to build the recommendation engine.

This was the general road map, during the project a more iterative development process was adopted, which means that some parts needed to be revisited and modified numerous times in order to adapt to the fluid changes made throughout the project.

3.1 Gathering and pre-processing the data

The majority of the data is gathered from Arbetsförmedlingens website that a provider, such as AlphaCE has access to. The main purpose of this website is to store all of the enrolled participants, both past and present, as well as all of the documents associated with them, that is, all of the aforementioned documents that are explained in the registration process in Section 2.1. The website is primarily used by the coach of a participant to submit various documents such as a periodic report, so that Arbetsförmedlingen are always updated on the participant's progress during their enrollment at AlphaCE.

The data was gathered from the website using web scraping tools. Web scraping is used to automatically access a website, often using a bot or similar automated software to gather relevant information instead of manually copy and pasting over everything. This was all done by Assedon themselves before the project started. By already having the data gathered and available meant that more focus was laid on pre-processing the data. While gathering the data, all of a participant's sensitive data was omitted, such as full name, home address and other general contact information, this was done to maintain user privacy and anonymity. To further increase the anonymity, instead of including a participant's genuine age, a total of six age-spans were created, 18-25, 26-35, 36-45, 46-55, 56-65 and 66+, which could then be assigned to each participant instead of using their real age.

The data was stored in text files using the JSON representation. To begin, the files were categorized based on the type of document. Then, they were divided based on different kinds of parameters, e.g., the files containing the participants were sorted based on the location of the AlphaCE office. The majority of the other documents, such as the deliveries and invoices, were divided based on a time span of approximately a year. There were only one file which contained all of the orders for each participant. So, in total, there were four different folders, each containing the different types of documents: par-

ticipants, deliveries, invoices and orders. With all of the different documents gathered and neatly stored, the pre-processing could begin.

In order to keep it simple, that is, to not have an overabundance of files for each document type, all of the files for each type were merged together to form one individual file for said type. Merging multiple JSON files in Python was very straightforward, mainly due to the vast number of libraries that exists for Python. The files for each type of document contain a lot of data, however, only a small portion of the data were actually used, therefore, we will only focus on the relevant parts of the data for each file. The structure for a participant with example data and the irrelevant data omitted can be seen in Appendix A.1, this is how each participant entry is represented as a JSON object in the file.

The majority of the keys in the entry should be self-explanatory and nothing completely new, however, there are still some keys which will need to be clarified. Starting with the key `delivery_adress`, this key corresponds to the location of the AlphaCE office, this is usually where the participants' coach is located. The keys nested inside of `af_supervisor` corresponds to the participants' supervisor at Arbetsförmedlingen, along with their general information, such as email, phone number and office. Finally, the keys which are nested inside of `documents_list` store the more interesting parts of a participant, namely almost all of the documents associated with the participant. The majority of the documents here are essentially the same documents which are mentioned in Section 2.1.

The first key, `type`, corresponds to the specific type of document it applies to, the second key, `registration_date`, is the date when the document was uploaded, if the value is empty next to some other keys, the document does not exist for that specific participant. The `ka_number` can be seen as an identification number for the office. Each of a provider's delivery areas are assigned a unique ka-number from Arbetsförmedlingen, each delivery area can also span multiple municipalities, offices located in different municipalities that are placed in the same delivery area share ka-number. Finally, the last two keys, `coach` and `pronoun` is the participants' coach and the participants' preferred pronouns respectively.

Along with these general keys, there are also some document type-specific keys which only exist for that specific document. These keys can be seen for both of the result documents, namely result 1 and result 2. Starting with the key `R1_reason`, the value for this key contains information whether the participant enrolled in education or got employed, whereas `R1_start` is the date when their employment/training is set to begin. For result 2, there are some keys which are the same as in result 1, with only the name changed, however, their definition stays the same. There is a new key called `R2_client`, which contains information about the company if the participant was

employed, or information about the school if the participant pursued education. The last and final key, `R2_info` stores further information about their employment/education, the number represents either the study pace, the amount of work hours per week or, if the participant got a part-time employment, the work pace.

The structure for an invoice with example data can be seen in Appendix A.2. It is much slimmer than a participant entry, while still storing relevant data.

In order to determine which invoice correlate to what participant and order, there is an identifier to both, as well as a unique identifier to the invoice itself. The invoice can be seen as a summarization of the entire month for a participant, it contains the amount of active workdays as well as the various activities that they practiced over those days. Inside of `articles` is first the identifier of the article, then the actual name of the article, which is the same as the participants' track, followed by the amount of days, AlphaCE's price for one unit (day) and lastly, the total price. Inside of `activities` is the name of the activity they practiced, there can either be one or multiple activities.

Moving on to the third document, the order. An order with example data inserted can be seen in Appendix A.3.

The order is almost indistinguishable from the invoice, apart from not containing any activities, the only difference is the amount of the article, it stores the total amount of days the participant is enrolled at the provider.

The last and final document to exemplify is the delivery. The entries inside are nothing new, it is basically the same as the invoice, right down to the number. This is perfectly reasonable because of how the monthly cycle, explained in Section 2.1, works. The structure for a delivery entry with sample data can be found in Appendix A.4.

During the pre-processing of the data, Python was primarily used. For the file containing the merged participants, instead of having the pronoun key nested deep inside each of the document's `documents_list`, it was also added as a key next to `participant_number` and the other root keys. The pronoun could also appear in two different forms, as seen in Listing A.1, namely, *henne*, which is Swedish for *her* and *hon*, Swedish for *she*, with the latter as the preferred pronoun. In order to get the correct form of the pronoun, when an occurrence of *henne* were found, *hon* was instead added as the value for the root key, `pronoun`. Occasionally, the pronouns nested inside of `documents_list` were empty and missing a value, in these cases, the added value to the `pronoun` key would instead be *None*, to indicate that the participant did not have an applicable pronoun.

Some participants included in the gathered data did not appear to have any documents associated with them, these participants were skipped completely during the merging of

participants since they were determined that they lacked any relevant or beneficial data. Another inconsistency that were present in the gathered data was that the participant did not have an assigned Arbetsförmedlingen supervisor, all of the values for the keys nested inside `af_supervisor` were empty. To solve this, a placeholder supervisor were added to these participants instead that they all share. Another rare occurrence regarding the supervisor, sometimes the values for certain keys would be swapped around, i.e., the supervisor's office were stored in the phone number, in addition to this, the phone key were also always missing during this occurrence, since it held the value for the office instead of the actual phone number. This was a pretty undemanding and easy fix, by using Python, the values of the keys were simply swapped to their actual value and the phone number value was set to an empty string.

Certain participants that had achieved a result 2, in other words, they had an entry in their `documents_list` for the document type result 2, were at times missing the `R2_info` key completely. Instead, the numbers that specified the work/study pace were placed inside of `R2_client`, often in the middle of the client name. Refer to Listing 1 below for an example.

```
{
    "type": "Resultat 2",
    "registration_date": "2020-05-05",
    "ka_number": "99999",
    "R2_client": "2021-02-10 2451 Hac1ker0Aca0demy",
    "R2_reason": "Ersättningsgrund: Utbildning"
}
```

Listing 1 Example how it would appear when a result 2 is missing the key R2_info in a participant.

By observing the last part of the value for `R2_info` in the example above, it can be seen that inside of the fictional school name, *HackerAcademy*, there are digits present. More specifically, the number 100 is hidden inside of the name, which in this case corresponds to the participant's study pace at 100%. To get the value of the study/work pace into the actual key, `R2_info`, an algorithm was written in Python, which first checks if the key is missing, then, it extracts the digits from `R2_client` and places them into `R2_info`.

The participants were also complemented by additional data from Arbetsförmedlingen in the form of the participants language. While gathering the data, a supplemental file was created which consisted of additional data for each of the participants, one of these additions was the native language of each participant. Again, by using Python, the

participants could be mapped with their native language and form a new root key inside the file of the merged participants called `language`. In the supplemental file, there were some entries which lacked a language, the value for the language was *None*, the participants which contained this value were then assumed that their native language was Swedish.

Some other various data cleaning was done on certain keys values, one of these were on a key in the `articles`. The key which holds the amount of days for each article. It essentially says the amount of days which were working days during that specific time interval. To better prepare it for the database and to be able to manipulate and use it as a real number, the suffix, *dag(ar)*, Swedish for *day(s)*, needed to be removed. Once again, with the assistance of an algorithm written in Python, the documents which stored articles, i.e., the invoices, deliveries and orders, had their amount trimmed so only the numbers remained. Similar to this, the prices were also trimmed to only contain numbers as well, effectively removing the decimals and currency at the end of the price.

3.2 Designing the database

When modeling a graph database, it is important to determine what part of the data should be represented as a node and what should be represented as a relationship to another node. In addition to this, it is equally important to establish what to model as a property on a node, and what to store as a property on the relationship itself. Finally, while designing the database, it is also necessary to consider the specific type of queries that one intends to run against the database. In Neo4j, when it comes to naming nodes and relationships, the most common convention to use is *UpperCamelCase* for nodes and *CAPITALIZED_WITH_UNDERSCORE* for relationships.

The original design of the database consisted of a participant node as the center of the model, since it functions as a root for each of the other documents associated. The participant had directed relationships towards their characteristics and properties, also represented in the model as individual nodes, such as the pronoun, coach, age span, track and program.

A visualization of the design of this model can be seen below in Figure 11.

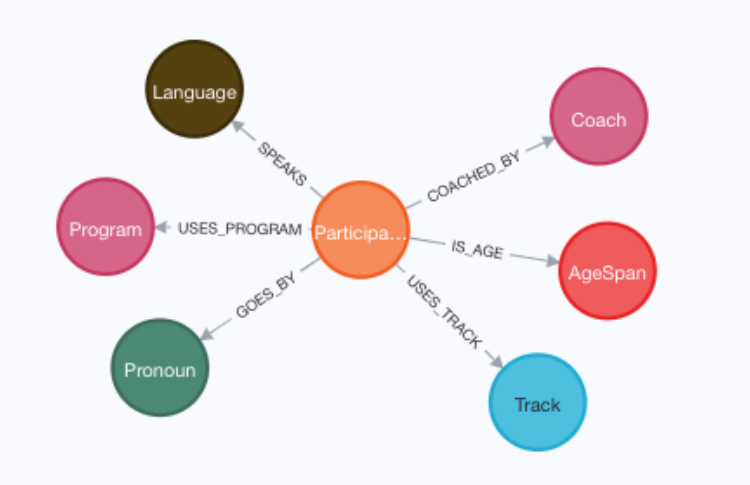

Figure 11 The first iteration of the model, containing only the general properties of a participant.

While naming the nodes, brief and self-explanatory names were chosen, e.g., the label *Language* were given to each of the languages and the label *AgeSpan* were given to all of the different age spans. The same guideline was used while naming the relationships, for example, the participant node had a directed relationship named *SPEAKS* towards a language node.

Connecting each participant with their unique characteristic, such as their language and pronoun, opens up the possibility to see the total amount of participants in the database with the same attribute, including all of the participants who share this attribute, this can of course be applied to all of the other characteristics the participant possess as well. More generally, it enables a quick way to get statistics in the database, as well as finding similar nodes using any given characteristic or attribute.

With each of a participant's general characteristics added, other various economic data could now be inserted into the model, such as the invoice, along with their most interesting component: activities. The first approach to how these documents should be modeled in the database was simple, design them the same way as they are modeled in Figure 2 in Section 2.1. While it is often clearer and more intuitive to have a single shared unified model both theoretically and practically since it helps make it easier to understand certain concepts that would otherwise be difficult to grasp, however, this proved not to always be the case. Expanding the model, adding new nodes and edges for the additional documents using the aforementioned figure as a reference, resulted in a graph with numerous amounts of unnecessary extra nodes and edges required to cross in order to access the activities and article. With this graph, the total distance between the participant and activity node became exactly four edges, which is an excessive amount

of edges that needs to be visited just to retrieve the monthly activities for a participant. Since the periodic report, delivery and invoice practically contain the same information (as seen in the previous subsection 3.1), with only the exception of the invoice, which includes the activities, the other economic documents become redundant when it comes to modelling purposes.

By omitting the redundant documents, the graph became much cleaner and compact, with the participant instead having a direct relationship with the invoice, without the need to visit other nodes in order to be able to reach it. This minor change to the graph also halved the distance between the participant and the activity.

Similar to the invoice, the other documents associated with the participant, such as the absence report, result 1 and result 2 could be represented as nodes in the same manner. Incorporating all of the previously mentioned alterations and additions, the second iteration of the model can be seen in Figure 12.

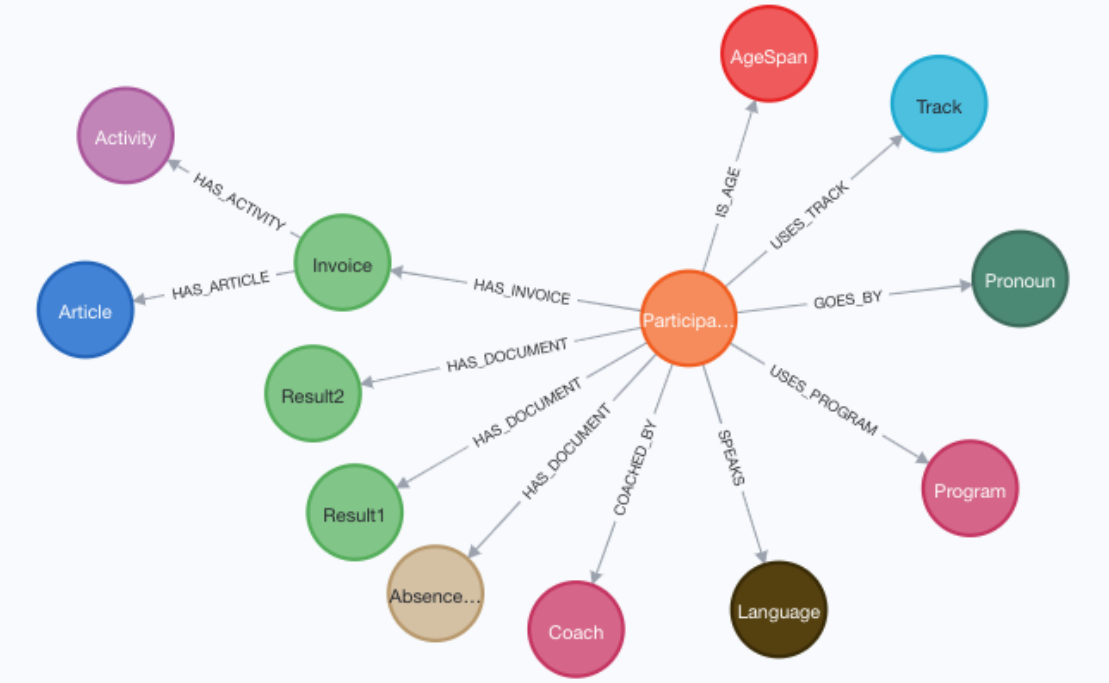

Figure 12 The second iteration of the database model, this iteration includes the additional documents.

By observing the figure above, it can be confirmed that with the new iteration, two edges are required to cross in order to access the activities connected to each invoice.

Similar to the first iteration, the new documents were added using the same straightfor-

ward naming convention for both nodes and relationships. All of a participant's documents were connected using the *HAS_DOCUMENT* relationship, except for the relationship to the invoice, which was given its own unique name, *HAS_INVOICE*. However, in order to maintain a general form of consistency within the database, when it comes to naming the relationships towards a document, there were only two mutually exclusive options. The first option is to let the relationships towards the different documents have separate unique names based on the document type, e.g., *HAS_INVOICE* for each of the invoices and *HAS_RESULT1* if the participant has a result 1. The second option that could be used would be to have a single unified name, such as *HAS_DOCUMENT* for all of the relationships to the different types of documents, similar to what result 1, result 2 and the absence report have in the previous figure.

To better understand the differences between the two options and their impact on the queries, consider the two queries in Listing 2 and 3 below. The two queries represent two different ways to retrieve a participant's result 1, with the first query using a shared name for the relationships, while the second query uses separate unique names.

```
MATCH (p:Participant)-[:HAS_DOCUMENT]->(rs1:Result1)
RETURN rs1
```

Listing 2 The Cypher query for retrieving a participant's result 1 using a shared relationship name for all of the documents.

Since the query above is using a common name for the relationship, there is an additional requirement needed to be specified in the pattern, namely, a label for the node to be retrieved, which in this case is result 1. Otherwise, if this is not explicitly stated, all of a participant's documents will instead be retrieved while executing this query.

```
MATCH (p:Participant)-[:HAS_RESULT1]->(rs1)
RETURN rs1
```

Listing 3 The second Cypher query used for retrieving a participant's result 1, now using a unique name for the relationship.

By using a unique name for the relationship, there is no need to specify the label for the node with the variable *rs1* name at the end of the pattern, since by matching a pattern using the unique relationship name only given to nodes with the label result 1, will return only result 1 nodes.

While having different names for the relationships could help increase the readability and make some queries easier to comprehend rather than using the same name for all, ultimately, due to how a query can be formulated in Cypher, it does not have any major impact on the intelligibility of the query itself. In fact, it could even be clearer and more organized by having all of the documents under the same relationship name, furthermore, by having a fixed name for all types of documents avoids the need of having to remember multiple and unnecessary long names such as *HAS_ABSENCE_REPORT*.

Modifying the relationship name of the invoice to the unified name and inserting the data that has not yet been added into the model yields the third iteration of the model which can be seen below in Figure 13.

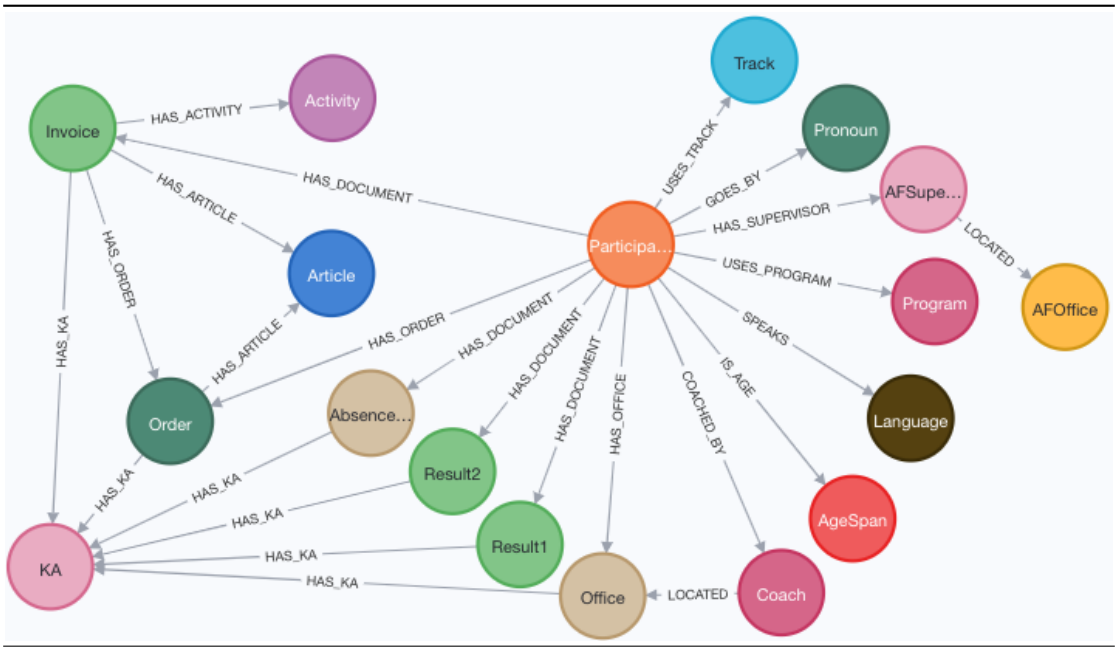

Figure 13 Third iteration of the database model, including all of a participant's characteristics along with their documents and both their coach and supervisor.

With the third iteration of the model as can be seen above, the data became significantly more connected compared to the previous versions. Having highly connected data in the database is optimal and preferred considering the database's specific use case, making a recommendation system. Primarily since it gives a complete picture of a participant with all of their individual interactions and attributes, which in turn increases the probability of making better and more accurate recommendations.

Among the new nodes inserted into the graph was the office, which was taken from the delivery address of the participant, the coach for the participant is also connected to this

office, since they was deemed to be located at the office with the same delivery address specified in the participant's documents. Another node added is the node representing the ka-number of the office, which can be seen in the model as the node with the label *KA*. This node also has a multitude of incoming connections with the name *HAS_KA* from various documents, the reason for this is to enable the possibility to see additional statistics solely based on the ka-number, for instance, the total amount of result 1's generated by a ka. There is also a node with the label *AFSupervisor*, this node represents the participant's supervisor at Arbetsförmedlingen, similar to the coach, the supervisor is also connected to their corresponding office. The last economic document, the order, was also added in this iteration, this node has a connection towards an article node, due to the order containing the total amount of days that a participant will be registered at the provider, which in this case is indicated by the amount of an article correlating to the participant's track, previously seen in the structure for an order entry in Appendix A.3.

Lastly, by observing the figure above, it can be seen that the invoice has a connection towards the order, the reason for this is due to each invoice entry containing a direct reference to the participant's order, furthermore, this relationship also assists in keeping the data in the database connected.

Using the final design of the database, an arbitrary participant could, for example, be represented in the database as in Figure 14 below.

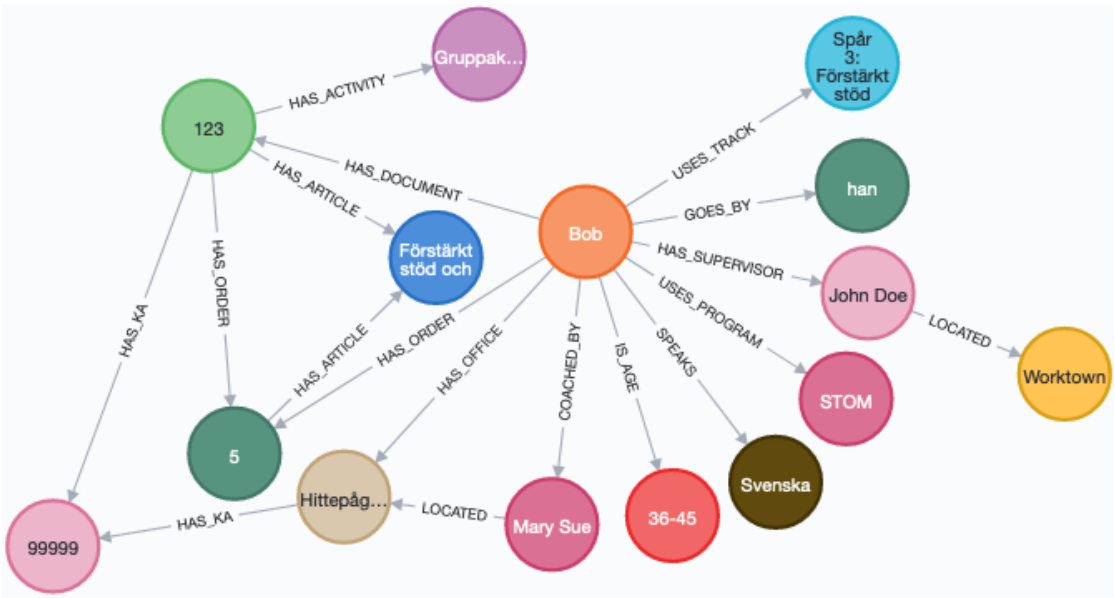

Figure 14 Example participant Bob with his corresponding characteristics and documents.

3.2.1 Python integration

Initially, the Cypher queries were written and executed in Neo4j's own browser, this was done in order to properly learn the Cypher syntax and how a query should be formulated, the browser was also used when there was a need to visualize the nodes and relationships in the graph.

Using Neo4j's official Python driver, a connection from Python to the Neo4j database could be established [18]. This connection made it possible to formulate and execute any Cypher query from Python code. The motivation for utilizing Python as the database query manager, was primarily due to the simple work flow it enables, for both writing the queries and handling the queries returned data. In the Python code, each unique query was represented as their own defined function, this was required in order to properly manipulate and aggregate the results, and, if needed, to utilize them as inputs for other queries, such as auxiliary functions. In the end, the Neo4j browser was rarely used, due to all of the queries being executed using Python, even the queries used to populate the database, however, in rare occasions the browser was used to confirm that a query was formulated correctly and working as intended.

3.3 Populating the database

With a solid foundation laid out for the database model, the database could now be populated using the data given by AlphaCE.

As mentioned previously, when inserting data into the database, an important step is to decide which value should be a property on the node and what value should be a property on the relationship. While creating the nodes, properties were added using straightforward and self-explanatory names, for instance, a *Participant* node has a property called `number` which contains the participant's unique number. Likewise, the *Track* and *Program* nodes, has a property called `name`, which, as the name suggests, is the name of the track or program.

Similar measures were used when naming properties on other nodes, the only instance where a relationship has a property is the relationship *HAS_ARTICLE* from both the *Invoice* and the *Order* node to an *Article* node. The relationship contains the property `amount`, that represents the specific amount of an article the invoice has, which correlates to the participant's workdays that month, and the order contains the overall total amount of workdays.

3.4 Building the recommendation algorithm

In order to be able to give specialized recommendations to the different types of participants, it is important to have an individual similarity score calculated for each pair of participants before a recommendation can be made. Initially, the algorithm was based simply on the collaborative filtering method due to using similar participants to recommend activities and coaches, however, it later evolved into a hybrid recommendation system, especially when it came to recommending coaches.

3.4.1 Node similarity

In Neo4j there exists a wide range of additional plugins that can easily be imported into a specific database. The Graph Data Science Library (GDSL) is a library which adds additional functionalities in the form of algorithms that can be utilized in the database, these algorithms are centered around providing extensive graph database analytical information, such as community detection, node similarity, path finding, etc. [19]. In the scope of this project, the node similarity algorithm was very beneficial, since it could be used to find and determine the various similarities between a pair of participant nodes. Given a set of nodes, the algorithm can compare the set and measure the similarity of a pair from that set based on the nodes they are connected to, the pair of nodes are deemed similar depending on the total amount of mutual nodes. In order to determine a pair's similarity score, the Jaccard similarity coefficient is calculated.

To be able to find participants with matching characteristics using the algorithm, a projected graph had to be generated. A projected graph can be derived from any existing graph in Neo4j, it is usually more compact and flexible than the original graph. A requirement while creating a projected graph is that the projected graph's nodes and relationships must be specified by name, this allows the creation of a graph with only the relevant nodes and relationships that is necessary for the algorithm.

Creating a projected graph including only the participants, their office and their unique characteristics such as the age span, language, pronoun and track yielded a projected graph which could be used as input for the node similarity algorithm. Next, after running the algorithm, in order to save each pair of participants' similarity score, the algorithm could also be used to write the score as a property to the original database. Thus, all of the participants now had a new outgoing relationship called *SIMILAR*, with the property `score`, to another participant, where the `score` holds the Jaccard similarity coefficient as a value for the specific pair of participants.

Now that all of the participants' similarity scores were established in the database, given

any participant, it was now possible to retrieve their identical participants and the corresponding similarity score to this specific participant using the short and simple Cypher query in Listing 4 below.

```
MATCH (participant)-[rel:SIMILAR]->(sim)
WHERE rel.score > 0.6 and participant.number = $number
RETURN DISTINCT sim.number, rel.score
```

Listing 4 The Cypher query for retrieving all of a specific participant's similar participants.

In the listing above, the parameter `$number`, represents the unique identifier of an arbitrary participant. Furthermore, the score constraint was set to only retrieve the similar participants with a score higher than 0.6, which means that participants with almost the same correlating characteristics (such as age span, language, pronoun and track) will be retrieved, whereas a score of exactly 1.0 corresponds to a similar participant where all of the characteristics are matching. Modifying the query so it also returns the mutual characteristics as nodes could for example, result in the graph in Figure 15. In this example, the orange nodes correlates to the participants with the nodes in the middle representing their mutual characteristics.

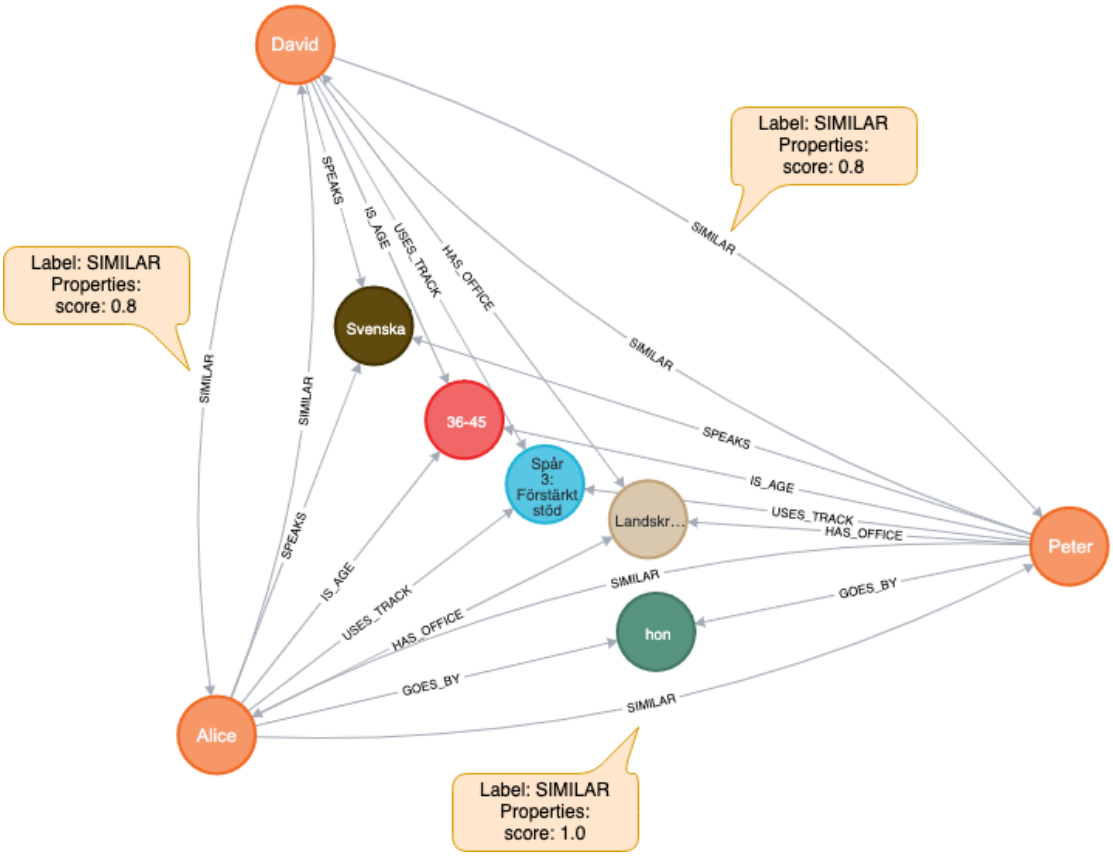

Figure 15 Graph visualization of how the similarity score is calculated. The two bottom participants, Alice and Peter, share every single characteristic, which amounts to a Jaccard similarity score of 1.0, while the top participant, David, only shares 4 out of the 5 characteristics, resulting in an overall lower score.

For the above example, it is possible to see that participants Alice and Peter are a perfect match to each other, since they both have a score of 1.0, while participant David at the top only has a score of 0.8 to each of these participants. The reason for this is that David is missing a valid pronoun altogether and as mentioned previously in Section 3.1, occasionally, the value for a participant's pronoun in the dataset was missing a valid value, therefore, the string *None* was added as the value instead. In the database, each participant that had missing as their pronoun did not get a relationship to a node with the label Pronoun, this was done to help improve the overall accuracy of the recommendations.

If David would instead have had a different valid pronoun entry, which would lead to a relationship with a pronoun different than Alice and Peter, the similarity scores would suffer a decrease from 0.8 to 0.66. When the participant's pronoun information is missing, there is a slight possibility that the pronoun could be the same as the bottom

participants' pronoun, however, if the top participant would have had a different pronoun altogether, then it is exactly known that the pronouns are not shared, hence why the score is higher for the former.

When all of the similar participants had been gathered, their corresponding relevant information, such as their practiced activities along with their assigned coached were retrieved using another identical query. The activities and coaches were then aggregated, assembled and stored in separate tables in descending order. Thus far, a basic recommendation system that utilizes the collaborative filtering algorithm has been designed, which gives average recommendations.

3.4.2 Recommending coaches

While using the collaborative filtering method exclusively is probably sufficient enough to give a somewhat accurate recommendation when it comes to the coach, there are other interesting strategies to recommend coaches using Cypher queries. Including the previous method that only used similar participants to recommend coaches, a total of three different methods for recommending coaches were made.

3.4.2.1 Naive approach

The second, naive approach was to retrieve a participant's characteristics and gather only the coaches which have had participants with exact matching characteristics, counting each occurrence of the matching participants and then aggregating the results to form a table that contains the name of the top 3 coaches accompanied by the count, the higher the count, the better the coach. While this is a relatively quick and cheap solution without much effort put into it, it does yield some favourable recommendations, however, there are instances when this method could fail to give satisfying results. For instance, consider a coach that have had many participants that matches the chosen participant's characteristics, but not a participant with the exact same characteristics, therefore, according to the algorithm, the coach will not be considered, even though the coach's characteristic distribution makes them a suitable coach for the participant.

3.4.2.2 Penalty system

The third approach, which was a little more researched, was to gather every single coach and check their participant distribution based on the characteristics from the selected participant. For instance, consider an arbitrary participant P, that possess unique

characteristics, where P_t represents their assigned track, P_l is their spoken language, P_p is their preferred pronoun and P_a is the age span to which they belong. To find a suitable coach for this participant, let C be a possible coach, where C_{tot} correlates to their total amount of participants, both past and present, while C_t, C_l, C_p and C_a corresponds to the amount of participants based on a given characteristic from P, in this case C_t is the amount of participants from C_{tot} which use the track P_t, the same applies to C_l, which states how many of the coach's participants (C_{tot}) speak the language given by the chosen participant, P_l, etc. With each value known, the coach's score is calculated by taking each characteristic and dividing it by the total, which results in the formula that can be seen in Equation 5 below.

$$C_{score} = \frac{C_t}{C_{tot}} + \frac{C_l}{C_{tot}} + \frac{C_p}{C_{tot}} + \frac{C_a}{C_{tot}} \tag{5}$$

Using the above formula yielded a value such that $0 \leq C_{score} \leq 4$, while this formula would be a decent scoring system for coaches, it was plagued by a few significant flaws. For starters, depending on the participant's characteristics and the coach's participant characteristics distribution, it always heavily favours coaches that have a low amount of total participants. For example, given a participant with a specific set of characteristics, if the coach has only had a single participant with the exact same characteristics previously, according to the formula, the coach will be seen as the perfect match, since the coach's participant's characteristics distribution of those exact characteristics is 100%. Another flaw that the scoring system suffered from was that, similar to the naive approach, it did not consider the office locations of both the coach and participant.

In order to account for these flaws and biases, the first solution was to implement a bonus system. After calculating the score using the formula above, if the coach's number of total participants amounted to greater than or equal to a parameter β, the coach's score was increased by a small amount, likewise, if the coach was stationed at the same office as the participant, the score was increased further by a small amount. While experimenting with this bonus system using a multitude of different values for β, the scores continued to give coaches with few participants higher scores, meaning that the bonus system did not work as intended.

The second solution was to try and further improve on the bonus system, the first change was to modify and invert the bonus system completely, which turned it into a penalty system instead. Meaning that, after the coach's scores had been calculated using the formula, if the coach had total participants less than the parameter β, a specified amount were subtracted from the score, the same applied for the coach's office location. This solution helped contribute in minimizing the effects of both the flaws and biases introduced with the scoring system. Testing and trying different parameter values for both

β and the penalties made it possible for coaches with a few total participants to still get a fair chance at getting recommended. In the end, the values that gave the best results were 0.25 for both types of penalties, followed by a β value of 10, meaning that if a coach has less than 10 total participants, a penalty of 0.25 is subtracted from the score.

The final tweak made to the formula was to scale it down, in order to get a score using the formula with a matching interval that is similar to the Jaccard similarity score, the entire formula was divided by 4. Omitting the penalty values, the new formula for calculating a coach's score can be seen in Equation 6 below.

$$C_{score} = \frac{\frac{C_t}{C_{tot}} + \frac{C_l}{C_{tot}} + \frac{C_p}{C_{tot}} + \frac{C_a}{C_{tot}}}{4} \tag{6}$$

Using the new, revised formula, yields a value, $0 \leq C_{score} \leq 1$.

Consider participant Bob in Figure 14 that possess characteristics, where P_t = Track 3, P_l = Svenska, P_p = han and P_a = 36-45. A possible coach for this participant could, for instance, be Coach Simon, who has had a total of 16 participants (C_{tot} = 16), where all of the 16 participants belonged to Track 3, resulting in C_t = 16, the majority of them spoke Swedish, C_l = 12, but only half of the participants went by the pronoun *han*, C_p = 8, and were of ages 36-45, C_a = 8. Furthermore, Coach Simon is also located at the same office as Bob. With this known, the score score for Simon can be calculated using the formula seen in Equation 7 below.

$$\begin{aligned} C_{score} &= \frac{\frac{C_t}{C_{tot}} + \frac{C_l}{C_{tot}} + \frac{C_p}{C_{tot}} + \frac{C_a}{C_{tot}}}{4} \\ &= \frac{\frac{16}{16} + \frac{12}{16} + \frac{8}{16} + \frac{8}{16}}{4} = 0.6875 \approx 0.69 \end{aligned} \tag{7}$$

Due to Coach Simon having a total of 16 participants the first penalty is not applied, since $C_{tot} > \beta$, in addition to this, since they are located at the same office, the second penalty is not deducted from the score as well. Since the penalties were not applied, the final score for Coach Simon is approximately 0.69 making Simon a suitable coach for Bob.

3.4.3 Front end

In order to properly display recommendations for any given participant, a simple user interface which the coach could use was needed. To build this basic user interface,

the micro web framework Flask was used. Flask is a library that can be imported into Python which can then be used to create and host a website, the *micro* in the description emphasizes that Flask is meant to be simple to use while also being scalable [23].

Using Flask, a basic but informative user interface was built, the interface enabled a clear and quick view of any given participant along with a summary of their characteristics and activities. The index page of this website can be seen in Figure 16 below.

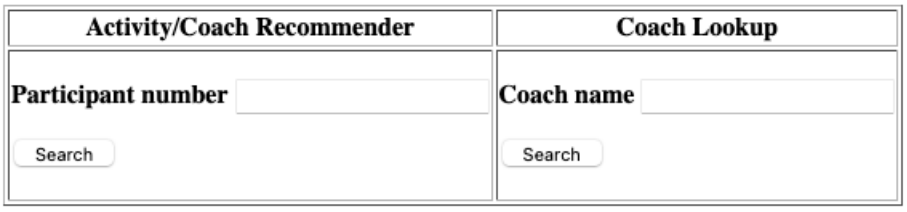

Figure 16 The home page of the user interface, where the coach can type in any participant. It is also possible to lookup any coach.

When first accessing the website, the coach is greeted by an index page containing two input fields, the first can be used to type in any participant number to get a clear view over the participant, entering any coach name in second field makes it also possible to get a clear view of the given coach.

After the coach has put a valid participant number into the leftmost field, the coach is then redirected to the participant view page, which is displayed in Figure 17 below. To increase the comprehensibility, the page in the figure is divided into different parts which is explained in more detail.

Info for 123

number	123
coach	Mary Sue
track	Spår 3: Förstärkt stöd och matchning
lang	Svenska
pronoun	hon
age_span	36-45
r1	missing
office	**Address**: Hittepågatan 5 **Postal Code**: 59054 **Postal Area**: Jobbvik **KA number**: 99999

Current Activities for 123

Name	Count
Stöd med att hitta och söka jobb	6
Studie- och yrkesvägledning	1

(a) A summary of the participant including all of their characteristics and currently practiced activities. The value for the key *r1* is referring to if the participant has achieved a result 1.

Popular activities among similar participants

Name	Total Count
Stöd med att hitta och söka jobb	15
Studie- och yrkesvägledning	1

Popular coaches among similar participants

Coach Name	Counts
John Doe	6
Anna Annsson	2
Susie McDonnell	1
Aloy Finn	1

(b) Popular activities and coaches among all of the participant's similar participants.

Top 3 Coaches recommended overall

Coach Name
Anna Andersson
John Doe
Sabrina Smith

Top 3 Coaches based on weights

Coach Name	Score
Mary Sue	0.72
Aloy Finn	0.65
Anna Annsson	0.57

(c) Coach recommendations for the participant, the leftmost recommendation uses the naive approach while the rightmost utilizes the penalty system.

Figure 17 The participant view page, which gives a clear view of a given participant, containing their characteristics, assigned coach and practiced activities along with their recommendations.

In the above page, the top part aims to give a quick and clear summary of the given participant, while the bottom part is more centered towards displaying the personalized recommendations, in this example, the summary for the participant with the unique identifier *123* can be seen, along with their activities and recommendations.

The first recommendations are essentially activities and coaches extracted from the participant's similar participants, which are then combined to form two clear tables, hence the titles mentioning popular. For instance, the given participant has been assigned the coach Mary Sue, meanwhile, among similar participants, the most assigned coach is

John Doe with a total of 6 similar participants having this coach. The bottom left recommendation is the coach recommendation featuring the first naive approach, while the bottom right is the second approach using the bonus system and weights. By observing the coach recommended by the weights, it is possible to see that the recommended coach is the actual coach that was assigned to the participant, making the recommendation very accurate.

Scrolling further down on the page reveals all of the participant's similar participants, along with their characteristics and practiced activities, this can be seen below in Figure 18.

Similar participants found: 10

Errand		Activities
number	124	
coach	John Doe	
track	Spår 3: Förstärkt stöd och matchning	
lang	Svenska	
pronoun	hon	
age_span	36-45	
r1	missing	
office	**Address**: Hittepågatan 5 **Postal Code**: 59054 **Postal Area**: Jobbvik **KA number**: 99999	
score	**1.0**	

Name	Count
Stöd med att hitta och söka jobb	2

Figure 18 How it looks when scrolling down to inspect each similar participant, in this example only one similar participant is included.

Among the similar participant's characteristics and activities is the similarity score in relation to the given participant, in the above example the score is 1.0, which indicates a perfect match, where all of the characteristics are matching, this can be further confirmed by comparing the characteristics between the two. These are the same participants that were used when producing the popular activities and coaches tables. Furthermore, as seen by the hyperlink, it is possible to click on the number of each participant to get redirected to their specific page, this is essentially the same as typing their number into the home page.

Likewise, by either clicking on the hyperlink that is attached to each coach name or by typing the coach name in the home page, it is possible to get to the page which displays the coach's information, along with their participant characteristic distribution. Clicking on the participant's coach, *Mary Sue*, leads to the page seen below in Figure 19

General info for Mary Sue

name	Mary Sue
Participants	16
Total Result1	18.8%
office	**address**: Hittepågatan 5 **postal_code**: 59054 **postal_area**: Jobbvik **ka**: 99999

Participant division based on characteristics for Mary Sue

Track	Count	Result1
Spår 3: Förstärkt stöd och matchning	16	18.8%

Agespan	Count	Result1
36-45	6	33.3%
46-55	5	0%
56-65	3	0%
18-25	2	50%

Language	Count	Result1
Svenska	16	18.8%

Pronoun	Count	Result1
hon	8	37.5%
han	7	0%
None	1	0%

Figure 19 Summary page for each coach, featuring their information as well as their participant characteristic distribution.

In the above page, to the left side, it is first possible to see general information about the coach and the total amount of participants that the coach has. The *Total Result1* indicates how many of the coach's participants that have achieved a Result1, in other words, the amount of participants that have managed to land an employment.

The other side of the coach view reveals the participant characteristic distribution of the coach, which aims to give a clear and simple view of the various types of participants the coach has. In the above figure, it can be seen that his coach has only had participants who speaks Swedish as their native language, furthermore, observing the pronoun distribution, it can be seen that this coach has an almost equal amount of experience with either pronoun. Likewise, the *Result1* at the side of each table, indicates the amount of participants with the given characteristic that have achieved a Result 1. For instance, while this coach has almost had an equal amount of participants with each pronoun, the participants with the pronoun *hon* have a higher amount of Result1 than participants with the pronoun *han*.

3.5 Evaluation

In order to determine if the recommendation system gave accurate and favourable recommendations, each individual recommendation had to be properly assessed, especially the various methods for the coach recommendation. When a participant is first registered to be enrolled at a provider, the provider spends some time to properly find and determine the most suitable coach possible for the given participant.

Since the coach is manually selected for the participant, the proposed evaluation method was to pick a reasonable amount of participants, then compare each of these participants' assigned coach to all of the coaches recommended, if the assigned coach was included in the recommendations, then the recommendation was considered favourable,

where the higher the coach is on the list of recommended coaches, the more accurate the recommendation. This could properly be measured by calculating the mean reciprocal rank of each recommendation methods' produced list of coaches. Since this statistic measurement focuses on the position of the assigned coach in the list, it makes it an appropriate evaluation method for this project, since the lists produced by the recommendation methods are small and there is only one relevant/correct item in the list which is sought after.

4 Results and Discussion

A fully functional recommendation system which can be accessed using any browser was built. Given any participant, inserting their unique identifier into the system's front end, *Activity/Coach Recommender*, will display a quick overview of the participant such as the participant's track, language spoken and age span, including their current practiced activities as well as their coach and office location. The system will also retrieve all of a participant's similar participants and aggregate their activities and coaches to form two tables, which will give a clear overview of the most popular activities and coaches among all of the similar participants.

Among the popular activities and coaches of similar participants, the recommendation system will try to find and retrieve the top 3 best coaches overall, specifically based on all of the participants individual characteristics, except for the office. Since this recommendation is focused solely on retrieving the most appropriate coaches without regard to the participant's, nor the coach's office, it cannot guarantee that the coaches are located in the same office where the participant is based in, therefore the recommendation could appear as less precise than the other recommendations. Fortunately, as a response to the COVID-19 pandemic, all of the current coaching done at a provider is handled remotely, this means that a participant can choose a provider with an office independently of their current location.

The last recommendation that the system yields is the top 3 coaches based on a penalty system. Unlike the last coach recommendation, which did not take the office of both the participant and coach into account, this recommendation does. The recommendation gathers all of the coaches, and uses the participant's characteristics to find the coach with the most participants that share these specific characteristics, and gives each coach their own unique score based on the percentage of participants. Then, in order for the system to not be biased and recommend a coach that has had, for example, a total of one or two participants with only these characteristics, penalties are applied. More specifically, if the coach has had less than a specific amount of total participants, or if the coach is not located at the same office as the participant, a small amount is subtracted from the coach's score.

4.1 Experiments

To give a clear picture of how the recommendation system worked in practice, multiple experiments were conducted, where each recommendation method was tested and measured using the reciprocal rank on an average of 100 participants out of a total of 2164

participants in the database. In order to get as a precise measurement as possible, the mean reciprocal rank was calculated after trying a various different number of iterations.

While running the following experiments, the selection and retrieval of each set of participants were all entirely random and automated using Python. Although the majority of the collected participants was unique and random for each iteration, occasionally, due to how the retrievals happen, the same participant could sometime occur in multiple different chunks. While measuring the amount of duplicate participants, on average, around 5 participants in 100 were duplicates, which leaves an error margin of around 5%. It is also worth to note that due to certain participants missing a valid native language entry in the data, their native language was defaulted to Swedish, this may introduce a potential source of errors.

The first experiment was conducted on completely arbitrary selected participants, meaning no particular characteristic was specified while retrieving these participants. The calculated mean reciprocal ranks of each recommendation method from the experiment, including the number of iterations followed by the standard deviations can be seen in Table 1.

When it comes to determining whether the calculated mean ranks of the methods can be classified as good or bad, since each list of recommended coaches contains three potential coaches, it could be argued that an acceptable rank would be $\frac{1}{3}$. With this rank, the actual coach is present in every single list, albeit not placed first.

	# of iterations	**Mean reciprocal rank**	**Std. deviation**
Similar participants	1	0.46	0.42
Naive approach	1	0.15	0.29
Penalty system	1	0.34	0.38
Similar participants	5	0.49	0.42
Naive approach	5	0.23	0.46
Penalty system	5	0.35	0.39
Similar participants	10	0.48	0.42
Naive approach	10	0.22	0.36
Penalty system	10	0.34	0.39

Table 1 Results from the first experiment, featuring 100 randomly selected participants, their respective mean reciprocal ranks, their standard deviations and the amount of iterations.

In the above table, the first row, *similar participants*, represents the recommendation method which gathered all of the participant's similar participants and then summarized all of their assigned coaches to a list in descending order. The second row, *naive ap-*

proach, is the naive recommendation method which only checks for coaches that has had participants with the exact same characteristics. Lastly, the third row, *penalty system*, represents the recommendation method which utilizes the penalty system.

Observing the above table, it can be seen that among all of the different methods, the similar participants was the one that performed the best by a large margin, growing in rank as the amount of iterations increase. This simple, yet effective recommendation method managed to predict the participant's assigned coach and place it at the top of the list, nearly half of the times. This is not surprising, considering the possibility that if two participants are similar, meaning they share every characteristic, including being located at the same office, chances are that there is a high probability that they share the same coach as well. The standard deviation is quite large for this method, indicating that the data is widely spread. In the case of this method, a mean rank of almost 0.5 with a consistent standard deviation of 0.42 means that while the rank is good, it is very inconsistent with values scattered. However, this is not necessarily undesirable, due to the utilization of the mean reciprocal rank, the possible values for the ranks can either be $1, \frac{1}{2}, \frac{1}{3}$ or 0, which explains the high overall standard deviations for all methods.

Moving on the to the second recommendation method, the naive approach which had on average rank of 0.2. It is not a surprise that the rank is low, since in order for it to be accurate for a specific participant with certain characteristics, there have to be coaches that has had a history of participants with the exact same characteristics. When it comes to the standard deviation, with an average standard deviation of 0.33, means that it is more consistent than the previous method, similar participants. While conducting the tests, occasionally, the list of coaches retrieved by the recommendation method was empty, meaning that no applicable coaches were given by the method, which could have impacted the score significantly. Perhaps having access to more data, and therefore more participants, could have helped in minimizing the amount of empty lists and lowering the standard deviation as well.

The last recommendation method featuring the penalty system, performed slightly better than the naive approach, but much worse than the similar participants method. With an average rank of 0.35, means that the method performed better than the pre-determined *good* rank of $\frac{1}{3}$. The method also had a lower average standard deviation of 0.27, which is worse than the naive approach but better than similar participants.

In the next experiment conducted, instead of retrieving random participants, participants from specific tracks were retrieved, featuring participants from all tracks 1 to 4. The following test for each track had iterations of 5, the results can be seen in Table 3 below.

It is worth to note that out of all 2164 participants, tracks are not distributed evenly, as a result of this, due to how the retrievals of the participant sets happens, it was not

possible to retrieve a consistent set of 100 participants for each iteration during this test, therefore, for each test made, the size of the used participant set is displayed in its own column. The full track distribution among the participants can be seen in Table 2.

Track 1	Track 2	Track 3	Track 4	Total
678	123	1029	334	2164

Table 2 The track distribution of the participants, displaying how many participants are using each track.

	Track	Mean reciprocal Rank	Std. deviation	Set size
Similar participants	1	0.43	0.41	59
Naive approach	1	0.16	0.32	59
Penalty system	1	0.31	0.39	59
Similar participants	2	0.49	0.40	31
Naive approach	2	0.56	0.44	31
Penalty system	2	0.54	0.44	31
Similar participants	3	0.47	0.42	100
Naive approach	3	0.15	0.31	100
Penalty system	3	0.31	0.37	100
Similar participants	4	0.50	0.44	42
Naive approach	4	0.47	0.45	42
Penalty system	4	0.38	0.39	42

Table 3 Experiment two, using only participants from every single track. The new column, set size, contains the amount of participants used while conducting the test. Each test was made using 5 iterations.

Inspecting the table above, like the previous experiment, the similar participants method still remains very consistent, having only a decrease in rank for track 1. The data is a little bit more inconsistent and scattered than the last experiment, as can be observed by the small increase in the standard deviation.

The naive method performed inconsistent, having only an increase in ranks during the test which featured participant sets of small size, such as the tests featuring participants of track 2 and track 4. The standard deviation is significantly increased for the naive approach in track 2 and track 4 as well, which could be due to the smaller participant set containing a higher set of outliers. The sudden increase in rank could be due to the small set of participants not including participants who have been recommended empty lists of coaches, which leads to an overall increase in rank. Similar to this, due to the

small set, outliers within the data could ultimately have a higher effect on the rank. Another explanation for the sudden increase could be that due to the small set, duplicate participants could have a higher chance of appearing and, depending on the rank of their individual recommendation, could have a large impact on the final rank.

Like the similar participants method, the penalty system managed to also maintain consistency, having only a minor decrease in the overall rank. Due to the consistency in the overall rank, the same consistency can be seen in the standard deviation. It can be said that when it comes to the penalty system, it performs consistent, regardless of the type of participant, which is surprisingly good.

The last experiment featured only participants whose native language was Swedish. The results from this test are very similar to the results from the first experiment, meaning that specifying the language did not have have any significant impact on the ranks of the recommendation methods.

5 Conclusion

In this thesis, a recommendation system utilizing three different recommendation methods was created, thus achieving the goal this project set out to accomplish. Out of all the different methods, the best performing method was the similar participants method. While the other two methods did often have decent mean reciprocal ranks, it can be interpreted from the results of this project, that in order to develop a good performing recommendation method, advanced and complicated formulas are not necessarily always required. Although the penalty system only had acceptable performance, it was the overall most consistent method managing to have the lowest standard deviation. With that said, the penalty system definitely has potential, there just needs to be some further testing of the parameters and the values they deduct. Once the sweet spot for both the parameters and the values has been found, then the method will truly shine, and possibly outperform the similar participants method.

The other main goal for this thesis was to utilize an appropriate statistic measurement, that could be used to evaluate all of the recommendation methods, calculating the mean reciprocal rank proved to be a suitable heuristic. Since it only considers the single highest-ranked relevant item, it is appropriate for each list of recommended coaches, due to there being only one relevant item located in each list, which is the participant's assigned coach. Furthermore, another way to properly test this system is to use it in practice, which means it would need to be deployed and used by the coaches of a provider, in addition to this, it would also be interesting to have had some real input from a provider and their opinion regarding the list of recommended coaches that each method produces.

Opting to use a graph database, such as Neo4j during this project proved to be both helpful and beneficial. The highly connected data within the database helped improve the accuracy of actually retrieving similar participants, which in the end led to better recommendations for the similar participants method. The use of the graph algorithms that could be added in the Neo4j database, such as the node similarity algorithm, meant that more time during the project could be spent on improving the various recommendation methods. If needed, expanding the database by adding more participants should not be a problem due to the scalability and flexibility of Neo4j.

6 Future work

Due to time constraints imposed by the project, a few improvements and additions have been left to the future work section.

As it is now, in order to determine the similarity between a pair of participants (nodes in the database), the included graph algorithm in Neo4j, node similarity, is used. This algorithm is based on the Jaccard metric to determine the similarity between a pair of nodes. In the graph algorithm library of Neo4j, there are a wide range of other algorithms which can be used to determine similarity between the nodes, albeit, with the exception of the node similarity algorithm, the majority of them are still experimental and in alpha, however, they can still be used. An interesting alternative that was not tested during the project was to try other similarity functions and then compare all of their different performances and their effects on the recommendations, to see if it could have possibly improved them.

An important aspect of any recommendation system is having access to large amounts of data that can serve as a foundation for the database. During this project, a dataset consisting of 2164 unique participants was used to populate the database and assisted in producing the recommendations. While a dataset of this size is probably sufficient enough in order to make well-performing recommendations, still, improving the database by adding more participants would most likely lead to the recommendation system producing even better recommendations. Furthermore, there are some potential improvements that could be made to the recommendation system's methods, such as the naive approach and the penalty system, especially for the penalty system. As it is now, there are only two penalties, each depending on a different set parameter, which deducts a specified value from a coach's score. For starters, these parameters as well as the values they deduct from the score can most likely be tinkered with further in order to possibly increase the overall performance of the method. Another addition would be to add more parameters with values that either deduct, or, depending on the performance, add to the coach's score, transforming the penalty system to a hybrid system.

Currently, Arbetsförmedlingen is planning to phase out their most primarily used program, Stöd och matchning (STOM), aiming to replace it with a new, similar program called Rusta och matcha (KROM). In KROM, the everyday work will essentially function the exact same way as it did with STOM, as a result of this, the recommendation system could be adapted to work with KROM with only some minor modifications. One of the different changes that KROM brings, are the removal of tracks and the addition of levels instead, namely, A, B and C, which functions like STOM's tracks, where C represents that the participant is far away from a job and A that they are very close to getting a job.

References

[1] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734–749, 2005. [Online]. Available: https://ieeexplore.ieee.org/document/1423975

[2] AlphaCE. AlphaCE. [Online]. Available: https://www.alphace.se/

[3] Arbetsförmedlingen. Arbetslöshet - Arbetsförmedlingen. [Online]. Available: https://arbetsformedlingen.se/statistik/arbetsloshet

[4] ——. Introduktion till arbete - Arbetsförmedlingen. [Online]. Available: https://arbetsformedlingen.se/other-languages/english-engelska/extra-stod/stod-a-o/introduktion-till-arbete

[5] ——. Stöd och matchning - Arbetsförmedlingen. [Online]. Available: https://arbetsformedlingen.se/other-languages/english-engelska/extra-stod/stod-a-o/stod-och-matchning

[6] ——. Våra tjänster - Arbetsförmedlingen. [Online]. Available: https://arbetsformedlingen.se/om-oss/for-leverantorer/vara-tjanster

[7] Assedon AB. Assedon - IT-lösning för kompletterande aktörer. [Online]. Available: https://www.assedon.se

[8] J. Barrasa. (2017, Aug.) RDF Triple Stores vs. Labeled Property Graphs: What's the Difference? [Online]. Available: https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference/

[9] BluePi. (2015, Nov.) Classifying Different Types of Recommender Systems — BluePi. [Online]. Available: https://www.bluepiit.com/blog/classifying-recommender-systems/

[10] R. Boyd, M. Hunger, and W. Lyon. (2016, Mar.) RDBMS Graphs: SQL vs. Cypher Query Languages. [Online]. Available: https://neo4j.com/blog/sql-vs-cypher-query-languages/

[11] S. Das, J. Srinivasan, M. Perry, E. Chong, and J. Banerjee, "A tale of two graphs: Property graphs as rdf in oracle," Mar. 2014. [Online]. Available: https://www.researchgate.net/publication/264702160_A_Tale_of_Two_Graphs_Property_Graphs_as_RDF_in_Oracle

[12] DB-Engines. DB-Engines - popularity ranking of database management systems. [Online]. Available: https://db-engines.com/en/ranking

[13] ——. DB-Engines Ranking - popularity ranking of graph DBMS. [Online]. Available: https://db-engines.com/en/ranking/graph+dbms

[14] E. Eifrem, I. Robinson, and J. Webber, *Graph Databases*. O'Reilly, Jun. 2015.

[15] C. A. Gomez-Uribe and N. Hunt, "The netflix recommender system: Algorithms, business value, and innovation," *ACM Trans. Manage. Inf. Syst.*, vol. 6, no. 4, Dec. 2016. [Online]. Available: https://doi.org/10.1145/2843948

[16] F. Isinkaye, Y. Folajimi, and B. Ojokoh, "Recommendation systems: Principles, methods and evaluation," *Egyptian Informatics Journal*, vol. 16, no. 3, pp. 261–273, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1110866515000341

[17] D. McCreary. (2018, Mar.) The Neighborhood Walk Story. [Online]. Available: https://dmccreary.medium.com/how-to-explain-index-free-adjacency-to-your-manager-1a8e68ec664a

[18] Neo4j, Inc. (2020) Neo4j Python Driver 4.2 – Neo4j Python Driver 4.2. [Online]. Available: https://neo4j.com/docs/api/python-driver/current/

[19] ——. (2020) The Neo4j Graph Data Science Library Manual v1.5 - Neo4j Graph Data Science. [Online]. Available: https://neo4j.com/docs/graph-data-science/current/

[20] Neo4j Staff. (2021) Company - Neo4j Graph Database Platform. [Online]. Available: https://neo4j.com/company/

[21] F. Ricci, L. Rokach, and B. Shapira, "Introduction to recommender systems handbook," pp. 1–35, 2011. [Online]. Available: https://doi.org/10.1007/978-0-387-85820-3_1

[22] W. Shalaby, B. AlAila, M. Korayem, L. Pournajaf, K. AlJadda, S. Quinn, and W. Zadrozny, "Help me find a job: A graph-based approach for job recommendation at scale," in *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017, pp. 1544–1553.

[23] The Pallets Projects. (2020) Welcome to Flask Documentation – Documentation 1.1.x. [Online]. Available: https://flask.palletsprojects.com/en/1.1.x/

[24] World Wide Web Consortium. (1999, Feb.) Resource Description Framework (RDF) Model and Syntax Specification. [Online]. Available: https://www.w3.org/TR/PR-rdf-syntax/Overview.html

[25] ——. (2004, Feb.) Resource Description Framework (RDF): Concepts and Abstract Syntax. [Online]. Available: https://www.w3.org/TR/rdf-concepts/#section-data-model

A Document structures

A.1 Participant


```
{
    "participant_number": "123",
    "track": "Spår 1: Grundläggande stöd och matchning",
    "age_span": "56-65",
    "period": "2021-01-01 - 2021-04-01",
    "order_number": "12345",
    "delivery_address": "Hittepågatan 5 59054 Jobbvik",
    "af_supervisor":{
        "name": "John Doe",
        "email": "john@workplace.com",
        "phone": "123456",
        "office": "Workplace AB"
    },
    "documents_list":{
        "0": {
        "type": "Avvikelserapport",
        "registration_date":,
        "coach":,
        "pronoun":
        },
        "1": {
        "type": "Periodisk rapport",
        "registration_date": "2021-02-05",
        "ka_number": "99999",
        "coach": "Mary Sue",
        "pronoun": "hon"
        },
        "2": {
        "type": "Resultat 1",
        "registration_date": "2021-02-10",
        "ka_number": "99999",
        "R1_start": "2021-02-10",
        "R1_reason": "Ersättningsgrund: Utbildning",
        "coach": "Mary Sue",
        "pronoun": "henne"
        },
        "3": {
        "type": "Resultat 2",
        "registration_date": "2021-06-10",
        "ka_number": "99999",
        "R2_client": "2021-02-10 23345 Secret Academy",
        "R2_info": "2021-02-10 40",
        "R2_reason": "Ersättningsgrund: Utbildning",
        "coach": "Mary Sue",
        "pronoun": "henne"
        }
    }
}
```

A.2 Invoice


```
{
    "invoice_id": "123456",
    "period": "2021-01-01 - 2021-01-31",
    "participant_number": "123",
    "ka_number": "99999",
    "order_number": "12345",
    "articles":{
        "0":{
            "0": "5",
            "1": "Förstärkt stöd och matchning",
            "2": "19 dag(ar)",
            "3": "10,00 kr",
            "4": "190,00 kr",
        }
    },
    "activities":{
        "0": "Motiverande samtal",
        "1": "Gruppaktiviteter",
    }

}
```

A.3 Order


```
{
    "order_number": "12345",
    "period": "2021-01-01 - 2021-04-01",
    "participant_number": "123",
    "ka_number": "99999",
    "articles":{
        "0":{
            "0": "5",
            "1": "Förstärkt stöd och matchning",
            "2": "63 dag(ar)",
            "3": "10,00 kr",
            "4": "630,00 kr",
        }
    }
}
```


A.4 Delivery


```
{
    "order_number": "12345",
    "invoice_number": "123456",
    "period": "2021-01-01 - 2021-01-31",
    "participant_number": "123",
    "ka_number": "99999",
    "articles":{
        "0":{
            "0": "5",
            "1": "Förstärkt stöd och matchning",
            "2": "19 dag(ar)",
            "3": "10,00 kr",
            "4": "190,00 kr",
        }
    }
}
```