



UNIVERSITE CATHOLIQUE DE
LOUVAIN

ECOLE POLYTECHNIQUE DE
LOUVAIN



The Facebook Love Connector

Supervisor: YVES DEVILLE
Readers: FRANÇOIS AUBRY
CYRILLE DEJEMEPPE

Thesis submitted for the Master's degree
in computer science and engineering
options: Networking and Security
by THOMAS DEWANDELEER

Louvain-la-Neuve
Academic year 2015-2016

Contents

1	Abstract	3
2	Acknowledgements	4
3	Introduction	5
4	The Graph Theory	6
4.1	Introduction to the Graph Theory	6
4.1.1	Undirected Graphs	6
4.1.2	Directed Graphs	7
4.1.3	Labeled Graphs	8
4.2	The concept of graph in this work	9
5	Graph Matching	10
5.1	Structure and content analysis	11
6	Algorithms for graph matching	12
6.1	Graph Edit Distance	12
6.1.1	Properties	12
6.1.2	Operations	13
6.1.3	How to compute Graph Edit Distance	14
6.2	Graph Edit Distance Variations	15
6.3	Greedy Search	16
6.4	Tabu meta-heuristic	17
6.5	Reactive Tabu search	18
7	Neighbour Matching by Mladen Nikolic	20
7.1	Introduction to the algorithm	20
7.2	Mathematical Explanation	21
7.3	Algorithmic Explanation	24
7.4	Implementation Explanation	27
7.4.1	Initialisation	27
7.4.2	Iteration	27
7.4.3	Optimization function	27
7.4.4	Similarity Computation	28
7.5	The Hungarian Algorithm	29
7.5.1	Introduction	29
7.5.2	How does it work ?	29
7.5.3	Integration in the work	33
7.5.4	Implementation	33

7.6	Advantages and Drawbacks	35
8	Evaluation Function	36
9	The Application	39
9.1	Presentation	39
9.2	How does the Application work ?	45
9.2.1	Main Menu	45
9.2.2	Profile Information	45
9.2.3	Choosing Page	45
9.2.4	Comparison Page	46
9.3	Video Presentation	48
10	Technologies Used	49
11	Facebook	50
11.1	Facebook API	50
11.2	What is used	50
12	Conclusion	52
13	Further Work	52
14	Appendix	53

1 Abstract

This work focuses on the study of graphs and more precisely, a graph comparison method. It has been done during the academic year 2014-2015 at UCL (Université Catholique de Louvain) and has been done as my work at the end of study. The work has been completed during the academic year 2015-2016.

The context of this work was to try developing a method that would quantify the similarities between two people based on their Facebook information.

Nowadays, a lot of comparison methods for graphs are based on the structure of the graphs but there are very few methods that do the comparison between graphs regarding the structure of it but also the information that is embedded in it.

This work tries to find a good solution to this problem by using the combination of an algorithm that compares the structure of two graphs and an algorithm that compares the information embedded in the graphs.

In order to illustrate the work, an android application applying those algorithms was developed.

2 Acknowledgements

I thank Professor Yves Deville, responsible for this work, for his support and advice during the implementation of this work. I also thank François Aubry who helped me enormously during the elaboration of the work.

This work was realized as part of my studies in computer science and is my final assignment study. This work thus marks the end of my studies at the Université Catholique de Louvain.

I also thank Mr Vincent Simons (Art Director at BBDO) for his help with the making of the design of the application.

3 Introduction

Nowadays, a lot of information is available on the internet and on social networks. People tend to put information about themselves online and this information could be used to do many things. In this work, a method will be developed to compare Facebook friends with each other. The goal is to find which of your friends are most like you and thus help you find love.

In order to do so, many things will be done. The first thing is to find out how to represent the data that we will retrieve from Facebook. This will be the point of chapter four.

After that, investigations must be made about algorithms that can be used to compare the data. Most algorithms that are discussed in scientific papers are algorithms that deal with structural similarities of the data. The goal, in this work, is to make an algorithm that deals with the structural similarities of the data but also with the content of these. This will be discussed from chapter five to chapter eight.

You may think there are a lot of images and representations in this work. This was done because some concepts discussed in the work are easier to understand by using drawings.

4 The Graph Theory

4.1 Introduction to the Graph Theory

The graph theory is a theory that takes place in the scientific and mathematics world. The graph theory is the study of graphs. The word “graph” was first used in this sense by J.J. Sylvester in 1878 [17]. A graph is a structure composed by “vertices” that can also be called “nodes” and by “edges” that are represented by the lines between the nodes. An entire theory is dedicated to the graphs because there is a very large amount of graphs with different characteristics and so, many theories around them.

4.1.1 Undirected Graphs

The first point that must be introduced is the graph. There are a lot of different definitions of graphs but we will try to make it the most simple. A graph is an ordered pair $G = (V, E)$ where G represents the graph, V is the set of vertices and E is the set of edges. As said earlier, a vertex is a node, so it can be represented by its name unlike an edge which can not be. Indeed, an edge is a connection between two different vertices. Thereby, an edge is represented by the “name” of the two different vertices that it connects. The two vertices belonging to a certain edge are called the endpoints of the edge. In this first definition of a graph, it can be represented with an unordered pair of two elements of the set of vertices (currently we do not care about the starting point or the destination of the edge). We call this type of graph undirected or simple.

A simple graph can be represented like this:

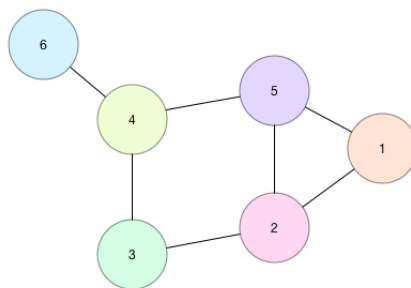


Figure 1: A Simple Graph

As said in the previous paragraph, we can represent this graph with two different sets. The first set is the set of the vertices which is $V=1,2,3,4,5,6$. And the sec-

ond set is the one of the edges which is $E = \{\{1, 2\}, \{1, 5\}, \{2, 3\}, \{2, 5\}, \{3, 4\}, \{4, 5\}, \{4, 6\}\}$.

The order of a graph is the number of vertices composing the graph. In 1, the order of the graph is 6. We call the size of the graph, the number of edges that are in the graph. In the example, the size of the graph is 7.

The degree of a vertex is the number of edges connected to the vertex in question. If a vertex is connected to itself by an edge, the edge is counted twice in the degree of the vertex. For example, node “4” has a degree of 3.

Now that the base of the definition of a graph is explained, the other types of graphs can be explained.

4.1.2 Directed Graphs

The first type that can be presented is the directed graph. Directed graphs are much alike simple graphs. But unlike the simple graph where the order of the edge does not matter, in directed graphs, the starting point and the destination of the edge do matter. As it is said in the name of the graphs, they are directed so the set of edges is now a set of ordered pairs of vertices. Directed graphs are expressed $G = (V, A)$. V being the set of vertices and A being the set of edges.

With this type of graphs comes also a new vocabulary. For an edge $e = (X, Y)$, coming from the node X to the node Y , we can say that the head of the edge is X and its tail is Y . Because of the fact that now, we care about the direction of the edges, we can also refine the definition of degree of a node. In the previous definition, the degree of a node was the amount of edges connected to it. For directed graphs, two new terms are added: the in-degree and the out-degree. Those represent the number of edges going out of the node or coming to the node. The graph in 1 can be modified to become a directed graph. It would then look like this:

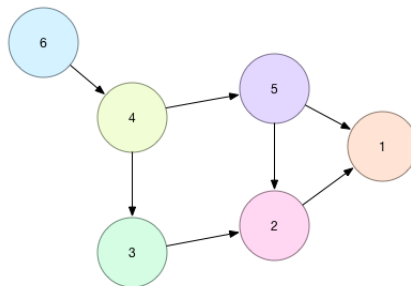


Figure 2: A Directed Graph

As the previous graph, this one can also be described with the help of 2 sets (ver-

tices and edges). The set of vertices is the same as the previous graph but the set of edges is different and looks like $E=\{(2, 1), (5, 1), (3, 2), (5, 2), (4, 3), (4, 5), (6, 4)\}$. In directed graphs, there is also a notion of predecessor and successor. If we have two vertices X and Y and the node Y is reachable from the node X, we can say that Y is a successor of X. In this case, we can also say that X is a predecessor of Y. If they are directly connected, we can say that they are direct successor or direct predecessor. In the example, node 5 is a successor of node 6 and node 3 is a direct predecessor of node 2.

4.1.3 Labeled Graphs

We have also labelled graphs. Labelled graphs are graphs that have labels attached to their vertices or edges. The labels can be made words or numbers. But if the labels are number that are used to represent informations like distances or costs, the graphs will be called weighted graphs.

The labels can be used to add information into a graph and to give nodes a certain identity. While a simple graph can represent many different things, a labelled graph can be used to concretely illustrate a precise data.

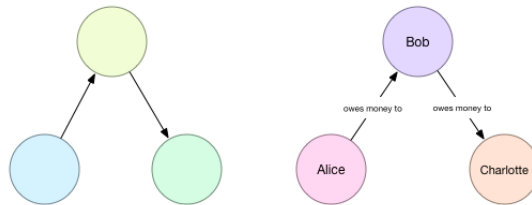


Figure 3: The utility of Labelled Graphs

4.2 The concept of graph in this work

As explained earlier, there are a lot of types of graphs. In this work, we are working with a certain type of graphs which will be explained. Like in normal graphs, the graphs in this work are made of nodes and edges. The nodes represent entities and edges represent relations between the entities. The edges are directed, which means that they have a direction. The direction can be used to explain the relation. For example, we can have a Facebook user who has his birthday on a certain day. The following drawing can represent this relation:

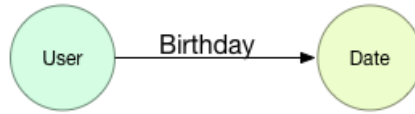


Figure 4: Classic labeled graph

Nodes and edges have labels in order to differentiate them. In our work, we used labels in a quite different kind of way. The labels on the edges are used to name the relation between two entities, which is like a regular labelled graph. But on the nodes, we have two different labels which is unusual. The first label is used to give the entities a type, like "name", "city", ... And the second label is used to give the entities a value. In our situation, we need to have entities that can have a type and a value. The label on the edges is informative and does not really play a role in the algorithm.



Figure 5: Graph in this work

5 Graph Matching

Now that the graph theory has been explained, we can go to the next step. The goal of the work is to find a way to compare two graphs. This must be done in order to know if two people seem to have similar profiles or very different ones. There is a lot of research going on graphs similarities measurements. Multiple techniques are being developed and tested by many people. All of those ways to compute the distance between graphs are based on different characteristics of the graphs. Some are comparing the graphs as a whole, other ones are comparing each node separately.

This variety of algorithms will set a first problem that will be to determine which of these algorithms will fits our needs. Indeed, because of the fact they are all based on different principles, they will all have different characteristics that will be useful (or not) for the resolution of the problem.

For the current problem, there are properties that we want. First of all, it would be nice if the result of the algorithm can be interpreted on its own. We want the measure of the distance between two graphs to be meaningful by itself and not imposing any comparison to other measures to make sense. We want the result to be bounded. There must be a minimal value and a maximal value. That way, we can say with only one measure that the 2 graphs are similar or not.

Of course, it is possible to modify algorithms that do not give a bounded result in order for them to give a bounded result. But we must not forget to do so if we choose to use one of them.

One other property that we want for our algorithm is its ability to work on labelled graphs. In our problem, we want to evaluate the distance between graphs. But in those graphs, we do not have only one kind of node, we have plenty of different nodes. We have to use labels on the nodes in order to differentiate every kind of node. So, the algorithm must be adapted to compare nodes that have labels.

5.1 Structure and content analysis

As said earlier, the goal of this work is to develop a method in order to compare different people based on their Facebook characteristics. To do so, our method will have to compare two different aspects of the graphs that will be used to represent the people. These two aspects are the structure of the graph and the content thereof.

For the first part, we will have to search for an algorithm as explained in the previous point. The algorithm will have to fit the work and have certain desired properties. Different algorithms will be presented and we will explain why they are suitable, or not, for the project.

For the content comparison, an other algorithm will have to be developed. The goal of this algorithm is to compare only the content of the graphs and it must give a score that quantifies the similarities between the two graphs. We will call this algorithm "the evaluation function". This point will be discussed after the graph matching techniques

6 Algorithms for graph matching

Now that we have identified what we are looking for, we can search for a method to compute the distance between two graphs having the properties that we want. This part of the work will present the algorithms that could potentially be used.

6.1 Graph Edit Distance

The graph edit distance is a method of graph matching operating at the structural level of the graphs.

This method is defined by a set of edit operations. The operations are delete, insert and substitute. Those operations can be used to modify the first graph until a sub-graph isomorphism to the second graph is found. Each operation comes with a cost and so each time an operation is performed, the value of the edit distance gets bigger.

To compute the overall effect of the edit operations that were performed on the first graph, the cost of each operations can be added. We can see that two very different graphs will have a big edit distance while nearly similar graphs will have a small one.

The "edit distance" is not only used in order to compare graphs. It can also be used to compare strings in order to determine the "distance" between them. The "edit distance" is also used in bio-informatics, in order to quantify the similarity of macromolecules like DNA. Indeed, those molecules can be represented by strings containing the letters G, A, T and C.

6.1.1 Properties

The graph edit distance has several properties.

The first one is that every operation except one has a positive cost above zero. The only operation with a zero cost is the substitution operation when substituting a part of the graph with itself.

The following formula expresses another property of the edit distance (A is a graph): $d(A, A) = 0$. This is easily understandable since the number operations to get to the same graph as the initial one is zero.

From this property, we can make the following formula (A and B are a graph): $d(A, B) > 0$ when $a \neq b$. Because it is needed to do an operation with a cost greater than zero to get to B from A.

An other property is that for every operation, the inverse operation exists and has the same cost.

Due to the previous property, we can determine this formula (A and B are a graph): $d(A, B) = d(B, A)$.

6.1.2 Operations

As explained, there are three different operations available. The name of those operations are quite representative of what they do. The first one is the delete operations. This can be used to remove a node or an edge of the current graph.

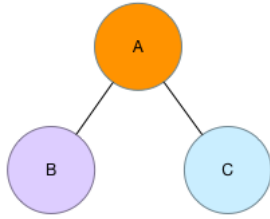


Figure 6: Basic graph

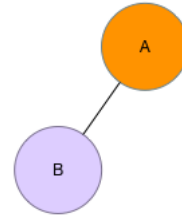


Figure 7: Deletion of node C

The second one is insert. It can be used to add a node or an edge to the current graph.

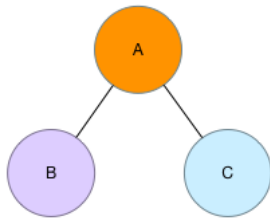


Figure 8: Basic graph

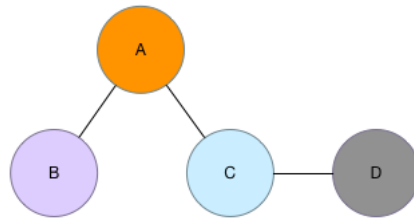


Figure 9: Adding node D

The last one is substitute. It is maybe the less expressive one. This operation can be used to change the label of a node or an edge. As explained in the introduction to the graph theory, graphs can have labelled nodes or edges. This method can be used to substitute one label for another one.

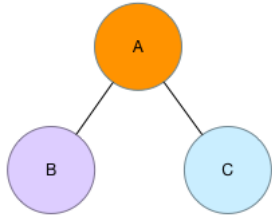


Figure 10: Basic graph

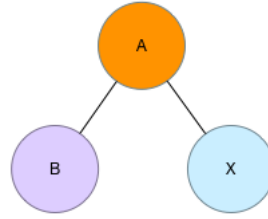


Figure 11: Substitution of node C by node X

Now that this nice graph matching method has been presented, we must check that it can fit our needs.

The method is easy to understand. The basic idea of the algorithm is quite nice and simple which makes it an interesting one. But it is not the only matching algorithm that has been explored in this work.

6.1.3 How to compute Graph Edit Distance

A very widely used method for exact computation of the GED is the A*GED. This is a GED computation based on the A* search algorithm.

A* Algorithm The A* search algorithm is a path search algorithm in a graph between an initial node and an end node both given. It uses a heuristic evaluation on each node to estimate the best path through it, and then visit the nodes in order of this heuristic evaluation. This is a simple algorithm, requiring no preprocessing, and consuming only small amount of memory.

In a more formal way, a node p in the search tree has 2 different values. Namely, his current cost and the estimated cost to the end node. Those values are represented by $g(p)$ (the current cost) and $lb(p)$ (the estimated cost to finish).

It is important that the heuristic function does not underestimate $lb(p)$ if we want to guarantee the optimality of the solution. The heuristic function must also be faster than the function that can be used to compute the exact final solution.

The sum of the current cost and the estimated remaining cost represents the total cost attached to a node. The algorithm will expand and explore the node with the smallest total cost.

The exploring and expansion is repeated when the end node is reached and so when we have transformed the first graph into the second one.

6.2 Graph Edit Distance Variations

This method is not really different from the one above but it is presented to show the multiple variations of the graph edit distance. As said above, the graph edit distance is quite simple and easy to understand. This method can be used in many different fields. It can be tempting for developers to create their own version of the algorithm. The basic graph edit distance only has three different operations: delete, insert and substitute. Several variations are based on the regular graph edit distance which is extended by new operations.

The article "Graph Edit Distance with Node Splitting and Merging, and Its Application to Diatom Identification" [12] is an example of work that adds certain operations to the three basic operations of the graph edit distance. In this article, we see two new operations namely the node splitting and the node merging. These new operations are inverse operations meaning that a node split can be undone with one node merge.

The first operation is the node split. The name of the operation is quite meaningful. This operation allows to split a node in multiple nodes.

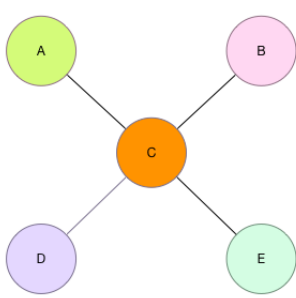


Figure 12: Basic graph

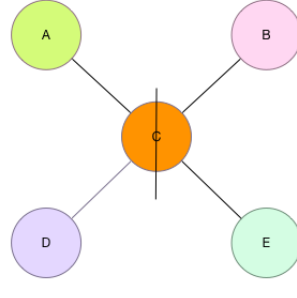


Figure 13: Splitting node C

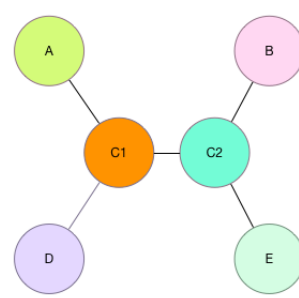


Figure 14: Result graph

This is an illustration of the node splitting operation. In the first graph, we have five nodes. In the second drawing we can see the intention of node splitting. We want to split the node C in two parts. We can note that the splitting line is vertical, cutting the node in two parts where one part is still connected to A and D whereas the second part is still connected with B and E. The last drawing represents the graph after the split operation. We have two new nodes C1 and C2.

The other operation is merge. This operation allows to merge nodes. Here is a small example of node merging.

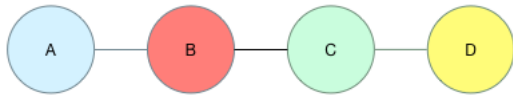


Figure 15: Basic graph



Figure 16: Merging node B and C into a new node E

We can see that this operation is the exact inverse operation compared to the split operation. In this example, we merge nodes B and C and we create a node E sharing the neighbours of B and C.

These examples of variations of the basic graph edit distance are interesting because they carry the same properties as the basic version but they are more efficient due to the new operations that were added. Some can make the computation shorter because they can quickly find a solution and other can reduce the number of operations needed to come to a solution. Another advantage is that these kind of algorithms can be used in different contexts by extending their operations, they open to a wider scope.

6.3 Greedy Search

A greedy algorithm is an algorithm that follows the principle of doing, step by step, a local optimum choice. In some cases, this approach allows to reach a global optimum but it does not ensure the optimality of the solution.

This kind of algorithm is widely used and in many different fields. It can, for example, be used in a machine that give change back.

In the current European currency, the available coins are 1, 2, 5, 10, 20, 50, 100, 200 (in cents).

Using a greedy algorithm will always give an optimal solution in terms of the number of returned coins. For example, if the machine has to give back 37 centimes, using a greedy algorithm, it will make a coin of 20 cents, one coin of 10 cents, a coin of 5 cents and a coin of 2 cents. Which is the optimal solution.

This type of algorithm can also be used in graphs problem. One algorithm has been discussed in the paper "Measuring the similarity of labelled graphs" written

by P.-A. Champin and C. Solnon [7].

The algorithm begins with an empty set called "m" representing the mapping between the two compared graphs. It will iterate and at each iteration it will add a couple of vertices taken from the candidate set "cand". The "cand" set is defined by $((V \times V') - m)$. V is the set of vertices of the first graph, V' is the set of vertices of the second graph and so $(V \times V')$ is the mapping between the vertices of the first graph with the ones from the second graph. During each iteration, the couple of vertices is chosen in a greedy way. This means that the algorithm is going to select the couples that most increase the similarity measure.

After the selection process, there may be more than one selected candidate. Only one couple is added by iteration so the algorithm has to break the tie. This can be done using several techniques. The easiest one is choosing randomly in the candidate. But we can also imagine techniques that will look for certain properties of the candidate that can break the tie (number of in-neighbours, number of out-neighbours,...).

The iterations will go on until a local optimum is found which means until the addition of any couples left does not increase the similarity measure anymore.

The time complexity of this greedy algorithm is polynomial ($O(|V|X|V'|^2)$). This is a rather low complexity which is good, but this algorithm also has drawbacks. First of all, due to its conception, this algorithm does not backtracks and is not complete. And above all, this algorithm does not find the best solution every time and if it does, the algorithm can not be used to prove that it found the best solution.

6.4 Tabu meta-heuristic

The tabu search is another algorithm that can be used to measure graphs similarities. This was firstly introduced by Fred W. Glover in 1989 [10].

The idea of the Tabu search is, from a given position, to explore the neighbourhood and choose the position in this neighbourhood that minimizes the objective function (in a minimization problem). It is essential to note that this may lead to increase the value of the function. That is the case when all points of the neighbourhood have a higher value.

The algorithm has the ability to escape local minima. However, there is a risk that during the next step, we fall into the local minimum we just escaped. That is why it is necessary to the heuristic to have a memory. The principle of the

mechanism is to prohibit (hence the taboo name) certain movements or certain parts of movement.

Already explored positions are stored in what is called the tabu list of a given size, which is an adjustable parameter of the heuristic.

In *Reactive Tabu Search for Measuring Graph Similarity* written by Sebastien Sorlin and Christine Solnon [15], the authors explain a method for measuring graph similarity using the tabu search.

For the algorithm, during the iterations, the best neighbour is chosen according to the same rules as in the greedy search. We must keep in mind that when a neighbour is chosen to be added in the mapping, its adding may lead to a not optimized similarity measure. This way we can get trapped into local maximum.

As said earlier, the idea of the tabu search is to keep a list that represents the memory of the algorithm. The list has a fixed length which is determined during the coding of the algorithm.

The choice of the size of the list is very important. If the list is too short, it will lead to a too strong intensification. The memory will remember only a few moves and it will not explore many different path.

If the list is too big, the algorithm will converge much slowly because of the too big diversification. Because of the fact that the list is big, it will remember many moves and so it will be able to explore a lot of different path. That leads to a late discovery of a solution.

6.5 Reactive Tabu search

The reactive tabu search is an improvement of the classic tabu search.

The difference with the tabu search is the length of the tabu list. In the tabu search, the length of the tabu list is set at the beginning of the algorithm and it doesn't change. As we can see in the name of the method, this method is a tabu search with a reactive feature. During the progress of the algorithm, the length of the tabu list will be adapted so that the algorithm is more efficient than the regular one.

While the algorithm is running, it will check if a mapping is explored multiple times. If it is the case, it means that there is a need in more diversification. The

tabu list is thus expanded. Inversely, if the search does not see the same mapping during a certain amount of moves, it means that the tabu list can be shortened.

Notice

During the explanation of the four previous algorithms (6.3 Greedy Search, ?? Local Search, 6.4 Tabu Search, 6.5 Reactive Tabu Search), there is mention of a similarity measure that we must increase or maximize. There is no explanation about the similarity measure formula because there are many ways to define the similarity between 2 graphs when we have a matching between the two graphs. The four previous algorithms are techniques that can be applied to every similarity measure computation formula.

7 Neighbour Matching by Mladen Nikolic

After reading about many solutions, we decided to choose the algorithm proposed by Mladen Nikolic in the paper "Measuring Similarity of Graph Nodes by Neighbour Matching" [11].

This section of the work will explain the algorithm in different ways. Firstly, the formulas will be explained in a mathematical way. After that, the algorithmic logic will be explained. We will see how the formulas must be applied to graphs in order to get a graph similarity measure. Finally, the implementation of the algorithm will be explained.

7.1 Introduction to the algorithm

The paper "Measuring Similarity of Graph Nodes by Neighbor Matching" written by Mladen Nikolic introduces a method that can be used to compare graphs. This method works with a refined concept of node similarity. To compute the matching of two nodes, we are going to look at the matching of their neighbours.

The algorithm comes with certain properties that are desirable. The first one is the fact that if a graph is compared to itself, each node should be most similar to itself. This property can be quite obvious and it is normal we want the algorithm to have this property but there are methods that are used to measure graph node similarity in which we can see that a node is more similar to an other one than to itself.

The second one is implied from the first one and says that a node compared to itself should have the maximal similarity value. Due to this property, if a graph is compared to itself and all the nodes match themselves with the maximal value, the result is that the similarity value of a graph compared to itself gives the maximal value.

The result of the algorithm is a value with a fixed range. Because of the way it is computed, the result of the algorithm is always between 0 and 1; 0 meaning that the two graphs are totally different and 1 meaning that they are similar. The consequence of this property is that the measure is meaningful by itself. We know that the result of the algorithm will be between 0 and 1 so if we have a score close to 0, we know that the graphs are quite dissimilar and if the score is close to 1, they are quite similar.

If two nodes do not have ingoing or outgoing edges, they should be considered similar. This seems normal because as said earlier, the concept of similarity in this method is that nodes nodes are similar if their neighbours are similar. So, if we have two nodes without any neighbour, it is as if they had the same neighbours

which are "NoNeighbours". So they should be considered similar.

The idea of the method can be illustrated by the following analogy. When we look at our hands, we think that they are very similar. But it is not because all the fingers of one hand are very similar to all the fingers of the other hand, it is because when we look at our left hand, we can find fingers and each of these fingers corresponds to one finger on the other hand. This can be expressed as : if we have two different nodes, namely i and j , we can say that they are similar if the nodes belonging to the neighbourhood of i can be matched to the nodes belonging to the neighbourhood of j . From this concept comes the name of the method which is "neighbour matching".

7.2 Mathematical Explanation

The purpose of this section is to analyse the formulas used by the algorithm in a mathematical way.

The first formula is the update rule of the algorithm. The algorithm is iterative and so an update rule must be developed in order to compute the next value of a node for each iteration.

$$x_{ij}^{k+1} \leftarrow \frac{s_{in}^{k+1}(i, j) + s_{out}^{k+1}(i, j)}{2}.$$

Figure 17: The update rule of the algorithm

In the above formula, x_{ij}^{k+1} is the value of the similarity between node i and node j at after $k+1$ iterations. It is defined by the average between the in-similarity (S_{in}^{k+1}) and the out-similarity (S_{out}^{k+1}) of these two nodes.

After this, we have the following formulas:

$$s_{in}^{k+1}(i, j) \leftarrow \frac{1}{m_{in}} \sum_{l=1}^{n_{in}} x_{f_{ij}^{in}(l)}^k g_{ij}^{in}(l)$$

$$m_{in} = \max(id(i), id(j))$$

$$n_{in} = \min(id(i), id(j))$$

Figure 18: The computation of the in-similarity

$$s_{out}^{k+1}(i, j) \leftarrow \frac{1}{m_{out}} \sum_{l=1}^{n_{out}} x_{f_{ij}^{out}(l)}^k g_{ij}^{out}(l)$$

$$m_{out} = \max(od(i), od(j))$$

$$n_{out} = \min(od(i), od(j))$$

Figure 19: The computation of the out-similarity

These formulas represent the computation of the in- and out-similarity of the nodes at a certain iteration. Here, $s_{in}^{k+1}(i, j)$ represents the in-similarity between node i and j after $k+1$ iteration.

The m_{in} , m_{out} , n_{in} and the n_{out} function are simple. The “m” functions take the maximal value and the “n” functions take the minimal value. The “in” functions are related to in-degree and the “out” functions are related to out-degree.

For the sum, we are going to take the neighbours of the node with the smallest amount of neighbours and match them with the neighbours of the node with the biggest amount of neighbours. We are going to take the best matching between the neighbours of the two nodes and we are going to add the similarity score of the match. This is done with the “f” and “g” functions.

The functions f_{ij}^{in} and g_{ij}^{in} are the enumeration functions of the optimal matching of in-neighbors of nodes i and j with weight function $w(a, b) = x_{ab}^k$.

The paper [9] gives a good definition of the “f” and “g” functions:

Definition Let A, B be two sets of elements.

A matching of them is a set of pairs $M = \{(i, j) | i \in A, j \in B\}$ such that no elements are matched to more than one element of the other set.

The enumeration functions $f : \{1, 2, \dots, k\} \in A$ and $g : \{1, 2, \dots, k\} \in B$ are defined such that $M = \{(f(i), g(i)) | i = 1, 2, \dots, k\}$ where $k = |M|$.

The weight function $w(a, b)$ is a function assigning weights to any pairs of elements (a, b) where $a \in A$ and $b \in B$.

The weight of a matching is the sum of the weights of the pairs in it.

A matching is optimal if the pairs in it are formed in such a way that the weight of the matching is minimized or maximized depending on the problem.

After this, we have the following formula:

$$s_{in}^1 = \frac{\min(id(i), id(j))}{\max(id(i), id(j))} \quad s_{out}^1 = \frac{\min(od(i), od(j))}{\max(od(i), od(j))}$$

Figure 20: The computation of the initial values

This formula is the computation of the initial value of the similarity values between two nodes. It is quite easily understandable. The min and max function selects the minimal and the maximal value between the two input values. the id and od function are the functions that give the in-degree and the out-degree of the nodes.

$$s(G_A, G_B) = \frac{1}{n} \sum_{l=1}^n x_{f(l)g(l)}.$$

Figure 21: The computation of the graph similarity

$$n = \min(|V_A|, |V_B|).$$

This formula is the computation of the similarity between two graphs. For the sum, we are going to take the nodes of the graph with the smallest amount of node and match them with the nodes of the graph with the biggest amount of nodes. The “x” represent the similarity value of two nodes. But this “x” means nothing alone. We must provide two argument in order to know which similarity we are talking about. This is the purpose of the “f” and “g” functions. For a certain node, the function will return the best matching possible for the nodes. In other words, the whole function is going to compute the best matching possible for every

nodes. It will sum up the similarity scores and it will divide the result by the number of nodes that were matched.

The value n is the minimal value between the amount of nodes the two graphs have.

7.3 Algorithmic Explanation

Let's dive into the calculation. Here, we are going to expose how the algorithm works. First of all, it must be known that this is an algorithm that computes the similarity scores of the nodes with an iterative procedure during which the scores are updated until the scores reach a fixed point. In the algorithm, the notion of similarity is divided into two categories, namely in-similarity and out-similarity. In-similarity is the similarity score between two nodes in terms of ingoing edges and out-similarity is the similarity score between two nodes in terms of outgoing edges.

The update rule of the algorithm is the following one:

$$x_{ij}^{k+1} \leftarrow \frac{s_{in}^{k+1}(i, j) + s_{out}^{k+1}(i, j)}{2}.$$

Figure 22: The update rule of the algorithm

We can see that the next similarity score of two nodes is computed by taking the average between the in-similarity and the out-similarity between those nodes. We compute the in-similarity and the out-similarity by using the following formulas:

$$s_{in}^{k+1}(i, j) \leftarrow \frac{1}{m_{in}} \sum_{l=1}^{n_{in}} x_{f_{ij}^{in}(l)g_{ij}^{in}(l)}^k$$

$$m_{in} = \max(id(i), id(j))$$

$$n_{in} = \min(id(i), id(j))$$

Figure 23: The computation of the in-similarity

$$s_{out}^{k+1}(i, j) \leftarrow \frac{1}{m_{out}} \sum_{l=1}^{n_{out}} x_{f_{ij}^{out}(l)}^k g_{ij}^{out}(l)$$

$$m_{out} = \max(od(i), od(j))$$

$$n_{out} = \min(od(i), od(j))$$

Figure 24: The computation of the out-similarity

For the four small formulas, we can easily understand their meaning, every "m" function takes the maximum value and every "n" function takes the minimum value. The "in" and "out" mean that we are looking at the number of in-going edges or out-going edges.

The two biggest functions are a little more complicated to understand in this form. We are going to take the neighbours of the node with the smallest amount of neighbours and try to match it with the neighbours of the node with the biggest amount of neighbours. For each of those nodes, we have their similarity score so we can look for the optimal match. When the optimal match is found, we are going to sum up the similarity scores. We will then divide this number by the following one. We will look for the compared nodes which has the most neighbours and we will take the number of neighbours it has. In this equation, 0 divided by 0 is defined to be 1. This is done so that nodes without in or out neighbours are considered similar.

As said in the paper, the method can easily be extended to graphs with labels. The update rule has to be modified in order to set that if two nodes are compared and do not share the same label (the same "type"), they are considered to be dissimilar and so their similarity score must be set to 0.

Because of the fact that it is an iterative method, an initial value must be chosen for every similarity score. As some other methods, the initial value is set to 1. At the beginning of the algorithm, all nodes are considered similar but that will change when the algorithm will begin its computation. For the initial value, a shortcut can be taken since we see that the values of the similarity scores after the first iteration will be:

$$s_{in}^1 = \frac{\min(id(i), id(j))}{\max(id(i), id(j))} \quad s_{out}^1 = \frac{\min(od(i), od(j))}{\max(od(i), od(j))}$$

Figure 25: The computation of the initial values

So when implemented, we can set the initial value according to these formulas in order to reduce the number of iterations.

The termination condition of the algorithm is quite simple. As said earlier, the method is an iterative procedure that will stop until a fixed point is found. Each time an iteration is finished, we will look at the difference between the current similarity scores and the similarity scores of the previous iteration. If the maximum difference is smaller than a certain value, namely epsilon, the iteration stops. Epsilon can be chosen at will but the smallest epsilon we choose, the more accurate we will be. The paper says that alternative termination condition can be used but it does not mention any.

Now that we have the similarity scores of every nodes, we can compute the similarity value of the graphs. This can be done using this formula:

$$s(G_A, G_B) = \frac{1}{n} \sum_{l=1}^n x_{f(l)g(l)}.$$

Figure 26: The computation of the graph similarity

$$n = \min(|V_A|, |V_B|).$$

Figure 27: Definition of n

We will take the nodes of the graph with the smallest amount of neighbours and try to match them with the nodes of the graph with the biggest amount of neighbours. We will get the optimal matching and sum up the similarity scores. After that, we will divide the result by the number of neighbours that is possessed by the compared node with the smallest amount of neighbours. This will give us a number which will be the similarity score between the two compared graphs.

We have seen that in the update formula and in the final formula, to compute the similarity between the graphs, we are talking about optimal matching between the nodes. This matching can be done using different algorithms. Here, the author speaks about the Hungarian algorithm which is an algorithm made for that. We will talk about it and its way of operating later.

7.4 Implementation Explanation

The purpose of this section is to explain how this algorithm was implemented. As said earlier, it is an iterative algorithm. Because of that, we will have several parts for this algorithm which have been implemented in different methods.

7.4.1 Initialisation

As explained earlier, we work with in and out similarity for the nodes. In order to maintain those values, we will have tree tables (`double[][]`). One will contains the in-similarity of the nodes, one will contains the out-similarity of the nodes and the last will contains the "global" similarity of the nodes. In this phase, we will simply fill every tables with the initial values following the rules defined earlier.

7.4.2 Iteration

This phase is more complicated than the previous one. We will iterate several times on the tables in order to get the similarity of the graphs.

In each iteration, we will firstly update the in-similarity of the nodes (the order for updating in-similarity and out-similarity does not matter). We will check if the nodes have the same type. If the type is different, the nodes are considered to be different and so their similarity is set to 0. If the nodes have the same type, we will apply the formula defined previously. We will mainly use the optimization function defined in the next section which will give the result of the sum in the formulas of the computation of the in-similarity and the out-similarity. After the calculation, the value of the in-similarity of the nodes will be update. The same process will be repeated for the out-similarity.

When the in-similarity and the out-similarity have been updated, we will update the similarity values by using the formula defined earlier.

While updating the similarity values, the maximal difference between two iterations will be computed. If the value is smaller than the chosen epsilon, the iteration will stop. If the value is bigger, an new iteration will begin.

7.4.3 Optimization function

As previously said, this function returns the result of the sum in the formulas of the in-similarity and the out-similarity. It uses the Hungarian Algorithm described in the next section. This function works as follow.

When comparing two nodes, we will take the node with the smallest amount of neighbours (if they have the same amount of neighbours, we take one a random).

We will make a table with these nodes and their similarity values for the neighbour of the other node. This will be represented by a double table. Here is an example of what we can have:

	Node 1	Node 2	Job 3
Node A	0	0,6	0,2
Node B	1	0,32	0,47

In order for the Hungarian Algorithm to work well, the values will be slightly modified but this will be explained later. This table will be used by the Hungarian Algorithm to compute the best matching for these nodes. It will return the best matching in term of which node should matches with which node. The function will then need to get the similarity values of the matches and add them.

7.4.4 Similarity Computation

At the end of the algorithm, we have a similarity table that shows the similarity between the nodes of both graphs. To compute the similarity, the algorithm just has to run the optimization function but this time it is not only on certain nodes, it is on every node of the graph. We will then have the sum of the similarity of the nodes. We will then just have to divide this number by n which is defined in the formula explained above.

The implementation was inspired by Dulanga Sashika [3].

7.5 The Hungarian Algorithm

7.5.1 Introduction

The Hungarian Algorithm or the Hungarian method (sometimes called the Kuhn-Munkres algorithm) is a combinatorial optimization algorithm that was published in 1955 by Harold Kuhn. The name "Hungarian algorithm" was given to the algorithm because it was largely inspired by the work of Dnes Knig and Jen Egervary, two Hungarian mathematicians.

The utilisation of the algorithm can be illustrated by a small problem. We have three workers: A, B and C. They can do the same three jobs but they ask a different amount of money in order to perform the tasks. The problem can be illustrated using a matrix.

	Job 1	Job 2	Job 3
Worker A	3	3	3
Worker B	3	2	3
Worker C	3	3	2

What is the best job assignment that we can make in order for us to spend less money ? The Hungarian algorithm can be used in this type of problem in order to solve the previous question. It will look for the best assignment in order to minimize the cost and return it.

	Job 1	Job 2	Job 3
Worker A	3	3	3
Worker B	3	2	3
Worker C	3	3	2

7.5.2 How does it work ?

For the explanation of the algorithm, we are going to say that we have n workers and n tasks that must be completed. The goal is to have an assignment that minimizes the total amount of money required to hire all workers. There are several steps to follow in order to complete the algorithm.

This is not a description of the implementation of the algorithm. It is the way it works when doing it "by the hand"

Step 1 :

The first thing to do is arranging the given information into a matrix with the workers on the left column and the jobs on the top row.

	W	X	Y	Z
A	10	19	8	15
B	10	18	7	17
C	13	16	9	14
D	12	19	8	18
E	14	17	10	19

Step 2 :

If the matrix is not a square matrix, we can add dummy rows or dummy columns. By convention, the dummy row/column contains the same number which is the biggest number that can be found in the matrix.

	W	X	Y	Z	Z'
A	10	19	8	15	19
B	10	18	7	17	19
C	13	16	9	14	19
D	12	19	8	18	19
E	14	17	10	19	19

Step 3 :

The rows must be reduced. This is done by subtracting the minimum value of each row from that row.

	W	X	Y	Z	Z'
A	2	11	0	7	11
B	3	11	0	10	12
C	4	7	0	5	10
D	4	11	0	10	11
E	4	7	0	9	9

Step 4 :

If columns remain without a zero, the reduction we have done on the rows must be applied to the columns. In this step, we also subtract the minimum value of each column from that column.

	W	X	Y	Z	Z'
A	0	4	0	2	2
B	1	4	0	5	3
C	2	0	0	0	1
D	2	4	0	5	2
E	2	0	0	4	0

Step 5 :

In this step, we must cover every zero findable in the matrix with lines. We must cover them with the minimum number of lines it is possible to cover them with. At this point, if the number of lines that is used to cover the zeros is equal to the number of rows, we can go to step 9. If not, we can go to step 6.

	W	X	Y	Z	Z'
A	0	4	0	2	2
B	1	4	0	5	3
C	2	0	0	0	1
D	2	4	0	5	2
E	2	0	0	4	0

Step 6 :

The minimum uncovered element must be added to every covered element. If two lines are standing on one element, the minimum uncovered element must be added twice.

	W	X	Y	Z	Z'
A	1	5	2	3	3
B	1	4	1	5	3
C	3	1	2	1	2
D	2	4	1	5	2
E	3	1	2	5	1

Step 7 :

Now, we must subtract the minimum element from the matrix from every element in the matrix.

	W	X	Y	Z	Z'
A	0	4	1	2	2
B	0	3	0	4	2
C	2	0	1	0	1
D	1	3	0	4	1
E	2	0	1	4	0

Step 8 :

In this step, we must do as in step 5, we must cover the zeros findable in the matrix and if the number of rows is not equal to the number of lines used to cover the zeros, we must go back to step 6.

	W	X	Y	Z	Z'
A	0	4	1	2	2
B	0	3	0	4	2
C	2	0	1	0	1
D	1	3	0	4	1
E	2	0	1	4	0

	W	X	Y	Z	Z'
A	0	3	1	1	1
B	0	2	0	3	1
C	3	0	1	0	1
D	1	2	0	3	0
E	3	0	1	4	0

Step 9 :

The best matching is here ! We must select zero elements in order to have only one selected zero in each row and in each column.

	W	X	Y	Z	Z'
A	0	3	1	1	1
B	0	2	0	3	1
C	3	0	1	0	1
D	1	2	0	3	0
E	3	0	1	4	0

Step 10 :

The final step is to transpose the matching found at step 9 into the original matrix. Now we have our matching that minimizes the total amount of money required to hire all workers.

	W	X	Y	Z
A	10	19	8	15
B	10	18	7	17
C	13	16	9	14
D	12	19	8	18
E	14	17	10	19

7.5.3 Integration in the work

As explained earlier, this algorithm is used in the matching algorithm in order to find the best matching possible between the neighbours of the nodes. There are several algorithms that can be used to do this job but in the paper, the author speaks about this one, so we chose to use it.

As said earlier, the Hungarian algorithm is used to have a matching which minimizes the total cost whereas in the matching algorithm, we want to have the higher compatibility between the nodes. So, the utilization of the Hungarian algorithm had to be slightly modified. Instead of putting numbers and getting the smallest amount, for each element, we take zero and we subtract the element to zero. This way we get negative numbers but our smallest number becomes the bigger one and the bigger number becomes the smallest one. We then run the Hungarian algorithm on the new matrix and we apply the matching that is found on the original matrix.

The algorithm used in the app was strongly inspired the algorithm of Manish Bhojasia found on the sanfoundry website. [2]

7.5.4 Implementation

The purpose of this section is to explain how the algorithm explained above was implemented in the work.

The method of the algorithm is a bit different of the above method but it gives the same result. Indeed, in the algorithm there is no "covering the zeros with lines".

The algorithm starts by reducing the initial cost matrix. This is done the same way as in the above method. The algorithm will subtract the minimum value of each row from that row. It will then also subtract the minimum value of each column from that column.

In order for the algorithm to work well, it will compute an initial feasible solution. That is a solution for which each worker gets the cheaper job for him regardless of other workers.

When it is done, the algorithm tries a greedy match. It will try to match each worker with his first best job. If the work is available, the worker is assigned to it and if the job is not available, the worker is affected to no job.

After that, the algorithm will look for an unmatched worker. If every worker are matched with a job, it means that the best matching has been found and so, the result is returned. If there is one or more unmatched worker, the algorithm will have to modify the match in order to match every worker. In order to do this, the algorithm will compute the second best match that we could have for the workers. Thanks to that, it will change the match that will have the least impact on the result in order for the unmatched worker to be matched. It will not be its best match but it is the best that can be done. This operation will be repeated until every worker are matched.

When it is done, the result is returned in a queue like this:

X1	X2	X3	...
----	----	----	-----

In this queue, the first element is the best job for worker 1 and its best job is X1. The second element is the best job for worker 2 and the job is X2. And so on.

7.6 Advantages and Drawbacks

As every other algorithm, this one also has advantages and drawbacks.

The first desired property that this algorithm has is that the score that is found at the end of the computation is bounded. We will always have a score between zero and hundred. This is interesting because we can deduce more information with bounded score. If we have a bounded score and we have only two graphs to compare, we can say if they look alike and we know that if we have a score close to zero, it means that they do not look alike and conversely, if we have a score close to hundred, they look alike a lot. With unbounded score, we only get a number. This number must be compared to other numbers. So, we must have more than two graphs in order to deduce something. And even then, if we have two scores from two comparisons, we can say that in one comparison, the graphs are more alike the graphs from the other comparison. But we can not clearly say that the scores are high or low because the scores are unbounded. Both scores are maybe high or they can both be low.

An other interesting feature of the algorithm is that in order to get the similarity score between two graphs, each node is compared with every other nodes. Therefore, at the end of the computation, we have the similarity score of the graphs but we also have the similitude score of every nodes which can be useful for certain work or studies.

One of the negative points of this algorithm is that it is not suitable for labelled graphs. In the original version of the algorithm, all nodes of the graphs are considered to be of the same type. This is clearly not the case in this work. All nodes are different because they represent data and they must not be confused with other nodes. This problem was solved by modifying the original algorithm to make it work with labelled graphs.

8 Evaluation Function

As said earlier, in this work, we try to compare graphs and get a score representing the similarities between those graphs. The algorithms that have been presented earlier are methods that could give us a similarity score between two graphs but only regarding the structure of these graphs; not regarding the content of these.

In this section, we will present a possible solution to this problem. The idea is to have an evaluation function that will compare the content of the graphs in order to give a similarity score based on the data contained in the graphs.

To do so, we must create "comparison rules" for each type of content within the graphs that we want to compare. Each function will evaluate the distance between the two contents and return a score that will be used in order to get the distance between the two graphs in terms of content.

As explained, the information that is compared comes from Facebook. We took every information we could get from there. This implies that some of the information is not evaluable. For example, we can not compare the identification number of two people and get a similarity score from it.

The characteristics that are compared are the birthday, the gender of the user, the interests of the user, his relationship status, the inspirational people of the user, the sports of the user, the languages spoken by the user, his favourite athletes, his location and his education.

The evaluation function will have to compare those characteristics. The way it works is the following one: It will count the amount of compared characteristics. Beside, the evaluation will give a certain amount of points. The maximal amount of points that can be given for the evaluation of one characteristic is 1. At the end of the comparison, the number of points will be divided by the number of compared characteristics. This way the maximal value of the evaluation is 1. And as the algorithm that compares the structure of the graphs, the score will be meaningful on its own.

We have built two different types of rules: we have the rules for the characteristics without categories and the rules for the characteristics with categories. Here is the explanation of the rules:

Birthday :

For this characteristic, we take the birthday of both people, and we check the difference of age. If the difference is under 5 years, one point is given. If the difference is between 5 and 10 years, 0.5 points is given. And if the difference is bigger, 0 points is given.

Gender :

To evaluate this characteristic, we look if the users are compatible. If there is one man and one woman, the evaluation gives 1 point. Else, it will give 0 point.

Interested in :

This rule follows the previous one. This characteristic tells if the user is interested in male or in female. Using this and the previous one, we can give 1 point if they are compatible and 0 if they are not.

Relationship status :

For this characteristic, we just have to look if both users are single. If it is the case, then the function will give 1 point. If one or both are in a relationship, 0 point is given.

Languages :

Here, we are only going to look if the users speak the same language. If they do speak the same language, 1 point is given and if they do not speak the same language, 0 point is given.

Athletes :

The rule for this characteristic is pretty much the same as for the two following characteristics. These characteristics have a type and a value as the other but they also have a category. For example, the first user can like a certain skier and the other user, another one. This is not a perfect match between both but it is more or less a half-match. To take this kind of matches into account, if we have a perfect match, 1 point is given. If there is no perfect match, we are going to count how many characteristics with the same category were seen. We are going to divide this number by the number of characteristics with the same type. And because of the fact that these are no real matches, we are going to divide the result by 2. This amount of points will be added.

Inspirational People :

This works like Athletes. If two users like two different people but those people are both painters. There will be a half-match.

Location :

Location works like athletes, this is done this way so if we have two users coming from different villages but both in the countryside, they will have a half-match on this characteristic.

Education :

As location, education follows the same principle. This is done so if we have two users coming from different universities, they will have a half-match.

Sports :

This also works like the above one. If a user likes ski and the other one likes snowboard, there will be a half-match because both like a winter sport.

These rules were invented so that the application can work. But one must not forget that this algorithm can be applied to another field through the act of creating the rules for the new information that we want to compare. The algorithm can also be adapted if Facebook changes its information. It will only require the rules to be adapted.

9 The Application

The purpose of this section is to show the application and explain what is done behind the screen.

9.1 Presentation

This screen is the first to be shown to the user. This is a loading screen waiting for the application to make the connection with Facebook.

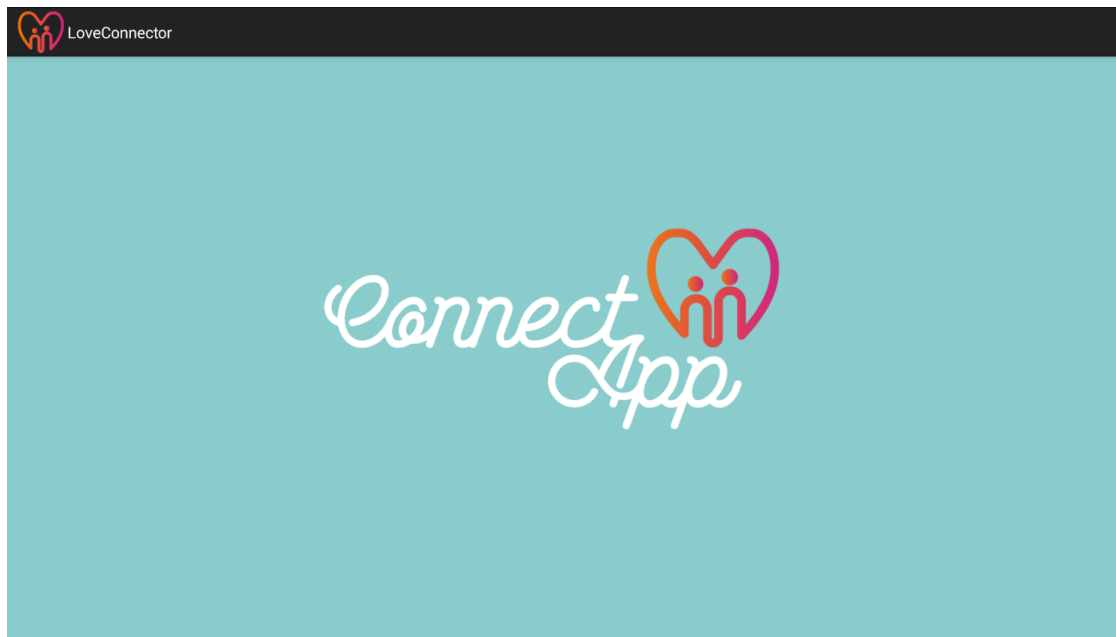


Figure 28: The Splash Screen

The next screen is the home page (29). This is where you can choose what you want to do if you are already connected to Facebook. If you are not connected to Facebook, you will automatically get on a page to log you in.

- You can click on "Compare my Friends" to dive into the heart of the application.
- You can click on "Get My Information" in order to see which of your data is available on Facebook.
- You can click on "About" to have some information about the application.
- You can log you out.

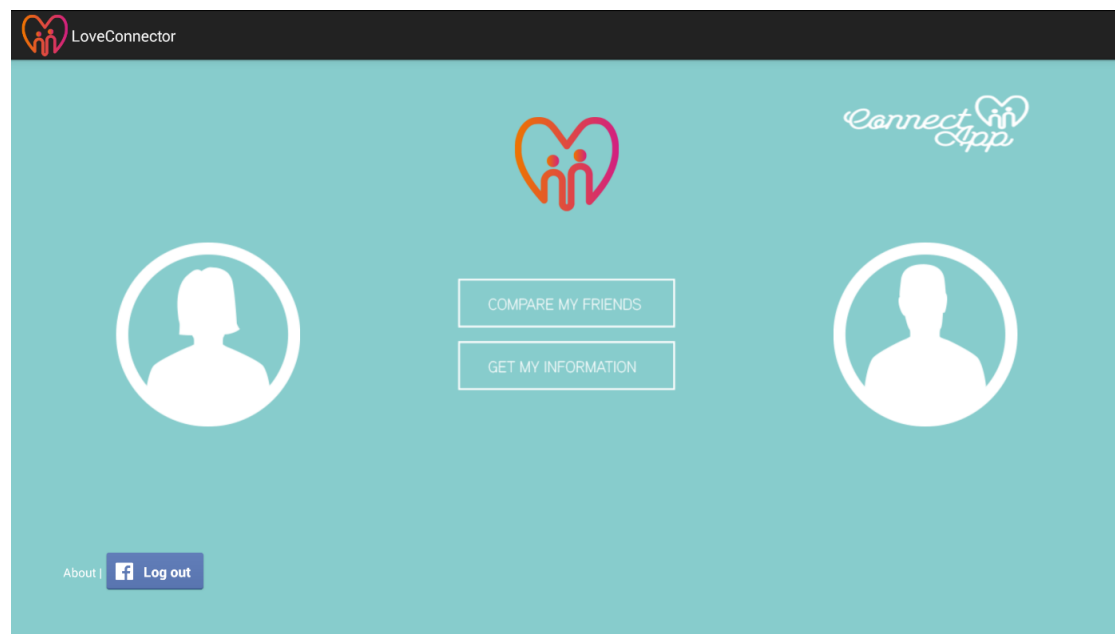


Figure 29: The Home Page

When the user clicks on "Get my information", he/she gets on the following page (30). This is a summary of all his information available on Facebook. This is a good way to check which of his/her data is available for everyone.

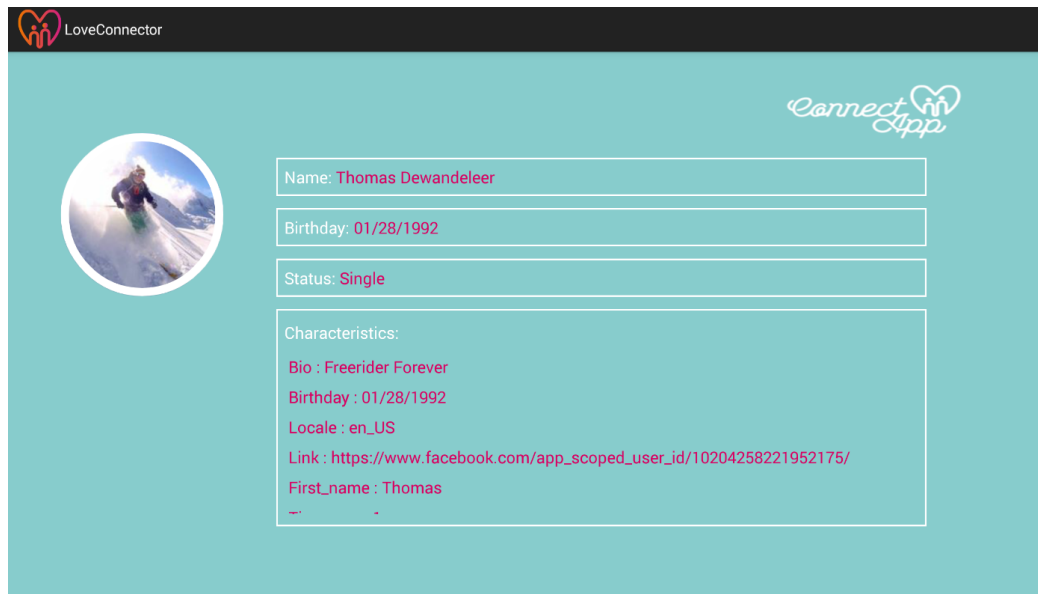


Figure 30: The User Information Page

If the user clicks on "Compare My Friends" on the home page, he/she gets on the following page (31). Here he/she can choose which friend he wants to be compared with.

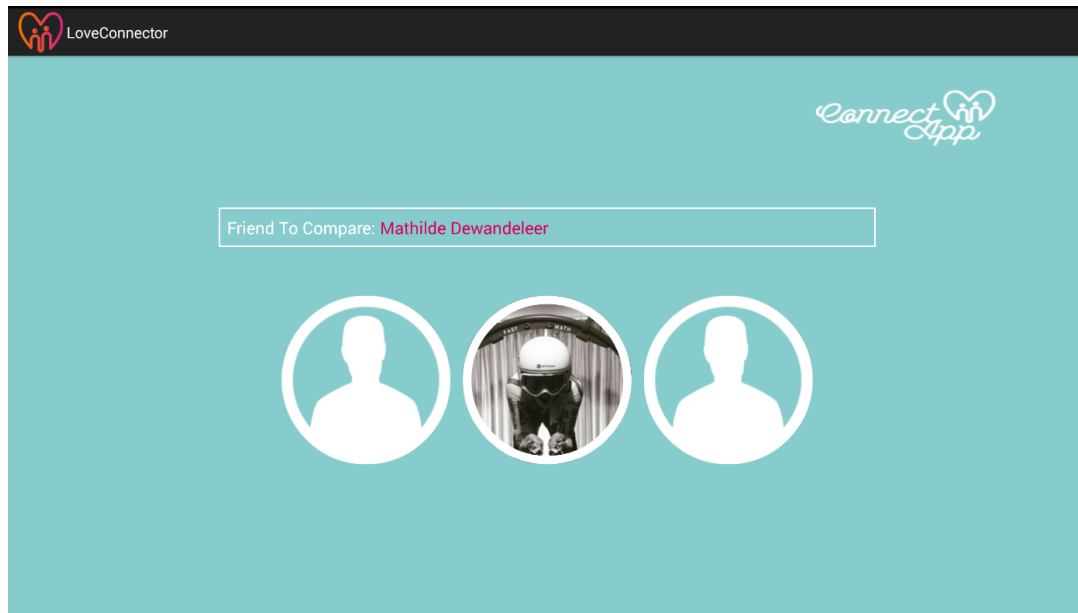
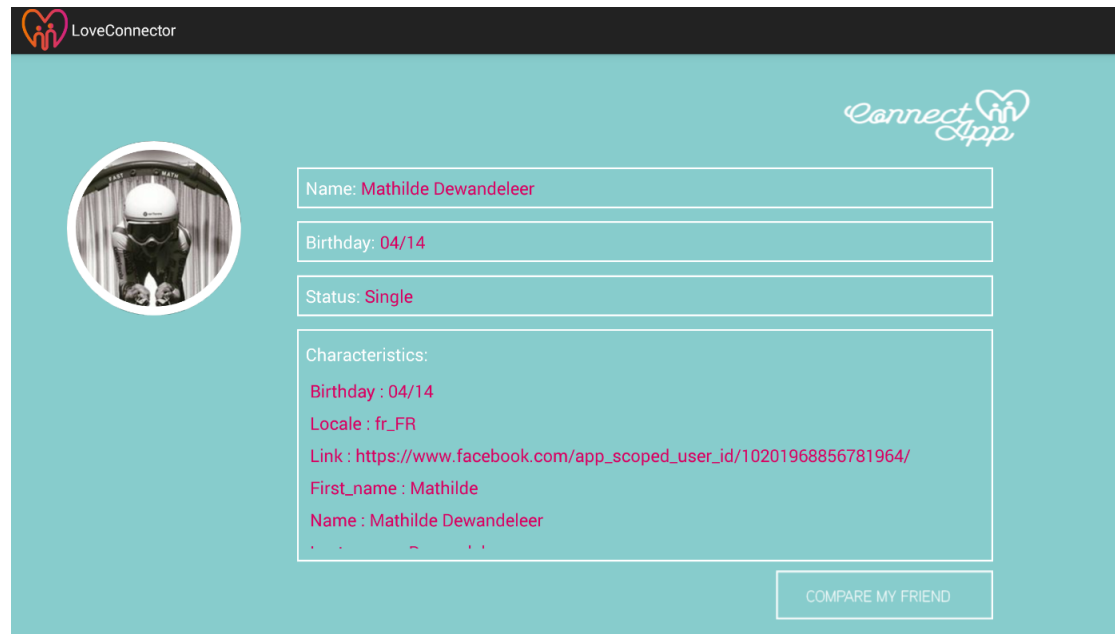


Figure 31: The Choosing Page

This is the page of a friend. As on the user page, we have a summary of all the friend's information available on Facebook. This time, the "Compare my friend" button has appeared.



The screenshot shows a web interface for 'Connect App' with a teal background. At the top left is the 'LoveConnector' logo. On the right is the 'Connect App' logo. On the left side, there is a circular profile picture of a person in a white helmet. To the right of the profile picture, there are several input fields containing user information: 'Name: Mathilde Dewandeleer', 'Birthday: 04/14', and 'Status: Single'. Below these is a larger box labeled 'Characteristics:' containing 'Birthday : 04/14', 'Locale : fr_FR', 'Link : https://www.facebook.com/app_scoped_user_id/10201968856781964/', 'First_name : Mathilde', and 'Name : Mathilde Dewandeleer'. At the bottom right, there is a button labeled 'COMPARE MY FRIEND'.

Figure 32: The User Information Page

If we click on "Compare this friend", we get here. There is only one button that will do everything. It will query for the information of the user and the information of the friend. It will use the graph matching algorithm in order to compare the structure of the graphs and show the result near the "Primary Test" label. It will compare the information via the evaluation function and show the result near the "Secondary Test" label.

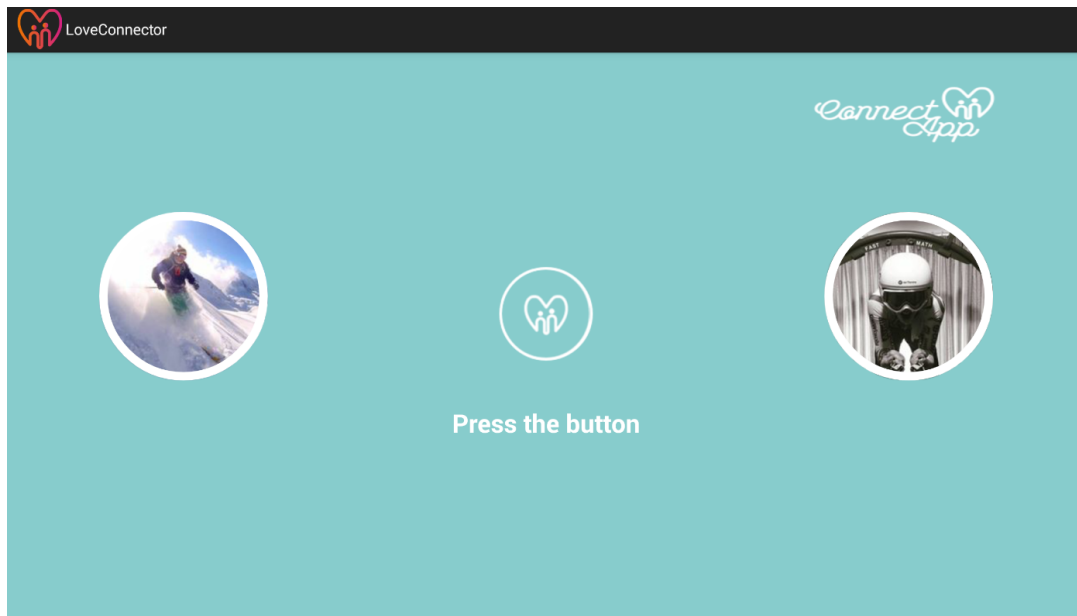


Figure 33: The Comparison Page

Here is an example of the comparison page with the results of the algorithms:

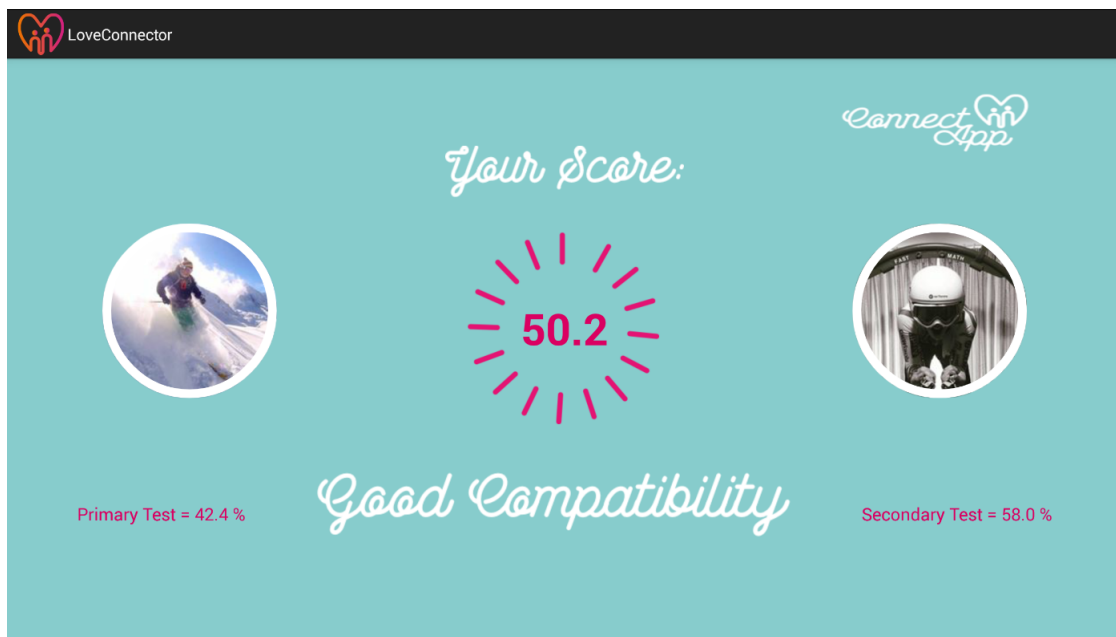


Figure 34: The result of the Comparison Page

We can see a comment between both results summarizing the scores. There are 5 possible level (from the lowest to the highest): "Not compatible", "A bit compatible", "Moderately compatible", "Good compatibility" and "You should date her/him".

9.2 How does the Application work ?

The purpose of this section is to explain in a simple way how each of the application page does work. It must be known that the application will handle the Facebook connection automatically. It will maintain the connection open when needed and it will close it when the user wants it.

9.2.1 Main Menu

This part of the app is simple.

If you are not connected to Facebook, a button will be there to log you in.

If your are connected to Facebook, you will get on a page like in figure 29. Here, you can choose what you want to do. Since this page is just a menu which will guide the user through the different features of the app, it will not be deeply explained.

9.2.2 Profile Information

This page shows information about a profile. This page is shown in figure 30. The way it works is quite simple and work in two steps.

The first step is the request to get the information about the profile. Here we will get the textual information about the profile. This is done via the Request object of the Facebook API. When a connection to Facebook is open, the user can request his information or the information of one of his friends by using the Facebook id of the user he/she wants to get the information. The application will receive a JSON object. This object will be parsed by the application in order to extract each information. The parsing of the information will put them into a List. The application just have to run through the List in order to add the information into the ListView.

The second step is the request to get the profile picture of the profile. This is done by calling `https://graph.facebook.com/id/picture?type=large` where id is the id of the profile. This is a direct link to the picture. It will be handled via the BitmapFactory class of Android. The application just has to update the ImageView with the image retrieved via the URL.

9.2.3 Choosing Page

This page allows the user to choose which of his Facebook friend will be compared. This page is shown in figure 31.

The application will request the list of the user's friend. This is done via the Request object of the Facebook API. The application just have to use it with the argument: `"/me/friends"`. Here we will get the name and the id of every friend of the user that he is able to compare.

In order to have a nice list, a custom list adapter has been coded for the application. It will create a nice looking list and request the profile picture of every friend. The request for the profile picture is done the same way it is done in the profile information page.

9.2.4 Comparison Page

This page is the heart of the app. This is where the magic happens. This page is shown in figure 33.

It will firstly request the profile picture in order to have a nice page.

After that, the user can click on the "Make the comparison" button in order to run the algorithm.

In order to do the comparison, the application must get the information of the two users it will compare. The app will use the Request object to get the information and it will parse it using the same parsing method as in the Profile pages.

The evaluation function will run in first place. Since the function does not have to rearrange the data, it can be launched right after getting it.

When the evaluation function is done, the graph matching algorithm will be launched. For this function, it will be needed to rearrange the data. As it is said, it is a graph matching algorithm, and so, it compares graphs. The application will convert the data into graphs. This is done in the following way:

The application will make a node representing the user. It will then add nodes representing the characteristics of this user and it will link them to the user.

After creating the user node, it will browse the characteristics list. There are two different types of characteristics: The ones with a type and a value (Birthday, Gender,...) and the ones with a type, a value and a category (Home-town, School,...).

When a characteristic without category is seen, it will create a node with the correct type and value and then link it to the user node. When a characteristic with category is seen, it will create a node with the correct category and a node with the type and value and link the first one to the second one and the second one to the user node.

Here is an example of a graph with one user and a few characteristics:

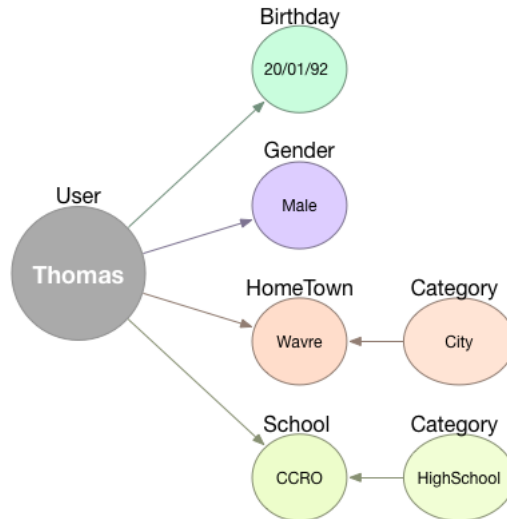


Figure 35: Graph with one user

Here is an example of a graph with two users and a few characteristics:

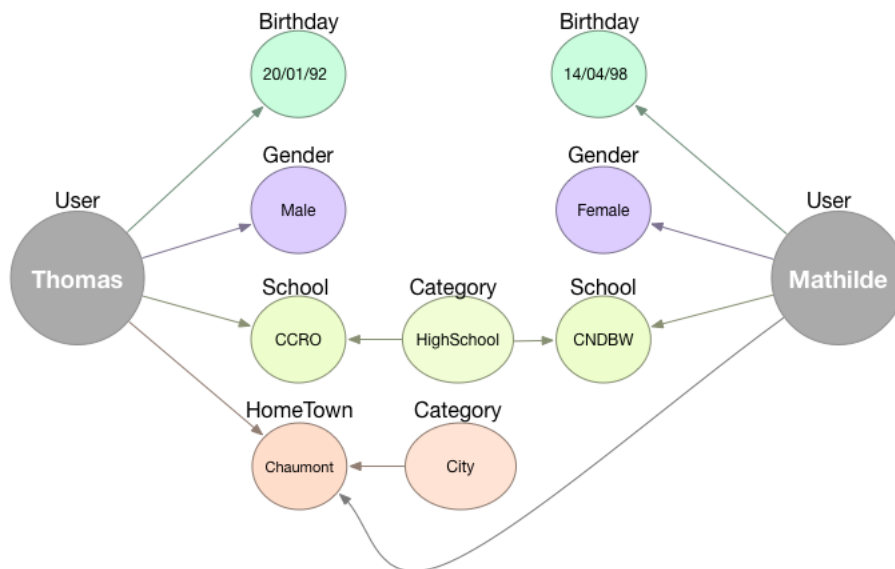


Figure 36: Graph with two users

After constructing the graphs, the app will run the graph matching algorithm on the graphs.

When both algorithm are done, the app will show the result of those and a small sentence summarizing the result.

The graph matching algorithms and the evaluation function will not be explained since they have been explained in previous sections.

9.3 Video Presentation

A video presenting the application has been made and is available on <https://www.youtube.com/watch?v=-C17E8oDGjo>

10 Technologies Used

The purpose of this section is to describe the technologies that have been involved in this project.

The algorithm presented in the section 7 was coded in Java using Eclipse (<https://eclipse.org/downloads/>).



In order for the integration into an application not to be too complicated, I have decided to make an Android application. Indeed, the logic part of an Android application can be coded in Java.



The graphical part of the application is coded in XML. All the design element of the application were made via Adobe Illustrator (http://www.adobe.com/be_fr/products/illustrator.html).



The application is compatible with Android 4.0 and every above version.

11 Facebook

Although Facebook is not part of the research, we wanted to explain some details about its use in the work.

11.1 Facebook API

In order to collect the data for the algorithm, we used the Facebook graph API. This gives us the possibility to make queries to Facebook server in a quick and clean way. We can make requests and get responses in JSON asynchronously. This can be used to query data, post new stories, manage ads, upload photos and a variety of other tasks that an application might need to do.

11.2 What is used

For this work, every data available via Facebook API were gathered and used. This was done in order to test the algorithm on complete graphs. For every information that we want to query via the API, we must get the authorisation of the user. Here are a few permission that are asked when you log yourself into the application : "user_likes", "user_location", "user_interests".

The following information is gathered from Facebook:

- The Id number of the user
- The biography of the user
- The first and the last name of the user
- The birthday of the user
- The gender of the user and the gender of his (future) partner
- The hometown and the current location of the user
- The list of schools of the user

- The favorite athlete of the user
- The list of inspirational people of the user
- The languages spoken by the user
- The list of sports of the user
- The list of works of the user

For the six last elements, more information is gathered than the name of the item. For example, if one inspirational people is an athlete, the category "athlete" will be gathered and it will be added into the analysed graphs.

12 Conclusion

There are quite many researches about the graph theory, the techniques that can be used for graph matching,... The purpose of this work was to explore only a small part of the graph's world. Indeed, graphs can be used in a lot of different fields and can be applied in a lot of problems.

The solution proposed in this work to solve the initial problem is certainly not the only one. But I think this can be a good solution.

This work taught me a lot ! I had never made a work of this size. We had an initial problem and from there had to find out possible solutions. I had to do a lot of researches in order to find possible solutions which were then evaluated to determine which could fit our needs at best. Everything needed to be adapted to the initial problem.

Despite the difficulties encountered in the realization of this work, I finally got into it and I'm very happy of it because it taught me many things like for example, completing a big work "alone" from conception to final realization. I am sure that this will help me in the active life in which I look forward to get into because I have a lot of exciting projects in mind.

13 Further Work

Further work can be done on this. For example, it would be nice to make a system that could easily accept inputs so that the application will not be entirely dedicated to Facebook but could also receive information from any source.

An other thing that could be improved is the evaluation function. It would be nice to find a convenient way to modify the evaluation rules. Currently, we have to modify an entire method and this is not really convenient.

An other aspect of the application that we could look into is the performance of the application. Currently, it is quite slow and when we ask it to run the comparison, it takes time. We could look for process optimization in order to have this one run faster.

14 Appendix

The code of the application is available at <https://bitbucket.org/tdewandeleer/loveconnector>. All the sketches and graphs were made thanks to OmniGraffle for Mac.

References

- [1] How to use the hungarian algorithm. <http://www.wikihow.com/Use-the-Hungarian-Algorithm>.
- [2] Java program to implement the hungarian algorithm for bipartite matching. <http://www.sanfoundry.com/java-program-implement-hungarian-algorithm-bipartite-matching/>. Author: Manish Bhojasia.
- [3] Measuring similarity of graph nodes by neighbor matching. <https://wadsashika.wordpress.com/2014/09/19/measuring-graph-similarity-using-neighbor-matching/>. Author: Dulanga Sashika.
- [4] Using the graph api, android. <https://developers.facebook.com/docs/android/graph/>. Author: Facebook.
- [5] V. K. Balakrishnan. Graph theory. *McGraw-Hill*, 1997.
- [6] Claude Berge. Theorie des graphes et ses applications. *Dunod, Paris: Collection Universitaire de Mathematiques*, 1958.
- [7] P.-A. Champin and C. Solnon. Measuring the similarity of labeled graphs. *Lecture Notes in Artificial Intelligence 2689-Springer-Verlag*, 2003.
- [8] Pierre-Antoine Champin and Christine Solnon. Measuring the similarity of labeled graphs. *Computer Science*, 2003.
- [9] Patrick P.F. Chan and Christian Collberg. A method to evaluate cfg comparison algorithms. *Quality Software (QSIC)*, 2014.
- [10] F. Glover. Tabu search - part i. *Journal on Computing*, 1989.
- [11] Mladen Nikolic. Measuring similarity of graph nodes by neighbor matching. *Intelligent Data Analysis*, 2012.

- [12] S. Fischer R. Ambauen and Horst Bunke. Graph edit distance with node splitting and merging, and its application to diatom identification. *Computer Science*, 2003.
- [13] S. Gelatt S. Kirkpatrick and M. Vecchi. Optimisation by simulated annealing. *Science*, 1983.
- [14] G. Kendall S. Petrovic and Y. Yang. A tabu search approach for graph-structured case retrieval. *STAIRS*, 2002.
- [15] Sebastien Sorlin and Christine Solnon. Reactive tabu search for measuring graph similarity. *Graph-Based Representations in Pattern Recognition*, 2005.
- [16] Alberto Del Bimbo Stefano Berretti and Pietro Pala. Graph edit distance for active graph matching in content based retrieval applications. *The Open Artificial Intelligence Journal*, 2007.
- [17] J. J. Sylvester. Chemistry and algebra. *Nature*, 1878.
- [18] Jean-Yves Ramel Zeina Abu-Aisheh, Romain Raveaux and Patrick Martineau. An exact graph edit distance algorithm for solving pattern recognition problems.