LINKING ENTITY PROFILES

by

RAMESH VENKATARAMAN

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTERS OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2014

To my parents and brother who made it possible for me.

Acknowledgements

It is with great pleasure that I would like to thank those people without whom it would have been impossible to complete my thesis. First and foremost, I would like to thank my supervising professor Dr. Chengkai Li for constantly motivating and encouraging me and for his invaluable advice during the course of my research work. I would also like to thank Dr. Li for his concern and his appreciation at several stages of my thesis work. I would also like to take this opportunity to thank Dr. Li for being available always and steering me in the right direction.

I wish to thank my committee members Dr. Ramez Elmasri under whom I took 2 database courses (DB1 and DB2) and Professor David Levine under whom I took the cloud computing course. I would really like to thank these professors for the courses they taught which helped me learn a lot of new information and also for their acceptance to be in my committee.

I would also like to thank my current job as Systems Administrator in CSE department which helped me in many ways. I would really like to thank my manager Bito Irie for giving me the opportunity to be part of the CSE services team and his valuable advice during my tough phases.

Finally, I would like to thank my friends Praveen and Nandish who helped me out in the difficult situations of my thesis work. Their constant motivation and guidance really helped me in shaping my career as a Master's student. I would also like to thank my fellow IDIR lab mates Gensheng, Nafi, Abol and Somu who all helped me when I was stuck with some problem. Thanks to the members of the Stack Exchange community sites, especially unix.stackexchange.com for answering many questions in connection with my thesis work.

July 14, 2014

Abstract

LINKING ENTITY PROFILES

Ramesh Venkataraman, MS

The University of Texas at Arlington, 2014

Supervising Professor: Chengkai Li

Entity linking allows one to have collections of data from multiple sources as a global dataset and then query those data. Entity linking allows us to do knowledge discovery on this global dataset which might result in the discovery of some interesting facts and information. Microsoft Academic Search (MAS) is a free public search engine for academic papers and contains the bibliographic information for papers published in journals, conference proceedings and respective citations. As of February 2014, it has indexed over 40 million publications and 20 million authors. LinkedIn is a social networking service used for professional networking. LinkedIn has an estimated 259 million users worldwide. Linking the author from MAS to the person from LinkedIn produces a bigger dataset. The resulting dataset enables us to find more interesting measures about an author such as the author's educational institutions, previous work experiences and social groups. We are effectively collecting missing pieces of information about an author from LinkedIn as part of forming an extensive dataset. In this process, we are resolving the ambiguity of multiple persons with the same name as the author and classifying them. Our experimental results indicate that we can attain a higher precision of 98% if we have a higher threshold of 2.8.

Table of Contents

List of Illustrations

List of Tables

Chapter 1

Introduction

A single entity is modeled across multiple data sources and it is valuable to link entities together. After entity linking, we have more comprehensive information available about that entity. The entity linking is known as “Linked Data” which describes a method of publishing structured data so that it can be interlinked and become more useful. It enables data from different sources to be connected and queried. The adoption of the Linked Data best practices has led to the extension of the Web with a global data space connecting data from diverse domains such as people, books and scientific publications [1]. It is important that we link data from multiple sources to derive some meaningful information from this dataset. Discovering the correspondence of an entity across different data sources is a crucial prerequisite for many interesting inter-network applications such as link recommendation and community analysis using information from multiple data sources [2].

However, the process of entity linking has several challenges. The same entity is represented in different ways across multiple data sources. For instance, in this work, we link an author profile available from Microsoft Academic Search (MAS) with a person’s profile from LinkedIn. The first challenge we face is that the author name in MAS is represented using first name, middle name and last name, but a person name in LinkedIn is represented using just the first name and last name. The second challenge is that the author skillset in MAS is a subset of a person’s skillset available from LinkedIn. The third challenge we face is that the author profiles from MAS contains duplicate data. The fourth challenge is that we will not always have a matching profile in LinkedIn for an author profile from MAS and vice-versa. We address the problem in a more generalized manner for any two data sources.

We have two sets where set M represents the distinct profiles from entity network A and the set L represents the distinct profiles from entity network B. Our goal is to find a one to one mapping between these 2 sets. To be precise, our expected outcome is a partial injective function from M to L (written as $f: M \nrightarrow L$). This is partial because we do not force the function to map every element of M to an element of L. This is injective because we look for a one to one function that preserves distinctness. This is illustrated in Figure 1.

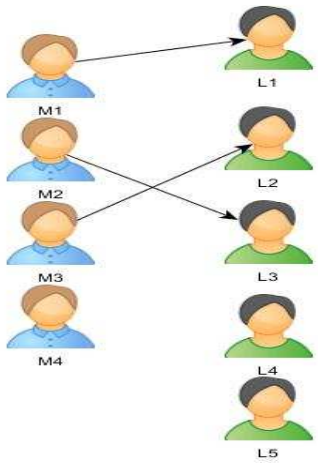


Figure 1 – Problem Definition

Represent the entities from MAS and LinkedIn using a complete bi-partite graph. Then, assign an edge weight between the nodes of MAS and LinkedIn and solve the problem using maximum weight perfect matching in a bipartite graph [3]. The name, organization and skills are the common parameters between MAS and LinkedIn. For an author from MAS, we search for various name combinations in LinkedIn. We are able to search the name combinations in LinkedIn because it always follows the same pattern for any person profile which is *www.linkedin.com/pub/dir/firstname/lastname*. We use web scraping to extract the HTML source of the person's LinkedIn profile. Then, extract the common parameters such as the name, organization and skills from the HTML source

page of these LinkedIn profiles and compare them with the name, organization and skills available from MAS. Then calculate the edge weight between an author profile in MAS and LinkedIn profiles based on the common parameters. After calculating the edge weight for all the authors, use a threshold on the edge weight to remove weaker edges from the bipartite graph. After removing the weaker edges, use the Hungarian Algorithm [4] to find the maximum weight perfect matching in the bipartite graph.

The Hungarian Algorithm works only on smaller datasets. We have implemented our own algorithm which is a modified version of the greedy approach that executes equally well on larger datasets.

On analysis of our entity network data, we observe that there are duplicate records in the data. The first problem can be viewed as duplicate records detection in the entity network data. There is a set of public profiles from the entity network which contains duplicate records for the same profile. Eliminate all such duplicate records in the entity network so that the list contains only distinct profiles.

In our MAS data, we have duplicate ids for the same author. We eliminate all the duplicate records. For each author, MAS has the details about author name, author's collaborators and the author's skills. We detect the duplicate entries for an author based on these 3 parameters. We assume that a person's profile in LinkedIn is always distinct and no person can have 2 profiles for himself in LinkedIn.

Human evaluators helped us to evaluate our work. Based on their evaluation, our experimental results for duplicate authors' detection have a higher recall value of 1. The evaluation for finding pairs between MAS and LinkedIn shows that a higher threshold value results in a better precision rate. We have discussed these algorithms and techniques in the following chapters and we have also presented our experimental results towards the end.

Chapter 2

Problem Modeling

As mentioned in Chapter 1, we have 2 entity networks. We use the author data from MAS as one data source and profile information from LinkedIn as another data source. We can view the problem as a complete bipartite graph where the vertices can be partitioned into two subsets $V1$ and $V2$. $V1$ represents the set of distinct MAS authors and $V2$ represents the set of distinct public profiles of persons in LinkedIn. Each vertex of the first set ($V1$) is joined to each vertex in the second set ($V2$) by exactly one edge. That is, it is a bipartite graph $(V1, V2, E)$ such that for every two vertices $v1 \in V1$ and $v2 \in V2$, $v1v2$ is an edge in E. Thus we can define the complete bipartite graph as $K_{m,n}$ where m is the partition size of $V1$ and n is the partition size of $V2$.

In the above complete bipartite graph we find a match G' such that $G' = (M, L, E')$. It is to be noted that there may be vertices $v_k \in M$ that does not form any edge with $v_j \in L$ and vice versa. We define the graph G' in such a way that the edges $e' = (vi, vj)$ satisfies the below criterion.

$$\forall\, e' \in E', v_i \in M, v_j \in L$$

$$\nexists\, e \in E' \; such\; that\; e = (v_i, v_k) or\; e = (v_p, v_j)$$

As per our definition of the match, for our complete bipartite graph K, we will get multiple matches. We define the total matches as the number of matches possible in our complete bipartite graph. We have illustrated 2 possible matches in Figure 2. We can formulate the total matches using mathematical equation. Assuming we have m nodes in MAS and n nodes in LinkedIn, the total matches possible for our complete bipartite graph can be represented by a mathematical equation as,

Total Matches = $\sum_{i=1}^{\min(m,n)} \binom{m}{i}\binom{n}{i} i!$

As per the definition of total matches even for a smaller case when m=2 and n=3 the total matches is 9.

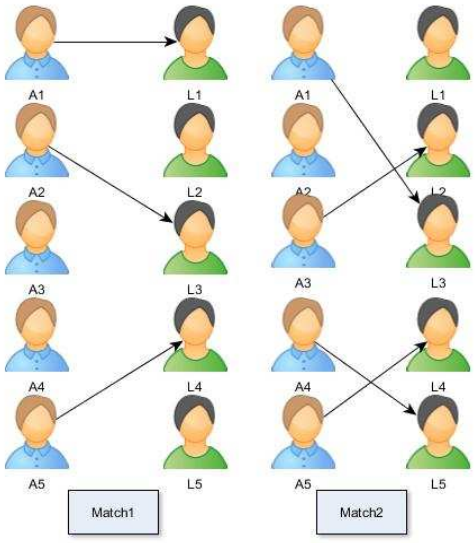


Figure 2 – Possible Matches

From all available matches, we find a single match which has the maximum weight. In the next section, we will discuss how we calculate the edge weight. We will look into the mathematical formula of edge weight followed by an example. The pseudo code on how we calculated the edge weight can be found in Appendix-A, Appendix-B and Appendix-C.

2.1 Mathematical Representation Of Edge Weight Calculation

The weight definition discussed here is specific to the datasets we used in our thesis work. Before defining the edge weight, let us look at the information that is common between the MAS and LinkedIn. Table 1 shows some of information available from MAS.

Table 1 – MAS Information for an author common with LinkedIn data

Name	Organization	Skills
Nicholas Chia	University of Illinois Urbana Champaign	Microbiology, Oncology
Kaiwen Xia	University of Toronto	Mechanical Engineering, Geophysics

In Table 1, Name column corresponds to M_{name}. Organization column corresponds to M_{org}. Each skill in the Skills column of Table 1 is represented as Y_{M_i}. Table 2 below shows the information available from LinkedIn for persons. In Table 2, the Name column corresponds to L_{name}. The Organizations column corresponds to L_{org}. Each skill in the Skills column of Table 2 is represented as X_{L_i}. We mention the variables we use to represent each column of Table 1 and Table 2 here is because we use these variables later when we define the mathematical equations.

Table 2 – LinkedIn information of a person common with MAS data

Name	Organizations	Skills
Nicholas Chia	Mayo clinic, Institute for System Biology, University of Illinois at Urbana Champaign, The Ohio state University, Georgetown University	Microbiome, Microbiology, Computational Biology and 12 more skills.
Kaiwen Xia	University of Toronto, California Institute of Technology, University of Science and Technology China	Rock Mechanics, Geophysics, Simulation and 15 more skills.

Both Table 1 and Table 2 above list the information that is available in common from MAS and LinkedIn. As we see, we have names, organizations and skills as the common parameters between the 2 datasets. Therefore, the edge weight is defined as a function of the sum of *namescore*, *orgscore* and *skillscore*.

$$Weight = namescore + orgscore + skillscore \quad (1)$$

Before defining the *namescore*, let us see the various combinations that we use for *namescore* calculation. MAS follow the pattern as first name, middle name and last name to represent an author's name. LinkedIn just uses first name, last name to represent a person. Hence we try various name combinations. The various name combinations that we try are as follows.

If an author has only first name and last name in MAS, then we use 2 name variations to search in LinkedIn. The URL patterns that we search in LinkedIn are,

- ✓ *www.linkedin.com/pub/dir/firstname/lastname*
- ✓ *www.linkedin.com/pub/dir/initials of first name./lastname*

For example, if the author name in MAS is *Bart Selman*, the URL patterns that we search in LinkedIn are,

- ✓ *www.linkedin.com/pub/dir/bart/selman*
- ✓ *www.linkedin.com/pub/dir/b./selman*

If an author has first name, middle name and last name in MAS, then we use 5 name variations to search in LinkedIn. The URL patterns that we search in LinkedIn are,

- ✓ *www.linkedin.com/pub/dir/firstname/lastname*
- ✓ *www.linkedin.com/pub/dir/firstname.middlename/lastname*
- ✓ *www.linkedin.com/pub/dir/firstname/middlename.lastname*
- ✓ *www.linkedin.com/pub/dir/initials of firstname.middlename/lastname*
- ✓ *www.linkedin.com/pub/dir/initials of firstname./middlename.lastname*

For example, if the author name in MAS is *Reid.G.Simmons*, the URL patterns that we search in LinkedIn are,

- ✓ *www.linkedin.com/pub/dir/Reid/Simmons*
- ✓ *www.linkedin.com/pub/dir/Reid G./Simmons*
- ✓ *www.linkedin.com/pub/dir/Reid/G.Simmons*

- ✓ *www.linkedin.com/pub/dir/R.G./Simmons*
- ✓ *www.linkedin.com/pub/dir/R./G./Simmons*

This is how we calculate the *namescore*. The namescore is defined as,

$$namescore = 1 - \left\{ lev_{M_{name}, L_{name}(|M_{name}|, |L_{name}|)} \div \left(len\ (M_{name}) + len\ (L_{name}) \right) \right\} \quad (2)$$

We use Levenshtein distance [4] for calculating the *namescore*. Levenshtein distance is an edit distance algorithm. Edit distance is a way of quantifying how dissimilar two strings are to one another by counting the minimum number of operations required to transform one string into the other. The reason for using Levenshtein distance is because it is the most common variant generally used for calculating the edit distances. Informally, Levenshtein distance between two words is the minimum number of single character edits (i.e. insertions, deletions or substitutions) required to change one word to the other. Mathematically, the Levenshtein distance between two strings a, b is given by $lev_{a,b}(|a|, |b|)$,

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) & if\ \min(i,j) = 0 \\ min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & otherwise. \end{cases} \quad (3)$$

Where $1_{(a_i \neq b_j)}$ is the indicator function equal to 0 when $a_i = b_j$ and 1 otherwise.

For example, the author name in MAS is *NicholasChia* and a person name for a public profile from LinkedIn is *NicholasCSChia*. The Levenshtein distance between these 2 names is calculated as 2. We have provided the calculation of Levenshtein distance in Appendix D. However, we normalize the Levenshtein distance. The final value calculated for namescore is 0.92 in this example.

Now, let us see how we define *orgscore*. We define *score* between any 2 pair of organization as below.

$$score\ (M_{org}, L_{org}) = lev_{M_{org}, L_{org}(|M_{org}|, |L_{org}|)} \div \left(len\left(M_{org}\right) + len\left(L_{org}\right)\right) \quad (4)$$

$$orgscore = 1 - max\left\{\forall_{a \in M_{org}, b \in L_{org}} score\ (a, b)\right\} \quad (5)$$

We have one organization in MAS but for a person in LinkedIn there are multiple organizations (current organization, previous organization, educational institution) for a person. We calculate the *orgscore* in the same way as we calculate the *namescore* except that we calculate the maximum *orgscore*.

The *skillscore* is calculated as below.

$$skillscore = 1 - \sum_{i=1}^{n} \left[X_{L_i} \cap Y_{M_i}\right] \div Y_{M_i} \quad (6)$$

Where,

$$Y_{M_i} = \begin{cases} 1\ if\ author\ possesses\ a\ skill\ in\ MAS \\ 0\ otherwise \end{cases}$$

$$X_{L_i} = \begin{cases} 1\ if\ author\ possesses\ a\ skill\ in\ linkedin \\ 0\ otherwise \end{cases}$$

The reason we use the skills from MAS in denominator is because the skills set in MAS is a subset of the skills available in LinkedIn. For example, as we saw in the table, the author *Nicholas Chia* in MAS and the person named *Nicholas Chia* from LinkedIn both have Microbiology as their common skill. In this case we calculate the skillscore as 0.5 as we have 2 skills in MAS.

2.2 Example On Calculating Edge Weight

For each author from MAS, we find different name combinations for that author so that we can search those name combinations in LinkedIn. After obtaining the name variations for all the authors from MAS, we check if these links are valid in LinkedIn. By valid, we mean if the links actually exist. The valid LinkedIn URLs may correspond to either a single public profile page of a person or multiple public profile pages of several

persons. The maximum number of URLs for a multiple persons public profile pages is restricted to 25. This is illustrated in the Figure 3. The figure shows the name combinations we try for an author and the valid LinkedIn URLs for those names.

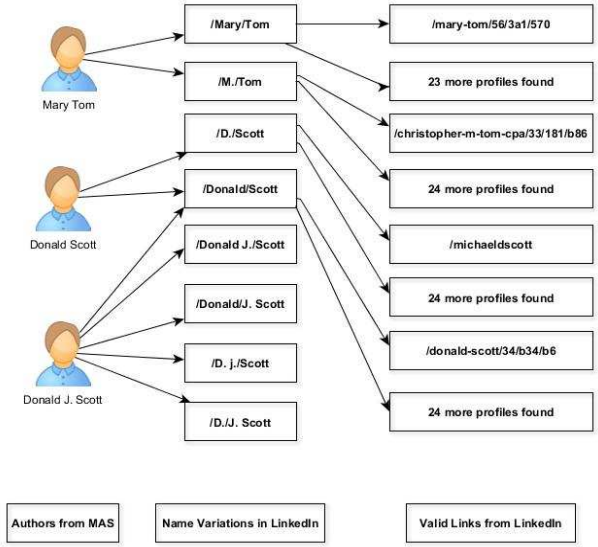


Figure 3 – Name Variations and Valid URLs

After finding the valid LinkedIn profiles, we need to define the weight over the edge that maps the nodes between these 2 sets. We have illustrated it in Figure 4.

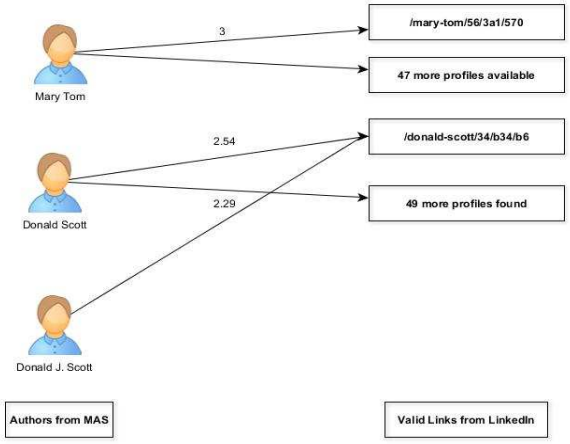


Figure 4 – Assigning Edge Weights

However, as we can see from the Figure 4 there will be some edges which are very weak. To avoid such weak edges, we apply the threshold and select only the edges that are greater than the threshold value. In this particular example, we use the threshold value as 2. This can be illustrated as in Figure 5.

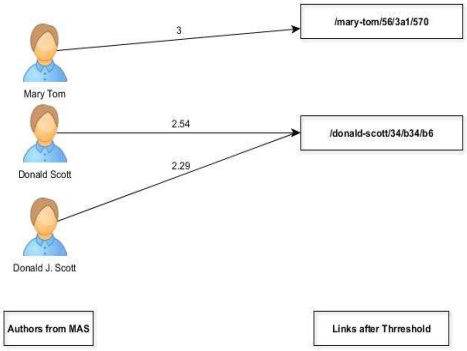


Figure 5 – Applying Thresholds

We will get several matches after applying threshold as illustrated in Figure 6.

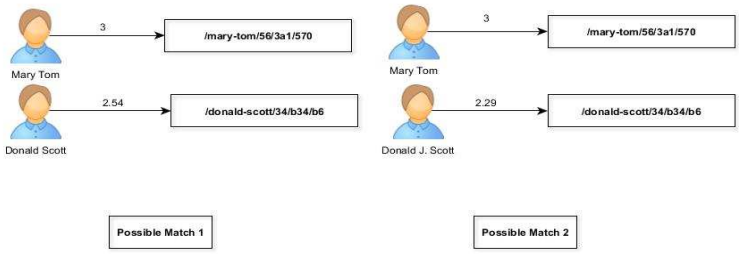


Figure 6 – Possible Matches after Threshold

We find the match with maximum possible edge weight. We have discussed the algorithm for finding the match with maximum weight in Chapter 3. The final output that we expect is the possible Match 1 in Figure 6. Let us consider a complex example as illustrated in Figure 7.

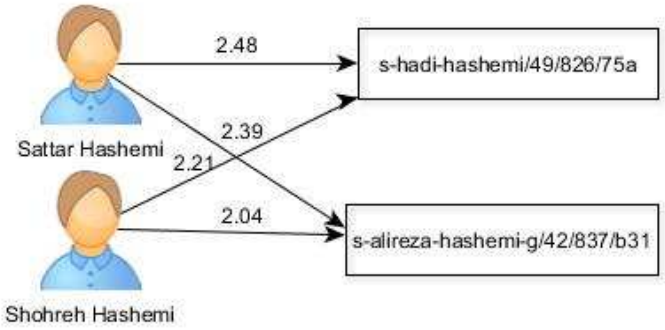


Figure 7 – Complex Example

For Figure 7, we can have 2 possible matches as illustrated in Figure 8.

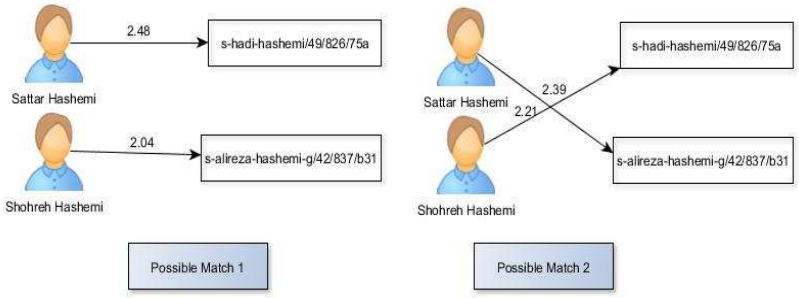


Figure 8 – Possible matches for a complex case

For possible match1 the sum of edge weights is 4.52 and for possible match 2 the sum of edge weight is 4.6. Hence, we choose the possible match 2 as our final output.

Chapter 3

Algorithms To Find Match With Maximum Weight

We use the Hungarian algorithm [4] to find the match with the maximum weight. The pseudo code [6] of the Hungarian algorithm is presented in Appendix E. Let us consider the example as illustrated in Figure 9.

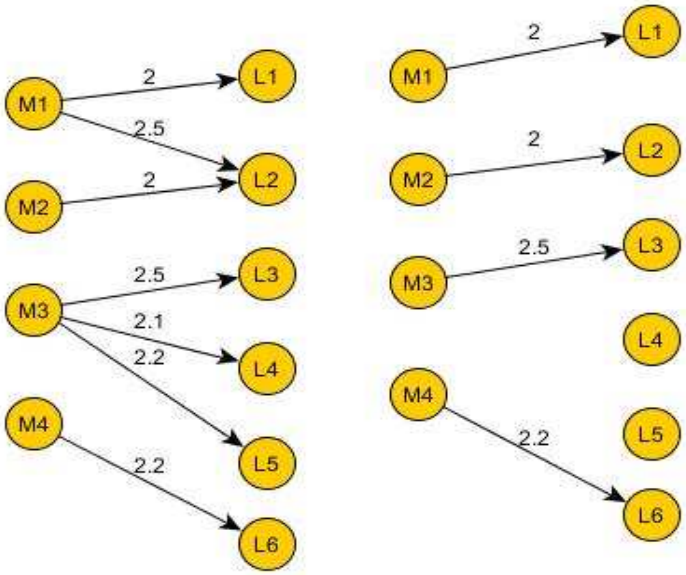


Figure 9 – Hungarian Algorithm Example

For the given figure in the left side of Figure 9, we are concerned with finding the maximum weighted bi-partite matching as in the right figure of Figure 9. To find the maximum edge weight in a bipartite graph, we can use the Hungarian algorithm as below.

Arrange the information from the bipartite graph in a matrix with the “authors from MAS” on the left and the “public profile information from LinkedIn” along the top, with the score for each pair in the middle. The *author – public profile* information can be

represented in a matrix as below. Since we are concerned with finding the maximum edge weight, we will first multiply the initial weights by -1.

	L1	L2	L3	L4	L5	L6
M1	-2	-2.5	0	0	0	0
M2	0	-2	0	0	0	0
M3	0	0	-2.5	-2.1	-2.2	0
M4	0	0	0	0	0	-2.2

Ensure that the matrix is square by the addition of dummy rows/columns if necessary. Conventionally, each element in the dummy row/column is the same as the largest number in the matrix. As we can see, the above matrix is not a square matrix. We will add row having the maximum value to make the matrix a square matrix. In our example, the maximum value is 0. Hence we will now add 2 more rows with all values as 0. The matrix would be as below.

	L1	L2	L3	L4	L5	L6
M1	-2	-2.5	0	0	0	0
M2	0	-2	0	0	0	0
M3	0	0	-2.5	-2.1	-2.2	0
M4	0	0	0	0	0	-2.2
	0	0	0	0	0	0
	0	0	0	0	0	0

Reduce the rows by subtracting the minimum value of each row from that row. For each row, we will subtract the minimum row value from the values and update the

matrix. For example, the minimum value in the first row is -2.5. We will subtract -2.5 from all the row values in the first row. After doing this step for all the rows the matrix will be updated as below.

	L1	L2	L3	L4	L5	L6
M1	0.5	0	2.5	2.5	2.5	2.5
M2	2	0	2	2	2	2
M3	2.5	2.5	0	0.4	0.3	2.5
M4	2.2	2.2	2.2	2.2	2.2	0
	0	0	0	0	0	0
	0	0	0	0	0	0

Reduce the columns by subtracting the minimum value of each column from that column. The minimum column value is 0 in all the 6 columns. Therefore, the matrix will stay the same as above.

Cover the zero elements with the minimum number of lines it is possible to cover them with. If the number of lines is equal to the number of rows, then we can choose a set of zeros so that each row or column has only one selected.

	L1	L2	L3	L4	L5	L6
M1	0.5	0	2.5	2.5	2.5	2.5
M2	2	0	2	2	2	2
M3	2.5	2.5	0	0.4	0.3	2.5
M4	2.2	2.2	2.2	2.2	2.2	0
	0	0	0	0	0	0
	0	0	0	0	0	0

Add the minimum uncovered element to every covered element. If an element is covered twice, add the minimum element to it twice.

	L1	L2	L3	L4	L5	L6
M1	0.5	0.5	2.5	2.5	2.5	2.5
M2	2	0.5	2	2	2	2
M3	3	3.5	0.5	0.9	0.8	3
M4	2.7	3.2	2.7	2.7	2.7	0.5
	0.5	1	0.5	0.5	0.5	0.5
	0.5	1	0.5	0.5	0.5	0.5

Subtract the minimum element from every element in the matrix. Cover the zero elements again.

	L1	L2	L3	L4	L5	L6
M1	0	0	2	2	2	2
M2	1.5	0	1.5	1.5	1.5	1.5
M3	2.5	3	0	0.4	0.3	2.5
M4	2.2	2.7	2.2	2.2	2.2	0
	0	1	0	0	0	0
	0	0.5	0	0	0	0

Now, from the available matches we will choose a match with the maximum weight which in our example corresponds to M1-L1 + M2-L2 + M3-L3 +M4-L6. This is already illustrated in the right figure of Figure 9.

However, for the Hungarian algorithm to execute, we needed to increase the heap size in the JVM because the input size was larger. However, if the input size to the Hungarian algorithm can be reduced, the execution of the Hungarian algorithm can be speeded up. This actually resulted in a much better implementation which is more time efficient and memory efficient than the Hungarian algorithm.

The graph used for Hungarian algorithm can be represented using files. The input file contains the columns of authorID, edgeweight and LinkedInURL. The pseudo code of the modified greedy algorithm can be summarized as below.

1. Initially, sort the input file on the authorID column.
2. The input file is split into 2 files such that the first file contains only the rows having unique author IDs (*unique.txt*) and the second file contains all the rows that are repeated (*duplicate.txt*).
3. Repeat steps 1 and 2 but this time after sorting the input file on the LinkedInURL column.
4. For the duplicate rows obtained as a result of step 3, check if the authorID already exists in unique.txt and remove the line if already exists.
5. Do a greedy approach to select the maximum weighted sum in duplicate.txt file.

Let us consider the below example illustrated in Figure 10 to understand how we are applying the modified greedy algorithm to our problem.

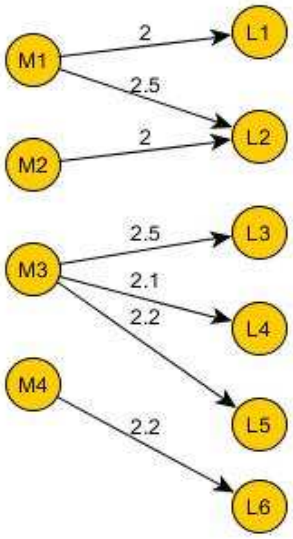


Figure 10 – Modified Greedy Approach Example

After sorting the input file on authorID column, we are splitting our input file into *unique.txt* and *duplicate.txt* files as illustrated in Figure 11.

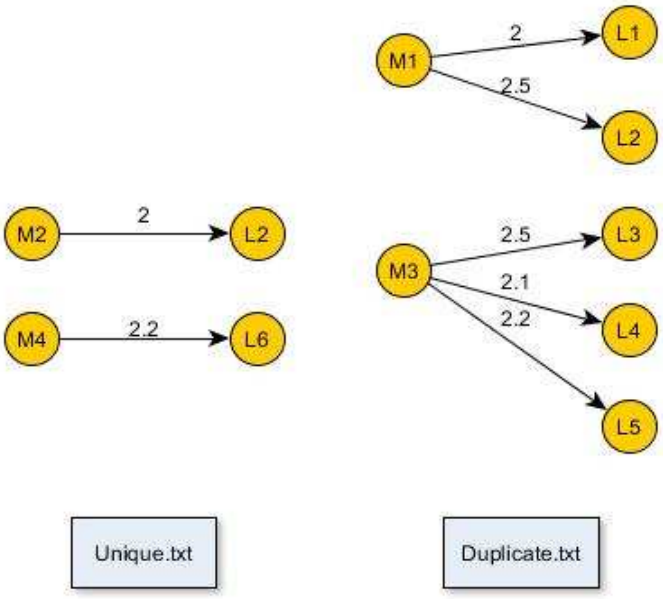


Figure 11 – Split into 2 files

We sort the input file on LinkedInURL column. After this step, we have our 2 files as illustrated in Figure 12.

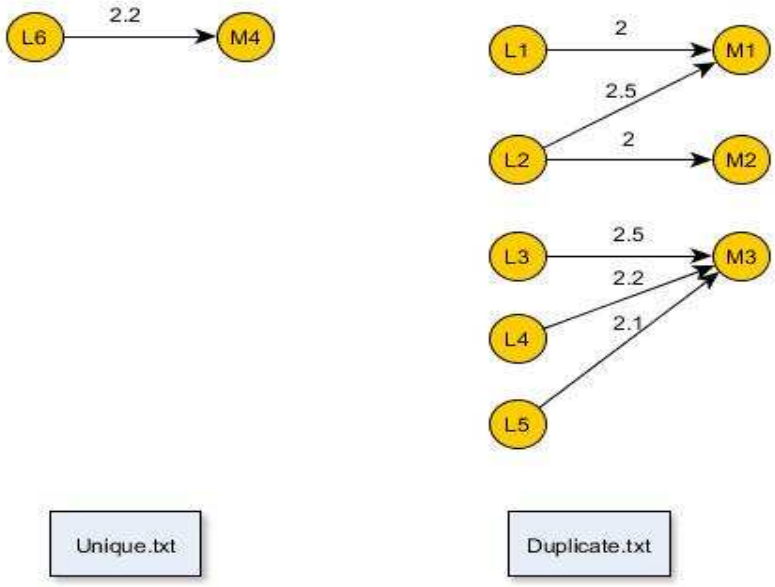


Figure 12 – 2 files after processing

As we can see from the Figure 12, the edge M2-L2 is not present in unique.txt because it shares an edge with M1 as well. Now, we select the maximum greedy weight from duplicate.txt. After the above step our 2 files would look as illustrated below in Figure 13.

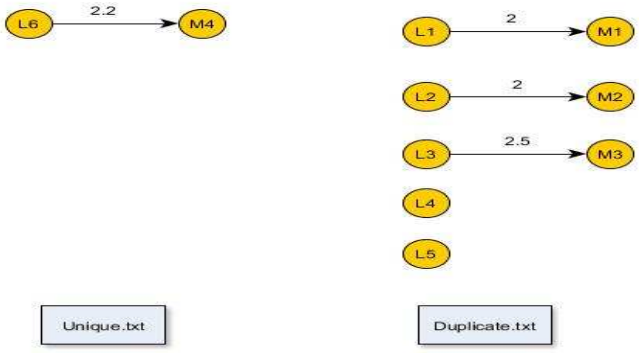


Figure 13 – Sorting the file

Chapter 4

Preprocessing Input Data

We need to pre-process the input data before applying our algorithms. The presence of duplicate records is a major data quality concern in large databases [7]. This is done because the input data contains duplicate records for the same Author. For example, if we look at Figure 14 for 2 authors from MAS, it represents the same author but using 2 different ids.

Figure 14 – Duplicate Authors from MAS

To check for duplicate ids for the same author, we compare all the authors that have the exact same name (i.e.) same first name, same middle name and same last name. For each author, we have the author's organization information and the specific skill sets of that author available from MAS. We also have the list of co-authors who have collaborated with the author. All these fields are highlighted in the Figure 14. We use the Levenshtein distance for finding the organization score between 2 authors having same

name. We use the Manhattan distance for finding the skills score and the collaborators score. The pseudocode that we use to detect the duplicate records in MAS is provided in Appendix F.

Chapter 5

Experiments

We evaluated our work for duplicate author profile detection in MAS and the possible profile match between MAS and LinkedIn. The MAS dataset contains only the authors who published papers in conferences related to computer science. The total authors in the MAS dataset are 696876. The total profiles extracted for these authors from LinkedIn are 8218476. The Hungarian algorithm takes $O\,(n^3)$ time to finish execution. The Hungarian algorithm fails with a java heap memory error for larger datasets. If we use the modified greedy approach, the input size could be considerably reduced and this input could be used for Hungarian algorithm. We evaluated our work based on the measures of precision,recall and f-measure. For calculating these measures, we got help from some human evaluators. We use the below formula for calculating the measures.

$$Precision = \frac{TP}{(TP + \ FP)}$$

$$Recall = \frac{TP}{(TP + \ FN)}$$

$$F1 = 2 \times \frac{Precision \ \times Recall}{(Precision + \ Recall)}$$

For duplicate author profile detection in MAS, our algorithm detected 3866 duplicate author pairs. However, since it is not possible to verify all the 3866 pairs, we carried our evaluation work on a smaller number of pairs as below.

We have an evaluation page for calculating precision which is illustrated in Figure 15. For calculating the precision, we got help from 5 human evaluators. All the human evaluators evaluated more or less the same number of pages. The true positive (*TP*) cases occur when the evaluators agree with the duplicates detected by our algorithm. The false

positive (*FP*) cases occur when the evaluators do not agree with the duplicates detected by our algorithm. When the evaluator opens the page, we present 2 authors randomly whom our algorithm has detected as duplicate and ask the evaluators to determine if it is true or not. Based on the user responses, we calculate the precision.

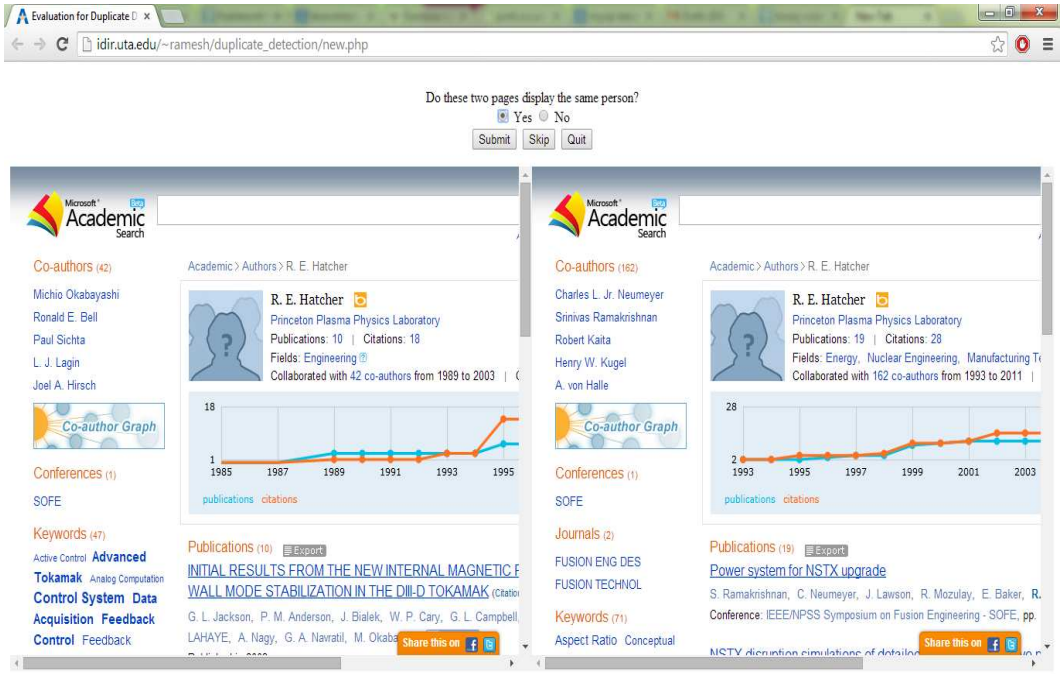


Figure 15 – Duplicate Author Detection Evaluation Page

To calculate the recall, we randomly selected some authors and shared the author names from MAS to the human evaluators. We asked the human evaluators to verify if there are duplicate entries for the shared authors in MAS. For calculating recall, we got help from 2 human evaluators. TP cases are when both the algorithm and evaluators agree that the entity is distinct and doesn't have duplicate. FN cases are when the evaluators find duplicates that are not detected by the algorithm. Again, the workload

was equally distributed between the 2 human evaluators. Based on the user responses, we calculate the recall.

Based on the evaluators' response, we present our experiment results as below.

Table 3 – Duplicate Author Detection Experiment Results

Precision	**Recall**	**F1-Score**
$\frac{79}{(79+51)} = 0.6076$	$\frac{68}{(68+0)} = 1$	0.7559

As the recall value suggests, our algorithm has not left out any author who has duplicate profile in MAS. The reason for the higher recall value is because we have considered all authors who have exact first name, middle name and last name.

The second evaluation was on determining whether the pairs detected by our algorithm between an author from MAS and a person's profile in LinkedIn are correct or not. We use threshold values to remove the weaker edges. We use three threshold values. When we set the threshold value greater than 2.8, our algorithm returns 3020 pairs. However, if we have a slightly lower threshold value of 2.5 our algorithm returns 5415 pairs. The total pairs returned by our algorithm are 17648 if we choose our threshold value as 2. Again calculating the precision and recall for such large number of pairs is not possible and hence we got help from the human evaluators. The precision calculation was carried out in the same way as we did earlier for duplicate author detection. We designed an evaluation page and asked the human evaluators to evaluate our work. The screenshot of our evaluation page is illustrated in Figure 16. For calculating precision, we got help from 5 human evaluators whose workload was equally distributed. The true positive and false positive values are determined in the same way as we did for duplicate author detection in MAS.

Figure 16 – Evaluation Page for Linked Data

For recall calculation, we selected some authors randomly from MAS and prepared an excel sheet with a hyperlink to the authors MAS page. After that, we explained the evaluators on how to get the LinkedIn URL based on the author's MAS page. We got help from 2 human evaluators and the workload was equally distributed among the evaluators.

Based on the users' evaluation results, we have prepared the following table which summarizes our experimental results. As we can see, we have tested our algorithm for various thresholds which we have specified as the experimental condition in the below table.

Table 4 – Experiment Results for pair detection between MAS and LinkedIn

Experiment Condition	Precision	Recall	F1 Score
Threshold > 2	$\frac{35}{(35+16)} = 0.68$	$\frac{21}{(21+9)} = 0.7$	0.689
Threshold > 2.5	$\frac{45}{(45+6)} = 0.88$	$\frac{15}{(15+10)} = 0.6$	0.713
Threshold > 2.8	$\frac{50}{50+1} = 0.98$	$\frac{18}{(18+17)} = 0.51$	0.674

As per our experimental results, it is clearly evident that the higher threshold values will have a higher precision rate.

Appendix A

Pseudocode on Finding Name Variations

We will first see the pseudo code for different name variations that we use for a name in MAS in LinkedIn. This function is implemented using VBA script. We have all the author names in our tables. We select the author name and find the name variations for this author and insert the possible LinkedIn URL into the table. The LinkedIn URL for any person will always follow the same structure (www.linkedin.com/pub/dir/FN/LN).Let us now see the pseudo code on how to find the LinkedIn URLs for various name combinations.

```
'This function finds all the LinkedIn URLs for a name from MAS.
'This pseudocode is implemented using VBA script.
Function LinkedInURLsFind()
  'Initialize the variables used.
  Dim AuthorFN, AuthorLN, AuthorID, AuthorMN,Ln;
  Ln="http://www.linkedin.com/pub/dir/";
  sql="SELECT id, first_name,last_name,middle_name FROM Author”;
  'Store the mysql values into the local variables.
  Do Until obj.EOF
    AuthorID = obj.Fields(0).Value;
    AuthorFN = obj.Fields(1).Value;
    AuthorLN = obj.Fields(2).Value;
    AuthorMN = obj.Fields(3).Value;
    'If no middle name is present, we are constructing 2 URLs.
    If Len(AuthorMN) = 0 Then
      Dim AuthorInitials, LinkedIn_URL1, LinkedIn_URL2;
      AuthorInitials = left(AuthorFN, 1);
      AuthorInitials = AuthorInitials & ".";
      LinkedIn_URL1 = "Ln" & AuthorFN & "/" & AuthorLN;
      LinkedIn_URL2 = "Ln" & AuthorInitials &  "/" & AuthorLN;
      sql = "INSERT INTO Author_LinkedIn (AuthorID,
             AuthorFN, AuthorMN, AuthorLN, LinkedIn_URL1)";
      'This if loop is to ensure that the author first name
       is not the same as initials. If they are the same, no
       need to insert author initials.
       If AuthorInitials <> AuthorFN Then
         sql = "INSERT INTO Author_LinkedIn(AuthorID,
                AuthorFN, AuthorMN, AuthorMN,LinkedIn_URL2)";
       End If

    Else
    'Initialize variables for authors who have middle names.

      Dim Author_FN, Author_LN, AuthorMNInitials,
          AuthorFMcombined,LinkedIn_URL5, LinkedIn_URL3,
          LinkedIn_URL4;
```

```
      'This variable has the FN and MN combined.
       Author_FN = AuthorFN & " " & AuthorMN;
      'This variable has the MN and LN combined.
       Author_LN = AuthorMN & " " & AuthorLN;
      'This variable gets the initials for MN.
       AuthorMNInitials = left(AuthorMN, 1);
       AuthorMNInitials = AuthorMNInitials & ".";
       'This variable gets the initials for the FN.
       AuthorInitials = left(AuthorFN, 1);
       AuthorInitials = AuthorInitials & ".";
       'This variable has FN and MN initials combined.
       AuthorFMcombined = AuthorInitials + AuthorMNInitials;
       'URL constructed with FN and LN.
       LinkedIn_URL1 = "Ln" & AuthorFN & "/" & AuthorLN;
       'URL constructed with {FN,MN} and LN.
       LinkedIn_URL2 = "Ln" & Author_FN & "/" & AuthorLN;
       'URL constructed with FN and {MN,LN}.
        LinkedIn_URL3 = "Ln" & AuthorFN & "/" &  Author_LN;
       'URL constructed with FN initials and {MN,LN}.
        LinkedIn_URL4 = "Ln" & AuthorInitials & "/" & Author_LN;
       'URL constructed with {initials of FN & MN },LN
        LinkedIn_URL5 = "Ln" & AuthorFMcombined & "/" & AuthorLN;
        sql = "INSERT INTO Author_LinkedIn (AuthorID,
               AuthorFN, AuthorMN, AuthorLN, LinkedIn_URL1)";
        sql = "INSERT INTO Author_LinkedIn (AuthorID,
               AuthorFN, AuthorMN, AuthorLN, LinkedIn_URL2)";
        sql = "INSERT INTO Author_LinkedIn(AuthorID,
               AuthorFN, AuthorMN,AuthorLN, LinkedIn_URL3)";
        If AuthorInitials <> AuthorFN Then
          sql = "INSERT INTO Author_LinkedIn(AuthorID,
                 AuthorFN, AuthorMN, AuthorLN, LinkedIn_URL4)";
        End If
        sql = "INSERT INTO Author_LinkedIn(AuthorID, AuthorFN,
               AuthorMN, AuthorLN, LinkedIn_URL5)";
      End If
    obj.MoveNext;
  Loop
End Function
```

Appendix B

Pseudocode On Determining If The Links Are Valid

After finding the possible LinkedIn URLs for an author, we are checking if the link is valid. By valid, we mean if it actually exists. We use the below pseudo code to check if the URL is valid. We implemented the below pseudo code using bash shell script.

```
#####################################################################
#Check if the links are valid.
#By valid, we mean to check if the links actually exist.
#We check if URL exists and update the value back in our table.
#We used bash script to implement this pseudocode.
#####################################################################

#We are selecting the URLs from table.

url=(mysql "SELECT LinkedIn_URL from Author_LinkedIn");

while read url;
do
  #The variable checkurl is used to check if the URL exists.
  #curl is used to check if the page actually exists.
  checkurl=$(curl -s --head "$url" | head -n 1 |grep HTTP/1.[01][23]..");
  #Check if the variable is null. If null, there is no URL that exists.
    if [ -z "$var" ]
    then
      Update LinkedIn_URL_exists=-1 where LinkedIn_URL="$checkurl";
    else
        Update LinkedIn_URL_exists=1 where LinkedIn_URL="$checkurl";
    fi
done
```

Appendix C

Pseudocode On Calculating The Edge Weight

Now, we have all the valid LinkedIn URLs available. We need to calculate the edge weight. The initial function which we use to select the values from database can be presented as below.

```
#*****THE MAIN EXECUTION OF THE SCRIPT STARTS HERE*********

sql=("select authorid, authorname, authororg, Author_LinkedIn_URL from
      Author_LinkedIn where LinkedIn_URL_Exists=1") > Results.txt;
#The pagination class is not present in the html source of single
 profile page.
#pagination class is present in html source of multiple profiles which
 can be upto 25 (max).

while IFS is not null
do
    page="$(curl ""$Author_LinkedIn_URL"" | grep "div
            class=\"pagination\"")";
    #Single profile page and so we are calling the function.

    if [ -z "$page" ]
    then
        Extract_Skills_Organization_LinkedIn "$AuthorID"
        "$Author_LinkedIn_URL" "$AuthorOrg" "$AuthorName";
    #Multiple profile page and so we will extract each URL and then call
     the function.

    else
        curl "$Author_LinkedIn_URL" | grep "<a href=.*title=" | grep -v
        Directory | grep -v class= > Multiple_Profiles.txt;
        while read -r line;
        do
            Extract_Skills_Organization_LinkedIn "$AuthorID" $line
            "$AuthorOrg" "$AuthorFN" "$AuthorMN" "$AuthorLN";
        done < <( awk -F\" '{print $2}' Multiple_Profiles.txt);
    fi
done < Results.txt
```

In the main script, we have made a call to **Extract_Skills_Organization_LinkedIn ()** **f**unction. The pseudo code for it can be presented as below.

```
#*****************************************************************
#This function accepts 6 input parameters.
#param1 - AuthorID
#param2 - AuthorURL
#param3 - Author organization in MAS
#param4 - Author First Name
#param5 - Author Middle Name
#param6 - Author Last Name
#*****************************************************************

Extract_Skills_Organization_LinkedIn ()
{
  #Dump the HTML source page into foo.txt file.
  curl ""$2"" -s |  w3m -dump -T text/html > foo.txt;
  #I extract the name from LinkedIn.
  LinkedIn_Name=$(sed -n '6p' foo.txt | sed -e 's/ //g');
  MSAS_Name="$4""$5""$6";
  #I am converting the names from MAS and LinkedIn to lower cases.
  LinkedIn_Name_Levenish=$(echo "$LinkedIn_Name" | awk '{print tolower($0)}');
  MSAS_Name_Levenish=$(echo "$MSAS_Name" | awk '{print tolower($0)}');
  #Lenvenshtein algorithm implementation for the Name score.
  #I call the Levenshtein_Match () function to calculate the name score.
  #I use length of names to normalize the values.
  name_score=$( Levenshtein_Match "$LinkedIn_Name" "$MSAS_Name"
"${#LinkedIn_Name_Length}" "${#MSAS_Name_Length}" );
  #I am subtracting the score from 1 so that higher value corresponds to better
match.
  name_score_value=$(bc <<< "scale=5; 1 - $name_score");
  #Extracting the skills from the main HTML file.
  awk 'f;/Skills & Expertise/{f=1}' foo.txt > foo1.txt;
  #This option checks if the file is non-empty. If the file is empty, there is
no skills assosciated.
  if [ -s foo1.txt ]
  then
    while read line;
    do
      echo $line >> skills;
    done;
  else
    break;
  fi
  sql_result=("SELECT MAS_Skill for author");
  cat sql_result >> skills;
  #Manhattan distance implementation for skills score.
  #common skills between MAS and LinkedIn.
  common_skills=$(sort skills | uniq -d | wc -l);
  #Total skills count.
  total_skills=$(sort skills | wc -l);
  nr_skills_score=$(($total_skills - $common_skills*2));
  dr_skills_score=$(($total_skills - $common_skills));
```

```
  skills_score=$(bc <<< "scale=5; $nr_skills_score/$dr_skills_score");
  #Same approach as name score as higher value in DB means better.
  skills_score_value=$(bc <<< "scale=5; 1 - $skills_score");
  #Extracting organization information from LinkedIn.

  #The organization section starts here.
  #If no MAS organization is found, there is no need to calculate the
org_score.
  if [ "$3" == "Not found" ]
  then
    organization_score=1;
  else
    #Extract information from the main HTML file.
    sed -n -e '/Overview/,/Connections/ p' foo.txt | grep "^ " | sed 's/^[
]\+//g' > organization_updated.txt;
    local LinkedIn_Organization_Array=();
    while read -r line;
    do
      #I put all the LinkedIn organizations into an array.
      LinkedIn_Organization_Array+=( "$line" );
    done < <( perl -pe 's/.+? at //' organization_updated.txt);
  #I am calling the Levenshtein algorithm from Organization_Exists () function.
  #The reason is I need to return the highest score for Levenshtein algorithm.
  #We are checking an author's organization against LinkedIn current
organization,
  #past work, past educational institutions.
  organization_score=$( Organization_Exists "$3" LinkedIn_Organization_Array[@]
);
  fi
  organization_score_value=$(bc <<< "scale=5; 1 - $organization_score");
}
```

We are checking the MAS organization against all the available LinkedIn organizations such as the current organization, past organization, educational institutions.

```
#*****************************************************************
#This is the function that checks if the MAS org and LinkedIn org match.
#For each organization in LinkedIn, I use the levenshtein distance algorithm.
#From the values returned, I select the minimum value.
#Selecting the minimum value is to make sure that the 2 strings are closely
matched.
#*****************************************************************
#This function accepts 2 parameters.
#Param1 - MAS Organization
#Param2 - Organizations from LinkedIn in an array.
#By organizations in LinkedIn, we mean the current, past and educational
institutions.
#*****************************************************************

Organization_Exists ()
{
  declare -a argAry1=("${!2}");
  local e;
  local Organization_MSAS_LinkedIn=();
  for e in "${argAry1[@]}";
  do
    tmp_org=$( Levenshtein_Match "$1" "$e" "${#1}" "${#e}" );
    Organization_MSAS_LinkedIn+=("$tmp_org");
  done
  Organization_Score=1;
  len=${#Organization_MSAS_LinkedIn[@]};
  i=0;
  while [ $i -lt $len ]
  do
    val=`echo "${Organization_MSAS_LinkedIn[$i]}" `;
    if [ $(echo "$Organization_Score < $val"|bc) -eq 0 ];
    then
      Organization_Score=$val;
    fi
    let i++;
  done
  #This is important as this acts as the return statement.
  echo "$Organization_Score";
}
```

We are using the Levenshtein algorithm to find the edit distance between two strings. The pseudo code for the implementation of Levenshtein algorithm can be presented as below.

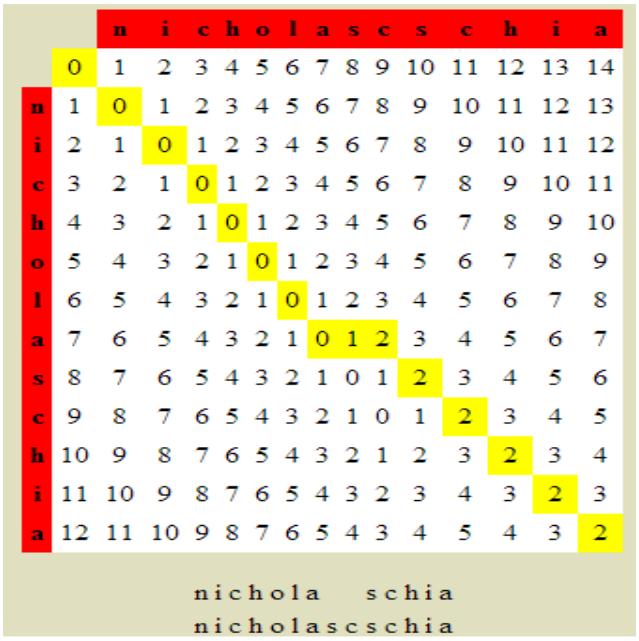
```
#*****************************************************************
#Levenshtein distance is used to find the edit distance between 2 strings.
#I am using this algorithm to find the name score and the organization score.
#I have used the mathematical formula lev dist/(sum of lengths of 2 strings).
#The above formula is to normalize the value before putting it in the database.
#This function takes 4 parameters. The 2 strings to be searched
#and the lengths of 2 strings.
#This function will be used to update the name score
#and the organization score.
#*****************************************************************
#However, I have made it accept 4 input parameters.
#Param1 - MAS Name/Organization
#Param2 - LinkedIn Name/Organization
#Param3 - MAS Name/Organization length
#Param4 - LinkedIn Name/Organization length
#*****************************************************************

Levenshtein_Match() {
ret=$(awk '
        function min(x, y) {
        return x < y ? x : y
        }
        function lev(s,t) {
        m = length(s)
        n = length(t)
        for(i=0;i<=m;i++) d[i,0] = i
        for(j=0;j<=n;j++) d[0,j] = j
        for(i=1;i<=m;i++) {
        for(j=1;j<=n;j++) {
        c = substr(s,i,1) != substr(t,j,1)
          d[i,j] = min(d[i-1,j]+1,min(d[i,j-1]+1,d[i-1,j-1]+c))
        }
      }
      return d[m,n];
    }
     BEGIN {print lev(ARGV[1], ARGV[2]) / ( ARGV[3] + ARGV[4] ); exit}' "$1"
"$2" "$3" "$4");
     #This return is important as this acts as the return statement.
     echo "$ret";
}
```

Appendix D

Example On Calculating Levenshtein Distance

There are several calculators available online to see how the Levenshtein edit distance is calculated between 2 strings. We used the calculator from http://odur.let.rug.nl/kleiweg/lev/.



		n	i	c	h	o	l	a	s	c	s	c	h	i	a
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
n	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13
i	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12
c	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11
h	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10
o	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9
l	6	5	4	3	2	1	0	1	2	3	4	5	6	7	8
a	7	6	5	4	3	2	1	0	1	2	3	4	5	6	7
s	8	7	6	5	4	3	2	1	0	1	2	3	4	5	6
c	9	8	7	6	5	4	3	2	1	0	1	2	3	4	5
h	10	9	8	7	6	5	4	3	2	1	2	3	2	3	4
i	11	10	9	8	7	6	5	4	3	2	3	4	3	2	3
a	12	11	10	9	8	7	6	5	4	3	4	5	4	3	2

nichola schia
nicholascschia

Appendix E

Pseudocode On Hungarian Algorithm

The pseudocode [7] of Hungarian algorithm is given as below.

```
procedure munkres is
   Row,Col : is array(1..n) of integer;        -- maintains
record of which row/columns are covered.
  stepnum : integer;                            -- covered =
1, non-covered = 0
  done : boolean;

  function step1(stepnum: in out integer) is
    :
  function step2(stepnum: in out integer) is
    :
  function step3(stepnum: in out integer) is
    :

begin
  done:=false;
  stepnum:=1;
  while not(done) loop
    case stepnum is
      when 1 => step1(stepnum);
      when 2 => step2(stepnum);
      when 3 => step3(stepnum);
      when 4 => step4(stepnum);
      when 5 => step5(stepnum);
      when 6 => step6(stepnum);
      when others => done:=true;
    end case;
  end loop;
end munkres;
```

In each pass of the loop the step procedure called sets the value of stepnum for the next pass. When the algorithm is finished the value of stepnum is set to some value outside the range 1..6 so that done will be set to true and the program will end. In the completed program the tagged (starred) zeros flag the row/column pairs that have been assigned to each other. We will discuss the implementation of a procedure for each step of Munkres' Algorithm below. We will assume that the cost matrix *C(i,j)* has already been loaded with the first index referring to the row number and the second index referring to the column number.

Step 1:

For each row of the matrix, find the smallest element and subtract it from every element in its row. *Go to Step 2.* We can define a local variable called *minval* that is used to hold the smallest value in a row. Notice that there are two loops over the index *j* appearing inside an outer loop over the index *i*. The first inner loop over the index *j* searches a row for the *minval*. Once *minval* has been found this value is subtracted from each element of that row in the second inner loop over *j*. The value of step is set to *2* just before *stepone* ends.

```
procedure stepone(step : in out integer) is
   minval : integer;
 begin
   for i in 1..n loop

     minval:=C(i,1);
     for j in 2..n loop
       if minval>C(i,j) then
         minval:=C(i,j);
       end if;
     end loop;

     for j in 1..n loop
       C(i,j):=C(i,j)-minval;
```

```
      end loop;

    end loop;
    step:=2;
  end stepone;
```

Step 2

Find a zero (Z) in the resulting matrix. If there is no starred zero in its row or column, star Z. Repeat for each element in the matrix. Go to Step 3. In this step, we introduce the mask matrix M, which in the same dimensions as the cost matrix and is used to star and prime zeros of the cost matrix. If M(i,j)=1 then C(i,j) is a starred zero, If M(i,j)=2 then C(i,j) is a primed zero. We also define two vectors R_cov and C_cov that are used to "cover" the rows and columns of the cost matrix C. In the nested loop (over indices i and j) we check to see if C(i,j) is a zero value and if its column or row is not already covered. If not then we star this zero (i.e. set M(i,j)=1) and cover its row and column (i.e. set R_cov(i)=1 and C_cov(j)=1). Before we go on to Step 3, we uncover all rows and columns so that we can use the cover vectors to help us count the number of starred zeros.

```
procedure steptwo(step: in out integer) is
  begin

    for i in 1..n loop
      for j in 1..n loop
        if C(i,j)=0 and C_cov(j)=0 and R_cov(i)=0 then
          M(i,j):=1;
          C_cov(j):=1;
          R_cov(i):=1;
        end if;
      end loop;
    end loop;
```

```
    for i in 1..n loop
      C_cov(i):=0;
      R_cov(i):=0;
    end loop;
    step:=3;

  end steptwo;
```

Step 3

Cover each column containing a starred zero. If K columns are covered, the starred zeroes describe a complete set of unique assignments. In this case, Go to DONE, otherwise, Go to Step 4. Once we have searched the entire cost matrix, we count the number of independent zeroes found. If we have found (and starred) K independent zeroes then we are done. If not we proceed to Step 4.

```
Procedure stepthree(step : in out integer) is
    count : integer;
  begin
    for i in 1..n loop
      for j in 1..n loop
        if M(i,j)=1 then
          C_cov(j):=1;
        end if;
      end loop;
    end loop;
    count:=0;
    for j in 1..n loop
      count:=count + C_cov(j);
    end loop;
    if count>=n then
      step:=7;
    else
      step:=4;
    end if;
  end stepthree;
```

Step 4

Find a no covered zero and prime it. If there is no starred zero in the row containing this primed zero, Go to Step 5. Otherwise, cover this row and uncover the column containing the starred zero. Continue in this manner until there are no uncovered zeroes left. Save the smallest uncovered value and Go to Step 6. In this step, statements such as "find a noncovered zero" are clearly distinct operations that deserve their own functional blocks. We have decomposed this step into a main procedure and three subprograms (2 procedures and a boolean function).

```
procedure stepfour(step : in out integer) is
    row,col  : integer;
    done     : boolean;

    procedure find_a_zero(row,col : out integer) is
      i,j : integer;
      done: boolean;
    begin
      row:=0;
      col:=0;
      i:=1;
      done:=false;
      loop
        j:=1;
        loop
         if C(i,j)=0 and R_cov(i)=0 and C_cov(j)=0 then
            row:=i;
            col:=j;
            done:=true;
         end if;
          j:=j+1;
         exit when j>n;
        end loop;
        i:=i+1;
        if i>n then done:=true; end if;
```

```
        exit when done;
      end loop;
    end find_a_zero;

    function star_in_row(row : integer) return boolean is
     tbool : boolean;
    begin
      tbool:=false;
      for j in 1..n loop
        if M(row,j)=1 then
          tbool:=true;
        end if;
      end loop;
      return tbool;
    end star_in_row;

    procedure find_star_in_row(row, col : in out integer) is
    begin
      col:=0;
      for j in 1..n loop
        if M(row,j)=1 then
          col:=j;
        end if;
      end loop;
    end find_star_in_row;

  begin
    done:=false;
    while not(done) loop
      find_a_zero(row,col);
      if row=0 then
        done:=true;
        step:=6;
      else
        M(row,col):=2;
        if star_in_row(row) then
          find_star_in_row(row,col);
```

```
          R_cov(row):=1;
          C_cov(col):=0;
        else
          done:=true;
          step:=5;
          Z0_r:=row;
          Z0_c:=col;
        end if;
      end if;
    end loop;
  end stepfour;
```

Step 5

Construct a series of alternating primed and starred zeros as follows. Let Z_0 represent the uncovered primed zero found in Step 4. Let Z_1 denote the starred zero in the column of Z_0 (if any). Let Z_2 denote the primed zero in the row of Z_1 (there will always be one). Continue until the series terminates at a primed zero that has no starred zero in its column. Unstar each starred zero of the series, star each primed zero of the series, erase all primes and uncover every line in the matrix. Return to Step 3. You may notice that Step 5 seems vaguely familiar. We decompose the operations of this step into a main procedure and five relatively simple subprograms.

```
 procedure stepfive(step : in out integer) is
    count : integer;
    done  : boolean;

    r,c   : integer;

  procedure find_star_in_col(c : in integer; r : in
out integer) is
  begin
    r:=0;
    for i in 1..n loop
      if M(i,c)=1 then
```

```
        r:=i;
      end if;
    end loop;
  end find_star_in_col;

  procedure find_prime_in_row(r : in integer; c : in
out integer) is
  begin
    for j in 1..n loop
      if M(r,j)=2 then
        c:=j;
      end if;
    end loop;
  end find_prime_in_row;

  procedure convert_path is
  begin
    for i in 1..count loop
      if M(path(i,1),path(i,2))=1 then
        M(path(i,1),path(i,2)):=0;
      else
        M(path(i,1),path(i,2)):=1;
      end if;
    end loop;
  end convert_path;

  procedure clear_covers is
  begin
    for i in 1..n loop
      R_cov(i):=0;
      C_cov(i):=0;
    end loop;
  end clear_covers;

  procedure erase_primes is
  begin
    for i in 1..n loop
      for j in 1..n loop
```

```
        if M(i,j)=2 then
          M(i,j):=0;
        end if;
      end loop;
    end loop;
  end erase_primes;

  begin
    count:=1;
    path(count,1):=z0_r;
    path(count,2):=z0_c;
    done:=false;
    while not(done) loop
      find_star_in_col(path(count,2),r);
      if r>0 then
        count:=count+1;
        path(count,1):=r;
        path(count,2):=path(count-1,2);
      else
        done:=true;
      end if;
      if not(done) then
        find_prime_in_row(path(count,1),c);
        count:=count+1;
        path(count,1):=path(count-1,1);
        path(count,2):=c;
      end if;
    end loop;
    convert_path;
    clear_covers;
    erase_primes;
    step:=3;
  end stepfive;
```

Step 6

Add the value found in Step 4 to every element of each covered row, and subtract it from every element of each uncovered column. Return to Step 4 without altering any stars, primes, or covered lines. Notice that this step uses the smallest uncovered value in the cost matrix to modify the matrix. Even though this step refers to the value being found in Step 4 it is more convenient to wait until you reach Step 6 before searching for this value. It may seem that since the values in the cost matrix are being altered, we would lose sight of the original problem. However, we are only changing certain values that have already been tested and found not to be elements of the minimal assignment. Also we are only changing the values by an amount equal to the smallest value in the cost matrix, so we will not jump over the optimal (i.e. minimal assignment) with this change.

```
procedure stepsix(step : in out integer) is
    minval : integer;

    procedure find_smallest(minval : out integer) is
    begin
      minval:=integer'last;
      for i in 1..n loop
        for j in 1..n loop
         if R_cov(i)=0 and C_cov(j)=0 then
           if minval>C(i,j) then
              minval:=C(i,j);
           end if;
         end if;
        end loop;
      end loop;
    end find_smallest;

  begin
    find_smallest(minval);
    for i in 1..n loop
      for j in 1..n loop
```

```
        if R_cov(i)=1 then
          C(i,j):=C(i,j)+minval;
        end if;
        if C_cov(j)=0 then
          C(i,j):=C(i,j)-minval;
        end if;
      end loop;
    end loop;
    step:=4;
  end stepsix;
```

Appendix F

Pseudocode On Duplicate Authors Detection in MAS

```
# Algorithm: DuplicateRecordsDetection(L)
# L is the set of authors from MAS.
# Initialize the 2 sets used.
unique_authorids <- {};
duplicate_authorids [] <- {};
# Get only authors having same name more than once in MAS.
Duplist <- getDuplicateAuthors(L);
#Iterate over each subset of authors with same name.
foreach subset s ∈ Duplist
{
    foreach pair a1,a2 in s
    {
            score=findscore(a1,a2)
            if score >= θ then
            {
                if a1 is not in unique_authorids then
                    unique_authorids <- {a1};
                    duplicate_authorids[a1] <- {a2};
            }
            else
            {
                if a1,a2 not in unique_authorids then
                    unique_authorids <- {a1,a2};
            }
    }
}
#function to return only authors with same name more than once.
getDuplicateAuthors(L)
{
    duplicate_entries <- {mysql_query "select authors group by
authorname having count(authorname) > 1"};
    return duplicate_entries;
}
#This function calculates the score between 2 authors.
#We use levenshtein distance for organization score.
#We use manhattan distance for skills score and collaborators
score.
findscore (a1,a2)
{
    return (org_score + skills_score + collaborators_score);
}
```

References

[1] Bizer, Christian, Tom Heath, and Tim Berners-Lee. "Linked data-the story so far." *International journal on semantic web and information systems* 5.3 (2009): 1-22.

[2] Kong, Xiangnan, Jiawei Zhang, and Philip S. Yu. "Inferring anchor links across multiple heterogeneous social networks." *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*. ACM, 2013.

[3] Frank, András. "On Kuhn's Hungarian method—a tribute from Hungary." *Naval Research Logistics (NRL)* 52.1 (2005): 2-5.

[4] Kuhn, Harold W. "The Hungarian method for the assignment problem." *Naval research logistics quarterly* 2.1-2 (1955): 83-97.

[5] Levenshtein, Vladimir I. "Binary codes capable of correcting deletions, insertions and reversals." *Soviet physics doklady*. Vol. 10. 1966.

[6] Hassanzadeh, Oktie, et al. "Framework for evaluating clustering algorithms in duplicate detection." Proceedings of the VLDB Endowment 2.1 (2009): 1282-1293.

[7] [Online] Pseudocode on Hungarian Algorithm
http://csclab.murraystate.edu/bob.pilgrim/445/munkres_old.html

Biographical Information

Ramesh Venkataraman received his B.Tech degree from SASTRA University, India. After finishing his B.Tech, he worked for Infosys Technologies Ltd, India for 3.8 years. He joined the University of Texas at Arlington in the fall of 2012. His current interest lies in Systems Engineering and he wishes to be a development and operations (DevOps) Engineer in the near future.