Máster en Ingeniería Computacional y Matemática

Trabajo Final de Máster

Impact evaluation of clustering-based k-anonymity for recommendations

Miguel Ros Martín
Máster en Ingeniería Computacional y Matemática
Anonimización y transparencia

Jordi Casas Roma, Julián Salas Piñón
Juan Alberto Rodríguez Velázquez

15 de enero de 2017

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Impact evaluation of clustering-based k-anonymity for recommendations*
Nombre del autor:	*Miguel Ros Martín*
Nombre del consultor/a:	Jordi Casas Roma Julián Salas Piñón
Nombre del PRA:	Juan Alberto Rodríguez Velázquez
Fecha de entrega (mm/aaaa):	01/2017
Titulación:	Máster en Ingeniería Computacional y Matemática
Área del Trabajo Final:	Anonimización y transparencia
Idioma del trabajo:	*Inglés*
Palabras clave:	*k-anonimity* *clustering* *recommendations*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

SaNGreeA es un algoritmo determinista voraz basado en agrupación para obtener grupos k-anónimos en un grafo etiquetado, no dirigido. Actualmente es un clásico y la referencia en los algoritmos de k-anonimización por agrupación. Tiene una complejidad de tiempo cuadrática. Esto lo hace realmente lento para una red razonablemente grande (1 M nodos). Nuestro proyecto adapta SaNGreeA para hacerlo escalable para una red grande real y lo especializa para asegurar que los grupos generados sean útiles para un sistema de recomendaciones.

SaNGreeA siempre devuelve la misma solución para un mismo conjunto de datos, pero no es la mejor solución ya que el problema subyacente es NP-completo. Genera una solución que es óptima local. Nuestra adaptación genera cientos de soluciones similares gracias a randomización con sesgo y múltiples reinicios y se selecciona la mejor solución de entre todas ellas.

Hemos desarrollado nuevas métricas y un sistema de evaluación para comprobar la calidad de los grupos generados, que consideran la potencial utilidad para un sistema de recomendaciones.

Hemos sido capaces de ejecutar nuestro algoritmo en un conjunto de datos grande real, con 1M de nodos y 75M de relaciones. Los resultados son

alentadores para un sistema de recomendación ya que los atributos quasi-identificadores más importantes no están muy dañados.

El algoritmo se ha implementado en Spark y puede ser ejecutado en hasta cientos de procesadores en paralelo para reducir el tiempo de computación de cientos de ejecuciones en apenas un par de horas.

Abstract (in English, 250 words or less):

SaNGreeA is a greedy and deterministic clustering algorithm for achieving k-anonymous clusters on a labeled, undirected graph. It is nowadays a classic and the leading work in clustering based k-anonymity algorithms. It has a quadratic time complexity which makes it really slow for a reasonably big network (1 M nodes). Our project adapts SaNGreeA to make it scalable for a real world big network and specialises it to make sure the generated clusters are useful for a recommender system.

SaNGreeA always returns the same solution, but it is never the best solution as the underlying problem is NP-complete. It generates a solution that is a local optima. Our adaptation generates hundreds of similar solutions with a multi-start biased randomization procedure and selects the best one.

We have developed new metrics and evaluation framework that consider the quality of the clusters from a recommender system point of view.

We were able to run our algorithm in a real world dataset with around 1M nodes and 75M relationships. The results are encouraging for recommendations as the most important quasi-identifier attributes are not heavily damaged. The algorithm has been implemented in Spark, and can be run in up to a hundred of computing processors in parallel to reduce the computing time of hundreds of different clusterizations to a couple of hours.

Contents

1 Introduction

1.1 Context of this work

XING [1] is a social network for business. People use XING, for example, to find a job and recruiters use XING to find the right candidate for a job. At the moment, XING has more than 15 Million users and around 1 Million job postings on the platform. Given a user, the goal of the job recommendation system is to predict those job postings that are likely to be relevant to the user. In order to fulfill this task, various data sources can be exploited. Job recommendations are displayed on xing.com as well as in XING's mobile apps.

RecSys Challenge 2016 [2] is organized by XING and CrowdRec [3]. In this edition of the RecSys Challenge, the task is: given a XING user, predict those job postings that a user will click on. XING provides publicly a dataset with information about the users, job postings, and the user clickstream on those postings during a time window (e.g., 1 month).

Researchers can use the dataset to design their own recommendation algorithm. At any time they can evaluate their system by submitting their productions to a submission portal, getting a score value back, which determines the quality of their predictions. The predictions are compared to the real clicks done by the users on a smaller time window immediately after the published one, which is private, not available to researchers.

The dataset published has been anonymised by masquerading ids, not revealing any names, and creating fake users / job postings to try to k-anonymise based on quasi-identifier attributes (career level, industry, discipline). The social network relationships between users have not been published.

We want to evaluate a potential extension to this public data-set, providing the network relationships between users, without giving out sensible data and preserving the anonymity of our users.

1.2 Objective of this work

We want to evaluate what would be the impact if we enhance the data-set by publishing social relationships between users, using a k-anonymization algorithm based on clustering techniques. We want to preserve the anonymity of the real individuals present in the released dataset, so it is important that it is not possible to re-identify any individuals of the dataset. And at the same time, we want to preserve the utility of the dataset.

k-anonymity is a model of data protection. The concept was introduced by P. Samarati [4] and L. Sweeney [5]. Essentially, this model asserts that any individual in the dataset can not be distinguished from at least other k - 1 individuals from the dataset in terms of quasi-identifier attributes values.

The focus is to develop a clustering based algorithm. We want to achieve k-anonymised clusters (on the quasi-identifier attributes career level, discipline and industry), where the clusters have been calculated based on the distance between user profiles. That means taking into account their quasi-identifier attributes, but also the relationships between users (friendship).

After the anonymization, the relationships between users of the same cluster will be dropped, and all the relationships that fall between two clusters will be simplified to one edge representing them.

These anonymous edges will be undirected between clusters, and they will have only one attribute with the count of edges summarized into that one.

The challenge is to develop a clustering k-anonymization algorithm that is able to scale up to 1M users, and around 100M edges between them.

1.3 Related work

There is quite research done for the anonymity of graphs. Most of them focus on unlabeled graphs, so the main information to be preserved is the graph structure itself, while the labeled graphs get less research.

There is always a tradeoff between the utility and protection of the anonymised data. Several measures of protection have been studied for social networks. They have introduced more specific definitions of k-anonymity for graphs depending on the assumption of the attacker's knowledge like k-degree anonymity

[6], k-neighborhood, etc. All of them can be named as k-$\mathcal{P}$-anonymity [7]. For a given structural property $\mathcal{P}$ and a vertex in the graph G there are at least k - 1 other vertices with the same property $\mathcal{P}$.

One technique is the edge or vertex modification. This is the approach used originally for the RecSys challenge. Their dataset was essentially an users table, but without any relationships. They added new nodes (fake users) into the dataset in order to still ensure there are at least k individuals with the same quasi-identifier attributes.

Another even simpler technique is a random perturbation of the graph. It is usually a simple and low complexity algorithm that can be used for big networks. Hay et al. (2007) [8] proposed a method that consists of removing p edges from the real graph, and adding p new fake ones so the number of vertices and edges in the anonymised network does not change. The downside is that it adds additional noise to the anonymised dataset and it does not have strict privacy warranties.

Casas-Roma et al. (2015) [9] studied how to preserve the anonymity of graph structures. Their approach is to generalise an unlabeled, undirected network by generalising the edge structures while retaining as much data utility as possible for graph mining tasks. Their algorithm can be used also for graph dimensionality reduction, as their generalisation summarizes the graph. The summary can then be used to approximate the structural properties of the original graph.

J. Salas (2016) [10] proposes a sampling and merging approach for unlabeled, undirected networks. The idea is to increase the utility of the published data by giving two views of it. First a set of local neighborhoods is anonymised separately from the rest of the graph. Then all local neighborhoods are published separately, and in addition, a merged graph.

A. Campan and T.M. Truta developed SaNGreeA (2009) [11], a greedy and deterministic clustering algorithm for achieving k-anonymous clusters on a labeled, undirected graph. It is nowadays a classic and leading work in clustering based k-anonymity algorithms.

Juan et al. [12] (2011) presented a generic procedure for converting a deterministic heuristic algorithm into a probabilistic one by adding biased random behavior. They used it for solving vehicle routing problems which have NP-complete complexity. With this technique, given an algorithm that can

generate some good-enough solution for a problem, it can be extended so that a big set of alternative good solutions can be generated in a simple way. Then the best generated solution is chosen.

K. Santhi and N. Sai Lohitha (2014) [13] claimed their clustering algorithm outperformed SaNGreeA in terms of minimizing the information loss. Their approach is to initially assign nodes into clusters of k elements at random, and then sequentially keep swapping nodes into other clusters to decrease the information loss until a local optimum is reached. There is no guarantee that the algorithm finds the global optimum, so they execute it several times with different initial random clusterings to try to find better local optimums.

T. Tassa and Dror J. Cohen (2013) [14] developed a similar sequential clustering algorithm. They have an initil random partition of all nodes into clusters. Then the main loop keeps iterating and moving nodes into clusters that decrease the information loss. As they do not swap nodes, it may happen that some clusters may grow in size, while others decrease. Then they have to split and merge clusters to restore the size. They have the same local optima problem, and suggest a similar restart mechanism.

In this work, we have developed a multi-start biased random algorithm based on SanGreeA. It uses some of the techniques described by Juan et al. and K. Santhi et al. work for generating multiple good enough local optimum solutions and keep the best performing one. Thanks to the clustering based approach, we will have k-neighborhood-anonymity which is the most restrictive and implies all the other structural properties $\mathcal{P}$.

1.4 Project planning

We have divided the project into three big milestones that are the continuous evaluation *PEC*.

1.4.1 Milestone PEC1: Work schedule

For this first milestone, we had to plan what to do, how and the reach. We investigated the state of the art in k-anonymization algorithms, and decided which one we can try to adapt to our needs.

At the same time, we had to build the dataset. We had to query the source original XING data and generate a subset similar to the used for the RecSys Challenge.

1.4.2 Milestone PEC2: Follow up

After the algorithm decision was made, we did a first naïve implementation without any parallelism. It should just generate one clustering solution. That version was very inefficient in terms of computing time and memory consumption. Next step, we optimized the algorithm such that it was able to be executed several times and finish in a reasonable amount of time. Many tweaks were necessary until it was performant.

At that point, we did not have a way to evaluate how the clustering output look like, so we implemented a simple evaluation framework to check a few metrics in order to compare the different clustering solutions. That was needed also to spot a few bugs and issues with the clustering. We did a few adjustments into the clustering algorithm thanks to that.

Finally, we implemented the final and complete evaluation framework and we started to collect all results for the different experiments published.

1.4.3 Milestone PEC3: Documentation

The main output of this milestone is the almost finished documentation. After this milestone, the last work to do is to record the presentation video.

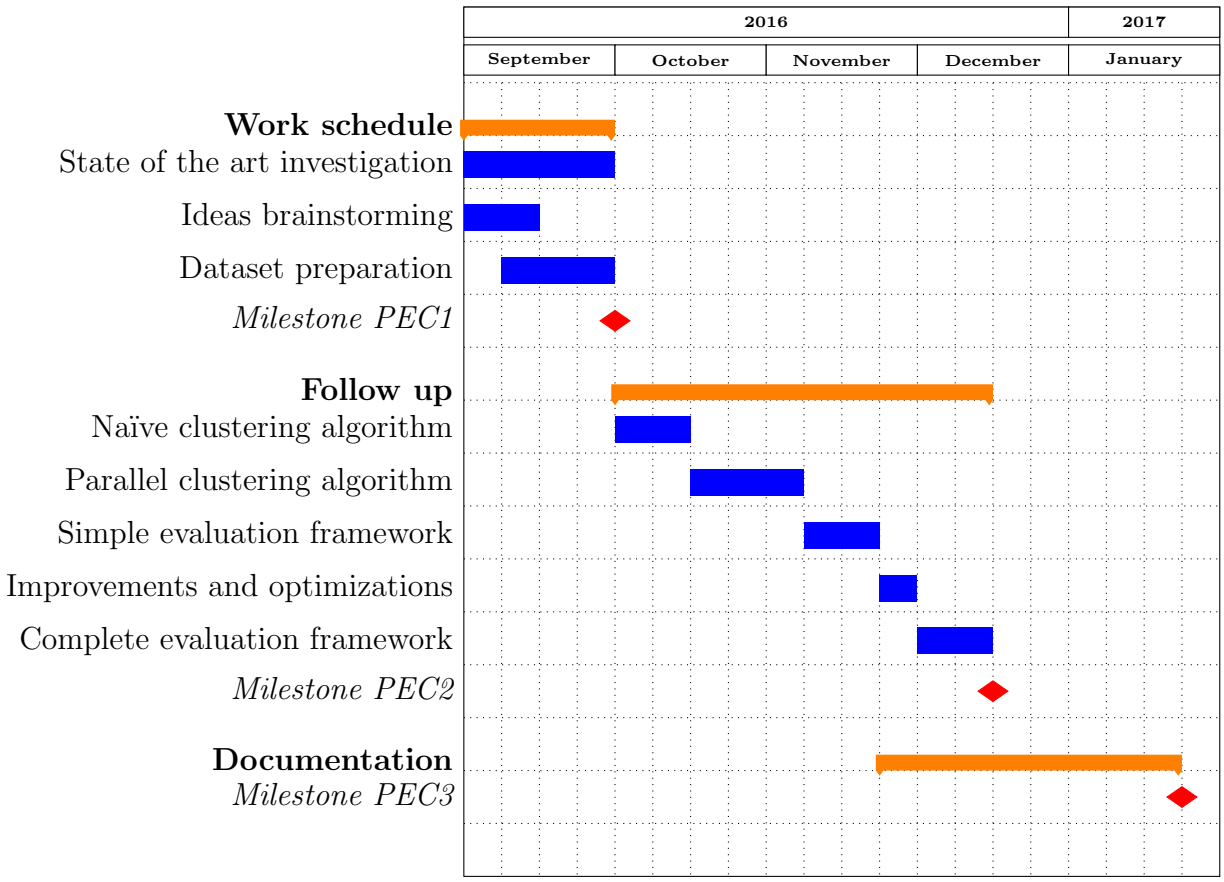
1.4.4 Project Gantt diagram

Figure 1 shows how these tasks where scheduled.

1.5 Product output summary

We have developed a scalable algorithm for k-anonymise a real dataset. It is based on SaNGreeA algorithm, but adapted to focus on the utility for a recommender system.

Figure 1: Gantt diagram for project schedule



We have converted it into a parallel and multi-start biased randomization algorithm. It will be run several times in parallel to calculate different solutions and keep the best one.

We have designed our own evaluation metrics that focus on the potential utility of the clustering output for a recommender system. Finally, we have empirically tested the computing time scalability and quality of the clustering results.

1.6 Sections summary

The rest of the sections are organized as follows. Section 2 overviews the dataset we have available. Section 3 explains the original SaNGreeA algorithm and our own modifications and implementation for both the clustering and evaluation framework. Section 4 describes our evaluation setup and discusses the results. Section 5 concludes the research and points future work directions.

2 Dataset overview

We have available a subset of the users of the full XING database. It is approximately the same size as the RecSys challenge database, with fewer attributes but adding the relationships.

The list of attributes on the dataset are shown in the following table:

Table 1: Dataset attributes

Field name	**Type**	**Average list size**	**Domain size**
User id	Identifier	-	813.000
Job roles	List of categorical values	3.8	11968
Career level	Categorical	-	5
Discipline	Categorical	-	22
Industry	Categorical	-	23
Region	Categorical	-	16
Relationships	List of user ids	155.9	813.000
Entities	List of categorical values	151.7	11000

It is important to note all except one attribute are categorical values, and we do not have available a hierarchy tree for such categorical values. Some of them have also a very big domain size like the list of job roles. Some example of job roles are "Data Scientist", "Software Developer" and "Senior Software Developer". Unfortunately, we do not have available for this dataset a hierarchy to relate these job roles in any way.

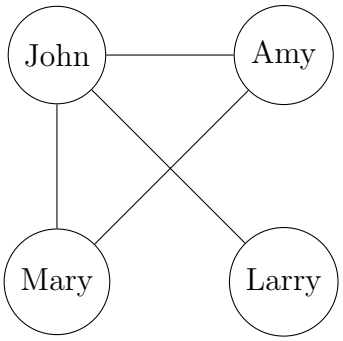
Discipline and industry are fields related, but different. For example, one user could have Human Resources discipline and IT industry values if he is an HR person working for an IT company.

The relationships network is provided as a list of relations for every user. As it is a symmetric relationship, if one user A has a user B in his relationships field, the user B will have the user A in his relationships field as well.

We will represent these relationships as a labeled, undirected graph G. Let $G = (V, E)$ where V is the set of nodes (users) and $E \subseteq V \times V$ the set of bidirectional edges (relations) in G. We use $e_{i,j}$ to define an undirected edge from vertex v_i to v_j. $|V| \approx 813.000$ and $|E| \approx 63M$. Every node v_i is a tuple

H

Figure 2: Small example network



with all the attributes in the above table. Every edge $e_{i,j}$ does not contain any additional information.

The figure 2 shows how an example network of 4 nodes could look. In the example, John user has a vertex degree of 3 (it has 3 relationships) and Larry has vertex degree 1. In our real dataset, the average vertex degree is around 156. These may feel as the network is densely connected, but this is not the case. As there are 813.000 users, the network is really sparse. That means it is very unlikely that two random users are connected.

3 Anonymization algorithm

We need to build clusters of k elements that are similar in attribute level and relationships level. The anonymization algorithm will need to find the clusters that minimise the information loss on these two directions, and ensure every cluster has at least k elements to maintain the desired anonymity level.

3.1 SaNGreeA scalable algorithm

The proposed algorithm is a modification of the classic SaNGreeA (Social Network Greedy Anonymization) algorithm. SaNGreeA is a greedy clustering algorithm that performs a partitioning of all graph nodes by grouping them into clusters of at least k elements.

3.1.1 Original algorithm

Nodes inside the clusters become indistinguishable, as every cluster collapses (generalises) all the attributes of the inner nodes, and all the relationships between two clusters are collapsed into a single undirected edge. The goal of the algorithm is to choose the partitioning such as the information loss on attributes generalisation and relationships is minimised.

It is a greedy algorithm that is driven by two criterion measures: minimisation of generalisation information loss and minimisation of structural information loss. On every iteration step, it adds a new node to the current cluster until it has k of them, and then it creates a new cluster. The node is selected such as it is the one that the most the two criteria described for the current cluster being built.

During the execution of the algorithm, the structural information loss can not be computed as the entire partitioning is not known, so the clustering process is driven by a different measure as a proxy. That means the solution will be a local optima, but not the best solution. Furthermore, the algorithm is deterministic, that means, two executions of the algorithm on the same data input will generate the same partitioning.

3.1.2 Time complexity

SaNGreeA has a quadratic asymptotic runtime complexity $O(N^2)$, where N is the number of nodes. That makes it a bit slow if we have a big number of nodes (1 million), as it would need to do in the order of 1M * 1M operations.

In our case, we will modify the amount of work the algorithm needs to do per iteration, such as in practice for our problem, the number of operations is far less than quadratic.

Concretely, each cluster will ensure all inner nodes have exactly the same discipline, career level and a location which is close enough (same region). That will allow to prebuild these buckets in advance, and do a fast search on runtime by looking only into the nodes in the corresponding bucket, which would be a very small subset of the network. That will reduce considerably the work to be done in each iteration, as long as there are not hotspots (a very few number of buckets with much more elements than the vast majority) and there are more buckets than machines.

This restriction also makes sense from product point of view, since it will ensure a minimum level of quality in the anonymization (minimising the information loss), while still maintaining the k-anonymization property.

3.1.3 Multi-start with biased randomization

In order to improve the clustering quality, we will transform the algorithm into a non-deterministic one. The idea is to apply some random perturbation into the node selection during each iteration step, while still taking into account the two criteria that drive the partitioning (generalisation information loss and structural information loss).

On every iteration, we will not just blindly choose the node that minimises the most the two criteria for the current cluster, but instead we will randomly chose it from the list of the top M candidates, with a probability proportional to their score on the criteria given.

Each full execution of the clustering algorithm will then generate a reasonably good solution, but different every execution. Even if most of the solutions

may have lower quality than the deterministic solution, by running the full clustering algorithm hundreds or thousands of times, we will hopefully escape from local-optima solutions and get some solutions which are actually better with a lower information loss. This will allow us to use a more complex evaluation of the solution scoring, more focused on the quality of the recommendations that we would get with the clustering. We will then just choose the clustering solution that damages the least our utility score.

3.1.4 Parallelization

The original SaNGreeA algorithm is inherently serial, since the work cannot be divided into independent tasks (it keeps adding nodes to the current cluster until it fills up, then it creates a new one). That means, it can not be directly sped up by splitting the work using a parallel framework like Spark or Hadoop.

In our case, we will keep it serial as well. The serial runtime will be just fine to be run in a single machine, since we have reduced its time complexity.

We will achieve linear parallelism from other perspective. As we are running the full algorithm multiple times with slightly different results by the randomization, we can very easily exploit that fact. We will simply run multiple copies of the clustering process in parallel, each with a different random seed. Once we have finished doing hundreds or thousands of full executions divided between the available number of machines, we will then just keep the best result as explained in the previous section.

3.2 SaNGreeA metrics for information loss

The authors of SaNGreeA paper developed a few metrics for the evaluation of the information loss caused by the clusterization for some small networks. There are two levels of generalisation, quasi-identifier attributes generalisation, and edge generalisation.

It is important to remark the networks used in their research are orders of magnitude smaller than the network in our dataset.

Concretely, one example network they use has just 300 nodes, and is really dense with an average vertex degree around 5 to 10. That means, given two nodes, they have a very high probability to be connected (greater than 1 %).

In contrast, our network is extremely sparse. We have around 1 million nodes, and an average degree between 100 and 200. The probability of two random nodes to be connected is tiny in comparison (lower than 0.01 %).

Given this difference in conditions, in this section we explain the metrics they developed for evaluation of their algorithm in very small networks, and why they are not very practical for our use case.

3.2.1 Generalization information loss

When the clustering has finished, we need to summarize the quasi-identifier attributes of all nodes assigned to a cluster. SaNGreeA authors developed two generalisation types depending on the nature of the attributes to be summarized.

- Categorical attributes: those are usually generalised using generalisation hierarchies specific to the domain of the attribute. The cluster node will take the lowest common ancestor in the hierarchy to all the inner nodes.
- Numerical attributes: the cluster node will take a range that goes from the minimum value of the inner nodes up to the maximum value.

None of the two types fits for our dataset. We are using essentially hierarchy-free categorical values, so this metric won't be very appropiate for our application. Even more, some attributes we have are a list of category values like list of job roles of an user. We do not have a clear hierarchy for them in our domain.

3.2.2 Intra-cluster information loss

The k nodes that have been collapsed into a cluster they may originally have some relationships between them. These edges do not point to other clusters, they are internal to the cluster. SaNGreeA represents that information by

adding a new attribute to the cluster which is just the cardinality of the number of intra-edges that have been collapsed.

Their metric for evaluation the amount of information lost by this summarization is called intra-cluster structural information loss. They quantify it as the probability of error when trying to reconstruct the structure of the initial social network from its masked version. The following formula is applied for every cluster cl and then summed over for all clusters.

$$intraSIL(cl) = 2 \cdot |\epsilon_{cl}| \cdot \left(1 - |\epsilon_{cl}| \,/ \binom{|cl|}{2}\right)$$

Their evaluation function intraSIL essentially means that we get zero loss, perfect information for a cluster cl if either:

- none of the nodes in cl share any edge
- all nodes in cl are interconnected between them

For example, for $k = 3$ we can have a maximum of 9 edges inside a cluster. The following shows the information loss function value for all the possible cases:

Number of edges	**intraSIL loss**
0	0.00
1	1.77
2	3.11
3	4.00
4	4.44
5	4.44
6	4.00
7	3.11
8	1.77
9	0.00

As we can see, for the extreme cases (no edges, or all possible edges), reconstructing the original edges is perfect and trivial as we have complete information. On the other hand, we get maximum loss if we have a number of edges that is the half of the pair of nodes in the cluster as there is more uncertainty to which original nodes belong every edge.

However from our application point of view, a good cluster will have users that are similar. We consider similar if their quasi-identifier attributes are

similar, or if they have many relations in common. We think that finding clusters with a few internal edges is far better than having no edges at all.

We do not care so much if the original information can be recovered, if it generates clusters with low usefulness for a recommendation algorithm. Why would we want to have clusters that do not share any edge between them? Even if that means we do not lose information when generalising it is counter-intuitive for our purposes. Given the sparsity of our network, it would be too unrealistic to think we can find many clusters where all nodes are interconnected between them.

SaNGreeA focuses on the damage from reconstruction point of view, but we aim to have a clusterization that retains the semantics of what we think is a good user similarity.

3.2.3 Inter-cluster information loss

This metric represents how much information do we lose if we generalise all the edges that were connected from a node belonging to a cluster cl_1, to a node belonging to a different cluster cl_2. All these nodes that fall between two concrete clusters are summarized to just one edge with a weight value that indicates how many edges are collapsed into this new one.

The interSIL metric they develop is identical to the metric for intra cluster but instead of all the edges that fall inside one cluster, it is evaluated for every pair of clusters cl_i and cl_j and then summed all over.

$$interSIL(cl_i, cl_j) = 2 \cdot \left|\epsilon_{cl_i,cl_j}\right| \cdot (1 - \frac{\epsilon_{cl_i,cl_j}}{|cl_i| \cdot |cl_j|})$$

So given two clusters cl_i and cl_j, the inter-cluster information loss metric has zero loss (a perfect reconstruction is possible) for these cases:

- there are no edges between cl_i and cl_j
- all nodes of cl_i were connected with all nodes of cl_j

We would get maximum loss if we have half the maximum possible edges. The function table is identical to the intraSIL one shown in the previous section. Again, we do not think this is a good metric for our use case. It is

not very different for us if we have two clusters that are totally linked with all possible edges between them or just a good subset of them.

Once there are more than a few edges between two clusters we can consider these two clusters to be heavily linked. That can be used by a recommender system to infer that these two clusters are really related and infer good recommendations for one cluster are potentially good for the other. But for that, it does not really matter too much if they have the maximum number of edges possible between them, just above a certain reasonable threshold. We may even choose that threshold to be around half of the maximum possible number of edges.

It is unrealistic to aim for achieving maximum linkage for a high number of clusters given our very high sparsity. Most of the pairs of clusters will have a very low number of edges crossing them.

Given this, the interSIL metric will essentially try to minimise the loss by either finding clusters that are completely unlinked, or fully linked. Those that fall in the middle are really penalised. As we can not aim to have many pairs of totally linked clusters, the algorithm will try to minimise the error from the other side, so that the clusters do not have too many edges between them. That is totally opposite to what we would like to have from a recommender point of view.

3.3 Our utility score for information loss

As the metrics used for study the original SaNGreeA algorithm are not applicable to our use case, we need to develop our own. We will develop a utility score, which will be used to select which full clusterization from the hundreds we run, is the one that we find the best and selected as final result.

First we need to find a different generalisation function for the quasi-identifier attributes, as we have categorical values without a clear hierarchy, and then a different scoring method for the clusterization output.

3.3.1 Quasi-identifier attributes generalisation

Every cluster represents a set of k nodes. Some quasi-identifier attributes are warrantied to have the same value as all its inner nodes (those used for the bucketing mechanism while clustering), but the rest of them will need to be generalised to represent the inner values while maintaining anonymity.

None of the quasi-identifier attributes in the dataset has a clear hierarchy. Even the job roles is essentially a flat structure on our dataset. Therefore taking into account what would be the best for the utility of the data in a recommender system, the easiest is just to report a list of all the values present in the inner nodes, without repetition.

That means, if we have a cluster covering users A, B, C, with industries respectively I1, I1, I2, the cluster will have as industries attribute the set [I1, I2].

As an advantage, we still know the concrete industries instead of more generic ones, thus avoiding increasing the damage caused by the clustering. If we had chosen a more generic one such that it covers both I1 and I2 (if we had such hierarchy), probably it will also cover other industries that would not be part of the inner nodes in the cluster, thus increasing the data damage unnecessarily.

The downside of this is that we are transforming the attributes that are single value to a list of values. In some applications there may be the need of some transformations (like generating nodes synthetically by pooling randomly from the cluster) to make it usable. But from the recommender application point of view, it is acceptable as it is very easy to incorporate a set as an attribute, and just recommend items doing an union to take into account results covering any of the values.

3.3.2 Relationships generalisation

We will generalise the relationships exactly as described in SaNGreeA paper. All intra cluster edges will be collapsed into just a new cluster attribute with the count value, and all inter cluster edges between two concrete clusters will be collapsed into just one edge with a count attribute.

We will consider the number of relationships in common of the inner nodes of a cluster and the intra edges for our scoring function. We want to obtain the clusters that collapse the most similar users. From a recommender system point of view, it is more important to find clusters with very similar users, than pairs of clusters with a very heavy link between them.

3.3.3 Clusterization scoring function

For every cluster, we will calculate one damage value per quasi-identifier attribute, and one additional one for the relationships, getting a tuple of damage values.

Essentially, for every attribute that have been generalised, the cluster has a set with all the different values present in the cluster. We will set the damage score of that attribute to the cardinality of the set.

Note the minimum score possible is 1, which is the perfect scenario where all the collapsed nodes have exactly the same value, as the resulting set would be just 1 element.

3.3.3.1 Relationships scoring

We will reduce the problem of scoring the relationships into the quasi-identifier attributes scoring one.

Given a cluster cl with users $u_1 \ldots u_k$, intra edges e_{cl} and inter edges e_{cl,cl_i}, we define the set of edges E_{cl} as the union of all intra edges and inter edges of cl.

$$E_{cl} = e_{cl} \ \cup \ e_{cl,cl_i}$$

Next, we define the nodes C_{cl} as all the unique nodes referenced by E_{cl}. That means, any node from the graph (internal or external), where at least there is one edge of cl referencing it.

$$C_{cl} = \{v \in V \mid \exists u \in u_1 \ldots u_k \ such \ that \ e_{u,v} \in E_{cl}\}$$

The damage score for the relationships is then calculated as:

$$damage\ score\ (cl) = \frac{|\{u \mid u \in u_1 ... u_k \cup C_{cl}\}|}{2 \cdot |E_{cl}| \cdot k}$$

Essentially, it is the number of all unique nodes inside the cluster or connected to the cluster (without repetitions), divided by the total number of edge extremes plus the cluster size (with repetitions).

The lower this score, the more shared relationships the cluster has. We can consider this value as the damage score for the relationships attribute, as we want to minimise it like quasi-identifier attributes damage score.

3.3.3.2 Example clustering damage score

For example, for a clusterization where we have a cluster C_1 that collapses the users u_1 and u_2, and the users have two quasi-identifier attributes A and B. If the users have the attribute values:

User	**Attribute A**	**Attribute B**
u_1	[A1, A2]	[B1]
u_2	[A2, A3]	[B1, B2]

Then when the cluster C_1 collapses users u_1 and u_2, we generalise the values by adding each different attribute value into a set. The cluster will then have the attributes:

Cluster	**Attribute A**	**Attribute B**
C_1	[A1, A2, A3]	[B1, B2]

The damage score tuple for cluster C_1 is equal to the cardinality of each attribute set values:

```
A = | [A1, A2, A3] | = 3
```

```
B = | [B1, B2] | = 2
```

So the damage score tuple of the cluster C_1 (dst_1) is equal to:

Cluster	**Damage score tuple**
C_1	(3, 2)

3.3.3.3 Final average damage score

Finally, after we have calculated all damage score tuples $dst_{1...M}$ for all the clusters C_1 to C_M, we sum all the tuples and divide by the number of clusters M. The final tuple would be the average number of different values per attribute for this full clustering execution.

average damage score = $\sum_{i=1}^{M} dst_i/M$

Let's say we have executed one hundred times the clusterization algorithm. For the 45th execution we have get an average damage score value of 2.33 for attribute A, 1.3 for attribute B and 67.4 for relationships. Then we would have a list of average damage score tuples, one per full clusterization execution like:

```
1st execution average damage score = (aaa, bbb, ccc)
```

...

```
45th execution average damage score = (2.33, 1.3, 67.4)
```

...

```
100th execution average damage score = (ddd, eee, fff)
```

3.3.4 Selection of the best clusterization

Once we have finished computing hundred of different clusterizations, and we have calculated their average damage score tuple, we need to choose which one of the clusterizations is the best.

As every clusterization execution has a tuple score, instead of just a number, it is not immediately obvious which one is the best. If it where just a number, we would then choose the minimum one, but we have a tuple instead.

We will calculate a single value final score per clusterization by first normalizing the tuple scores per attribute, and finally multiplying all normalized attributes to get a single value score.

Example: in the following table we show the average damage scores we would have get in an hypotetical case where we have executed the clustering algorithm 3 times:

Clustering execution	Attribute A	Attribute B
Execution 1	3.2	4.5
Execution 2	3.1	6
Execution 3	2.1	4.6

We then normalize the scores by dividing by the maximum damage per column:

Clustering execution	Attribute A	Attribute B
Execution 1	3.2 / 3.2 = 1.00	4.5 / 6 = 0.75
Execution 2	3.1 / 3.2 = 0.97	6 / 6 = 1.00
Execution 3	2.1 / 3.2 = 0.66	4.6 / 6 = 0.77

Finally we just multiply each value of the normalized tuple:

Clustering execution	Final score
Execution 1	1.00 * 0.75 = 0.75
Execution 2	0.97 * 1.00 = 0.97
Execution 3	0.66 * 0.77 = 0.51

The minimum normalized damage score is achieved by the 3rd execution, so that will be selected as the best clustering execution.

3.4 Technical implementation

The anonymization algorithm have been implemented in Apache Spark which is a framework for developing large-scale data processing algorithms.

Essentially every spark program consists of a main driver program that spawns several parallel operations on a cluster of machines. It is similar to the MapReduce framework, but with a higher level abstraction, making easier to write complex workflows.

3.4.1 Parallelism

As explained in previous section, we have exploited the parallelism in Spark by two axis:

1. Partitioning the users so that we can divide and calculate independently different sets of clusters

2. Running multiple independent full clusterization executions

Thanks to that two-level parallelism, we try to ensure the data processing machine nodes almost always have tasks to do that are not too big. That is important to avoid hotspots that will make the algorithm slower, as we would need to wait to the longest task, making some data processing nodes underutilized.

Finding the right amount of data partioning and number of tasks to be run in parallel have been a long trial and error process, as too many tasks create a lot of management overhead by Spark, and having too few means we may face hotspots by very long-running tasks.

3.4.2 Optimizations

We had to make several memory management optimizations in the data processing tasks. One of the issues was that the most trivial data structures were not performant enough, as we quickly run out of memory in the task nodes. We had to use high-performance memory-compact data structures from fastutil framework to make our algorithm fast and memory efficient.

One of the most expensive operations is to calculate the similarity between users. Also this is a very redundant operation, during a single execution we will need the similarity of some user A and B up to k times (when comparing the current cluster being built to all remaining users). Even more, the similarity value between user A and B does not change between clustering algorithm executions as it only depends on their attributes.

3.4.3 Execution steps

The execution steps are the following:

1. Similarity cache
2. $N \approx 100$ clusterization algorithm executions with biased randomization
3. Evaluation of the N clusterization algorithm outputs and selection of the best

For achieving computing efficiency, we have done a similarity cache step before starting any clusterization execution. It is executed just once, reading all user data, grouping them by the clustering bins, and computing all pairs similarity values per bin. That information is then stored in a compact table indexed by pair of users A, B with the similarity value.

Then every clusterization execution just loads that users similarity table as the input. That makes the clusterization algorithm very efficient on execution (the similarity is precomputed) and in memory usage as well, since the only thing it needs to load is such table, removing the need to read any user data which is several orders of magnitude bigger.

Finally, the evaluation needs as input the original users data and the clusterization output, in order to generate the damage scores.

4 Experiments and results

We have tested our algorithm with different parameters on our 830.000 users dataset. We are interested to evaluate how the utility of the k-anonymised dataset is damaged from a recommender system point of view. Also we have done scalability tests for the proposed parallel solution.

4.1 Damage evaluation

We have tested our algorithm with three similarity functions. They are used to drive the clustering greedy algorithm:

- Minimise both attributes and structural information loss
- Minimise only the attributes information loss
- Minimise only the structural information loss

Each of these similarity functions have been tested with different values of k. The number of full clustering executions have been fixed to $N = 100$.

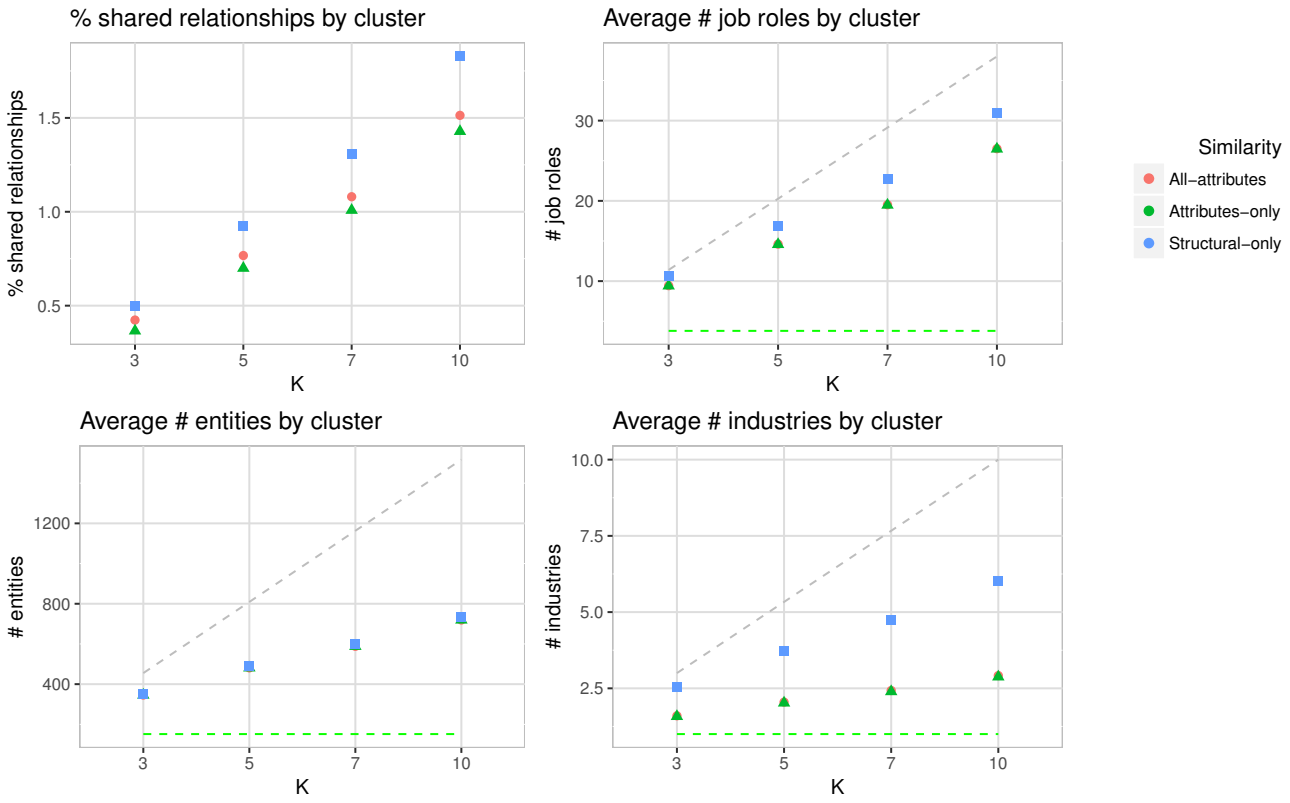
For the quasi-identifier attributes damage evaluation, we have plotted the average number of different values a cluster of size k has. In the ideal case, the average cluster quasi-identifier attribute length will match the average length of a random user (if for every cluster, all users of the cluster have identical values). In the worst case, the average value would be k times the ideal value (none of the clusters have any two users with a shared attribute value).

For example, for the industries attribute, as every user has only 1 value, the ideal case would be to have an average attribute length of 1 for a cluster, independently of k. But the worst case would be equal to k. Analogously, for the job roles attribute, the average length per user is 3.8 (see dataset overview section). Then the best case would be 3.8 independently of k, and worst case would be $3.8 \times k$.

Figure 3 shows the average number of different values for each of the quasi-identifier attributes as cluster size k grows, with a different color for the

similarity function applied. The gray dashed line is the theoretical worst case and the green one is the best case as explained before.

Figure 3: Average damage scores per cluster



We can clearly see the effect of the different similarity functions. The *structural-only* similarity function works well for increasing the shared relations, but not so well for the minimisation of the quasi-identifier attributes, except for the entities case which will be analyzed later.

The *attributes-only* and *all-attributes* similarity functions are nearly identical, except for the first graph, the shared relations. Obviously the *attributes-only* function is the worst performing one, as it does not take into account the relationships for building the clusters.

We can say the *all-attributes* similarity function is performing specially well

for the information loss minimisation, and still improving a bit the structural information loss (relationships damage). That is good news, since we want to minimise the damage score in both.

It is important to note the effect on the entities quasi-identifier attribute. It is a very sparse attribute as an average user has a list of 151.7 different values from a domain of 10.000 different entities. As all similarity functions have the same effect for it, it does not matter if we take it into account for building the clusters. We can conclude this attribute will be heavily damaged, and will be very unreliable to be used for driving any recommendation from the clusters.

The industries attribute is arguably the most undamaged one. For the *all-attributes* and *attributes-only* similarity functions, as k grows, the number of different industries barely grows, it is almost flat, and quite close to the theoretical minimum of 1 (ideal case, green dashed line). This quasi-identifier attribute is specially important for recommendations, so it is highly desirable to have the average size as small as possible.

Please remember there are 3 other quasi-identifier attributes which do not have any damage at all and do not need any evaluation: disciplines, career level and region. During the clustering process, we have guaranteed all users inside a cluster have the same value for these 3 attributes by partitioning the users beforehand.

We are able to just share around 1 % of the relationships inside a cluster. This is expected, as our network is really big and sparse. This is not as bad as it seems, as we can say two users are strongly connected when they share just a few (maybe a handful) of contacts.

4.2 Damage minimisation over N

We have evaluated how much the solution improves running the clustering algorithm N times. We are interested to see how the "best so far" solution damage score decreases as we are finding better solutions until we have completed the clustering stage by calculating N different solutions.

For this analysis, we have run the full clustering process for $N = 200$ and $K = 5$. The similarity function is *all-attributes*.

Figure 4: Damage minimisation over N

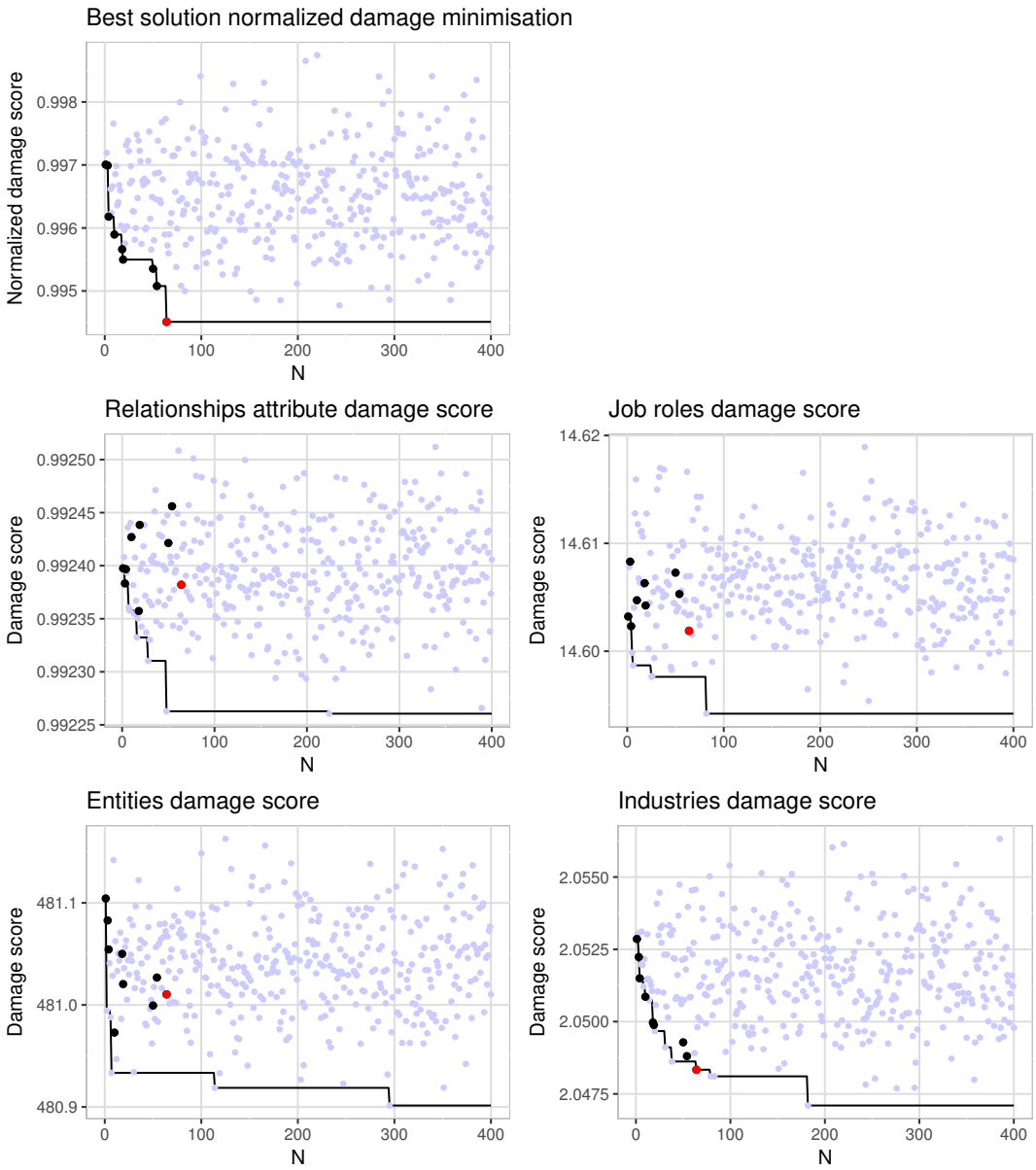


Figure 4 shows in blue dots the score value for each of the N clustering solutions. We have separated into a different chart per attribute, and a

leading one with the final solution score combining all attributes.

The black line is the trailing minimum score found so far for the specific attribute being represented. The first plot is the one that has the real function we are minimising and selects the "best solutions so far", those are marked as black dots in the other graphs. The absolute best solution with minimum score is marked as a red dot in all graphs.

Note how the best solution (marked in red in all plots) is found quickly after less than 100 iterations. The best solution coincides that it is one of the minimums of the industries attribute (follows its black line), but it is not a minimum in the other attributes, they are about in the middle of their range of solutions. It is not even the absolute minimum of the industries attribute, as an even lower value is found after much later (N approaching 200).

Since the best solution is the one that combining all attribute damage scores has the lowest value, it should be competent in all attributes at the same time. That means having a very low value for industries, but too high for the other attributes is not desirable. That is the reason why it sacrifices a bit of damage for industries for a bigger decrease of damage in the other attributes.

Nevertheless, we are able to adjust the importance of any attribute by applying higher weight when combining all the scores. That allows for fine tuning the search, if any of the attributes is specially important for the utility of the anonymized dataset.

4.3 Scalability

We want to evaluate how our algorithm scales as we increase our computing power. As our algorithm requires many full executions to improve the clustering results, we want to be able to run a large number of executions N and shorten the wall clock computing time by adding more computing nodes.

When we increment the number of executions N, if we increment the computing power by the same relative amount, in the ideal case we want it to finish in about the same time. If an algorithm has such ideal property, it is said the algorithm has perfect linear scalability.

4.3.1 Processing stages

As stated before, our full process has three stages that are executed sequentially with a different need of resources.

1. Similarity cache: compares all pairs of users of the same bin. It does not depend on N neither on k
2. N clusterization algorithm executions with biased randomization. We have set 40 computing tasks by default.
3. Evaluation of the N clusterization algorithm outputs and selection of the best.

As the first stage is fixed in size and complexity, it is independent of the number of clustering executions N and k. However, the second and third stages depend on k and N, so these are the ones we want to scale well. The target is to be able to run as many full clustering executions as we want, and keep adding more computing nodes in order to finish the full process in a reasonable total wall clock time.

We focus our analysis in the clustering stage which is the one that really requires high computing power. The evaluation is really lightweight in comparison. In our tests we have seen the time required for evaluating the results is mainly constant, with a very small increase in time as N increases.

Figure 5 and table 2 show the total wall clock time required to compute N full clusterizations and how much it decreases as we run it in more computing nodes in parallel.

Figure 5: Full clustering stage time

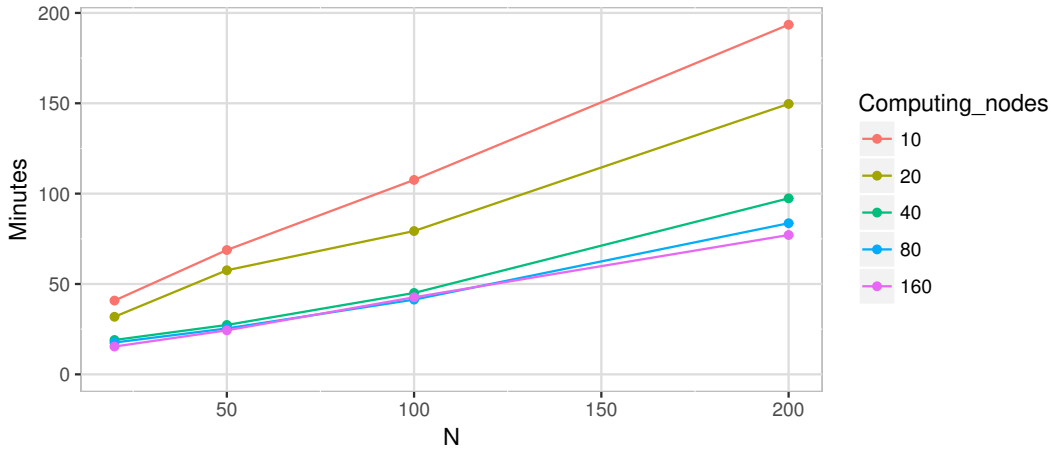


Table 2: Full clustering computing time (minutes)

	Computing nodes				
N	**10**	**20**	**40**	**80**	**160**
20	40.8	31.9	19.0	17.6	15.4
50	68.8	57.6	27.3	25.4	24.4
100	107.6	79.3	45.0	41.4	42.7
200	193.5	149.7	97.4	83.6	77.1

We can see it is not perfect linear scalability as if we double the number of nodes, we do not finish the same amount of work in half the time. In the table 3 we calculate relative computing time per row relative to 10 computing nodes (first column).

It scales up until we have between 40 and 80 nodes, then adding more nodes does not significatively decrease the total time. That is because we need to have much more work to do than nodes, so that we can ensure all of them are busy most of the time.

Another factor is the overhead work of data loading. Our clustering algorithm should first load the similarities table from the cache. Even that the table is precomputed, this table is quite big and needs some minutes for its loading.

Table 3: Percentage of time relative to 10 computing nodes

	Computing nodes				
N	**10**	**20**	**40**	**80**	**160**
20	100%	78%	47%	43%	38%
50	100%	84%	40%	37%	35%
100	100%	74%	42%	38%	40%
200	100%	77%	50%	43%	40%

This loading is done in serial, and it can not be reduced by adding more tasks.

There is also a communication overhead cost. Adding more nodes requires more data movement. If there is not enough work to do to keep the nodes busy, it would be a waste of time to move data into them. It can actually be finished earlier in half nodes if they have full usage, rather than double nodes with half usage.

For our dataset and $N = 200$, we claim it is acceptable to use 40 nodes to finish the clustering in less than two hours. There is no need to actually use more nodes as practically if a company needs daily generation can leave that computing process during night when there are usually other batch processes running. Even more, if a dataset is going to be published, it can be just a one time thing, so two hours is really not of an issue.

Furthermore, if we have the need for higher accuracy figures, we can increase N over 1000 and then we will see the benefit of adding more nodes to the clustering process.

5 Conclusions

We have successfully developed a k-anonymized clustering based algorithm that can scale to process almost a million users and can be run over tens of computing nodes in parallel. It can be executed hundreds of times to keep decreasing the damage caused by the clusterization. It is able to take into account both the structure of the network and the nodes attributes to drive the clustering greedy algorithm.

We are happy with the evaluation results. We have learned which attributes of the clustering output will retain their utility for a recommender system, and which ones should not be used as they are heavily damaged, which is expected by their own sparse nature. We have shown how the solution improves as we keep running more and more iterations of the full clustering algorithm and finding new local optima solutions.

We were able to develop our algorithm in a nice big data computing framework (Spark) which is one of the references in the field and use it to process a real world dataset which is big enough to be a challenge. There is not many research on clustering-based k-anonymized algorithms where a big dataset is processed as usually they focus on networks of less than a thousand nodes.

Initially we planned to develop a recommender algorithm alongside the clustering algorithm. Quickly afterwards we realised that would be too much work, and we left it out of scope. A recommender algorithm can be a full research project by itself easily. Nevertheless, the spirit to focus on the utility to a recommender system remains, as the clustering algorithm decisions and evaluation framework is heavily focused into real world needs for a recommender system.

5.1 Future directions

5.1.1 Recommender systems effect with k-anonymized data

One obvious future direction can be to further analyze the effect of the k-anonymized clusters into a real world recommender. That would mean to

use the clustered data for generating recommendations and evaluate how much those recommendations degrade in comparison of recommendations generated from the original dataset.

Recommender systems evaluation is hard. There are essentially two types of recommendations evaluation:

1. Offline evaluation: compare the recommendation system results against some truth dataset.
2. Online evaluation: compare how well the new recommendations work by checking real user behavior when they face them.

With online evaluation we are able to know if the utility of the recommendations have really improved or not. The downside is that it requires to deploy the new recommender system into some production environment and drive users into it. That can be risky if the new recommender system is very experimental and can do detrimental damage to the business if the new system does not performs well enough. However, the results are the most realistic ones.

The offline evaluation is a bit harder to be accurate. We can check how the new recommendations output is, their distribution, some key metrics, even compare them against some truth dataset without the need to deploy the system to production.

Let's say we have an online store and a recommender system A that never recommended any book product. Then the new recommender system B is able to recommend all products A was able, and additionally books. If we check offline the accuracy of this new recommender, it will have products never reported before by the previous version. How can we then compare against a truth dataset? We may need to manually enter good matches for some hand-made user profiles which can be time consuming, and we will never be able to think on all possibilities. Even more, are these books recommended what the user really wants? Selling these books will be more profitable than other products? How can we then develop a truth dataset to compare with? The utility of a recommender system is not only driven by the accuracy of the matches, but also from a business perspective. For this case maybe online evaluation is more appropiate.

5.1.2 Other k-anonymization algorithms

Another future research direction can be to apply our evaluation framework to K.Santhi et al. k-anonymization algorithm [13]. Their algorithm already has the same multi-start properties and multiple solutions generation, so it would mean to develop a scalable version that can handle our dataset size, generate the same set of metrics and compare the results.

6 Glossary

- XING [1]: a social network for business.
- RecSys Challenge 2016 [2]: it is a research challenge organized by XING and CrowdRec [3].
- Quasi-identifier attributes: attributes that are not by themselves identifiers, but they are correlated with a series of entities and combined can be used to identify some of the entities individually.
- k-anonymity: it is a model of data protection. The concept was introduced in 2002 [5]. Essentially, this model asserts that any individual in the dataset can not be distinguished from at least other k - 1 individual from the dataset in terms of quasi-identifier attributes values.
- Clustering: it consists of grouping a set of objects such that objects in the same group are more similar than those in other groups.
- SaNGreeA [11]: a greedy and deterministic clustering algorithm for achieving k-anonymous clusters on a labeled, undirected graph. It is nowadays a classic and leading work in clustering based k-anonymity algorithms.
- NP-complete problems: a set of problems for which we do not know yet any polinomial time algorithm that can solve it with exact solution. They are usually solved with heuristic algorithms that can get an approximation in polynomial time.

7 References

[1] XING `http://www.xing.com` 2016

[2] RecSys `https://recsys.acm.org/recsys16/challenge/` 2016

[3] CrowdRec `http://crowdrec.eu` 2016

[4] P. Samarati. *Protecting respondents identities in microdata release*. 2001

[5] L. Sweeney. *k-anonymity: a model for protecting privacy*. 2002

[6] K. Liu and E. Terzi. *Towards identity anonymization on graphs*. 2008

[7] S. Chester, B. M. Kapron, G. Ramesh, G. Srivastava, A. Thomo, S. Venkatesh. *Why Waldo befriended the dummy? k-Anonymization of social networks with pseudo-nodes*. 2013

[8] Michael Hay, Gerome Miklau, David Jensen, Philipp Weis, and Siddharth Srivastava. *Anonymizing Social Networks*. 2007

[9] Jordi Casas-Roma, François Rousseau. *Community-Preserving Generalization of Social Networks*. 2015

[10] Julián Salas. *Sampling and merging for graph anonymization*. 2016

[11] Alina Campan, Traian Marius Truta. *Data and Structural k-Anonymity in Social Networks*. 2009

[12] Angel A. Juan, Javier Faulin, Albert Ferrer, Helena R. Lourenço, Barry Barrios. *MIRHA: multi-start biased randomization of heuristics with adaptive local search for solving non-smooth routing problems*. 2011

[13] K.Santhi, N. Sai Lohitha. *Anonymization Of Social Networks By A Novel Clustering Algorithm*. 2014

[14] Tamir Tassa, Dror J. Cohen *Anonymization of Centralized and Distributed Social Networks by Sequential Clustering*. 2013