Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Data Base Charter

The quality of compression on bipartite graphs using dedensification algorithms

Bachelor Thesis

Kacper Sawicz

Supervisors:
dr. Nikolay Yakovets

Final version

Eindhoven, July 2020

Abstract

Continuously growing data stimulates the development of software and algorithms. One of the biggest issues when it comes to graph data is the running time of reachability queries. To optimize this process we can work on input, with dedensification algorithms, or directly on algorithms to develop them even further. The meta-studies combining those two have not been done before. In this study the optimization of the dedensification algorithms will be performed, giving a head start for future researchers working on this topic. The two algorithms, that are evaluated in this paper are the Partition algorithm (Feder and Motwani, 1995) and the Dedensification algorithm (Maccioni and Abadi, 2016). First of them is using the neighborhood trees to find a delta clique of a graph. The second one is based on the simple principle of combinations throughout the high degree nodes set. Those algorithms are run on 19 dependent bipartite graph datasets, to see the compression and optimize its parameters. The Partition algorithm based on the strong theoretical setup does not perform up to the expectations, for 17 out of 19 datasets the algorithm does not even attempt to find the clique. It is caused by the k-value formula used in the algorithm, which is based on the notion of the U and V set being similar sizes, which doesn't happen in real data that often. The Dedensification algorithm provides us with strong compression and shows the space for further improvement with optimizing its parameter. Based on the explanatory data, with the clustering technique, 5 main groups of graphs are discovered. For each one of them, a certain recommendation is provided, which will allow for having a strong choice of the parameter T, without optimization of this parameter, based on the graph.

Preface

I would like to thank dr. Nikolay Yakovets for supervision during my bachelor project, his great enthusiasm and comments were crucial for the progress of this work. I would also like to thank my parents and friends for all the support I received, I would never achieve it without you.

Contents

Chapter 1

Introduction

1.1 Context and Topic

A vast accumulation of data is currently the main principle of development for all scientific domains. As the study (Stevenson, Kording 2011) shows, we are not able to process the data with our machines, since the development of them is slower than the pace of the growing databases. To be able to fully use this newly introduced good, we need to improve the methods of saving and processing the data. In the time of popularized social media, companies of the enormous sizes, and the development of bio-medicine, handling the graph databases is the priority for the world's biggest actors and can have a significant influence on people's everyday life. For this reason, one of the rapidly growing domain at the moment is the graph processing. The main issues the researchers are facing in this field are the sizes of the graphs (problems with storing) and running the queries (the length of processing given the query). For storing the graphs the researchers are presenting the dedensification algorithms, which might by lossless (preserve all information from the original graph) or the approximating approach which preserves most of the graph information. Those algorithms are used to decrease the size of the storage needed to store the graph without the need for further development of the machines we are working on. Moreover, the compression might have a significant influence on speeding up the running times of vertex connectivity, edge connectivity, and all-pairs shortest paths. The problem that we have to solve when it comes to such a compression, is the partition into the bipartite cliques' problem. Its goal is to minimize the sum of the orders of the cliques, where the order is the number of vertices in them, by partitioning the edges of the graph G (Orlin, 77). This problem is proven (Orlin, 77) to be NP-complete.

1.2 State of the Art

Dedensification algorithms are being created for different types of graphs. One of them is a bipartite graph which is characterized by having two disjoint sets of nodes called U and V, from which one set is the origin for every edge and the other one is the destination of every edge. The first algorithm created for dedensification is quite old, taking into account the recent development of the domain (Feder and Motwani, 1995). This paper is the first answer to the NP-complete problem of finding the minimum total order of the cliques, stated by Orlin. Paper by Feder and Motwani presents a way of proving the existence of the partition of the small total order in a sufficiently dense graph and also describes the algorithm for compression, after finding this clique (called delta-clique in the paper). Because it was written as a guideline for the future researchers, with the main concept of compression based on the Orlin findings, the paper had to be highly mathematical with multiple statements proven. To focus on the algorithmic part, the creators of the 1995 paper decided to write it based on bipartite graphs. This is the case since the mathematical proving of the concepts and formulas creation is easier in the bipartite graph setting. Nonetheless, the formulas used in the paper can be generalized to the directed acyclic

graph (DAG). Moving 20 years forward the state of the art algorithm was created (Maccioni and Abady, 2016). It contains far easier concepts than the algorithm mentioned previously since the notion of the paper is no longer to describe to Orlin's phenomenon but to solve the issue by an algorithm. In oppose to solving mathematical problems and writing an algorithm based on that, the focus is shifted to the core of the problem and looking for an easy solution. The Dedensification algorithm (as it is called in the Maccioni and Abady paper) is written for DAGs, which means we can also use it for bipartite graphs. Both of the algorithms have parameters, with which we can optimize the quality of the compression. While shortening the time of running reachability queries is the biggest concern of the graph processing researchers, most of them tend to focus on the optimization of the algorithms on which those queries are running. Those algorithms are being regularly improved (Su, Zhu, Wei, and Yu, 2017) by using different ideas and upgrading the existing ones, thus there exist meta-studies that are focused on comparing those algorithms on various datasets. Combining the research from the dedensification domain with the algorithms for reachability queries to see how dedensification is improving the time of running the queries has not been done before.

1.3 Research Question

As creating the meta-study by properly constructing multiple algorithms for dedensification and reachability queries is a work for multiple researchers, in this study I would like to provide a good start for the future work in that direction. I focus on the dedensification algorithms part with optimizing their parameters given a bipartite graph as an input. For this purpose, I re-create the dedensification algorithms from the pseudo-codes given in the papers by their creators, and based on the results I try to make general patterns for the parameters. The Algorithms I will re-create are Feder and Motwani algorithm (Partition) and Maccioni and Abadi algorithm (Dedensification). Both of them have parameters, that are working in different ways. The influence of those parameters as well as the description of the algorithms can be found in the second section of this paper. The study will use 19 dependent bipartite graphs of various sizes and densities, which are the results of querying the Yago2 (KDE Group, 2020) dataset. Since the graphs are with various sizes, densities, and U to V ratio, the results can be partially generalized and useful for other kinds of data, that other researchers can use to further solve the problem of storing the graph data and decreasing the time of reachability queries.

1.4 Method or Approach

The method section is divided into two parts, since the work is done on two separate algorithms which does not have that much in common, besides striving for the same result. Both algorithms are re-created in Python, which introduces several limitations (i.e. problems with using pointers to the node of the tree), thus the running times of those algorithms may diverge from the running time described in formulas from the papers they originated, meaning that the running times of the algorithms will not be evaluated.
For the Feder and Motwani algorithm, 3 separate algorithms have to be created. First I will create Neighbourhood Trees algorithm, which created neighborhood trees for every node in the set U (the origin set of nodes) of graph G. Next, Clique Stripping algorithm, which will look for the delta-clique (for the explanation refer to section 2 with preliminaries) and correctly update the neighborhood trees as the search produces more results. The last one called Partition is combining the two mentioned previously and producing the cliques from the graph. Given the 19 datasets, I will look on the size of the compression, how changing the parameter delta is affecting the results and with the complete understanding of how the algorithm is working, I will explain it performance.
For the Maccioni and Abadi algorithm, 2 algorithms have to be created. The first is for the correct input, as it requires the division for low/high degree nodes (described in section 2). The second

one is the Dedensification, which produced the new graph with additional nodes and fewer edges. As previously, given the datasets I will look into the compression and how the change of parameter T influences it.

1.5 Findings

Results

For the Partition algorithm, only 2 out of 17 algorithms achieved the k-value (explanation in section 3.1) higher or equal 2. One of the two datasets achieved $k = 2$ and another one achieved $k = 4$.
For the Dedensification algorithm, the perfect T was found for all of the datasets (table in section 4). All of the sizes of the graphs were partially decreased (table in section 4) and the explanatory data of each graph was described using densities and ratios to provide the foundation for the interpretation.

Interpretation

To run the Partition, we need a minimum $k = 2$ (explanation in section 3.1), this means that the algorithm does not provide any results for most of the graphs. For the 2 graphs for which it can run, all the cliques that are being found, given an (α, β) clique, have β bounded on the size of k. By only including those cases we are significantly decreasing the scope of the cliques we are looking for, thus not providing sufficient results. Given, that during k-value calculation the δ (explanation in section 4) is set to maximum, and the value of k is proportional to δ, optimizing it does not make sense.
The Dedensification algorithm provides us with a simple tool to significantly decrease the sizes of the graphs. Based on the 19 graphs used in this study, a decision tree to choose a value of T is created, based on the graph density ($|V|/|E|$, where E is the set of edges), nodes ratio ($|V|/|U|$) and the graph of the density for the single nodes of V.

Chapter 2

Preliminaries

2.1 Dedensification/Compression process

Before further introduction to the algorithms, it is important to understand the mechanism of the dedensification. The explanation given here is combined together with the correct input for the Maccioni and Motwani algorithm, showcasing the low and high degree nodes in addition to the dedensification process. The first picture is a graph with six nodes in set U and three nodes in set V. All nodes from set U are connected to all nodes from set V, forming a clique. In the current setup, we have $3 * 6 = 18$ edges in our graph.

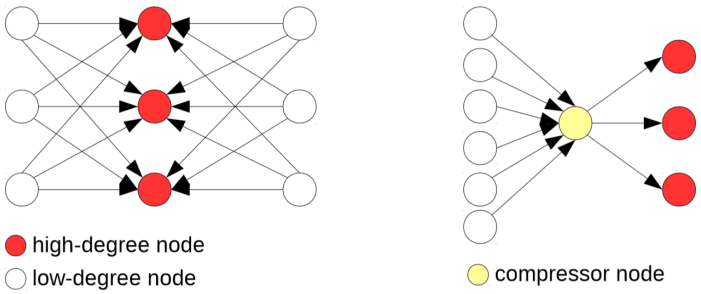


Figure 2.1: On the left a graph G is presented, on the right we can see the same graph after dedensification, as shown in Maccioni and Abadi 2016

The second picture is the graph after the dedensification. For every clique in a graph, we would like to introduce a new node, which will combine the connections and lower the number of edges. Since in our graph we have only one clique, only one new node is needed. After the dedensification, we have $6 + 3 = 9$ edges which decrease the size of the edge set by half. Those differences are growing in a quadratic manner with the sizes of the cliques, so it is in our great interest to find the biggest cliques possible in a graph.
Both of the algorithms are looking for those cliques in different ways, the way of re-writing the graph by introducing new nodes for cliques is used exactly in the same way for both of those graphs. It is important to acknowledge, that this method was not introduced in the Feder and Motwani paper, their expertise finished on the algorithm finding the cliques, but to show the differences between the algorithms reliably, I will use this compression technique for this algorithm as well.

2.2 Bi-clique and clique partition

A bipartite clique of a bipartite graph G is a complete subgraph of G. A clique partition of G, is a collection of bipartite cliques of G, such that edge sets of the bi-cliques form a partition of the edge set of the bipartite graph G.

2.3 Delta-clique

The last preliminary is the notion of the delta-clique used in the Feder and Abadi paper. With the use of Orlin findings, the formula for the clique of the maximum size, in the worst-case scenario, was created, to justify the work of the Partition algorithm. Every time Partition returns the clique, the number of edges is being adjusted together with k-value, which is dependent on them, which results in the finding of the maximum clique by the definition of the formulas, on every iteration of the algorithm.

2.4 Graph data description

In this paper YAGO2 is the graph database, on which everything is run on. On the website of the distributor we can read, that it is an extension of the YAGO knowledge base with focus on temporal and spatial knowledge. It is automatically built from Wikipedia, GeoNames, and WordNet, and contains nearly 10 million entities and events, as well as 80 million facts representing general world knowledge. An enhanced data representation introduces time and location as first-class citizens in the database. The wealth of spatio-temporal information in YAGO can be explored either graphically or through a special time- and space-aware query language.
The 19 graphs, that are used in this study are a result of the 19 queries run on YAGO2. The detail description of the queries used, can be find in the Figure A.1 and A.2 in the appendix. In this table you can find a regex of the graph edge labels for each of the graphs. All queries are of the form '?s, (isLocatedIn/owns/created)+, ?t' with '?s' and '?t' being source and the target variables respectively and the regex of the graph edge labels of the form '(isLocatedIn/owns/created)+' (query for the graph 1 shown in Figure A.1).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
nodes in set U	89888	639	567	70	1282	1521	269	114	100	130	124	257	73	144	59	14514	1883	361	316
nodes in set V	16	123	83	132	157	374	152	119	329	189	93	184	61	1067	112	1481	909	80	3221
amount of edges	169796	17857	5331	573	1659	5251	596	330	578	5439	2211	313	164	13000	185	327128	100554	3050	421157

Table 2.1: The descriptive statistics about the sizes of the datasets used in this study

Chapter 3

Algorithms

The core of the research on the dedensification algorithms is understanding how they are built, and what do they produce. In this chapter, the step by step explanation of both of the algorithms is presented together with the examples. This will give insights for the interpretations of the results and the role of the parameters in both of the algorithms.

3.1 Partition Algorithm

The Feder and Motwani algorithm is the first algorithm produced in the dedensification domain. It was made in 1995 and the main purpose of it was to show the direction of future development to the researchers. The mathematical tools used in it were aiming for the full mathematical explanation of the finding the large clique's phenomenon based on the number of nodes and density of the bipartite graph. Since the algorithm is based on the notion, that given a certain input, we can always find a delta clique, the mathematical proof is created for the maximum delta clique, that always exists in such a graph. The main theory used in the prove is pigeon-hole theorem, which because of its simplicity, gives fairly weak compression, even in the theoretical setting. It is crucial to understand, that we are looking at the algorithm which creates the compression for the worst-case scenario. Not only it means that in theory, it will produce poor results, but also that it is not scalable, since the bigger graph gets, the more cliques there are to optimize. To work not only with a graph which will illustrate well the work of the algorithm but also with a sufficiently dense graph, which is the prerequisite for the Partition algorithm to run (under the k-value computation), the explanation will be based on the graph G in Figure 3.1. It is important to note, that two different graphs are used to showcase the running for two algorithms described in the paper. It is the case because running them on similar graphs would not be as informative, as it is with the currently chosen graphs.

Neighborhood Trees is the first algorithm we need to produce, as it is required in an input. It creates a binomial tree for every node in set U (the origin of all edges) of height $log_2|V|$. Every tree represents the number of connections of the single node from set U to the subsets of the set V (the destination nodes). We start with a full V at the root, taking a left half of V on the left node below, right half on the right node below, and so on until a single tree node will represent a single node from set V, as shown in Figure 3.2. For a graph G, represented in Figure 3.1, those trees would look like the tree presented in Figure 3.3 and 3.4. To decrease the space needed for the operations on the trees, we encode the path as omega, which will work as follow: the root is encoded as omega = [] and for every move in the tree (left or right downwards) we add a 0 to the path for the left direction turn and a one to omega for the right direction turn. An example of such a pathing is shown in Figure 3.5. With this as a prerequisite, we can move to the next algorithm.

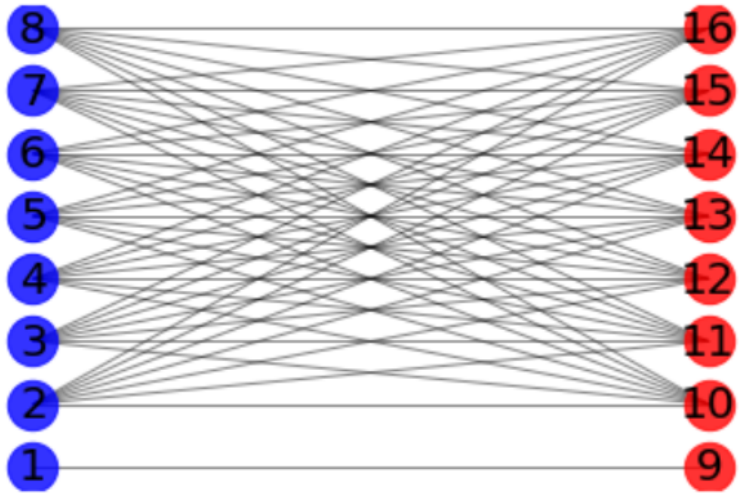


Figure 3.1: Bi-partite example graph G for showcasing neighbourhood trees usage

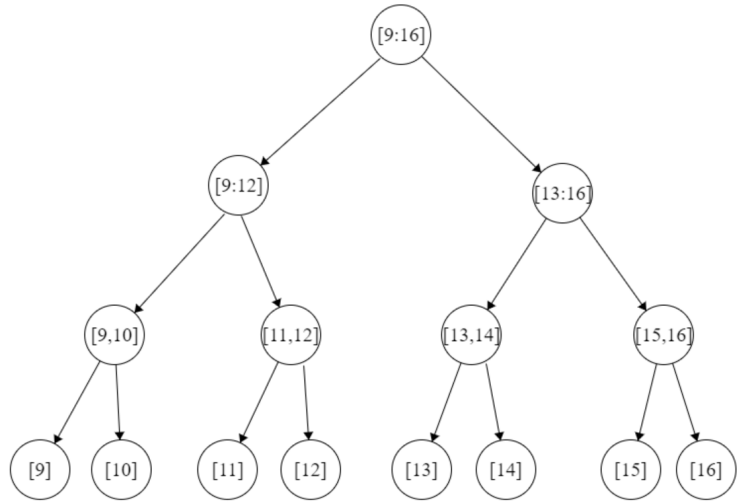


Figure 3.2: Degree tree for the graph G, for nodes 1-8, showing the subsets of V connected to each node

Clique Stripping algorithm is made to find the biggest delta clique in the graph G. We find the clique by using the formula shown in Figure 3.6 step 3.2, where the bold part with $(d_i–1)^{(k-t)}$ is the newton binomial of $(d_i–1)$ and $(k-t)$. This formula and the performance of clique stripping is highly connected to the k-value, which is calculated with the formula $k(n,m,\delta) = \lfloor \delta * log_2 n / log_2(2 * n^2 * m)) \rfloor$, with $n = |V|$ and $m = |E|$. It helps in determining the left and right turns, when it comes to searching through the Neighbourhood Trees for a node of set V, which is a part of delta-clique and it terminates the loop on which the size of the delta-clique is dependent on, as shown in Figure 3.6. The size of k does not go linearly with the density, the size of the graph works for its advantage, meaning that the more nodes we are getting, keeping the density constant, the higher k value we will receive. Not having a sufficiently high value of k, in the worst-case scenario immediately terminates the loop (for $k = 0$ and $k = 1$) or

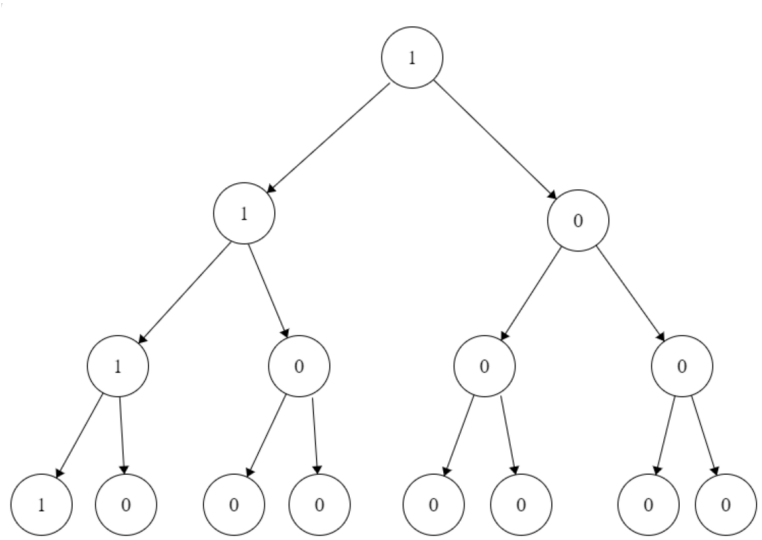


Figure 3.3: Degree tree for the graph G for node 1

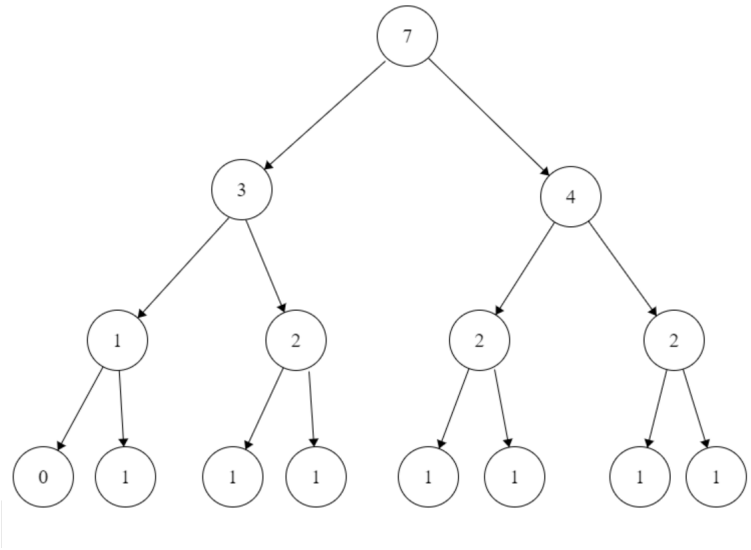


Figure 3.4: Degree trees for the graph G, for nodes 2-8 (all are the same)

finds meaningless or almost meaningless cliques which are not improving the running time (for $k = 2$). By the structure of the formula for k, it is tough to achieve the high k-values exceeding 10. The main issue is, that the formula works always for the worst-case scenario and thus it has a huge error margin in case of big graphs. As mentioned before, the given formula does not work for the smaller graphs, which creates a situation in which we can use this algorithm only on big graphs, since it would not run otherwise. On the other hand for those graphs, the improvement is minimized which is simply not enough for the desired outcome. On top of those problems, there is also a problem with finding big graphs with such high densities required by the algorithm to run, which is decreasing the usefulness of it to even narrower margin. For the optimization purposes, the creators of the algorithm are including a parameter δ in the formula for the k-value. The δ

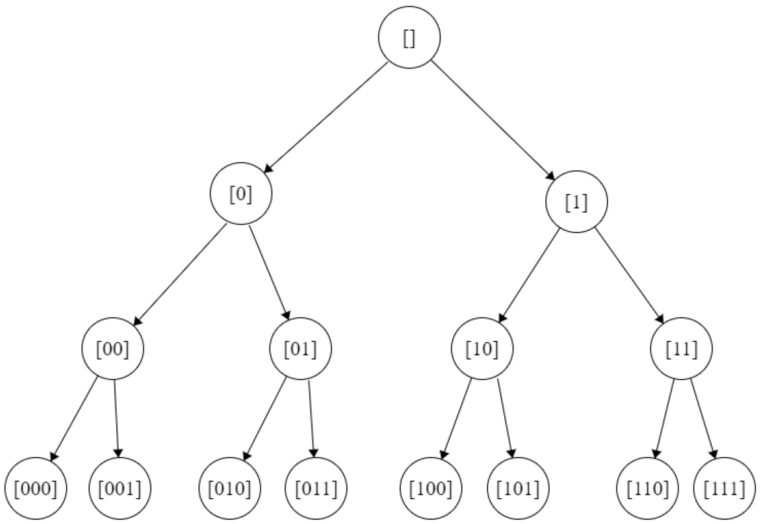


Figure 3.5: Degree tree for the graph G, for nodes 1-8, showing the encoded path to each node

can take the numbers from range 0 to 1 including those numbers. The whole formula is multiplied by delta, so the maximum k-value is always being achieved for $\delta = 1$. This parameter can be used to achieve more symmetric delta-cliques, instead of highly skewed ones to one of the sides. Since every decrease of δ is decreasing our k-value, we need to achieve a high k, to start thinking about the optimization for this algorithm.

The Partition Algorithm combines the previously mentioned algorithms as well as puts additional constraints on the termination of the algorithm. Since the main loop is inversely proportional to the size of δ, as shown in Figure 3.7, the lower δ parameter is, the more cliques will be found. As mentioned in the previous paragraph, the delta is also affecting the size of the k-value and symmetry of the delta-cliques. This means that the bigger δ is, the less amount of cliques will be found, but they are going to be of a larger size and if the δ is smaller we will find more cliques but smaller ones. While the delta-cliques are being stripped from the graph G, we continuously update the graph, the Neighbourhood trees, and the value of k. The last one is made to ensure, that we are always looking for the biggest delta-clique in the graph, thus after finding the delta-clique, we are removing those edges from the graph, adjust the degrees in Neighbourhood trees and after changing the k, we can look for another biggest delta-clique, until the k-value will drop to a threshold, which will not allow us to run the algorithm further ($k < 2$). In this algorithm the method of adding nodes and decreasing the number of edges is not implemented, thus I had to use the method from the Maccioni and Abadi algorithm to have the compression clear for the comparison when it comes to comparing the results with the initial state as well as using another algorithm.

CLIQUE STRIPPING ALGORITHM.

Step 1. Initialize a pointer to the root of each neighborhood tree T_i.

Step 2. $t \leftarrow 1$; $U_t \leftarrow U$; $\mathscr{B}_t \leftarrow \mathscr{B}$.

Step 3. Perform stage t of the binary search.

Step 3.1. $w \leftarrow \varepsilon$;

Step 3.2. Compute c_0 and c_1 using the pointers to node w in each T_i:

$$c_0 \leftarrow C(U_t, \mathscr{C}_{t, w \cdot 0}) = \sum_{u_i \in U_t} d_{i, w \cdot 0} \cdot (d_i - 1)^{\mathbf{k} - \mathbf{t}}$$

$$c_1 \leftarrow C(U_t, \mathscr{C}_{t, w \cdot 1}) = \sum_{u_i \in U_t} d_{i, w \cdot 1} \cdot (d_i - 1)^{\mathbf{k} - \mathbf{t}}$$

Step 3.3. IF $c_0 \geqslant c_1$ THEN $w \leftarrow w \cdot 0$ ELSE $w \leftarrow w \cdot 1$.

Step 3.4. Update each T_i's pointer to the node labeled by the new value of w.

Step 3.5. IF $|w| < r$ THEN GO TO Step 3.2 ELSE $y_t \leftarrow v$, where $V_w = \{v\}$.

Step 4. $V_{t+1} \leftarrow V_t - \{y_t\}$;
$U_{t+1} \leftarrow \{u \in U_t \mid y_t \in \Gamma(u)\}$;
$\mathscr{B}_{t+1}$ is the graph induced by the vertex sets U_{t+1} and V_{t+1}.

Step 5. Update the neighborhood trees T_i to correspond to $\mathscr{B}_{t+1}$. This is done by subtracting 1 from each d-value stored at nodes on the path from the root to the leaf labeled w. Also, initialize the three pointers to point to the root of each tree.

Step 6. IF $t < k$ THEN $t \leftarrow t + 1$; GO TO Step 3.

Figure 3.6: Clique Stripping pseudo code

ALGORITHM PARTITION.

Step 1. Initialize: $i \leftarrow 0$; $n \leftarrow |U|$; $\hat{m} \leftarrow |E|$.

Step 2. Compute the neighborhood trees, as described in Lemma 2.2.

Step 3. WHILE $\hat{m} \geqslant n^{2-\delta}$ DO

Step 3.1. Start ith stage: $i \leftarrow i+1$; $\hat{k} \leftarrow \lfloor \delta \log n / \log(2n^2/\hat{m}) \rfloor$.

Step 3.2. Using the *Clique Stripping Algorithm* and the neighborhood trees, determine sets of vertices $K \subseteq V$ and $U_K \subseteq U$ which form a δ-clique in $\mathscr{B}$. The ith δ-clique C_i has the vertex sets $U_i \leftarrow U_K$ and $V_i \leftarrow K$. The neighborhood trees are modified by the *Clique Stripping Algorithm* to reflect the removal of all edges in $U_K \times K$.

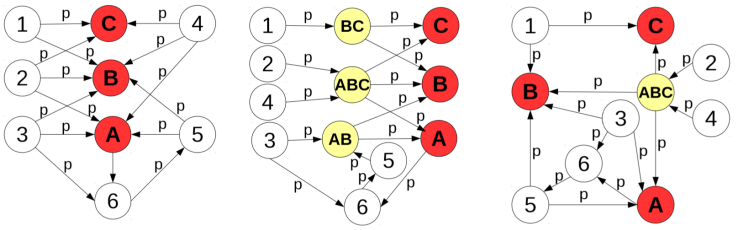
Step 3.3. Update $\mathscr{B}$ as follows: $E \leftarrow E - (U_i \times V_i)$; $\hat{m} \leftarrow |E|$.

Step 4. The remaining edges in E are partitioned into cliques consisting of a single edge each.

Figure 3.7: Partition algorithm pseudo code

3.2 Dedensification Algorithm

The Dedensification algorithm by Maccioni and Abadi is the state of the art algorithm made for the dedensification of all kinds of graphs. The algorithms are simple and do not require a complicated preprocessing on the data, (like in the case of Feder and Motwani, when we had to build Neighbourhood Trees) so we can start it easily and without a significant additional cost of the memory and time. The main idea is to give the algorithm a set of high-degree nodes, from which the cliques are calculated. The cliques we are producing in this way can be a combination of all the high degree nodes as the destination of the edges and all the nodes (including low-degree nodes as well as high-degree nodes) as the origin. Since in this study we are working with bipartite graphs, sets of high-degree and the origin set containing low/high degree nodes will be disjoint. In Figure 3.8 the work of the algorithm is presented, showing how the combination of high degree nodes is being created as cliques. The second and third part of Figure 3.8 is produced with the same value of the parameter $T = 3$ and the same high degree nodes set. The only difference between the two is applying the optimization constraint which uses only the cliques, which are resulting in compression. As you can see in the second part of Figure 3.8, some of the newly introduced nodes are not reducing the number of edges, thus we do not blindly use the full result, that we acquired, but rather the cliques of a bigger size. The same optimization based on the size of the cliques is used in case of Feder and Motwani algorithm, so the comparison between the two is fair and conclusive.



(a) A graph G (b) $\tilde{G}$ (c) Non-expansive $\tilde{G}$

Figure 3.8: The complicated example for new graph G, showcasing how the dedensification is created in the middle and how it can be optimised in the third figure, as shown in Maccioni and Abadi 2016

Dividing the nodes into a high/low degree is simply done by setting a parameter T, looping over all nodes, and choosing which ones were the destination of the edges T or more times ($>= T$). We do not want to include all the nodes that were destination since this will produce more of the useless cliques, that after applying the optimization constraint, will not become part of the compression, thus it is important to optimize the value of T for the graph.
The Dedensification algorithm is working by looping over all nodes of the graph G and adding a clique if the high degree destination of the node n was not in the set of cliques before or adding the node n to the set of origin of a clique if the clique is already there (pseudo-code of dedensification algorithm).

Since I am using python to write the code for those algorithms, sometimes some of the ways of writing the code are producing the same result with a different structure and shorter time. For this algorithm, the fastest one I produced is by using dictionaries. First, all the edges are converted to the dictionary, which groups them by the origin node, then we run the loop which terminates when there is no further improvement. Inside of the loop, we reverse the dictionary (keys with values) and we reverse it back, which is causing additional grouping to the cliques we had before. The work of this algorithm can be easily presented in the example below. I present the other way of combining those cliques for the future researchers who would like to use python

as their programming language, since using pointers is almost impossible and by using a loop we are significantly increasing the running time.

Example on the Dedensification based on dictionaries based on graph G from Figure 3.1:

Edges:
$[[1],[9]],[[2],[10]],[[2],[11]],...,[[2],[16]],\ldots,[[8],[15]],[[8],[16]]$

Dictionaries forming:
$[[1]->[9]],[[2]->[10,11,12,13,14,15,16]],...,[[8]->[10,11,12,13,14,15,16]]$

Reversing a dictionary:
$[[9]->[1]],[[10,11,12,13,14,15,16]->[2,3,4,5,6,7,8]]$

Reversing it back:
$[[1]->[9]],[[2,3,4,5,6,7,8]->[10,11,12,13,14,15,16]]$

Which are the cliques we were looking for.

We can run reversing and reversing back as long as the amount of keys in the dictionary gets smaller after double reverse. The cliques combining is happening on each of the reverse operation, which is not shown in the example above (improving only on the first reverse), because of its simplicity. This method is giving the same output as the 4-11 lines in the pseudo-code below.

```
Dedensification algorithm pseudo-code, lines 4-11 are forming the cliques and lines
    12-19 are for replacing the edges

Input: A graph database G_r = (N_h, N_t, N_c = [], E)
Output: A dedensified graph database G_r^

1.  W = [] // and auxiliary map for Constraint 1
2.  N_c = []
3.  E^ = E
4.  for each n_i in (N_h and N_t) do
5.      H <- select the outgoing high-degree nodes of n_i
6.      if H not empty then
7.          M <- W.get(H)
8.          if M is empty then
9.              W <- W and (H, n_i)
10.         else
11.             M <- M and {n_i}
12. for each <H_i, M_i> in W do
13.     n_c <- newNode()
14.     for each n_i in M_i do
15.         E^ <- E^\{(n_i, h)} : h in H_i
16.         E^ <- E^ and {(n_i, n_c)}
17.     for each h in H_i do
18.         E^ <- E^ and {(n_c, h)}
19.     N_c <- N_c and {n_c}
20. return G_r^ = (N_h, N_t, N_c, E^)
```

3.3 Comparison

The Dedensification algorithm can compress both of the graphs discussed in this chapter (Figure 3.1, Figure 3.8). The compression quality would be maximal in both of the cases, which means, that the algorithm will deduct the maximal amount of edges, given the dedensification process definition stated in chapter 2, section 2.1. A new figure for this algorithm was introduced to explain

how the optimization constraint works, which would not be possible with the first example. On the other hand, the Partition algorithm is only able to run on bipartite graphs. Even when the k-value in the given example is equal to 2, the compression is of such small quality, that the optimization constraint described in figure 3.8, which is also used for introducing new nodes in Partition algorithm, is stopping all of the cliques, that were found from being compressed. As a result, after running the Partition over Figure 3.1, we receive the same figure, without any changes. As stated before, this algorithm is working on sufficiently dense graphs, but also the quality of the cliques it finds is strictly connected to the k-value. K-value is based not only on density but also on the number of vertices. As you can see, the running prerequisites for Partition algorithm are not enough to make a compression, after the optimization constraint.
By the definitions included in Feder and Motwani paper, the bipartite graph G with n vertices and m edges can be partitioned into bipartite cliques of total order $p(G) = \mathcal{O}(m * log(n^2/m)/log(n))$. Further if $m = \Omega(n^{2-\epsilon})$, then we have $p(G) = \mathcal{O}(\epsilon * m)$ and if $m = \Omega(n^2)$ we have $p(G) = \mathcal{O}(m/log(n))$. Comparing it to the trivial upper bound $p(G) = \mathcal{O}(m)$ we can see the advantages of using this algorithm. The Partition algorithm aims to find the delta-clique with a stable upper bound, which is reflected in the running times. For the Dedensification algorithm, the case is different, since the aim of creating this algorithm is to accelerate the query processing times, by a compression. This is similar to achieving the best compression, but it is not the same. The description of how the Dedensification algorithm works are to simply combine all of the nodes who have the edges towards the high-degree nodes and group them when they are reaching the same nodes. No computations required, no looking for the best matching. The high-degree nodes set is the component, that ensures the choice of important destination nodes, and by simply running all combinations of them (indirectly, since we make checks by nodes) we create the cliques ready to be compressed. The compression is not perfect since we do not check if partial cliques (using not all connections from a node to high-degree nodes) are better. The complexity of this algorithm is $\mathcal{O}(m^2/n)$ if all of the edges are edges to a high-degree node. In reality, the complexity goes rapidly down, with an accurate high-degree node-set.

Chapter 4

Evaluation

4.1 Objective

The main question of the paper is, how good of a compression we can achieve by using the described algorithms. To address this question, the parameters of the algorithms have to be highly explored. If possible, finding the optimization constraints for the parameters will help future researchers with reliable work on the graphs. The compression comparison will help us understand what an improvement we achieved throughout the years as well as give as the insights of how strong the algorithms are. Optimizing the parameters is going to be an indicator of the flexibility of the algorithms given graphs of different sizes and densities.

4.2 Setup

To produce generalized results in this study 19 related (since drawn from the same dataset YAGO2 (KDE Group, 2020) using queries) bipartite graphs are used. All of them are going to be run through 2 algorithms, the Feder and Motwani algorithm and Maccioni and Abadi algorithm. Since one of the goals is to compare the results of both of the algorithms, a similar input has to be used. For the Feder and Motwani algorithm delta is set for 1, as we are interested in the biggest cliques we can find (since Maccioni and Abadi algorithm will also find the biggest cliques). For the second algorithm instead of setting a parameter T for a certain value for all of the datasets (which would not be a viable option, since the optimization of T depends on the particular graph), the disjoint property of the origin and destination will be used. The origin set U of the input and the V set of destination for Partition will be used for the Dedensification algorithm as the low-degree set and high-degree set. This works for the disadvantage of the Dedensification algorithm, which can be easily optimized in a better way, but since the theory behind the algorithms is giving the impression of how one-sided this comparison will be, the setup is created to compare the algorithms on the closest input possible. For the compression we will look at the number of new nodes introduced, to have the insight into how many cliques were found in the graph. Nonetheless, the crucial measure is the number of edges, which will allow decreasing the storage for the graphs.

4.3 Execution

The goal is not only to compare the compression but also to reliably optimize the parameters, separately for each of the algorithms. For this purpose, we have a list of steps which is followed in this study.
For the Partition Algorithm, the main question is the k-values per each graph. If the values are sufficiently big, optimizing delta should be considered. At the start, 10 values uniformly distributed over the $[0, 1]$ range will be used and if necessary, investigated further by smaller division in the

interesting ranges of the delta.
With the Dedensification algorithm, the optimization process is easier, since there is a way to always find the most optimal solution to every graph, with enough computational power. To make the computations less expensive, we need to have a list of potential Ts that might be optimal. This list is formed by calculating the densities for all of the nodes in set V (how many incoming edges have each node) and by extracting all unique density values. It works since T is the parameter describing the minimal density of a node to be included in high degree nodes. After calculating the set of potential T values we run the algorithms for all of them, on every graph with their unique T suspects. To justify the usage of the T optimizing, the calculations for $T = 0$ has to be shown next to the best values, so not only we investigate the quality of compression, but also the power of the parameter. To have the complete overview of the graph structure, the graph density ($|V|/|E|$) is introduced as well as the graph ratio ($|V|/|U|$). Since the number of connections per node may vary, the bar plots with nodes as the x-axis, indicating the size of set V and the densities (amount of incoming edges) per each note on the y-axis are sorted and shown all together. With that information, the decision tree based on the bipartite graph will be created and evaluated on how close to the perfect values it lays. In the case of a decision tree, we can have multiple approaches, from the most greedy one with all the parameters set to perfection to having all Ts as 0 (extreme variants), which means in this study, few of the approaches are going to be described.

4.4 Results

Calculating k-values per each graph is shown in table 4.1. Only 2 of those values are bigger or equal to 2. There is also one negative value of k. The further implementation stops here for the reasons described in the next section.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
k value	0	1	1	1	0	1	1	1	1	2	1	0	1	4	1	1	1	1	-8

Table 4.1: Values of k per graph, calculated via formula described in chapter 3 under Partition algorithm

Moving to the Dedensification algorithm, the implementation of the best T per graph can be checked in the appendix on Figures A.2 to A.50. The description table with more comprehensible results can be found in Table 4.2, 4.3, and 4.4. The compression quality is described by the division of the number of new edges by the number of old edges. As you can see in a few of the cases there is almost no compression, there are cases when optimizing the value of T does not improve the results, there are also graphs in which based on T optimization we can improve the compression from almost 0 deleted edges to thousands. Those results vary, nonetheless, there are visible patterns in them, which are going to be discussed in another section. Those are all of the descriptive results, that are used further, without counting the plot of the densities in Figure 4.1. The graphs are being sorted by indices, so the first index contains, graph 1, next graph 2, and so on. On the density plots we can spot few patterns, almost flat as in [5,3] (row, column), exponential curves as in [1,2] or [3,2], step structure as in [3,4], [4,4], [1,4], [5,3] or disconnected straight bars as in [4,2], [1,4], [5,1].

	edges	low degree nodes	high degree nodes	new nodes	U to V ratio	high nodes to edges ratio
1	169796	89891	11	229	0.00012237	6.47836e-05
2	17857	746	16	48	0.164879	0.000896007
3	5331	646	4	37	0.128483	0.000750328
4	573	158	44	2	0.835443	0.0767888
5	1659	1422	17	10	0.110408	0.0102471
6	5251	1887	8	24	0.198198	0.00152352
7	596	378	43	17	0.402116	0.0721477
8	330	224	9	5	0.53125	0.0272727
9	578	411	18	3	0.800487	0.0311419
10	5439	316	3	6	0.598101	0.000551572
11	2211	214	3	8	0.434579	0.00135685
12	313	376	65	3	0.489362	0.207668
13	164	132	2	2	0.462121	0.0121951
14	13000	1209	2	2	0.882548	0.000153846
15	185	161	10	1	0.695652	0.0540541
16	327128	15971	24	2174	0.0927306	7.33658e-05
17	100554	2786	6	61	0.326274	5.96694e-05
18	3050	399	42	4	0.200501	0.0137705
19	421157	3533	4	31	0.91169	9.49765e-06

Table 4.2: Part one of the description table

	edges after dedensification for T=0	edges after dedensification for best T	diff in edges for T=0 and best T
1	91534	91529	5
2	17843	17110	733
3	5227	4324	903
4	567	564	3
5	1647	1637	10
6	5170	5070	100
7	547	543	4
8	329	317	12
9	576	574	2
10	5439	5203	236
11	2211	2048	163
12	311	310	1
13	162	155	7
14	12999	12953	46
15	184	184	0
16	285375	276683	8692
17	100188	99549	639
18	2945	2943	2
19	421157	420407	750

Table 4.3: Part two of the description table

	abs diff in edges for T=0	abs diff in edges for best T	new to old edges ratio
1	78262	78267	0.539053
2	14	747	0.958168
3	104	1007	0.811105
4	6	9	0.984293
5	12	22	0.986739
6	81	181	0.96553
7	49	53	0.911074
8	1	13	0.960606
9	2	4	0.99308
10	0	236	0.95661
11	0	163	0.926278
12	2	3	0.990415
13	2	9	0.945122
14	1	47	0.996385
15	1	1	0.994595
16	41753	50445	0.845794
17	366	1005	0.990005
18	105	107	0.964918
19	-100000	750	0.998219

Table 4.4: Part three of the description table

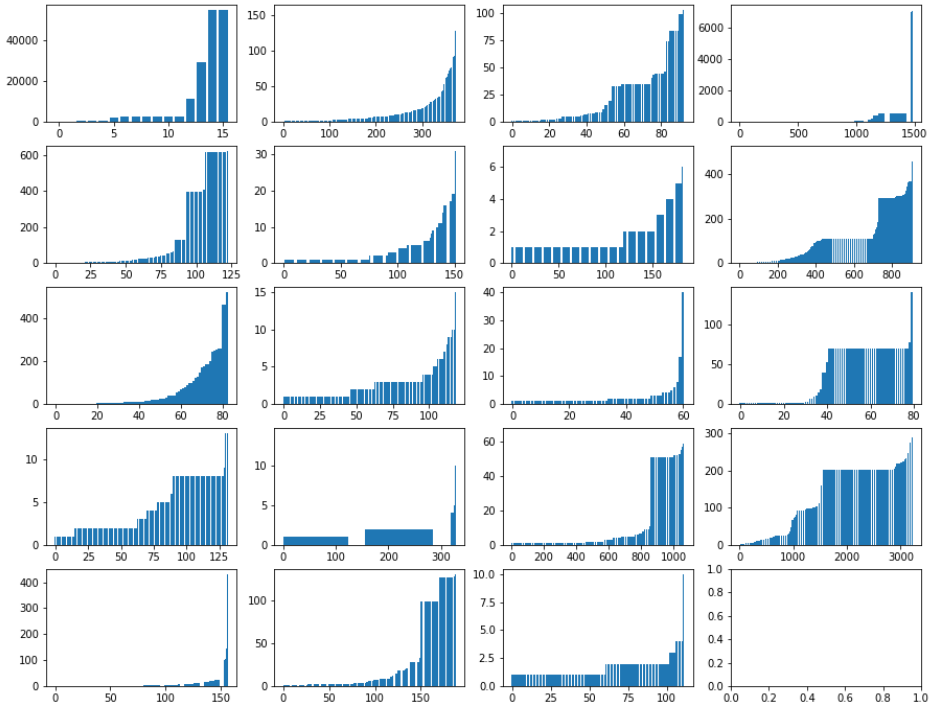


Figure 4.1: Bar chart for every of 19 datasets used in a study, showing the incoming edges per node after sorting the values

4.5 Interpretation

Having only 2 k-values sufficiently big to run the Partition algorithm, means that for all other 17 algorithms there is no improvement. Moreover one of then is k-value equal to 2, meaning the compression would be still close to 0 for this graph. With the maximum value of k being 4, there will be no attempt to optimize the delta, since any decrease would result in even smaller k. Those results show how unreliable and theoretical the Partition algorithm from 1995 is. The main reason for that is that the proves were created based on the Orlins paper, with the parameter delta not being flexible enough to significantly increase the k. Further increase above 1, would result in their formulas for the worst-case scenario being not valid anymore, thus this restriction was created. The setup for the k-value is strict and every small deviation from the perfect setup is highly influencing the results. Going back to the graph with $k = 4$, this means the biggest cliques we will find would be of the form $(\alpha, 4)$, which is simply not enough to capture the complexities in the various graphs. Any form of compression using this algorithm would be much weaker than the Dedensification, because of its multiple limitations, that is dictated by the need for a mathematical explanation of the phenomenon. One of the main reasons for this specific k-value behavior might be the simplified notation used in the formula, which assumes the set U and V to be of equal sizes. If this assumption would not be there and the formula correctly adjusted, the algorithm would at least run multiple times. It does not mean it would perform well, for the reasons described above, but this is the specific reason for the k-value being so small, together with the worst-case scenario approach.
The compression ratio for the graphs after running over the Dedensification algorithm is more than satisfactory. Since the algorithm does not require that much running time and the edge replacement is cheap, using it on huge graph networks can decrease the storage by a vast amount. Based on the results, we can classify each graph to one of the following groups:
$1 - >$ Highly compressible, with the density below 10^{-4} and the ratio below 10^{-2}, making it the best group for compression, and at the same time not requiring any T optimization, since the density levels and ratio are so extreme, that using all nodes in V is desirable. In our datasets, an example of such a graph is number 1.
$2 - >$ Decently compressible, with a density below 10^{-2} and the ratio below 1/5, making it the best group for the optimization, since all of the compression here is achieved by the correct choice of T. Graphs with those parameters are highly skewed to the right, so if there is not enough power to optimize them by trying their T values, the last 5 unique densities should be checked, or even only numbers 4 and 3, counting last five in a manner [5,4,3,2,1]. In our datasets, an example of such graphs are number 2,3,16.
$3 - >$ Moderately compressible, with density 0.01 to 0.1 and ratio not higher than 3/2. This group still gives a satisfying result while optimizing, but in opposite to the Decently compressible group, the optimization is not always all of the compression, which means we can achieve some of the compression in this group without setting a T. In our datasets, an example of such graphs are number 6,10,11.
$4 - >$ Slightly compressible, with density above 0.1 or V above 5, is the last group worth optimizing. More often then in 3, the optimization gives only partial results. In our datasets an example of such a graphs are number 5,7,8,13,17,18,19
$5 - >$ Compression comparable with the statistical error, happening especially in graphs with the number of edges below 500 and the extreme values of V ratio. The density above 0.1 and the ratio above 1 are the indicators. For those graphs optimizing the value of T is not advised, the density graphs in those cases are usually flat, so the middle choice of T (out of unique densities) is advisable here, without checking all the outcomes. In our datasets, an example of such graphs are number 4,9,12,14,15.
Those results are acquired by clustering the datasets by compression, density, ratio, and the graph curve.

Chapter 5

Conclusions

5.1 Conclusion

All in all, the algorithm by Feder and Motwani did not produce the results I was striving for in this study. The main reason for it is its complexity and multiple limitations, such as the sizes of the U and V sets. Because the authors used the description Orlin provided, the k-value calculation is based on the U and V of similar sizes. At the same time the k-value, responsible for the size of the cliques as well as the length of the loop is crucial in the Partition usage. Unfortunately, this algorithm does not produce any results in 17/19 cases and for other ones, it performs weakly.
From the other side, the Dedensification algorithm overachieved its expectations. The quality of compression with fast running time makes it the recommended algorithm for the dedensification. The parameter T seems to add flexibility, so the compression can be adjusted per individual graph. Since the datasets, that were used in this study are from the same bigger dataset, it is hard to tell if the notion of usually choosing one of the last 5 unique density values as T, per graph, is correct. Nonetheless, the clustering results based on the compression, density, ratio, and shape of the graph, are showing us the patterns being there, waiting for further exploration.

5.2 Discussion

The improvement done on the later studies, who would like to perform similar research centered on optimization of dedensification, would be to work on thousands of graphs. For such a work use of server would be needed. This will allow us to create a more precise decision tree with tight clusters and the perfect T parameter recommendation. If this algorithm is meant to be used on a larger scale, such a study is needed to fully exploit the algorithm.
Another idea would be to implement more of the dedensification algorithms and see if the results are comparable to the currently best option made by Maccioni and Abadi. An example of such an algorithm would be a recent study by Buehrer and Chellapilla (Buehrer and Chellapilla, 2008). It might be the case that with a combination of reachability queries algorithms, some of the dedensification algorithms outperform others, even if the compression is smaller. That is why other ones should also be implemented and checked while doing meta-studies combining dedensification and reachability.
Finally, the methods used to optimize the value of T may not be optimal. Maybe a general formula can be created so that after specifying the graph explanatory data, a short-range of T values is received from it. Clustering on such a small database as used in this study can lead to inaccuracies, thus the advice should be taken more like a general guideline, which provides a notion of how certain graphs will behave after the algorithm used.

Bibliography

Buehrer, G. and Chellapilla, K., 2008. A Scalable Pattern Mining Approach To Web Graph Compression With Communities. [online] Videolectures.net. Available at: <http://videolectures.net/wsdm08_buehrer_spma/> [Accessed 6 July 2020].

Feder, T. and Motwani, R., 1995. Clique Partitions, Graph Compression and Speeding-Up Algorithms. Journal of Computer and System Sciences, 51(2), pp.261-272.

KDE Group, H., 2020. YAGO2: Exploring And Querying World Knowledge In Time, Space, Context, And Many Languages — Bibsonomy. [online] Bibsonomy.org. Available at: <https://www.bibsonomy.org/bibtex/2bf85a360d67642eb76c5c1724758b7f3/jaeschke> [Accessed 21 July 2020].

Orlin, J., 1977. Contentment in graph theory: Covering graphs with cliques. Indagationes Mathematicae (Proceedings), 80(5), pp.406-424.

Maccioni, A., Abadi, D. J. (2016). Scalable Pattern Matching over Compressed Graphs via Dedensification. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. doi:10.1145/2939672.2939856

Stevenson, I. H., Kording, K. P. (2011). How advances in neural recording affect data analysis. Nature neuroscience, 14(2), 139-142.

Su, J., Zhu, Q., Wei, H. and Yu, J., 2017. Reachability Querying: Can It Be Even Faster?. IEEE Transactions on Knowledge and Data Engineering, 29(3), pp.683-697.

Appendix A

Appendix

The Figure A.1 and A.2 are showing all of the queries used to form 19 bipartite graphs used in the study. Only numbers [1,2,3,5,8,9,10,11,13,16,17,18,19,20,21,22,23,26,27] are used, since the rest of them did not create the expected results. Those queries are the ones we used for graphs, that are named graph 1-19, based on the order in the previous list. In Figures A.3 to A.50 you can find multiple tables, which are representing using all of the unique densities of the single nodes in set V as a parameter T in the Dedensification algorithm on 19 graphs (each one separately). Thus 19 tables with the following data are provided below. The T values are the column names and the full table is always for one dataset.

	13	173	349	737	1973	2576	11325	29199	54917
num of low degree nodes	89888	89889	89890	89891	89893	89894	89900	89901	89902
num of high degree nodes	16	15	14	13	11	10	4	3	2
num of new nodes	239	234	234	234	229	229	9	3	2
amount of new edges	91534	91530	91530	91529	92209	92208	103581	114888	114883

Table A.1: Parameter T usage on graph 1

	1	2	3	4	5	6	7	8	9	10	12	13
num of low degree nodes	639	646	655	660	664	667	675	677	681	685	686	691
num of high degree nodes	123	116	107	102	98	95	87	85	81	77	76	71
num of new nodes	1	1	1	1	1	1	1	1	1	1	1	1
amount of new edges	17843	17843	17843	17843	17843	17843	17843	17843	17843	17843	17843	17843

Table A.2: Parameter T usage on graph 2, part 1

14	15	19	20	23	24	25	27	30	31	36	39	43	55	58	60
692	693	694	697	698	702	704	705	706	709	710	714	717	718	721	722
70	69	68	65	64	60	58	57	56	53	52	48	45	44	41	40
2	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3
17815	17815	17815	17815	17815	17815	17815	17815	17815	17815	17815	17815	17801	17801	17801	17801

Table A.3: Parameter T usage on graph 2, part 2

64	127	398	399	406	409	618	619	624
723	724	732	743	744	745	746	760	761
39	38	30	19	18	17	16	2	1
4	5	5	23	26	30	48	2	0
17773	17759	17759	17439	17396	17362	17110	17242	17857

Table A.4: Parameter T usage on graph 2, part 3

	1	2	3	4	6	8	9	10	11	12	17	18
num of low degree nodes	567	581	587	597	599	600	601	603	606	609	610	613
num of high degree nodes	83	69	63	53	51	50	49	47	44	41	40	37
num of new nodes	22	22	22	22	22	22	22	22	23	24	24	25
amount of new edges	5227	5227	5227	5227	5227	5227	5227	5227	5226	5223	5223	5220

Table A.5: Parameter T usage on graph 3, part 1

20	23	27	32	37	39	41	52	57	68	76	78	90	97	99	108
614	617	618	620	621	622	623	625	626	627	628	629	630	631	632	633
36	33	32	30	29	28	27	25	24	23	22	21	20	19	18	17
26	26	26	28	31	31	32	34	34	34	37	37	32	35	37	39
5218	5220	5220	5210	5201	5200	5191	5184	5184	5184	5177	5172	5174	5157	5146	5139

Table A.6: Parameter T usage on graph 3, part 2

121	128	149	172	176	184	185	202	244	250	256	260	261	463	465	523
634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649
16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
41	44	50	56	72	89	98	107	113	108	92	73	37	12	2	0
5137	5124	5085	5024	4818	4718	4660	4614	4463	4435	4386	4338	4324	4468	4899	5331

Table A.7: Parameter T usage on graph 3, part 3

	1	2	3	4	5	6	8	9	13
num of low degree nodes	70	85	133	140	148	158	160	199	200
num of high degree nodes	132	117	69	62	54	44	42	3	2
num of new nodes	6	6	7	6	6	2	2	2	2
amount of new edges	567	567	564	565	565	564	564	564	564

Table A.8: Parameter T usage on graph 4

	1	2	3	4	5	6	7	8	9	10	11	12
num of low degree nodes	1282	1337	1361	1377	1383	1389	1394	1399	1403	1404	1408	1411
num of high degree nodes	157	102	78	62	56	50	45	40	36	35	31	28
num of new nodes	5	5	6	6	6	6	7	7	7	7	8	8
amount of new edges	1647	1647	1646	1645	1645	1645	1644	1644	1643	1643	1642	1641

Table A.9: Parameter T usage on graph 5, part 1

	query
1	(:<isLocatedIn>/:<owns>/:<created>)+
2	(:<hasCapital>/:<isLocatedIn>/:<dealsWith>)+
3	(:<owns>/:<isLocatedIn>/:<dealsWith>)+
4	(:<participatedIn>/:<isLocatedIn>/:<dealsWith>)+
5	(:<hasCapital>/:<participatedIn>/:<happenedIn>)+
6	(:<isInterestedIn>/:<influences>/:<hasAcademicAdvisor>)+
7	(:<isLocatedIn>/:<dealsWith>/:<hasCapital>)+
8	(:<owns>/:<isLocatedIn>/:<hasCapital>)+
9	(:<participatedIn>/:<happenedIn>/:<hasCapital>)+
10	(:<influences>/:<isMarriedTo>/:<hasChild>)+
11	(:<isMarriedTo>/:<influences>/:<hasChild>)+
12	(:<hasAcademicAdvisor>/:<isInterestedIn>/:<influences>)+
13	(:<isMarriedTo>/:<hasChild>/:<influences>)+
14	(:<isLocatedIn>/:<owns>/:<isConnectedTo>)+
15	(:<influences>/:<hasAcademicAdvisor>/:<isInterestedIn>)+
16	(:<dealsWith>/:<hasCapital>/:<isLocatedIn>)+
17	(:<dealsWith>/:<owns>/:<isLocatedIn>)+
18	(:<hasCapital>/:<owns>/:<isLocatedIn>)+
19	(:<owns>/:<created>/:<isLocatedIn>)+
20	(:<owns>/:<isConnectedTo>/:<isLocatedIn>)+
21	(:<hasChild>/:<influences>/:<isMarriedTo>)+
22	(:<created>/:<isLocatedIn>/:<owns>)+
23	(:<isConnectedTo>/:<isLocatedIn>/:<owns>)+
24	(:<isLocatedIn>/:<dealsWith>/:<owns>)+
25	(:<isLocatedIn>/:<hasCapital>/:<owns>)+
26	(:<happenedIn>/:<hasCapital>/:<participatedIn>)+
27	(:<isLeaderOf>/:<dealsWith>/:<participatedIn>)+

Figure A.1: The queries used on YAGO2 dataset to obtain bipartite graphs

14	15	16	18	19	20	21	23	24	25	27	36	57	99	104	143	431
1414	1415	1419	1421	1422	1425	1426	1427	1428	1431	1432	1433	1434	1435	1436	1437	1438
25	24	20	18	17	14	13	12	11	8	7	6	5	4	3	2	1
8	9	10	10	10	8	8	8	7	4	4	3	3	5	5	2	0
1641	1640	1638	1638	1637	1643	1642	1642	1644	1646	1646	1647	1646	1642	1641	1645	1659

Table A.10: Parameter T usage on graph 5, part 2

	1	2	3	4	5	6	7	8	9	10	11	12
num of low degree nodes	1521	1554	1626	1660	1686	1705	1717	1737	1751	1762	1771	1778
num of high degree nodes	374	341	269	235	209	190	178	158	144	133	124	117
num of new nodes	33	33	33	33	33	33	32	33	33	34	33	33
amount of new edges	5170	5168	5168	5167	5167	5167	5168	5166	5166	5169	5171	5170

Table A.11: Parameter T usage on graph 6, part 1

13	14	15	16	17	18	19	20	21	22	23	24	25	26	28	29	30
1784	1791	1798	1803	1807	1812	1821	1824	1827	1829	1830	1831	1835	1836	1838	1842	1844
111	104	97	92	88	83	74	71	68	66	65	64	60	59	57	53	51
32	34	36	35	33	35	32	30	30	28	27	25	24	24	24	21	21
5171	5167	5161	5161	5162	5160	5165	5168	5166	5171	5171	5173	5172	5172	5171	5182	5182

Table A.12: Parameter T usage on graph 6, part 2

31	32	33	34	35	36	37	38	40	43	44	49	53	54	60	62	64
1848	1850	1853	1854	1856	1857	1860	1861	1862	1863	1865	1867	1868	1869	1871	1872	1875
47	45	42	41	39	38	35	34	33	32	30	28	27	26	24	23	20
24	25	25	24	24	21	23	24	24	24	25	24	25	25	27	26	24
5178	5176	5176	5179	5179	5185	5184	5182	5182	5182	5178	5180	5172	5171	5168	5166	5165

Table A.13: Parameter T usage on graph 6, part 3

67	68	71	72	73	74	75	77	78	85	89	90	91	93	100	129	158
1876	1877	1878	1879	1880	1881	1882	1883	1885	1886	1887	1888	1889	1891	1892	1893	1894
19	18	17	16	15	14	13	12	10	9	8	7	6	4	3	2	1
22	22	27	27	31	33	36	36	33	26	24	20	12	15	9	2	0
5186	5186	5156	5153	5146	5141	5126	5121	5072	5072	5070	5096	5089	5154	5146	5160	5251

Table A.14: Parameter T usage on graph 6, part 4

	1	2	3	4	5	6	7	8	9	10	11	14	16	17	19	31
num of low degree nodes	269	345	361	370	378	391	398	399	400	403	405	409	410	415	417	420
num of high degree nodes	152	76	60	51	43	30	23	22	21	18	16	12	11	6	4	1
num of new nodes	16	16	17	17	17	15	14	14	14	12	10	10	10	6	5	0
amount of new edges	547	545	544	544	543	544	545	545	545	549	555	555	552	574	575	596

Table A.15: Parameter T usage on graph 7

	1	2	3	4	5	6	7	8	9	10	15
num of low degree nodes	114	160	177	210	217	220	224	226	227	230	232
num of high degree nodes	119	73	56	23	16	13	9	7	6	3	1
num of new nodes	1	1	1	4	5	5	5	3	3	2	0
amount of new edges	329	329	329	320	319	319	317	321	321	324	330

Table A.16: Parameter T usage on graph 8

	1	2	3	4	5	8	10	14
num of low degree nodes	100	251	398	411	424	426	427	428
num of high degree nodes	329	178	31	18	5	3	2	1
num of new nodes	2	2	2	3	0	0	0	0
amount of new edges	576	576	576	574	578	578	578	578

Table A.17: Parameter T usage on graph 9

	1	2	3	4	5	6	7	8	9	11	12	14
num of low degree nodes	130	157	185	207	221	223	229	235	244	247	249	252
num of high degree nodes	189	162	134	112	98	96	90	84	75	72	70	67
num of new nodes	0	0	0	0	0	0	0	0	0	0	0	0
amount of new edges	5439	5439	5439	5439	5439	5439	5439	5439	5439	5439	5439	5439

Table A.18: Parameter T usage on graph 10, part 1

18	19	21	22	25	28	33	98	99	126	127	130
255	263	264	267	268	269	279	280	299	300	316	318
64	56	55	52	51	50	40	39	20	19	3	1
0	0	0	0	0	0	0	0	3	6	6	0
5439	5439	5439	5439	5439	5439	5439	5439	5385	5337	5203	5439

Table A.19: Parameter T usage on graph 10, part 2

	1	2	3	5	6	7	8	9	12	16	19	33
num of low degree nodes	124	140	148	151	161	163	165	169	173	174	176	178
num of high degree nodes	93	77	69	66	56	54	52	48	44	43	41	39
num of new nodes	0	0	0	0	0	0	0	0	0	0	0	0
amount of new edges	2211	2211	2211	2211	2211	2211	2211	2211	2211	2211	2211	2211

Table A.20: Parameter T usage on graph 11, part 1

35	41	43	44	46	74	84	99	103
183	199	200	201	206	207	209	214	216
34	18	17	16	11	10	8	3	1
1	4	3	3	3	5	7	8	0
2209	2204	2203	2202	2202	2200	2180	2048	2211

Table A.21: Parameter T usage on graph 11, part 2

	1	2	3	4	5	6	7
num of low degree nodes	257	376	410	421	432	438	440
num of high degree nodes	184	65	31	20	9	3	1
num of new nodes	2	3	2	2	1	0	0
amount of new edges	311	310	311	311	312	313	313

Table A.22: Parameter T usage on graph 12

	1	2	3	4	5	6	8	17	40
num of low degree nodes	73	107	122	126	129	130	131	132	133
num of high degree nodes	61	27	12	8	5	4	3	2	1
num of new nodes	2	2	3	4	3	4	4	2	0
amount of new edges	162	162	161	160	159	158	158	155	164

Table A.23: Parameter T usage on graph 13

	1	2	3	4	5	6	7	8	9	10	11	15
num of low degree nodes	144	604	718	767	821	908	932	951	962	993	997	1000
num of high degree nodes	1067	607	493	444	390	303	279	260	249	218	214	211
num of new nodes	1	1	1	1	1	2	2	2	2	2	2	2
amount of new edges	12999	12999	12999	12999	12999	12998	12998	12998	12998	12998	12998	12998

Table A.24: Parameter T usage on graph 14, part 1

17	18	51	52	53	54	55	56	57	58	59	62	65
1003	1004	1005	1141	1176	1186	1187	1194	1198	1201	1203	1209	1210
208	207	206	70	35	25	24	17	13	10	8	2	1
2	2	3	3	3	3	2	2	3	3	3	2	0
12998	12998	12997	12997	12997	12997	12998	12996	12995	12995	12995	12953	13000

Table A.25: Parameter T usage on graph 14, part 2

	1	2	3	4	10
num of low degree nodes	59	120	161	165	170
num of high degree nodes	112	51	10	6	1
num of new nodes	1	1	1	1	0
amount of new edges	184	184	184	184	185

Table A.26: Parameter T usage on graph 15

	1	2	3	4	5	6	7	8	9	10	11	12
num of low degree nodes	14514	14717	14852	14959	15038	15073	15098	15111	15145	15183	15213	15220
num of high degree nodes	1481	1278	1143	1036	957	922	897	884	850	812	782	775
num of new nodes	2182	2186	2192	2195	2197	2205	2206	2204	2204	2206	2200	2199
amount of new edges	285375	285341	285252	285172	285138	284986	284885	284869	284741	284641	284546	284434

Table A.27: Parameter T usage on graph 16, part 1

13	14	15	16	17	18	19	20	22	23	24	25	26	27	29	31
15261	15276	15304	15326	15334	15353	15362	15370	15386	15406	15407	15415	15424	15432	15439	15453
734	719	691	669	661	642	633	625	609	589	588	580	571	563	556	542
2204	2208	2211	2205	2200	2197	2189	2187	2186	2185	2185	2180	2181	2189	2182	2192
284289	284118	283930	283920	283859	283717	283670	283628	283582	283448	283447	283399	283300	283150	283145	283019

Table A.28: Parameter T usage on graph 16, part 2

32	33	35	37	38	39	40	41	43	44	47	48	50	51	52	56
15456	15471	15472	15473	15479	15481	15487	15489	15495	15496	15499	15501	15511	15518	15521	15522
539	524	523	522	516	514	508	506	500	499	496	494	484	477	474	473
2193	2185	2180	2182	2187	2185	2190	2193	2178	2177	2178	2186	2187	2191	2180	2181
282874	282908	282921	282875	282705	282694	282577	282464	282548	282531	282443	282129	282081	281862	281886	281873

Table A.29: Parameter T usage on graph 16, part 3

57	59	60	61	63	69	70	73	74	77	79	85	86	88	101	102
15548	15549	15551	15553	15554	15559	15560	15561	15562	15563	15567	15568	15569	15571	15572	15573
447	446	444	442	441	436	435	434	433	432	428	427	426	424	423	422
2179	2177	2177	2171	2179	2177	2177	2176	2170	2164	2166	2162	2160	2154	2154	2152
281777	281822	281721	281753	281517	281388	281388	281378	281411	281435	281385	281403	281409	281555	281523	281610

Table A.30: Parameter T usage on graph 16, part 4

103	105	108	109	118	120	123	130	145	187	188	223	230	246	293	294
15578	15598	15599	15603	15606	15607	15615	15637	15640	15641	15644	15645	15647	15653	15654	15655
417	397	396	392	389	388	380	358	355	354	351	350	348	342	341	340
2151	2157	2158	2143	2120	2123	2133	2140	2139	2146	2146	2142	2138	2116	2124	2094
281404	281230	281197	281246	281254	281194	280868	280643	280612	280259	280410	280483	280602	280526	280284	280263

Table A.31: Parameter T usage on graph 16, part 5

295	334	419	450	568	2102	2881	3098	7040	7087	7138	7149
15656	15657	15658	15702	15703	15971	15977	15978	15979	15992	15993	15994
339	338	337	293	292	24	18	17	16	3	2	1
2079	2109	2104	2161	2173	2174	1755	1753	1849	6	2	0
280431	279059	279216	277319	276711	276683	281322	283857	280152	313066	320092	327128

Table A.32: Parameter T usage on graph 16, part 5

	1	2	3	4	5	6	7	8	10	11	12	13
num of low degree nodes	1883	1936	1968	1996	2015	2030	2042	2048	2062	2074	2075	2093
num of high degree nodes	909	856	824	796	777	762	750	744	730	718	717	699
num of new nodes	75	75	75	75	75	75	76	76	76	73	73	76
amount of new edges	100188	100188	100188	100188	100188	100184	100183	100183	100183	100186	100186	100177

Table A.33: Parameter T usage on graph 17, part 1

14	15	16	17	18	19	20	21	23	24	25	27	28	30	32	33
2102	2110	2115	2121	2130	2133	2139	2144	2151	2159	2166	2170	2172	2175	2187	2188
690	682	677	671	662	659	653	648	641	633	626	622	620	617	605	604
78	76	76	75	77	79	79	79	79	78	76	73	73	73	73	73
100174	100176	100176	100165	100163	100154	100154	100151	100150	100140	100144	100132	100132	100132	100131	100131

Table A.34: Parameter T usage on graph 17, part 2

34	35	37	38	39	40	41	42	43	44	46	47	48	49	51	53
2190	2191	2202	2207	2216	2217	2221	2222	2224	2226	2227	2231	2232	2238	2241	2243
602	601	590	585	576	575	571	570	568	566	565	561	560	554	551	549
75	75	71	70	71	72	72	72	71	69	69	69	67	67	67	68
100127	100116	100127	100129	100127	100126	100123	100119	100120	100135	100135	100131	100163	100163	100156	100137

Table A.35: Parameter T usage on graph 17, part 3

55	58	59	60	61	62	68	69	71	73	74	75	78	83	90	93
2244	2246	2247	2249	2250	2251	2253	2256	2258	2259	2262	2264	2267	2273	2274	2280
548	546	545	543	542	541	539	536	534	533	530	528	525	519	518	512
68	69	69	71	71	72	71	66	66	64	62	63	64	63	65	62
100137	100101	100056	100048	100048	100038	100042	100076	100076	100092	100088	100085	100084	100069	100063	100070

Table A.36: Parameter T usage on graph 17, part 4

94	97	104	106	109	113	126	127	134	152	155	161	174	186	211	227
2284	2286	2309	2312	2313	2583	2584	2587	2588	2594	2599	2600	2605	2606	2612	2613
508	506	483	480	479	209	208	205	204	198	193	192	187	186	180	179
60	60	63	61	62	62	61	59	58	61	61	59	60	55	58	61
100065	100053	100046	100062	100056	100038	100036	100040	100015	99984	99984	99966	99952	99958	99946	99918

Table A.37: Parameter T usage on graph 17, part 5

290	291	292	293	296	297	300	301	302	303	304	305	308	310	323	324
2614	2680	2684	2698	2700	2703	2706	2707	2708	2715	2739	2742	2749	2752	2754	2757
178	112	108	94	92	89	86	85	84	77	53	50	43	40	38	35
68	68	68	68	68	68	68	68	69	69	69	69	69	68	68	68
99868	99868	99867	99867	99867	99867	99867	99867	99866	99866	99866	99866	99866	99865	99865	99865

Table A.38: Parameter T usage on graph 17, part 6

338	344	349	362	365	368	384	439	455	503
2761	2762	2765	2766	2770	2771	2786	2788	2790	2791
31	30	27	26	22	21	6	4	2	1
68	67	62	66	37	35	61	38	2	0
99865	99853	99874	99859	100199	100170	99549	99732	100433	100554

Table A.39: Parameter T usage on graph 17, part 7

	1	2	3	7	9	10	15	18	39	52	70	77	140
num of low degree nodes	361	380	391	393	395	396	397	398	399	401	402	439	440
num of high degree nodes	80	61	50	48	46	45	44	43	42	40	39	2	1
num of new nodes	6	6	6	5	4	4	4	4	4	2	2	2	0
amount of new edges	2945	2945	2943	2944	2944	2944	2944	2943	2943	2977	2977	2977	3050

Table A.40: Parameter T usage on graph 18

	1	2	3	4	5	6	7	8	9	10	11	12
num of low degree nodes	316	327	382	389	424	459	481	574	600	613	614	653
num of high degree nodes	3221	3210	3155	3148	3113	3078	3056	2963	2937	2924	2923	2884
num of new nodes	0	0	0	0	0	0	0	0	0	0	0	0
amount of new edges	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157

Table A.41: Parameter T usage on graph 19, part 1

13	14	15	16	17	18	19	20	21	22	24	25	26	28	29	31
698	754	788	790	802	871	873	878	880	942	964	1205	1208	1213	1222	1232
2839	2783	2749	2747	2735	2666	2664	2659	2657	2595	2573	2332	2329	2324	2315	2305
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157

Table A.42: Parameter T usage on graph 19, part 2

32	33	34	35	36	37	38	40	43	46	49	50	53	59	60	67
1236	1240	1243	1246	1250	1251	1255	1264	1270	1271	1272	1282	1285	1287	1288	1289
2301	2297	2294	2291	2287	2286	2282	2273	2267	2266	2265	2255	2252	2250	2249	2248
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157

Table A.43: Parameter T usage on graph 19, part 3

68	69	70	72	73	74	76	80	89	93	94	95	96	97	98	99
1312	1322	1324	1325	1340	1341	1343	1353	1381	1382	1537	1544	1558	1559	1658	1674
2225	2215	2213	2212	2197	2196	2194	2184	2156	2155	2000	1993	1979	1978	1879	1863
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157

Table A.44: Parameter T usage on graph 19, part 4

100	101	102	103	104	105	106	107	108	109	112	113	115	116	118	119
1677	1700	1740	1741	1762	1763	1767	1772	1773	1777	1796	1804	1805	1807	1808	1811
1860	1837	1797	1796	1775	1774	1770	1765	1764	1760	1741	1733	1732	1730	1729	1726
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157

Table A.45: Parameter T usage on graph 19, part 5

120	122	125	134	139	146	147	150	158	159	161	165	168	170	172	176
1812	1814	1815	1818	1824	1826	1827	1832	1833	1834	1835	1839	1840	1841	1842	1843
1725	1723	1722	1719	1713	1711	1710	1705	1704	1703	1702	1698	1697	1696	1695	1694
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157

Table A.46: Parameter T usage on graph 19, part 6

192	193	194	197	199	201	202	203	204	205	206	207	208	211	212	213
1844	1845	1846	1847	1851	1853	3090	3103	3201	3210	3215	3217	3218	3220	3222	3223
1693	1692	1691	1690	1686	1684	447	434	336	327	322	320	319	317	315	314
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421156	421156	421156	421156	421156	421156

Table A.47: Parameter T usage on graph 19, part 7

215	217	218	220	221	222	223	224	225	226	227	228	229	230	231	232
3224	3227	3229	3307	3327	3331	3353	3364	3366	3378	3389	3390	3391	3393	3396	3408
313	310	308	230	210	206	184	173	171	159	148	147	146	144	141	129
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
421156	421156	421156	421156	421156	421156	421156	421156	421156	421156	421156	421156	421156	421156	421156	421156

Table A.48: Parameter T usage on graph 19, part 8

233	234	235	236	237	238	240	244	245	246	247	248	250	251	252	253
3418	3421	3433	3434	3436	3442	3448	3449	3450	3452	3455	3464	3466	3467	3469	3470
119	116	104	103	101	95	89	88	87	85	82	73	71	70	68	67
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157

Table A.49: Parameter T usage on graph 19, part 9

255	257	258	262	267	268	269	272	273	275	276	277	278	280	281	282
3473	3475	3477	3478	3479	3481	3482	3483	3484	3493	3504	3505	3507	3509	3511	3523
64	62	60	59	58	56	55	54	53	44	33	32	30	28	26	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157	421157

Table A.50: Parameter T usage on graph 19, part 10

283	285	287	288	289	292	300
3526	3528	3530	3532	3533	3534	3535
11	9	7	5	4	3	2
1	3	27	64	31	8	2
421148	421142	421004	420487	420407	420593	420861

Table A.51: Parameter T usage on graph 19, part 11