PRE - Research Project

Information and Communication Sciences and Technology

School Year: 2019 - 2020

Demand-Aware Network Designs of Bounded Degree

Confidentiality Notice Non-confidential report, can be published on the web

ENSTA Paris Supervisor :
François Pessaux

Universität Wien Supervisors :
Stefan Schmid
Maciej Pacut

Internship from 18/05/2020 to 31/07/2020

Universität Wien
Fakultät für Informatik, Währinger Straße 29, A-1090 Wien

Confidentiality notice

This present document is not confidential. It can be published online and read by the members of ENSTA Paris.

Acknowledgments

I would like to thank Prof. Dr. Stefan Schmid who proposed me this internship in the Communication Technologies Group of the University of Vienna. It allowed me to discover the research field in a group on the cutting edge of the Network Design area. I would like to express a special thanks to Dr. Maciej Pacut who allowed me a lot of time to discuss and think about the project. I appreciated its constant motivation which allowed me to keep moving forward. I would like to thank my ENSTA teacher François Pessaux who accepted to supervise my internship.

Abstract

New issues are emerging in the field of designing Datacenters. The research field about demand-aware networks is booming since Datacenters tend to be structured accordingly to the traffic load. This work focuses on finding improvements of a previous publication. The goal was to find a way to create an algorithm to build a network with the same performance in term of path length as the previous version but using less edges. The outcome is the description of an algorithm using 4 times less network links. This improvement of the previous work contributed to a submission for the 2020 INFOCOM conference.

Contents

List of Figures

Introduction

I had my research project with the Communication Technologies Group of the Faculty of Computer Science part of the Unviersity of Vienna. The theme was the design of demand-aware networks in order to respond to the demand of expanding Datacenters. My project was focused on a paper Prof. Dr. Stefan Schmid (head of the group) published in 2019 for the International Symposium on Distributed Computing (DISC) conference. I have been seeking to search some improvements of one algorithm and its analysis. This was for the purpose of a submission in the 2020 INFOCOM conference. The following report is divided in three chapters. The first two describe the problem of demand-aware networks and the algorithms designed for the DISC conference. Then in a third part I detail my contribution in the submission for INFOCOM conference. It contains an implementation of the algorithms, an analysis of one specific algorithm about sparse graphs and then the given improvements to it.

– Part 1 –

Presentation of demand-aware networks

1.1 The motivations

The research about design of demand-aware networks (*D*NAs) is motivated by the two following points. On the one hand, in order to reduce costs and increase performance, compagnies wish to design their data-centers in agreement with the traffic load in the networks. On the other hand, new technologies of connecting servers are emerging such as ProjecToR. The idea of this project in development is to replace physical wires between servers with open-air lasers which directions are controlled by ceiling mirrors. A simple implementation is to use a big flat mirror to the ceiling. In practice, spheric mirrors like disco balls will be used to reduce the shifting of laser angles for more accuracy.

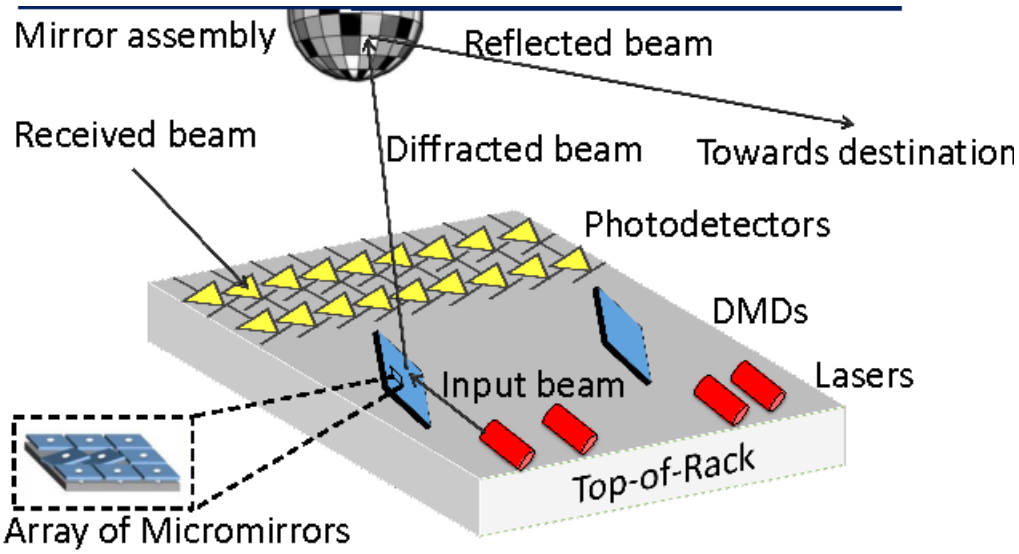

Figure 1.1: ProjecToR interconnect with unbundled transmit (lasers) and receive (photodetectors) elements.

This new technology required to find a way to design a network which satisfy a traffic demand in term of path length and congestion with a constraint on the number of emitters and sensors we can use per server rack. These components are placed on the top of rack of a server. That allows us to consider this problem as a graph theory problem, where the nodes of the graph are the top-of-rack servers and the edges are the established laser connections between top-of-rack servers.

The Communication Technologies Group of the Faculty of Computer Science of the University of Vienna led by Prof. Dr. Stefan Schmid, started to work recently on this topic. My work in the laboratory was mostly related to a paper the group published in 2019 including already some solutions to the problem [1]. This paper was written in collaboration with the Communication Systems Engineering Department of Ben Gurion University of the Negev in Israel.

The paper includes algorithms, results on its performance and proof of them. My project was to first implement the existing algorithms and then try to find improvements either in the algorithm definition, the performance analysis or both of them. The definition of the problem comes from the paper [1], and we will focus in the following part on its explanation.

1.2 Description of the problem

To have a coherent form we will stick to the notation already used in the published paper. The input of the problem is a demand distribution D of the traffic load and we consider the directed and weighted graph G_D which is associated. The nodes of the graphs correspond to top-of-rack servers. For each node i we construct edges to all others nodes j with a weight proportional to the amount of traffic between i and j. We then remove all edges with a weight of zero. Finaly we normalize the weights in order to have a sum on all weights equal to 1.

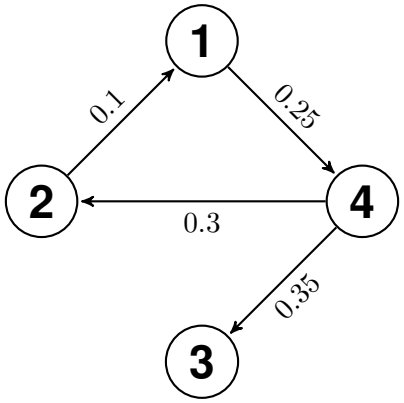

Figure 1.2: Simple example of a demand graph G_D

It is not illustrated in the above example, but edges can be directed in both directions.

Considering this input, the problem is to find an undirected and unweighted graph which minimizes the Expected Path Length (EPL) defined as follow:

$$EPL = \sum_{(i,j) \in V \times V} p_{i,j} \cdot d_{i,j}$$

where :

- V is the set of nodes
- $p_{i,j}$ the weight of edge from i to j
- $d_{i,j}$ the distance between i and j in the graph

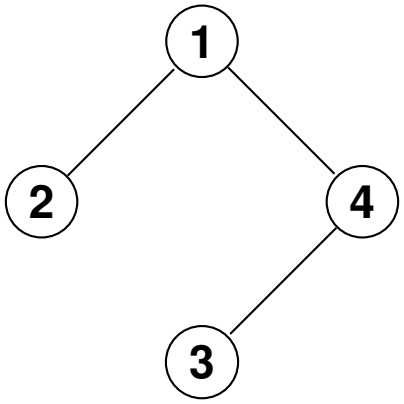

Figure 1.3: One possible result from the previous example

The above figure illustrated a possible solution which is not optimal. It has an EPL of 1.3. The best we can imagine is an EPL of 1 with a complete network for example. However for larger networks which correspond to real datacenters, we cannot use this solution. The problem is to find an algorithm which gives a result network and two upper bounds, one on the EPL and one on the maximum degree (ie. the maximum number of neighbours a node can have). Ideally, we would like to have a constant upper bound on the maximum degree, independent on the input graph. By doing so, industrials would be able to create *one size fits all* top-of-rack model. An other value to consider is the congestion which corresponds to the maximumum traffic load in one single edge. This consideration is not include in the paper I worked on, that is why it does not appear in my contribution part. However we will give later some ideas, the research worked on, to taking it in consideration.

1.3 Description of the algorithms

The topologies of real datacenters are not totaly random and we can mention three relevant categories of traffic demands : tree graphs, sparse graphs and regular and uniform graphs. For each one, the published paper described an algorithm and its performance about EPL and maximum degree. The division of separate categories allows to get more accurate results for some specific families of input demands. The three algorithms are similar and use quite the same methods of reduction of the degree in consideration of the EPL. That's why we will start by describing them.

1.3.1 Methods of reduction

The following reduction tools are the bricks in the construction of the algorithms. It will be important to recall it after in the improvements part.

Near optimal binary trees

We consider a node i and the subset of its out-neighbours (ie. the nodes j we can find and edge from i to j). In a input graph, the node is an important issue if the size of this subset is big, espacially if it is greater than the bound of the maximum degree we wish to reach. With the same reasoning, we need to care about nodes i with a large subset of in-neighbours (ie. the nodes j we can find and edge from j to i). This sort of issue can appear in a star distribution for example.

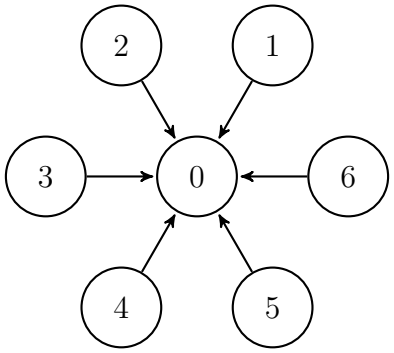

Figure 1.4: Example of a star input graph

Then the idea is to replace the neighbourhood by a near optimal binary tree. The *lemma 2* of the paper [1] claims that such a construction enables to have asymptotically optimal EPL. I chose in the implementation, as suggested in the paper, the Kurt Mehlhorn's method [2].

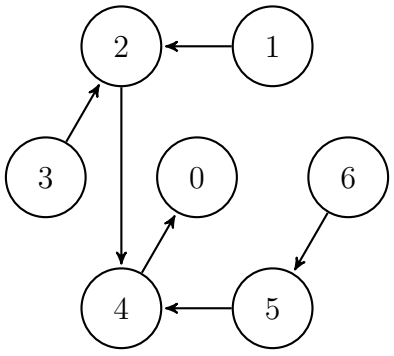

Figure 1.5: Possible binary tree construction from a star input

This construction enables to reduce the degree of the central node from 6 to 1 without creating another central node with high degree.

Classification of the nodes

The previous method of creating near binary trees permits to reduce the degree of nodes with an initial high degree. To define clearly the next algorithms, it is required to give a concrete definition to these so-called *high degree* nodes. The paper chose the *high degree* nodes as the $n/2$ ones with the highest degree. The other half is denoted the *low degree*

nodes.

Among the high-degree nodes, we can do another classification. Let's denote Δ_{avg} the average degree of the input demand graph G_D. Before going further, we recall an usefull relation between the number of nodes n, the number of edges m and the average degree Δ_{avg}.

$$\Delta_{avg} = \frac{2m}{n} \tag{1.1}$$

Then we denote *high-in degree* nodes (resp. *high-out degree* nodes), the nodes with a in-degree (resp. out-degree) greater than $2\Delta_{avg}$. In order to illustrate this definition, we recall our previous star example. The central node 0 with an initial degree of 6 is so a *high-in degree* node. Let's find Δ_{avg} thanks to (1.1). Because $n = 7$ and $m = 6$, we have $\Delta_{avg} = \frac{12}{7}$. Thus $6 > 2\Delta_{avg}$, that proves the classification of node 0 in the *high-in degree nodes*.

Use of helping nodes

The two previous tools enable, if there are used in a correct, to reduce the maximum degree of the output graph. However we may be carefull on the use of near binary trees. Let's imagine a troublesome case where, for a node i, we create near-binary trees from all its neighbourhood. That will involve i in each tree, and then multiply its initial degree. This issue appears especially in the case a *high-in degree* node is connected to many *high-out degree* nodes.

The last tool of degree reduction presented in the paper is the rerouting of edges from *high-out degree* node to *high-in degree* node, by replacing it with 2 intermediate edges passing through a another node we call *helping node*. To resolve the issue, this node can be a *low degree* one.

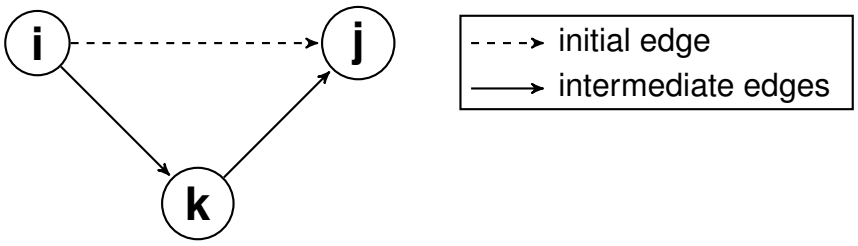

Figure 1.6: Explanatory scheme of the use of helping nodes

1.3.2 Algorithms

Tree graphs

This algorithm deals with input graphs G_D that can be regarded as a tree, the indirected version of the graphs (ie., no directed edges and merging of the double edges between two

nodes) is a tree (not necessarily binary).

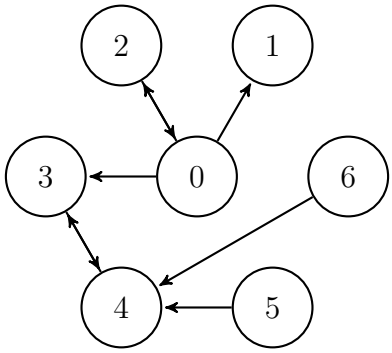

Figure 1.7: Example of a tree input graph

The algorithm is the following. For each node, we replace its initial *in-neighbourhood* and *out-neighbourhood* with two near-binary trees. Then the paper claims that the EPL is asymptotically optimal and the maximum degree is 8.

Sparse graphs

In the litterature, *the disctinction between sparse and dense graphs is rather vague* [3]. In this context, a graph topology is sparse, if by increasing its size, we have a constant Δ_{avg}.

The algorithm is the following. We first replaced edges from *high-in degree* nodes to *high-out degree* nodes thanks to helping *low degree* nodes. Then we replace *in-neighbourhoods* of *high-in degree* nodes and *out-neighbourhoods* of *high-out degree* nodes by near optimal binary trees. Then the paper claims that the EPL is asymptotically optimal and the maximum degree is $12\Delta_{avg}$.

Regular and uniform graphs

A graph is *regular* if all nodes have the same degree. It is *uniform* if all the weights of edges are the same. Then if we can find a spanner for D [4], we can use the following algorithm. We cut in half the nodes between *high* and *low degree* nodes. Then we replace edges between *high degree* nodes thanks to the help of *low degree* nodes. Finally, we create near optimal binary trees on the neighbourhoods of all *high degree* nodes. The paper claims that the EPL is asymptotically optimal and the maximum degree is $8\Delta_{avg}$.

– Part 2 –

Implementation and analysis of the performance

2.1 Implementation of the three algorithms

My first task during my internship was to implement the algorithms described in the paper the group published. It allows me to get some insights about the problem and understand the mechanisms in the algorithms. It also allowed us to compare the practical results with the theorical bounds found in the analysis. The utltimate goal was to make them match.

Because of the multiple librairies it offers for scientific computing, I followed the advice of my supervisors to use Python language. Futhermore researchers are used to code in Python, thus the group could reuse it to go further. It did require me to test functions and write a well structured and documented code. For example my supervisor could add some evaluations of the congestion I didn't focus on, and then accordingly modify the algorithms implementation.

2.2 Testing on different graph categories

Once the algorithms were implemented, my supervisors asked me to find some little modifications or details I could bring to the algorithms in order to increase performance, notably for the maximum final degree. The EPL is indeed asymptotically optimal. The paper precises indeed a lower bound on the value of EPL, that prevents us to find a better upper bound because it asymptotically match with the lower one. On the other hand, we wish to have a better bound on the maximum degree especially for the second and third algorithms that give a result dependent on the average degree Δ_{avg}. Ideally we would like to make it independent of Δ_{avg}.

For the rest of the project, I focused on the second algorithm and the sparse graphs. This algorithm is indeed a general one and we can use its results for every type of graphs. The hypothesis of a sparse graph is only used to assimilate Δ_{avg} as a constant and so the maximum degree bound as constant too. Furthermore the second and third algorithms are very close, thus a modification in the second algorithm might be relevant for the third.

Before adding modifications to the algorithm, I tested its performance on different graphs categories. I searched for graphs that have a heterogeneous distribution in order to have some nodes with degree greater than $12\Delta_{avg}$. Otherwise the input graph would be an optimal solution accordingly to the EPL and with a maximum degree lower than our algorithm can

guarantee. It must also be a sparse graph, ie. with a number of edges m proportionnal to the number of nodes n.

Highlands graphs

I first built some graphs with even distribution of high degree nodes we call *peaks*. The following part describes the definition of these graphs, I called *highlands graphs*. They are parametrized by an integer n which correponds to the number of nodes. Let index the nodes from 0 to $n-1$. We note $k = \lfloor \sqrt{n} \rfloor$ For each node i, if $i \equiv 0 \mod k$, i is a peak, then we create edges from i to every $j \in [\![i+1, max(n-1, i+k)]\!]$. We add a final edge from the last peak to a first one, otherwise we woud have a tree. Then we can show that $\Delta_{avg} \sim 2$ when n increases. Thus our bound on maximum degree is 24. The figure below is an example of the 9-highlands graph generated with the library networkx of Python.

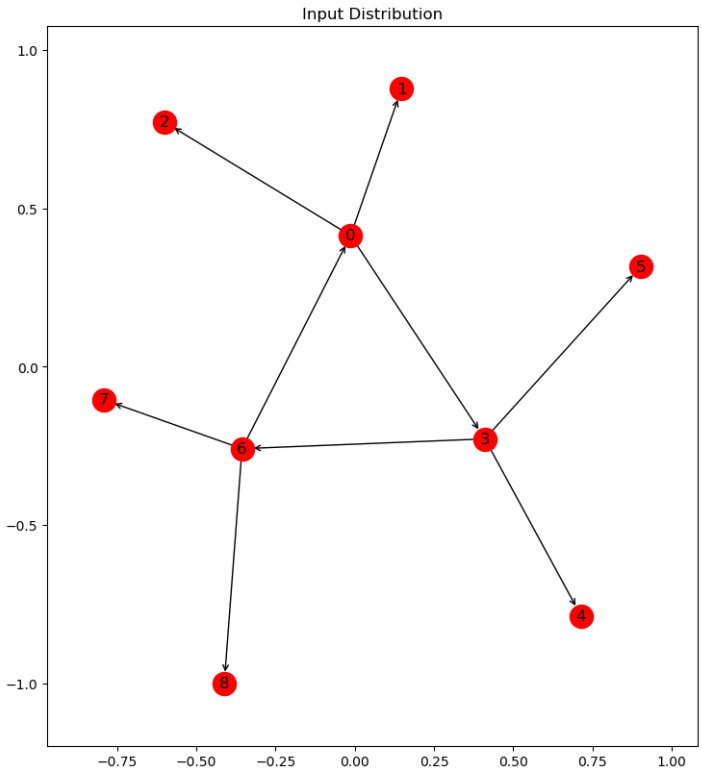

Figure 2.1: Example of a highlands graphs

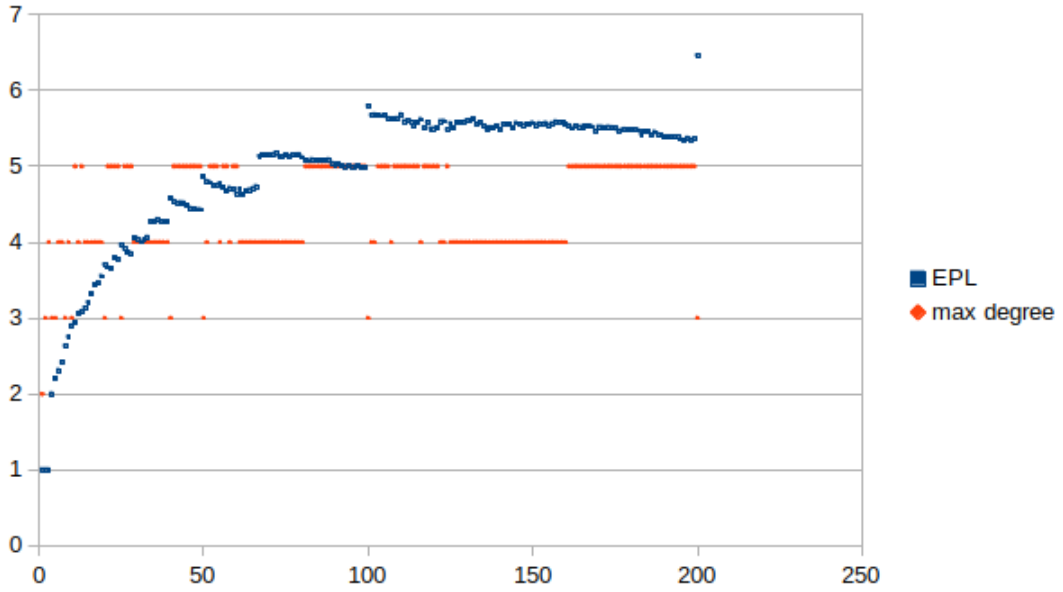

Figure 2.2: 200 first *highlands graphs*

The figure above shows the evolution of the EPL and the maximum degree of the result graph created by the algorithm. The maximum degree does not exceed 5 which is far from the theorical bound 24. We would find worst cases were the algorithm approaches the limit.

Composition of stars

Highlands graphs do not include *high-in degree* nodes, and thus they are no edges from *high-out degree* node to *high-in degree* node. Because we said they can be a real issue for the design of a bounded network, we need to test the algorithms using inputs including this kind of edges. I studied other categories of graphs, one made of a composition of stars. The definition is very close to highlands graphs one's. We build k star trees with l branches. It can be an ingoing star (with ingoing edges) or an outgoing one (with outgoing edges).

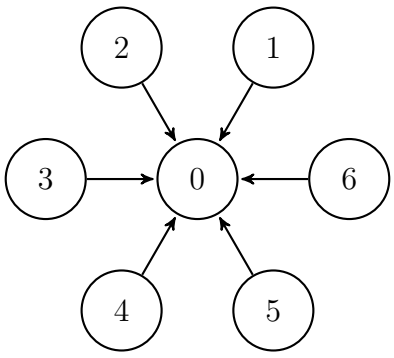

Figure 2.3: Ingoing star

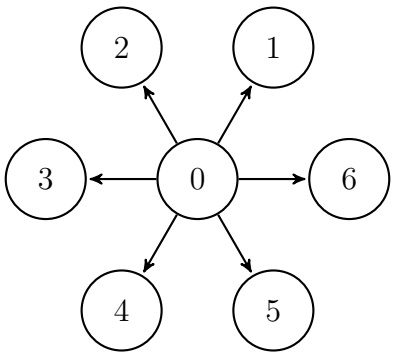

Figure 2.4: Outgoing star

Then we connect the centers of the stars with a bidirected cycle as shown in the figure below.

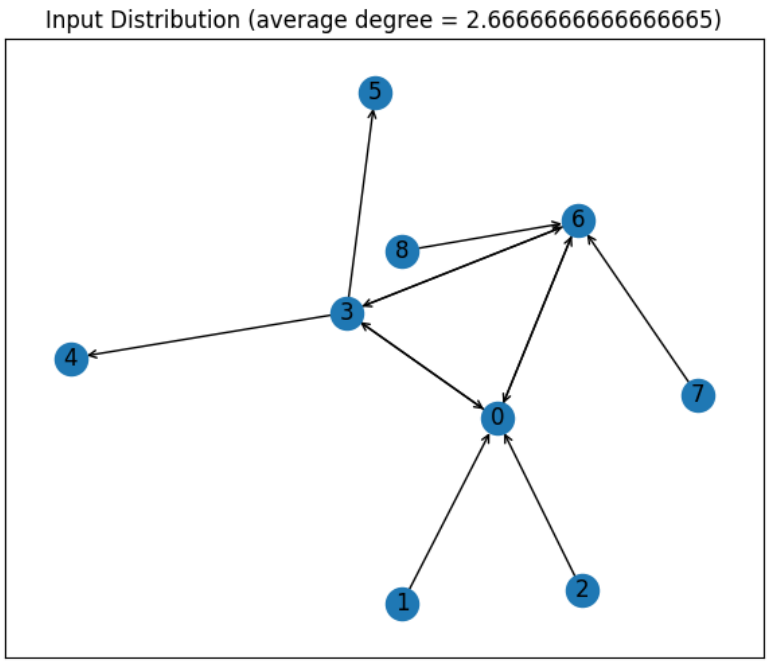

Figure 2.5: *Composition of star* with $k = 3$ and $l = 2$

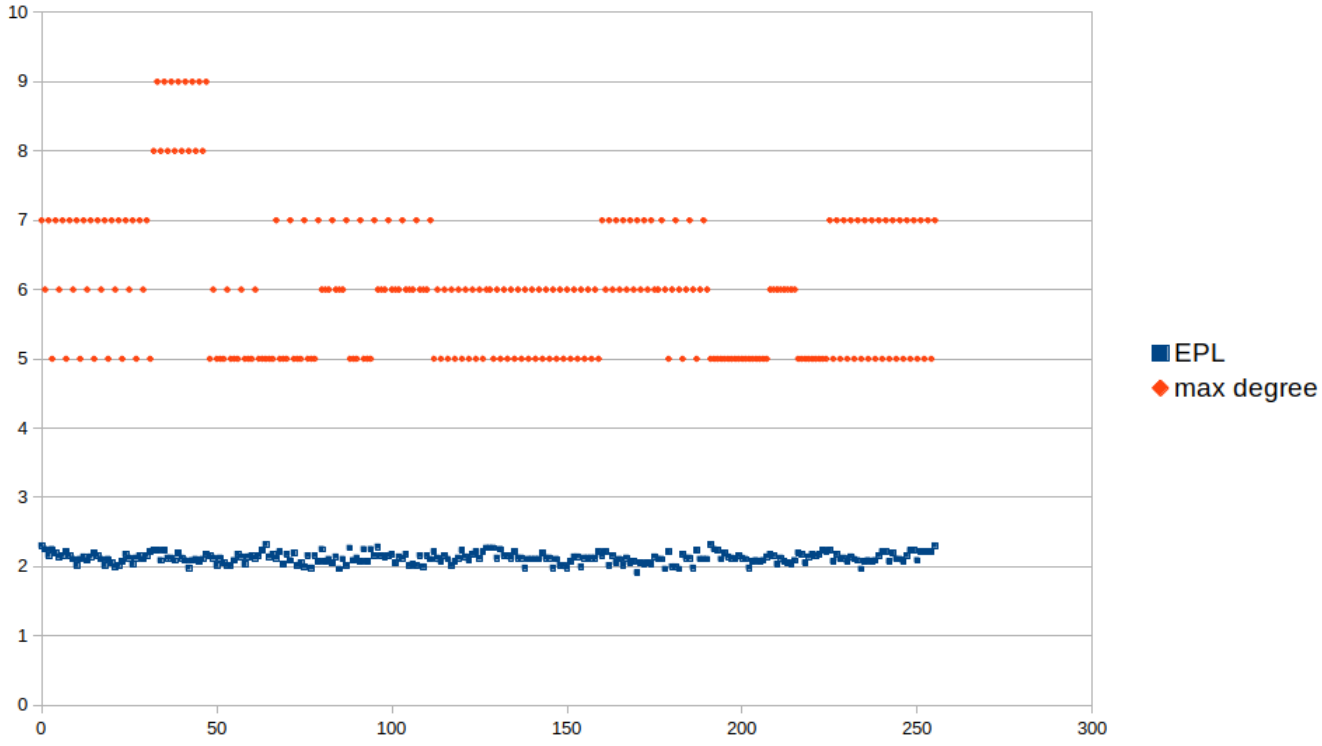

Figure 2.6: *Star graphs* with $k = 8$ and $l = 5$

I so tested the algorithm using the 256 compositions of stars with $k = 8$ and $l = 5$. For a such graph, we have $\Delta_{avg} = \frac{7}{3}$, so our bound on the maximum degree is $12\Delta_{avg} = 28$. Again, the worst result about the maximum degree, which is 9, is far from the theorical bound of 28.

I continued the testing of the algorithm performance, include some randomness in the

generation of the graphs. The conclusion was that the algorithm was way more efficient than the paper claimed for every tested input. That did not prove that we could not find a bad case that matches with the theorical bound. However it gives me the insight that eiher the algorithm or the analysis could be change a little in order to make results and theorical bound closer. Futhermore this range of inputs would enable us to analysis changes and compare two versions of the algorithm in term of EPL and maximum degree.

– Part 3 –

Modification brought to the algorithm and analysis

This chapter deals with the last part of my internship. After implementation and analysis, thanks to new insights about the problem, I made to my supervisors some proposals to modify the algorithm. The focus of these modifications was to reduce the theorical bound on the maximum degree, which was intially equals to $12\Delta_{avg}$. Before going further we need to recall the proof of this bound.

3.1 Proof of the upper bound on the maximum degree

The algorithm removes all edges from *high-out degree* nodes to *high-in degree* nodes and replace it with a 2-hop path through a *low degree* node we call *helping* node. Every *low degree* node can help at most Δ_{avg} edges. By doing so we are sure to help enough edges. There are at most m edges to help (m is the total number of edges). If each *low degree* node helps Δ_{avg} egdes, all in all, we are able to help $\frac{n\Delta_{avg}}{2} = m$ according to (1.1).

Then we replace *in-neighbourhood* of *high-in degree* nodes and *out-neighbourhood* of *high-out degree* nodes with near optimal binary trees. In order to prove the bound on maximum degree, the paper shows a different bound for each type of node and keep the greatest one.

3.1.1 *Low degree* nodes

Because of the *pigeonhole* principle [5], the initial degree of a *low degree* node is at most $2\Delta_{avg}$. In the worst case its initial neighbours can be *high-out degree* or *high-in degree* nodes and they may involve the *low degree* node in their near optimal binary tree. A node in a binary tree can have at most 3 neighbours. That's why the contribution of the intial neighbours in the final degree of this *low degree* node is at most $6\Delta_{avg}$.

Then this *low degree* node helps at most Δ_{avg} edges from a *high-out degree node* to a *high-in degree* node. Necessarily it will be involved in the trees from the *out-neighbourhood* of the *high-out degree* node and from the *in-neighbourhood* of the *high-in degree* node. Thus the node may be involved in $2\Delta_{avg}$ trees and add a contribution of $6\Delta_{avg}$ in the final degree.

In total we found a bound on the maximum degree of the *low degree* nodes which is $12\Delta_{avg}$.

3.1.2 *High-in degree* and *high-out degree* nodes

A node which is both *high-in* and *high-out degree*, will not be involved in any tree from an other node, because we would have replaced intial connections with *high-in degree* or *high-out degree* nodes. Furthermore, the other connections will be replaced by two edges to the binary trees made of its *in* and *out-neighbourhoods*.

In total the final degree of a both *high-in* and *high-out degree* is at most 2.

3.1.3 Either *high-in* or *high-out degree* node

Let's consider a node which is only *high-in degree* (the same reasoning is possible for an only *high-out degree* node). Its *in-neighbourhood* will be replaced with a connection to the root of a tree. The contribution of the *out-neighbourhood* in the final degree will be $6\Delta_{avg}$, because its out degree is lower than $2\Delta_{avg}$ (it is not a *high-out degree* node), and each connection can lead to a participation in a binary tree.

In total the final degre of an only *high-in* or *high-out degree* node is at most $6\Delta_{avg} + 1$.

3.1.4 *High* but neither *high-in* nor *high-out degree* nodes

A *high degree* node which is neither *high-in* nor *high-out degree*, will have, with the same reasoning, a final degree lower than $12\Delta_{avg}$. The contribution of the *in* and the *out-neighbourhood* is indeed $6\Delta_{avg}$.

We can sum up the data in the following table. In order to decrease the bound we need to focus on *low* and *high* but neither *high-in* nor *high-out degree* nodes. We denote the following subsets:

- L, the *low degree* nodes
- H, the *high degree* nodes
- HI, the *high-in degree* nodes
- HO, the *high-out degree* nodes

L	$H \setminus (HI \cup HO)$	$(HI \setminus HO) \cup (HO \setminus HI)$	$HI \cap HO$
$12\Delta_{avg}$	$12\Delta_{avg}$	$6\Delta_{avg} + 1$	2

Figure 3.1: Initial maximum degree for each type of node

3.2 First improvement to reach $6\Delta_{avg}$

3.2.1 Replace connections between *high degree* nodes

In order to reduce the maximum degree of high-degree nodes we can imagine to replace not only edges from *high-out degree* node to *high-in degree* node, but all edges between *high degree* nodes. That way it is sure that no *high degree* node will be involved in any binary tree from another node. We have more edges to help but it is still possible to help them all, because we are able to replace all edges. Thank to this update, if the in degree (.resp out degree) is greater than $2\Delta_{avg}$, then the contribution of the intial *in-neighbourhood* (.resp *out-neighbourhood*) in the final degree will be 1, else it will be at most $2\Delta_{avg}$ instead of $6\Delta_{avg}$. Let's sum up the results.

L	$H \setminus (HI \cup HO)$	$(HI \setminus HO) \cup (HO \setminus HI)$	$HI \cap HO$
$12\Delta_{avg}$	$4\Delta_{avg}$	$2\Delta_{avg} + 1$	2

Figure 3.2: Each edge between *high degree* node is replaced

We still have a global bound of $12\Delta_{avg}$ but we have made significant improvements for *high degree* nodes.

3.2.2 Decrease the limit on the number of edges a *low degree* node can help

We previously used the fact that the initial algorithm enables to help a large amount of edges, in fact all the edges. That allowed us to increase the number of edges replacements. The idea of this part is that the algorithm is still enabling a too high number of helps. And this high capacity to help has a cost, thus we would like to have a capacity close to the demand of helps. This cost is concerning the final degree of the *low degree* nodes. If we enable them to help a lot of edges, the analysis on the upper bound will lead to a high value.

My proposal was to diminish the number of edges a *low degree* node can help and make this number depend on the initial degree of the node. The number of edges a node can help decreases with its intial degree. For a node i, I chose to note this number β_i while the initial degree is denoted α_i. Recall that in the initial algorithm, for each node i, $\beta_i = \Delta_{avg}$. Now the new improvement is to choose β_i as follow.

$$\beta_i = \Delta_{avg} - \frac{\alpha_i}{2} \tag{3.1}$$

In order to well define the algorithm we need to justify that the number of edges we are able to help (ie. $\sum_{i \in L} \beta_i$) is greater than the number of edges between *high degree* nodes. Let note this number B. We also note A the number of edges where at least one *low degree* node is involved. Recall that m is the total number of edges, so we have :

$$A + B = m \tag{3.2}$$

Furthermore we consider the sum of the initial degrees of *low degree* nodes $\sum_{i \in L} \alpha_i$. For each edge taken in count in A, either it counts for one in the sum of inital degrees (edge connecting a *low degree* and a *high degree* node) or for two (edge connecting two *low degree nodes*). It leads to the relation :

$$\sum_{i \in L} \alpha_i \leq 2A \tag{3.3}$$

Proving that we can help a sufficent number of edges means to show $\sum_{i \in L} \beta_i \geq B$.

$$\sum_{i \in L} \beta_i = \frac{n\Delta_{avg}}{2} - \frac{1}{2} \sum_{i \in L} \alpha_i \geq m - A = B$$

Finally we need to measure the gain on the maximum degree bound. Let note γ_i the final degree of the *low degree* node i. As we repeated previously, there are two contributions in the final degree : the intial connections which are not replaced because i is *low degree* and the involvement in two trees per edge it helps. Thanks to the definition of β_i (3.1),

$$\gamma_i \leq 3(2\beta_i + \alpha_i) = 6\Delta_{avg}.$$

L	$H \setminus (HI \cup HO)$	$(HI \setminus HO) \cup (HO \setminus HI)$	$HI \cap HO$
$6\Delta_{avg}$	$4\Delta_{avg}$	$2\Delta_{avg} + 1$	2

Figure 3.3: Maximum degree for each type of node after the first improvement

According to the figure above, the bound on the *low degree* nodes is still the greatest but now its value is $6\Delta_{avg}$ instead of $12\Delta_{avg}$ in the initial paper. That result improves a lot the analysis, because without changing the idea of the algorithm, we manage to divide by two the bound on the maximum degree.

3.3 Tries to reach a constant bound

The last result satisfied me a lot and encouraged me to dig deeper and maybe find a better result for the degree bound. Remember that ideally we would like to find a constant value, independent of Δ_{avg}, and so independent of the input topology in order to create regular top-of-racks servers. Next are the ideas I tried to reach this goal.

3.3.1 Replacing Δ_{avg} with a parameter c

At this step the bound is $6\Delta_{avg}$. I didn't want to change the reduction tools I've presented at the beginning, because these are optimal for the EPL. My idea was to keep the structure of the algorithm and add some modifications to change the result of $6\Delta_{avg}$ into $6c$ with c a

parameter I would try to optimize, and maybe find a value independent of Δ_{avg}.

Let's consider the *low degree* nodes. To change the result of $6\Delta_{avg}$ into $6c$ we need to find where does the Δ_{avg} come from. By reading again the analysis, we find that it comes from the *pigeonhole* principle. Because the *low degree* nodes are the $n/2$ nodes with the lowest degree, their degree is lower than $2\Delta_{avg}$. Otherwise the sum on the total degree would be greater than $n\Delta_{avg}$ [5].

To replace in the analysis Δ_{avg} by a parameter c, we need to make the *low degree* nodes have a degree lower than $2c$. To do so I thought to change the number of *low degree* nodes. Let note x the ratio of the number of *low degree* nodes to n, the total number of nodes. Because the *pigeonhole* principle does not hold anymore, I had to find another upper bound on the degree of the *low degree nodes* dependent on x. Then I would choose x to have this bound equals to $2c$. To do so we note n_l the number of *low degree* nodes, $x = \frac{n_l}{n}$. Let sort the nodes in an ascending order. The sum of degrees of all nodes is then

$$n\Delta_{avg} = \sum_{i<n_l} \alpha_i + \alpha_{n_l} + \sum_{i>n_l} \alpha_i \geq \alpha_{n_l} + \sum_{i>n_l} \alpha_i \geq \alpha_{n_l}(n - n_l + 1),$$

where the last inequality follows as for all $i > n_l$ we have $\alpha_i > \alpha_{n_l}$ (the nodes are sorted). Than for all $i \in L$

$$\alpha_i \leq \alpha_{n_l} \leq \frac{n\Delta_{avg}}{n - n_l + 1} \leq \frac{n\Delta_{avg}}{n - n_l}.$$

$$\alpha_i \leq \frac{\Delta_{avg}}{1 - x} \tag{3.4}$$

I then replaced Δ_{avg} in the definition of β_i by c. Then I tried to optimize the value of c the lowest as possible, but the result was Δ_{avg}. The detailled reasoning is in the appendix. So with this method we found that the best we could do was to cut in half as previously to get a bound of $6\Delta_{avg}$.

3.3.2 Adding another parameter d in the β_i definition, $\beta_i = c - \frac{\alpha_i}{d}$

To resolve the issue we encounter previously, I tried to add another parameter d in the β_i definition, $\beta_i = c - \frac{\alpha_i}{d}$. The reasoning is also added in the appendix, but it leads unfortunately to the same conclusion as previously. The best result we could find was $6\Delta_{avg}$ when choosing $\beta_i = \Delta_{avg} - \frac{\alpha_i}{2}$.

The two analysis so elegantly concluded that $6\Delta_{avg}$ was the optimal bound, I though I've reached a fundamental limit of the problem. It took a little more time and energy to find new ideas to go further and improve the results.

3.4 Second improvement to reach $4\Delta_{avg}$

To make a step further we needed to find another idea. The last ideas do not enable us to go beyond the bound of $6\Delta_{avg}$. This corresponds to the bound of *low degree* nodes, for the *high degree* we have a bound of $4\Delta_{avg}$. Even if the goal of a constant bound seems hard to reach, we can try to obtain a lower value for example $4\Delta_{avg}$. It is not a constant value but it is still a good improvement from the initial one. It seems also reachable because we just have to improve the bound for *low degree* nodes. Recall that γ_i is the final degree of the node i, and for a node $i \in L$, we have, $\gamma_i \leq 3(2\beta_i + \alpha_i)$. We have $\beta_i = \Delta_{avg} - \frac{\alpha_i}{2}$, but we can imagine to have $\beta_i = \frac{2}{3}\Delta_{avg} - \frac{\alpha_i}{2}$. Thus we would have $\gamma_i \leq 4\Delta_{avg}$. However we must guarantee that β_i is positive. Thanks to (3.4), we have a bound on the degree of i. We need to choose x in order to have :

$$\frac{\Delta_{avg}}{2(1-x)} = \frac{2}{3}\Delta_{avg}$$

.

That leads to $\alpha_i = \frac{1}{4}$. In other words, we change the definition of the *low degree* nodes, and say that they are the $n/4$ nodes with the lowest degrees. Unfortunately without another modification of the algorithm, we cannot guarantee that we are able to help enough edges.

$$\sum_{i \in L} \beta_i = \frac{1}{3}m - \frac{1}{2}\sum_{i \in L} \alpha_i \geq \frac{1}{3}m - A$$

.

My supersivisor Maciej Pacut had the idea to distribute the load of helping edges, and make *high degree* nodes contribute in this task. This may sounds nonsense, because *high-degree* nodes are not suppose to help. But in fact it permits to help enough edges. We so define a β_i for $i \in H$. So we choose :

$$i \in H, \beta_i = max(2/3\Delta_{avg} - \frac{1}{6}\alpha_i, 0) \tag{3.5}$$

That way we have,

$$\sum_{i \in H} \beta_i \geq \frac{2}{3} \cdot \frac{3}{4} n\Delta_{avg} - \frac{1}{6}\sum_{i \in H} \alpha_i$$

.

Because $\sum_{i \in H} \alpha_i \leq \sum_i \alpha_i = 2m$, we conclude that

$$\sum_{i \in H} \beta_i \geq m - \frac{1}{3}m$$

.

All in all, $\sum_i \beta_i \geq m - A = B$. By doing so we can finally help enough edges and the algorithm is well defined. The bound on the final degree for *low degree* nodes is $4\Delta_{avg}$ as expected, but we must analyse it again for *high degree* nodes. This is done in the appendix, in a complete analysis of this improvement. Let's sum up again the bounds on the degree for

the different types of nodes. The calculations of these values are detailled in the appendix. Note that they differ from the following table, because to make simpler we didn't watch out to consider integer values. Then rounding considerations increase the values at most by 6.

L	$H \setminus (HI \cup HO)$	$(HI \setminus HO) \cup (HO \setminus HI)$	$HI \cap HO$
$4\Delta_{avg}$	$4\Delta_{avg}$	$2\Delta_{avg} + 2$	2

Figure 3.4: Maximum degree for each type of node after the second improvement

We managed to improve again, quite elegantly, the analysis of the algorithm. My internship was coming to an end, so we decided to write our ideas down, in order to submit a paper on these improvements. The appendix which details the reasoning was a good draft for a possible submission.

However when looking at our reasoning a second time, we were wondering if some details in the algorithm definition, we keep from the initial one, were still relevant. We were also wondering if we could go further with this idea of making *high degree* nodes contribute to the helping effort.

3.4.1 Last improvement to reach $3\Delta_{avg}$

Drop of the classification of *high-in* and *high-out degree* nodes

When we had read again the details of our paper, we noted that, because we changed the definition of *low degree* nodes - they represent no more the half of the nodes - the definition of *high-in degree* and *high-out degree* nodes was no more relevant. The limit of $2\Delta_{avg}$ came in the initial paper from the fact that we cut the nodes in half the nodes, and by doing so the *low degree* nodes had a degree lower than $2\Delta_{avg}$. But now it has changed, we should fix the limit at $\frac{3}{4}\Delta_{avg}$. If a node has its in degree (resp. out degree) larger than $\frac{3}{4}\Delta_{avg}$ than it would be a *high-in degree* (resp. *high-out degree*) node.

We could so, but I was wondering that because now we remove edges between all *high degree* nodes and not especially the edges from a *high-out degree* node and a *high-in degree* node, it was no more relevant to make this classification. Our idea was to drop the definition of *high-in* and *high-ou degree* node. That means that we would create binary trees from the neighbourhoods of all *high degree* nodes.

Drop of the classification of *high* and *low degree* nodes

Then I tried to add a parameter p in the analysis of the algorithm. The idea was to find the conditions, to have a maximum degree bound of $p\Delta_{avg}$ and if possible make p be less than 4.

Because we do binary trees from the neighbourhoods of all *high degree* nodes, for each $i \in H$ the contribution of the intial neighbours in the final degree is reduced to at most 2. We have $\gamma_i \leq 6\beta_i + 2$. Recall that for $i \in L$, we have $\gamma_i \leq 3(2\beta_i + \alpha_i)$. So now we introduce the paramter p we will try to minimize, in the definition of β_i :

- $i \in L, \beta_i = \frac{p}{6}\Delta_{avg} - \frac{1}{2}\alpha_i$
- $i \in H, \beta_i = \frac{p}{6}\Delta_{avg}$

That way, we have the following bounds on γ_i :

- $i \in L, \gamma_i \leq p\Delta_{avg}$
- $i \in H, \gamma_i \leq p\Delta_{avg} + 2$

Let's calculate $\sum_i \beta_i$ to be sure that we can help enough edges.

$$\sum_i \beta_i = \frac{p}{3}m - \frac{1}{2}\sum_i \alpha_i \geq \frac{p}{3}m - A$$

This was a really intersting result, because it means that if we choose $p = 3$, the algorithm is well defined in term of number of edges to help. Thus our final bound on the maximum degree would be $3\Delta_{avg}$, which improves again the initial algorithm.

Before claiming this result, we needed to make sure that β_i is postive for

$$i \in L$$

. We need to have $\Delta_{avg} >= \alpha_i$. From (3.4), we choose x to have $\Delta_{avg} = \frac{\Delta_{avg}}{1-x}$, and thus fill the condition on the positivity of β_i. However this equation leads to the surprising result of $x = 0$, in other words we need to consider all the nodes as *high degree* nodes, and by doing so drop the classification of nodes.

In fact that changed a lot the way the algorithm is constructed, and make it simpler. One important condition which was not detailled before, and which is needed soon as we enable *high degree* nodes to help, is that if a node i helps an edge of a node j, then i will be included in the the tree of j, but j will not be included in the tree of i. A complete explanation of the final algorithm can be found in the appendix, in the draft paper of the submission for the *INFOCOM* conference.

To conclude this part, we managed to divide by 4 the initial bound on the maximum degree in the initial algorithm. Note that this is not constant, it still depend on Δ_{avg}, but I didn't expect at the beginning to decrease so much the previous result.

Conclusion

This project focused on demand-aware networks enabled me to be involved in a work contributing to the state of art of networking. My main concern was to find some improvements of the work the Communication Technologies group had begun. However, I thought it would be little details about the way to implement the existing algorithm. I am really proud of the outcome because we managed with my supervisor to change the algorithm and its analysis in order to achieve the same performance in term of path length in the network using 4 times less edges. I also hope that the implementations of the previous and the new algorithms will enable them to move forward and find other important results. There is still a lot of problems to resolve in this field. A important consideration my project did not focus on, was the congestion. This leads to change a bit the reduction tools, such as replacing the near optimal binary trees with non-binary trees in order to distribute the traffic load. Nevertheless, I believe that the new algorithm definition can be used to deal with it. I look forward to the results of the submission of this work for the 2020 INFOCOM conference. This internship provided me an overview of the reasearch field, and the way a research group work together in order to find and claim new outcomes.

Bibliography

[1] Chen Avin, Kaushik Mondial, Stefan Schmid. *Demand-Aware Networks Designs of Bounded Degree*. 2019

[2] Kurt Mehlhorn. *Nearly optimal binary search trees. Acta Inf.*. 1975

[3] Definition of a dense graph. *https://en.wikipedia.org/wiki/Dense_graph)*

[4] Definition of a tree spanner. *https://en.wikipedia.org/wiki/Tree_spanner*

[5] Pigeonhole principle *https://en.wikipedia.org/wiki/Pigeonhole_principle*

– Part A –

Sparse graph : try to get a constant bound

A.1 Introduce a new parameter c to have $\beta_i = c - \frac{\alpha_i}{2}$

The idea is to classify the degrees in another way and then to do a quite same reasoning. Let's note C the constant degree we wish to obtain as a degree bound. We note: $c = \frac{C}{6}$ and $n_l = \frac{m}{c}$. n_l corresponds to the number of low degrees. In the previous version we had chosen $n_l = \frac{m}{\Delta_{avg}} = \frac{m}{\frac{2m}{n}} = \frac{n}{2}$ which is the half.

The other nodes are high degree nodes and we also change the definition of high-out(in) degree nodes. These are the nodes with out(in) degree $> 2c$.

Let's do the first step, we replace all the edges between high degree nodes. There are B edges to replace.We now decide to choose β_i that way:

$$\beta_i = c - \frac{\alpha_i}{2}$$

Before thanks to pigeon holes principle we were sure that β_i was positive. But now we have not this guarantee. To be sure that β_i will always be positive we need to study the worst case.

Let's sort the degrees. [i, ..., nl] are the low degree nodes and [nl+1, ..., n] are the high ones. We take a look at the total degree:

$$n\Delta_{avg} = \sum_{i<nl} \alpha_i + \alpha_{nl} + \sum_{i>nl} \alpha_i$$

$$n\Delta_{avg} >= \alpha_{nl} + \sum_{i>nl} \alpha_i$$

and if $i > nl$ then $\alpha_i > \alpha_n l$ because the nodes are sorted. So

$$n\Delta_{avg} >= \alpha_{nl}(n - nl + 1)$$

$$\alpha_{nl} <= \frac{n\Delta_{avg}}{n - nl + 1} <= \frac{n\Delta_{avg}}{n - nl}$$

And all the low degree nodes have this bound. Let's find a condition on c to have $\beta_i >= 0$:

$$c - \frac{n\Delta_{avg}}{2(n - nl)} >= 0$$

$$2(n - \frac{m}{c}) * c >= n\Delta_{avg}$$

$$c >= \frac{n\Delta_{avg} + 2m}{2n}$$

$$c >= \frac{2m}{n} = \Delta_{avg}$$

So we didn't found a better bound than previously... We could do the same reasoning but we would have a bound of $6c > 6\Delta_{avg}$.

A.2 Introduce a second parameter d to have $\beta_i = c - \frac{\alpha_i}{d}$

Let's find a sufficient condition on d to have $\sum_i \beta_i >= B$

$$\sum_i \beta_i = m - \frac{1}{d}\sum_i \alpha_i$$

So $d >= 2$ is sufficient because we still have $\sum_i \alpha_i <= 2A$.

Now we also need sufficient conditions on d and c to have $\beta_i >= 0$ Let's note $k = \frac{n_l}{n}$ which is the fraction of low degree nodes. We have

$$c = \frac{\Delta_{avg}}{2k}$$

and

$$\alpha_i <= \frac{n\Delta_{avg}}{n - n_l} = \frac{\Delta_{avg}}{1-k}$$

So

$$\beta_i >= c - \frac{\Delta_{avg}}{d(1-k)}$$

So by writing equivalent inequalities :

$$c - \frac{\Delta_{avg}}{d(1-k)} >= 0$$

$$\frac{\Delta_{avg}}{2k} >= \frac{\Delta_{avg}}{d(1-k)}$$

$$2k <= d(1-k)$$

$$d >= \frac{2k}{1-k}$$

So this is a sufficient condition to have $\beta_i >= 0$

Now let's calculate the bound on low degree node :

$$\gamma_i <= 3(2\beta_i + \alpha_i)$$

$$\gamma_i <= 3(2c - 2\frac{\alpha_i}{d} + \alpha_i)$$

$$\gamma_i <= \frac{3\Delta_{avg}}{k} + 3\alpha_i\frac{d-2}{d}$$

And because $\alpha_i <= \frac{\Delta_{avg}}{1-k}$

$$\gamma_i <= 3\Delta_{avg}\left(\frac{1}{k} + \frac{1-\frac{2}{d}}{1-k}\right)$$

Let's note

$$g(k,d) = \frac{1-\frac{2}{d}}{1-k}$$

We need to find a solution of the following system:

$$\begin{cases} \frac{1}{k} + g(k,d) < 2 \\ d >= \frac{2k}{1-k} \end{cases}$$

On one hand, because we have $d > 2$, we have $g(k,d) > 0$ So it means that we need to have $k > 0.5$ otherwise $\frac{1}{k} >= 2$.

On the other hand, thanks to the first inequation we have :

$$\frac{1-k}{2k} + \frac{1}{2} - \frac{1}{d} < 1 - k$$

And because of the second one we have

$$\frac{1-k}{2k} >= \frac{1}{d}$$

So

$$\frac{1}{d} + \frac{1}{2} - \frac{1}{d} < 1 - k$$

Finaly

$$k < 0.5$$

So unfortunately, there is no solution. We need to cut in half.

Then

$$\gamma_i <= 12\Delta_{avg}\left(1 - \frac{1}{d}\right)$$

And if we want this bound to be lower than $6\Delta_{avg}$ then $d = 2$ So it corresponds to the initial definition of β_i

– Part B –

Maximum degree of $4\Delta_{avg} + 7$

The algorithm partitions the nodes into two subsets: L, called the low degree nodes and H, called the high degree nodes. The low-degree nodes are the $\frac{n}{4}$ nodes with the minimal degree, and the high degree nodes are the remaining ones. Among the high degree nodes, we distinguish the set HO (resp. HI), called the high-out degree nodes (resp. high-in degree) that contain nodes whose out-degree (resp. in-degree) is greater than $2\Delta_{avg}$. Note that high degree nodes can be neither high-out nor high-in degree.

The algorithm replaces all the initial edges between high degree nodes with 2-hop paths through an intermediate node. We say that an intermediate node *helps* (is a *helper*) an edge between two high degree nodes. We say that the edges added to (and from) intermediate nodes are *intermediate edges*. The algorithm uses both low and high degree nodes as helpers. For a node i, we denote β_i as the limit on number of edges the node i can help. We denote α_i the initial degree of node i (prior to the execution of the algorithm).

We choose the values β_i in the following way:

- for $i \in L$ we set $\beta_i = \lceil \frac{2}{3}\Delta_{avg} - \frac{1}{2}\alpha_i \rceil$
- for $i \in H$ we set $\beta_i = max(\lceil \frac{2}{3}\Delta_{avg} - \frac{1}{6}\alpha_i \rceil, 0)$

We'll later justify that these values are positive.

The algorithm reduces the degree of HI (resp. HO) nodes by forming a Mehlhorn tree out of its ingoing (resp. outgoing) neighbors. For nodes that are both high in-degree and high out-degree we construct two separate Mehlhorn trees: an ingoing one and an outgoing one. It forms Mehlhorn trees out of both initial and intermediate edges.

High-in and high-out degree nodes may help other nodes (that are also H). When a high-out degree node i helps a high degree node j, we skip j while building an outgoing Mehlhorn tree of i (and similarly if i is high-in degree node). This is done for two reasons. First, this would be unnecessary because they are already connected directly, but second, we don't want a high degree node involved in too many Mehlhorn trees of its neighbors.

Now we argue that our choice of β_i is positive for low degree nodes. To this end, we sort the the nodes in ascending degree order, and then the first $\langle 1, \ldots, n/4 \rangle$ nodes are the low-degree ones and the last $\langle n/4 + 1, \ldots, n \rangle$ nodes are the high-degree ones. The sum of degrees of all nodes is then

$$n\Delta_{avg} = \sum_{i<n/4} \alpha_i + \alpha_{n/4} + \sum_{i>n/4} \alpha_i \geq \alpha_{n/4} + \sum_{i>n/4} \alpha_i \geq \alpha_{n/4}(n - n/4 + 1),$$

where the last inequality follows as for all $i > n/4$ we have $\alpha_i > \alpha_{n/4}$ (the nodes are sorted). Than for all $i \in L$

$$\alpha_i \leq \alpha_{n/4} \leq \frac{n\Delta_{avg}}{n - n/4 + 1} \leq \frac{n\Delta_{avg}}{n - n/4} \leq \frac{4}{3}\Delta_{avg},$$

and thus for all $i \in L$ we have $\beta_i \geq 0$.

Now we claim that we have sufficient number of edges available as helpers (ie., ths sum of β_i of all nodes is sufficient). Let B be the number of initial edges between high degree nodes. We need a helper for each of them, thus

$$\sum_i \beta_i \geq B \tag{B.1}$$

Recall that our algorithm helps only high degree nodes. Let A be the number of initial edges in which a low degree node is involved. In total, these sum to all edges,

$$m = A + B \ .$$

Now we bound the number of edges B the algorithm helps. Each edge counted in A involves at most 2 low degree nodes, thus $\sum_{i \in L} \alpha_i \leq 2A$. This gives us

$$B \leq m - \frac{1}{2}\sum_{i \in L} \alpha_i \ .$$

Now we show that our choice of β_i satisfies the condition (B.1). For low degree nodes L we have $\beta_i \geq \frac{2}{3}\Delta_{avg} - \frac{1}{2}\alpha_i$ and thus

$$\sum_{i \in L} \beta_i \geq \frac{2}{3} \cdot \frac{n}{4}\Delta_{avg} - \frac{1}{2}\sum_{i \in L} \alpha_i \geq \frac{m}{3} - \frac{1}{2}\sum_{i \in L} \alpha_i$$

For high degree nodes H we have $\beta_i \geq \frac{2}{3}\Delta_{avg} - \frac{1}{6}\alpha_i$ and thus

$$\sum_{i \in H} \beta_i \geq \frac{2}{3} \cdot \frac{3n}{4}\Delta_{avg} - \frac{1}{6}\sum_{i \in H} \alpha_i \geq m - \frac{1}{6}\sum_{i \in H} \alpha_i \geq m - \frac{1}{3}m,$$

where the last inequality follows from $\sum_{i \in H} \alpha_i \leq 2m$. Finally, we obtain a lower bound on $\sum_i \beta_i$

$$\sum_i \beta_i \geq m - \frac{1}{2}\sum_{i \in L} \alpha_i \geq B,$$

and we conclude that the algorithm has a sufficient number of helping edges, and thus it is well-defined.

Finally, we evaluate the maximal final degree of the nodes. Let γ_i be the final degree of the node i. The degree of each node may increase beyond its initial degree because it helps other nodes, and, more importantly, because it may participate in Mehlhorn trees of its neighbors.

A low degree node may be involved in Mehlhorn trees of all its neighbors (both initial and intermediate). The total number of the node's neighbors is the number of its initial neighbors

α_i plus at most two high degree nodes per each edge it helps, in total $\alpha_i + 2\beta_i$. A participation in each Mehlhorn tree adds at most 3 edges to a node, thus

$$\gamma_i \leq 3(2\beta_i + \alpha_i) \leq 4\Delta_{avg} + 6,$$

where the last inequality follows from $\beta_i = \lceil \frac{2}{3}\Delta_{avg} - \frac{1}{2}\alpha_i \rceil < \frac{2}{3}\Delta_{avg} - \frac{1}{2}\alpha_i + 1$.

A node i that is both high in and high out degree (i.e., $i \in HO \cap HI$), becomes the root of two Mehlhorn trees. The algorithm never involves such node in helper's Mehlhorn tree. All of i's initial edges are replaced with two edges that are leading to Mehlhorn trees that contain all of its neighbors (and possibly some intermediate nodes). The node i cannot help other nodes because $\alpha_i \geq 4\Delta_{avg}$, and thus $\beta_i = 0$, and finally $\gamma_i = 2$.

For $i \in HO \setminus HI$ (and similarly for $i \in HI \setminus HO$), we partition its initial degree α_i into initial out degree α_i^+ and initial in degree α_i^-. From $i \in HO \setminus HI$, $\alpha_i^+ \geq 2\Delta_{avg}$ and $\alpha_i^- > 2\Delta_{avg}$. All initial outgoing edges are replaced with Mehlhorn trees, so i's out-degree is 1 (from connection to the Mehlhorn tree) plus 6 per each edge it is helping (thus it is involved in at most 2 Mehlhorn trees, each adding at most 3 edges). Additionally, its in-going degree is α_i^-, as each of its neighbors is either directly connected with it, or this connection is replaced by an intermediate node (that exchanges one edge for one edge). In total, the final degree of i is

$$\gamma_i \leq 1 + \alpha_i^- + 6\beta_i$$

if $\beta_i = 0$, $\gamma_i \leq 2 + \alpha_i^- < 2 + 2\Delta_{avg}$ else

$$\gamma_i < 1 + \alpha_i^- + 6(\frac{2}{3}\Delta_{avg} - \frac{1}{6}\alpha_i + 1) = 4\Delta_{avg} - \alpha_i^+ + 7 < 2\Delta_{avg} + 7,$$

where the last inequality follows from $\alpha_i^+ > 2\Delta_{avg}$.

A node i that is neither high in nor high out degree (i.e., $i \in H \setminus (HI \cup HO)$), will not be involved in the Mehlhorn trees of its initial neighbors. The algorithm replaced its initial connections with high degree nodes using intermediate nodes. And recall that helpers do not involve the nodes they helped in their Mehlhorn trees. The final degree of i is

$$\gamma_i \leq \alpha_i + 6\beta_i < \alpha_i + 6(\frac{2}{3}\Delta_{avg} - \frac{1}{6}\alpha_i + 1) < 4\Delta_{avg} + 6,$$

where the second inequality follows from $\alpha_i < 4\Delta_i$ and $\beta_i = \lceil \frac{2}{3}\Delta_{avg} - \frac{1}{6}\alpha_i \rceil > 0$.

All of the above bounds combined guarantee that the algorithm produces a network with maximum degree of $4\Delta_{avg} + 7$.

– Part C –

Maximum degree of $3\Delta_{avg} + 8$

The algorithm arbitrarily assigns a node to *help* each edge in the demand graph. While doing so, it ensures that each node helps at most $\beta = \lceil \Delta_{avg}/2 \rceil$ edges.

To construct the network, we first construct an *auxiliary graph* G' that is initially equal to the demand distribution graph G_D. Then, we construct the network N from G'.

Now we construct the auxiliary graph G' based upon the helper nodes assignment. If the helping node k is chosen as either i or j, then do not modify any edges of G'. Otherwise, if the algorithm helps $k \neq i, j$ to help the edge (i, j), we replace the edge (i, j) in G' with 2-hop paths through k:

$$p(i, j) = 0$$

$$p(i, k) = p(i, k) + p(i, j)$$

$$p(k, j) = p(k, j) + p(i, j)$$

We say that the edges (i, k) and (k, j) added to (and from) intermediate nodes are *intermediate edges*.

Next, we construct the network N based upon the auxiliary graph G'. We start with an empty network N. In G', a node i has two types of new neighbors: the set G_i of intermediate nodes that replaced an initial edges of i, and the set H_i of nodes in whose edges i is helping. Among G_i we distinguish the set G_i^- (resp. G_i^+) of nodes that are connected with i with an ingoing (resp. outgoing) edges. For each node i, the algorithm constructs two Mehlhorn trees in N, one for G_i^- and another for G_i^+, and connects its roots to i. (reference to Figure of i,j and helper k)

Note that we skip the set H_i while building the Mehlhorn trees of neighbors of i. However, the connection (possibly indirect) between i and a node $j \in H_i$ appears while building the Mehlhorn tree of j.

Now we claim that the algorithm has a sufficient number of nodes available as helpers (i.e., the total number of available helpers $n \cdot \beta$ is sufficient to help all m edges).

$$n \cdot \beta = \frac{n\Delta_{avg}}{2} = m$$

and we conclude that the algorithm is well-defined.

Now, we upper-bound the maximal final degree of the nodes. A node i is involved in one Mehlhorn tree for each node it helped, in total at most 2β trees. Furthermore, the node i is connected with one edge to Mehlhorn trees G_i^+ and G_i^-. Note that the node i is not involved in the Mehlhorn trees of the intermediate nodes that replaced a node between i and another node. A participation in each Mehlhorn tree adds at most 3 edges to a node, thus its final degree γ_i is

$$\gamma_i \leq 6\beta + 2 \leq 6\left(\frac{\Delta_{avg}}{2} + 1\right) + 2 = 3\Delta_{avg} + 8.$$

We conclude that the algorithm produces a network with maximum degree of $3\Delta_{avg} + 8$.

Remarks. When a node is assigned to help one of its incident edges, it is the most efficient. However, the analysis holds for arbitrary assignments.

Now we claim that EPL are within the constant in comparison to optimum. This is a consequence of near-optimality of Mehlhorn trees.