Ministry of Science and Higher Education of the Russian Federation
ITMO UNIVERSITY

GRADUATION THESIS

SELF-ADJUSTING NETWORKS IN MATCHING MODEL

Author: Feder Evgeniy Alexandrovich _______________

Subject area: 01.03.02 Applied mathematics and informatics

Degree level: Bachelor

Thesis supervisor: Aksenov V.E., PhD _______________

Saint Petersburg, 2020

Student Feder Evgeniy Alexandrovich
Group M3439 Faculty of IT&P

Subject area, program/major
Mathematical models and algorithms in software engineering

Consultant(s):

a) Stefan Schmid, PhD, Professor in University of Vienna _______________

Thesis received "_____" ____________ 20___

Originality of thesis _____%

Thesis completed with grade ____

Date of defense ""

Secretary of State Exam Commission Pavlova O.N. _______________

Number of pages ____

Number of supplementary materials/Blueprints ______________________________

Ministry of Science and Higher Education of the Russian Federation

ITMO UNIVERSITY

APPROVED

Head of educational program

prof., doct. Parfenov V.G. ______________

“____” ___________ 20__

OBJECTIVES

FOR A GRADUATION THESIS

Student Feder Evgeniy Alexandrovich

Group M3439 **Faculty of** IT&P

Degree level: Bachelor

Subject area: 01.03.02 Applied mathematics and informatics

Major: Mathematical models and algorithms in software engineering

Thesis topic: Self-Adjusting Networks in Matching Model

Thesis supervisor Aksenov V.E., PhD, Researcher in ITMO University

2 Deadline for submission of complete thesis: “”

3 Requirements and premise for the thesis

The goal here is to try to generalize Self Adjusting Network algorithms. In all this alogorithms the cost of forwarding along one link is 1, and the cost of changing a link is also 1. In real networks, the cost of changing a link is higher than the cost of forwarding along one link. So we want to look at Self-Adjusting algorithms in a model where: the cost of forwarding along one link is 1, and the cost of changing a link is α>1, for some parameter α.

4 Content of the thesis (list of key issues)

Graduation thesis must explain basic definitions that will be explored: Self Adjusting Networks, Static Optimality and introduce Model in which we will work. Then thesis must explain basic algorithms that will used and then explain their modification and proof, that they better than straightforward known implementations.

5 List of graphic materials (with a list of required materials)

Supplementary materials and blueprints not required

6 Source materials and publications

a) Demand-Aware Networking:A Theory for Self-Adjusting Networks, by Chen Avin and Stefan Schmid;
b) ProjecToR: Agile Reconfigurable Data Center Interconnect, by Monia Ghobadi et al;
c) SplayNet: Towards Locally Self-Adjusting Networks, by Stefan Schmid, Chen Avin, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, Zvi Lotker.

7 Objectives issued on “”

Thesis supervisor ______________

Objectives assumed by ______________ “”

Ministry of Science and Higher Education of the Russian Federation

ITMO UNIVERSITY

SUMMARY
OF A GRADUATION THESIS

Student: Feder Evgeniy Alexandrovich
Title of the thesis: Self-Adjusting Networks in Matching Model
Name of organization: ITMO University

DESCRIPTION OF THE GRADUATION THESIS

1 Research objective: Adapt data structures for Matching Model and get algorithms that will have less than $O(\alpha)$ multiplier for the static optimality complexity.
2 Research tasks:

a) introduce a model which will approximate Self Adjusting Networks in data centers.
b) look for subcases of Self Adjusting Networks and invent idea to adapt them for model.
c) get static optimality complexities for the invented algorithms.

3 Number of sources listed in the review section: 12
4 Total number of sources used in the thesis: 12
5 Sources by years:

Russian			Foreign		
In the last 5 years	5 to 10 years	More than 10 years	In the last 5 years	5 to 10 years	More than 10 years
0	0	0	7	0	5

6 Use of online (internet) resources: none
7 Use of modern computer software suites and technologies: none
8 Short summary of results/conclusions

Lazy adaptations algorithms turned up better than versions of algorithms in Standard Model

9 Grants received while working on the thesis

10 Have you produced any publications or conference reports on the topic of the thesis?

Congress of Young Scientist

Student Feder E.A. _______________

Thesis supervisor Aksenov V.E. _______________

“_____” ____________ 20___

CONTENTS

INTRODUCTION 5
1. Introduction to the subject area and goals for the work 7
   1.1. The topology of the networks 7
   1.2. Self-Adjusting Networks 7
   1.3. Standard and Matching Model 9
   1.4. Static Optimal algorithms 11
   1.5. Related work and Overview 12
      1.5.1. Line Network 12
      1.5.2. Binary Tree Network 12
      1.5.3. Network with bounded degree 14
   1.6. Goals and Objectives 17
   Conclusions on Chapter 1 18
2. Lazy Nets 19
   2.1. List Topology with search requests 19
   2.2. Tree Networks 22
      2.2.1. Idea of Lazyness 22
      2.2.2. Search requests 23
      2.2.3. Routing requests 27
   2.3. Network with bounded degree property 29
   Conclusions on Chapter 2 32
CONCLUSION 34
REFERENCES 35

INTRODUCTION

Nowadays, the traffic in data centers becomes incredibly high. Thus, researchers all over the world want to reduce the communication cost between the nodes. Currently, networks in data centers are static and optimized for the worst case load. For example, one of the ways is to adapt a static structure for the traffic by reducing the routing distance between frequently communicating nodes.

Despite the fact that most of the state-of-the-art algorithms target to build optimized static network [2, 5–8] the recent research made it possible to dynamically change parts of the physical network [9]. Thus, it is now possible to have a dynamic network topology, by paying additional cost for the changes. One of the arising questions is how to create low-cost algorithms that change a network depending on the communication requests dynamically.

Usually, dynamic algorithms are provided in a model where the cost of passing along one link is one, and the cost of changing a link is also one. We call this model as *Standard Model* (SM). However, in next-gen real-life networks [9] we can change several links in parallel. So, SM does not reflect the real-life opportunities.

In this work, we want to tighten the gap between theory and practice. We talk about recently introduced model [1] — Matching Model (MM), in which the cost of passing along one link is one, and the cost of changing the whole network is $O(\alpha)$, for some fixed parameter $\alpha > 1$. Note that α can depend on the size of the network.

We investigate Self-Adjusting algorithms on the most popular types of networks: List, Binary Trees and Graph with bounded degree — in Matching Model, introduced here. The state-of-the-art algorithms: Move To Front [10] for List, Splay Tree [11] and SplayNet [12] for Binary Tree, ReNet [3] for Graph with bounded degree — applied straightforwardly, give us $O(\alpha)$ complexity of static optimality. We find $O(\alpha)$ bound to be too loose and we decide to adapt these algorithms for the new model. Our modification of Move To Front algorithm is $O(1)$-static optimal. For other algorithms, we introduce "Idea of Lazyness" that allows us to get a better ratio — $O(\min(\sqrt{\alpha}, \log|V|))$-static optimality in MM.

In Chapter 1, we provide the main definitions that we use in this work, for example, Self-Adjusting Networks and static optimality. We consider state-of-the-art Self-Adjusting algorithms on List, Binary Tree and Graphs with bounded degree. Finally, we talk about motivation and set a goal of our work. In Chapter 2, we present our modifications of the algorithms from Chapter 1 and provide the proofs of

their complexity of static optimality. In conclusion, we briefly overview the results presented earlier and describe directions for further studies.

The problem description and model were found together with the supervisors (it took approximately 3 month). All other results were obtained by the student, under the supervision. This work was presented on Congress of Young Scientist and it won in “report contest” section.

CHAPTER 1. INTRODUCTION TO THE SUBJECT AREA AND GOALS FOR THE WORK

In this chapter, we introduce basic definitions such as Self-Adjusting Network (SAN) algorithms, static optimality, and we define two models, Standard Model and Matching Model. Finally, we recall self-adjusting algorithms in SM on List, Binary Trees and Graph with bounded degree. Finally, we talk about motivation and set a goal of our work.

1.1. The topology of the networks

Let computational nodes and the connections between them form an undirected graph $G = (V, E)$, where G belongs to $\mathcal{G}$ and $\mathcal{G}$ is a topology, or family of undirected graphs.

In this work we consider three different topologies: List, Binary Tree and Graph with bounded degree.

Definition 1. *Connected topology* is a family of graphs, in which there exists a route between all nodes.

Definition 2. *List topology* is a connected topology, where two nodes (*head* and *tail*) have degree one, and all other nodes have degree two.

Definition 3. *Tree topology* is a connected topology, where each but one (*root*) node has *parent*. If each node has zero or two childs we name this as *Binary tree topology*.

Definition 4. Topology G satisfies *bounded degree property*, if a degree of each node is not more than constant Δ.

1.2. Self-Adjusting Networks

In this subsection we formalise the definition of Self-Adjusting Network algorithm and other related notions.

Definition 5. Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m) = ((u_1, v_1), (u_2, v_2), \ldots, (u_m, v_m))$, where $u_i, v_i \in V$, be a sequence of *routing requests* to forward a packet in between the pairs of nodes u_i and v_i.

Definition 6. If we assume that our topology has a distinguished node S, e.g., front for List and root for Trees, then instead of routing requests we perform *search requests* from node S (i.e., $u_i = S$ for all i) and the notation of these requests will be $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$, where $\sigma_i \in V$.

Our main goal is to build a graph, which will optimize the total cost for processing all requests. We can solve this task statically, i.e. the graph do not change during the execution, or dynamically, i.e. the graph can be changed.

Definition 7. *Static optimization task* is a task when given requests a priori we build a graph $G \in \mathcal{G}$ and this graph does not change until the end of all requests. This graph G needs to optimize the total cost function $\text{sumCost}(\text{static}, G, \sigma) = \sum_{i=1}^{m} l_i$, where l_i is a length in edges to process request σ_i and "static" means that we do not change G during requests.

One of the example of the static optimization task is "Building *Demand-Oblivious Network* task" [4]. In it we need to build a static network, which optimizes the cost of the "worst-case" request. If we know an additional information about requests we can build a more optimal network. For example, if we know the distribution of requests we can build *Demand-Aware Network* [4], which consider some pair of nodes to communicate more frequently than another.

Definition 8. In *dynamic optimization task*, we assume that we can change our graph after each request. Before requests, we are provided with an arbitrary graph $G_0 \in \mathcal{G}$. Our task is to build an algorithm $\mathcal{A}$ that adjusts our network and minimizes the total cost, which is calculated as $\text{sumCost}(\mathcal{A}, G_0, \sigma) = \sum_{i=1}^{m} (l_i + cost(G_i, G_{i+1}))$, where l_i is a length in edges of G to process request σ_i and $cost(G_i, G_{i+1})$ is an adjusting cost to rebuild the network from step i, $G_i \in \mathcal{G}$, to step $i + 1$, $G_{i+1} \in \mathcal{G}$. Example of this adjustments are presented on Figure 2 and 3 for two (routing and search) types of requests.

In this work, we assume that all changes to the network graph are controlled by a coordinator C. The coordinator is connected to all nodes from V and controls a structure of the network. We neglect the communication cost with the coordinator. Example of such a graph is presented on Figure 1.

Definition 9. The algorithm to perform the requests and adjust the network at each step from $G_i \in \mathcal{G}$ to $G_{i+1} \in \mathcal{G}$ is called *Self-Adjusting (Network) algorithm*.

On Figure 2 from [4] we can see an illustration of Self-Adjusting algorithm for routing requests. At first, we find a route between two nodes. Then we adjust our network hoping to exploit temporal locality as these requests are also likely to be relevant soon. And on Figure 3 from [4] we can see illustration of Self-Adjusting algorithm "Splay Tree"[11] performing search requests. After finding 7 we splay this node to the root.

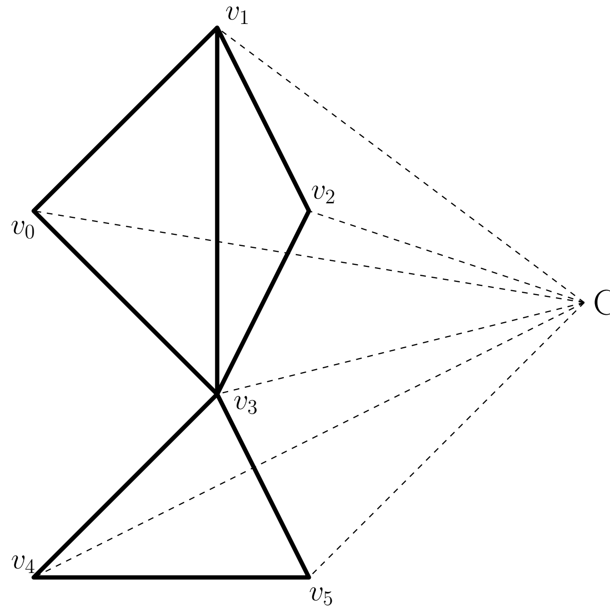

Figure 1 – All nodes are (logically and possibly physically) connected to node C, which can request arbitrary topology changes and has a complete information about the network and its statistics.

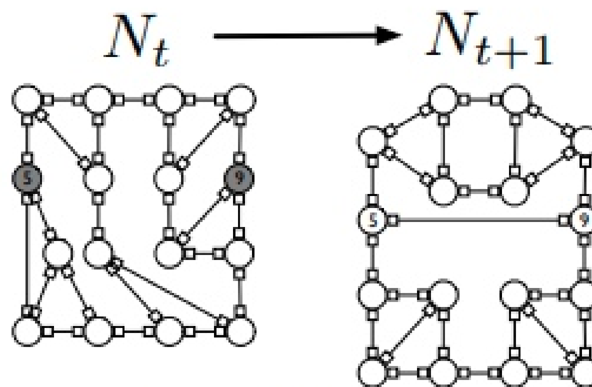

Figure 2 – Self-Adjusting algorithm for routing request illustration from [4].

1.3. Standard and Matching Model

Let algorithm $\mathcal{A}$ be a Self-Adjusting algorithm. We need to specify the cost of each operation.

Definition 10. In *Standard Model (SM)* the cost of changing one edge to another one is equal to 1. Thus, $\text{cost}(G_i, G_{i+1}) = r_t$, where r_t is a number of edge changes between G_i and G_{i+1}.

In this work, we talk about recently introduced model [1] — Matching Model (MM). The idea for this model comes from the physical properties of a next-gen dynamic network based on laser switches [9]. Each physical node has a constant

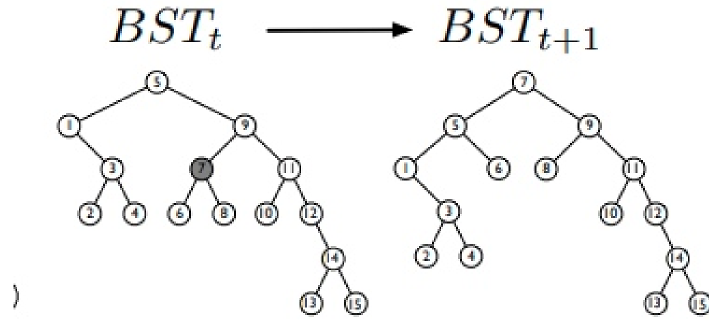

Figure 3 – Self-Adjusting Networks for search requests illustration from [4].

number Δ of physical sockets — a sender and a receiver. One socket represents one possible connection with another node.

Definition 11. *Edge coloring of the graph* is a color function $C : E \to \{1, \ldots, \Delta\}$, where incident edges have different color values.

Theorem 12 (Vising's theorem). Every undirected graph with bounded degree Δ can be provided with edge coloring using at most $\Delta + 1$ different colours.

Let Δ be the maximum degree of any graph from $\mathcal{G}$. Thus, by Vising's theorem, at each step t the edges of graph G_t can be edge colored using at most $\Delta + 1$ colors. Thus, our graph G_t can be represented by $\Delta + 1$ matchings (Figure 4).

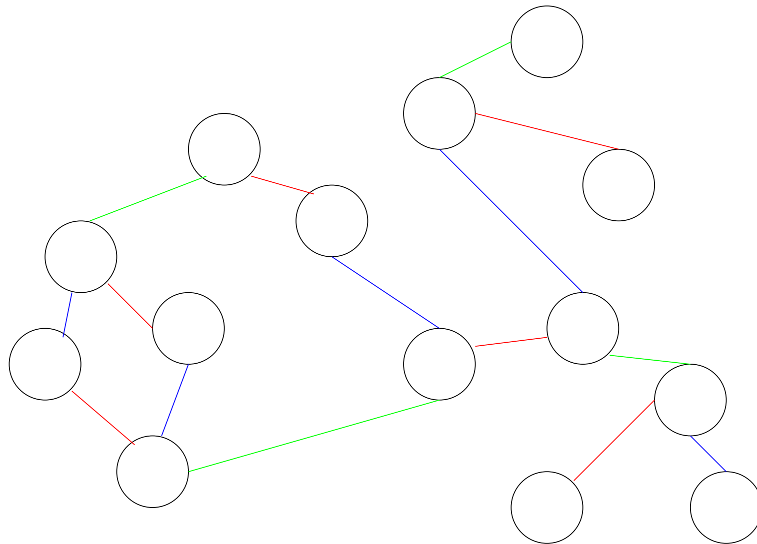

Figure 4 – Graph which is represented by matchings. The edges of one color represent one matching.

In MM, the coordinator C can perform the following command: ask all nodes simultaneously to change the edges of the matching with the chosen color paying the cost α. Since G_t has constant number of matchings (more precisely, $\Delta + 1$) by

Vising theorem, the network can be changed from G_t to any $G_{t+1} \in \mathcal{G}$ by paying the cost $\alpha \cdot (\Delta + 1) = O(\alpha)$.

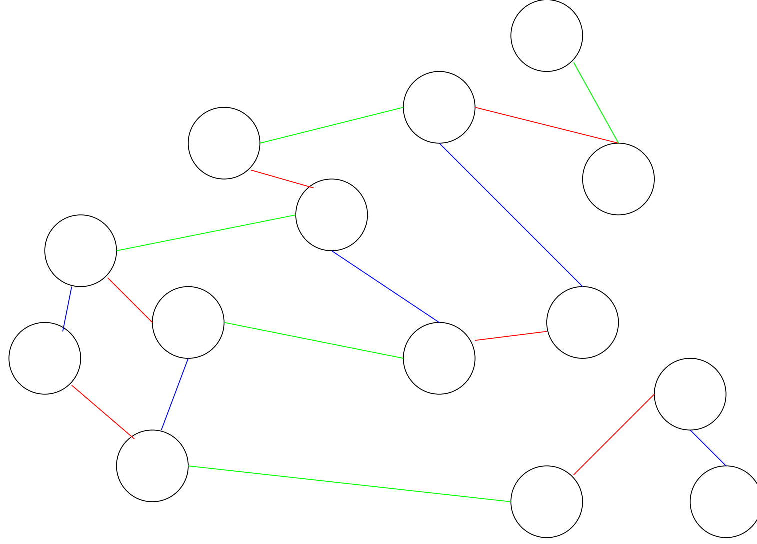

Figure 5 – The coordinator ask to change green matching from Figure 4.

Definition 13. In MM, the adjustment cost is equal to $\alpha \cdot m_i$, where m_i is the number of matchings that we changed between G_i and G_{i+1} on i-th request.

Note that the discussed cost of $O(\alpha)$ counts both the computation of the new topology G_{i+1} inside the coordinator C and the physical reconfiguration cost. In other words, we say that the computation of the new topology is small in comparison to the physical adjustment.

1.4. Static Optimal algorithms

We already discussed static algorithms, in which requests are known in advance, but the network can not change. However, in real life requests are not known a priori and by that we should consider dynamic networks. To reason about a complexity of dynamic algorithms the following notion is provided.

Definition 14. The network configuration $OPT(\sigma) \in \mathcal{G}$ is *static-optimal* on the sequence of requests σ if $\text{sumCost}(\text{static}, OPT(\sigma), \sigma)$ is less than $\text{sumCost}(\text{static}, H, \sigma)$ for any configuration $H \in \mathcal{G}$, where “static” means that a network does not change

Self-Adjusting algorithm $\mathcal{A}$ is called *c-statically optimal* if for each sequence of requests σ and for each start configuration G_0, $\text{sumCost}(\mathcal{A}, G_0, \sigma) \leqslant c \cdot \text{sumCost}(\text{static}, OPT(\sigma), \sigma)$

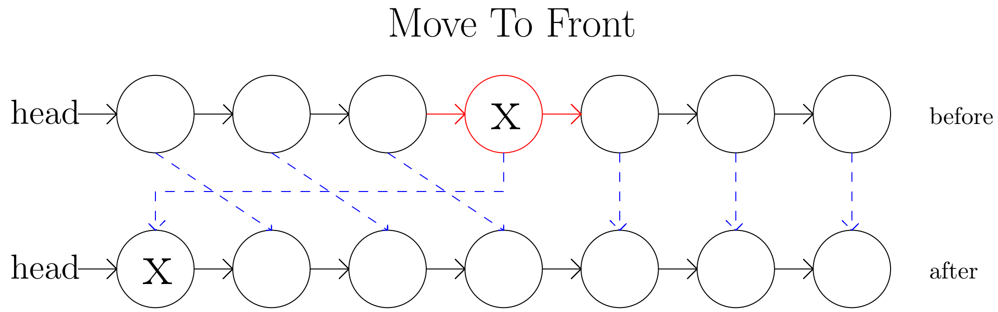

Figure 6 – Move-To-Front algorithm on request x

1.5. Related work and Overview

1.5.1. Line Network

The first type of considered networks is Line Network. Line Network is a network with Line Topology. One of the most popular dynamic algorithm to perform search requests on Line Network is Move-To-Front (MTF) by Sleator and Tarjan [10]. In a few words, after an access to the element x MTF moves x to the front of the list (see it at Figure 6). This algorithm is interesting for us since it is $O(1)$-statically optimal in Standard Model.

In Chapter 2, we introduce a Lazy modification of the MTF algorithm which appears to be $O(1)$-statically optimal in Matching Model.

The proof of $O(1)$ static-optimality in [10] uses the idea of amortized analysis with the potential function $\phi(x)$ which is the number of inversions in MTF list, i.e., the list maintained by MTF algorithm, with respect to the static optimal list (OPT). An inversion is a pair (x, y) of items such that x occurs before y in Move-To-Front list and x occurs after y in OPT list. We assume without loss of generality that Move-To-Front and OPT start with the same list so that the initial potential is 0.

Using such a potential function ϕ we can prove that the amortized cost of the MTF list is strictly less than twice the cost of OPT. Thus, MTF is $O(1)$-statically optimal.

1.5.2. Binary Tree Network

The second type of a considered typologies is a Binary Tree topology. Binary Tree Network is a network with Binary Tree topology. At first, let us consider search requests from the root of the tree and then let us discuss routing requests.

1.5.2.1. Search requests

The first type of requests are search requests. We remind that for these requests in Binary Tree Topology we have a special node, named “root”, and for all requests one of the nodes in pair is always the root.

One of the well-known Self-Adjusting Binary Trees is $\mathcal{ST}$, Splay Tree algorithm, introduced by Sleator and Tarjan [11]. One of the most important property of $\mathcal{ST}$ is that it is $O(1)$-statically optimal in SM, like MTF algorithm. Another point why we consider Splay Tree is that it is used as a basic block of the ReNet structure, introduced later in Section 1.5.3.

Suppose we are given search request x. At first, we search x like in a standard search tree. Then the algorithm performs the *splay* operation. The splay operation moves x to the root step-by-step. On each step, $\mathcal{ST}$ applies one of the operations: zig, zig-zig and zig-zag. You can see the illustration of each operation on Figure 7.

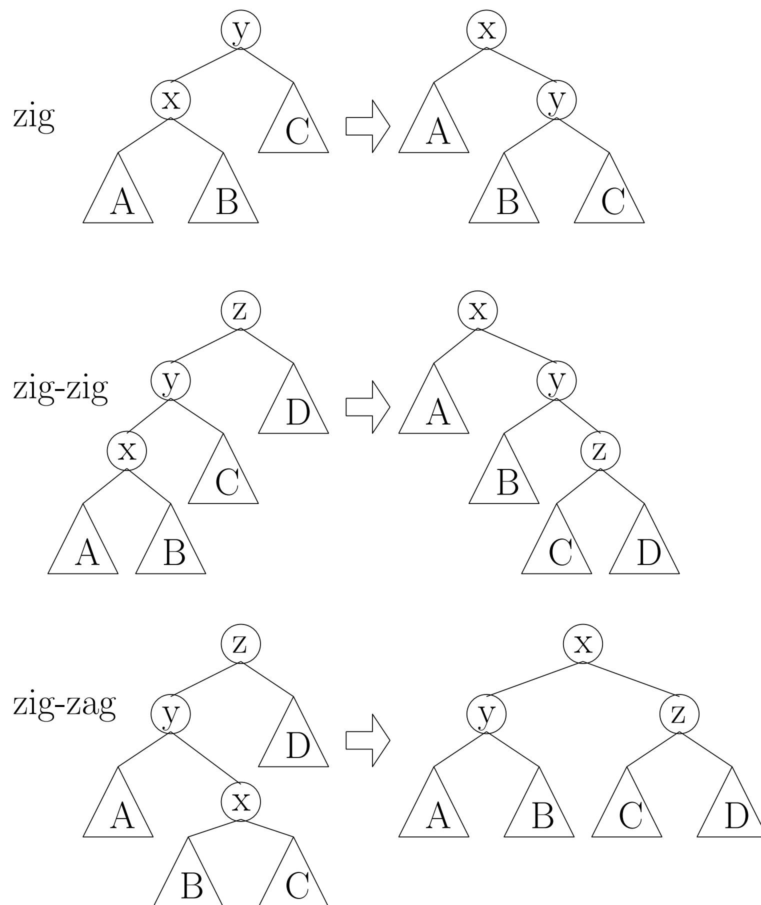

Figure 7 – Illustrarion of the splay step types.

Obviously, the straightforward version of $\mathcal{ST}$ in MM is $O(\alpha)$-statically optimal since a change of one edge costs $O(\alpha)$.

In Chapter 2 we explain how to modify Splay Tree such that it becomes $O(\sqrt{\alpha})$-statically optimal.

1.5.2.2. Routing requests

In the previous subsection we consider requests that are performed from the root. In this section, the data structure tries to answer general requests between any pair of nodes. In 2015, Schmid et al. [12] presented the study of self-adjusting data structures and introduced a new self-adjusting Binary Search Tree under general requests called *SplayNet*.

We shortly describe how it works. Consider a request (u, v). Node u asks the coordinator C for the common ancestor of u and v. Let it be node p. The route by which we pass a packet is $u-p-v$. After we find the route, we “splay” u to the place of p and then “splay” v to the position of the son of u. By that, SplayNet is a natural generalization of the classic Splay Tree algorithm which “splays” communication partners to their common ancestor. On Figure 8 you can see an example on how to process a request from node 1 to node 5.

For this data structure we do not have a complexity of static optimality, however, in Chapter 2 we propose its version obtained by the application of our “Idea of Lazyness”.

1.5.3. Network with bounded degree

Network with bounded degree (Bounded degree network) is a network of the topology that satisfies the bounded degree property. In this subsection, we consider ReNet algorithm [3].

Ideally, we would like to have each node $u \in V$ in graph be connected directly to all its communication partners in σ, achieving an ideal route length of one. However, this is infeasible, as (1) the communication partners of u are not known a priori and (2) u may have too many communication partners that would result in a large degree of u while our topology has bounded degree property.

ReNet is an algorithm, that solves the problem of large degree in a specific case where the request sequence is *sparse*. Sparse request sequence is a sequence in which a number of different communications pairs is linear, i.e., $O(|V|)$, while in the general case this number can be $O(|V|^2)$. If this precondition is satisfied, ReNet algorithm maintains a graph with bounded degree property: $\deg(v) \leqslant 6 \cdot \theta$ for all vertices v, where θ is a parameter of ReNet.

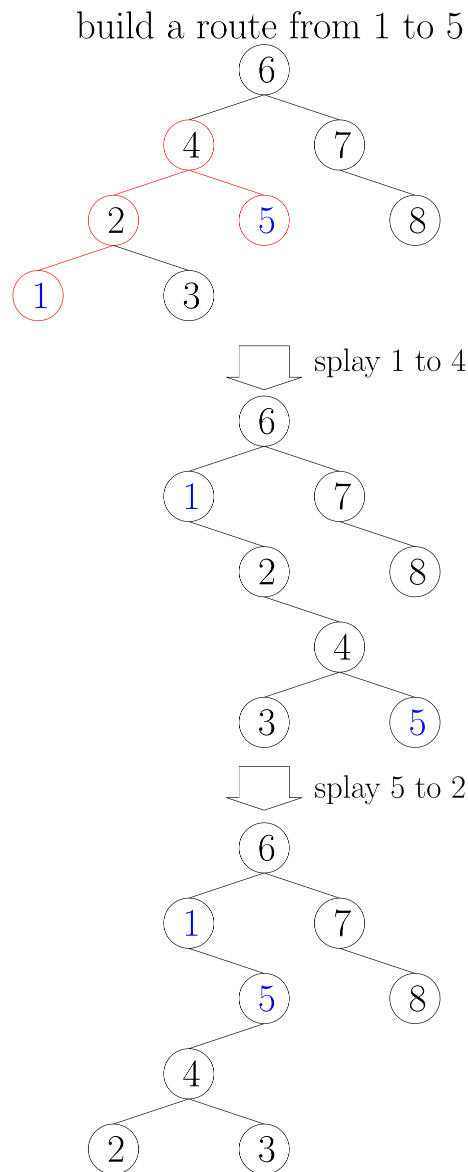

Figure 8 – Example on how to build a route and adjust in the SplayNet algorithm

We begin an explanation of ReNet by providing the main definitions for this structure.

The structure of ReNet is the following:

- Each node u in ReNet keeps track of its recent active communication partners, hoping to exploit temporal locality as they are also likely to be relevant in the near future. So, let *Working set*, $W_t(u)$, be the set of nodes v at request t such that either (u,v) or (v,u) is in $\{\sigma_r, \sigma_{r+1} ... \sigma_{t-1}\}$, where r is a moment of last reset (we explain what *reset* is later).
- Each node in a network can be one of two types. Node u is **small**, when $|W(u)| < \theta$, otherwise, the node is **large**. If the node is small it will connect to its communication partners directly. When the node becomes large it or-

ganizes partners into the Splay Tree (in paper these trees are called *ego-tree* of node u or $ego(u)$). The examples of the network topologies of small and large nodes can be seen on Figure 9.

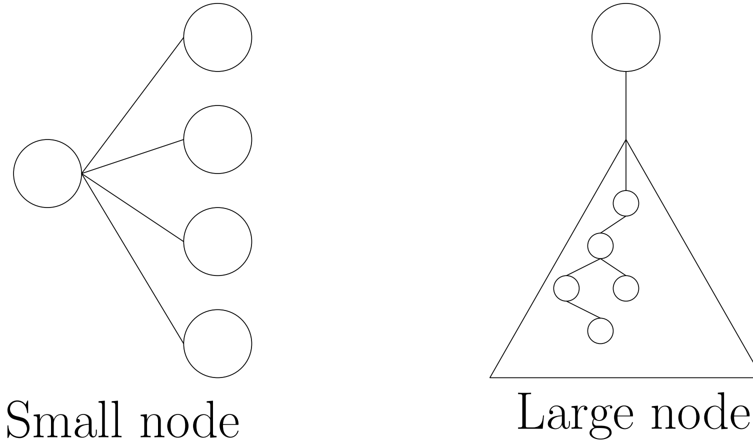

Figure 9 – Types of nodes in ReNet

— In addition, we have a *reset operation* that provides temporal locality. When $\sum_{u \in V} |W(u)|$ becomes $n \cdot \theta/2$ we clear everything: we remove all edges from the network and re-initialize all working sets. Then, we simply continue with requests.

— If we want to add an edge between nodes, we simply ask the coordinator C to add the edge by paying the cost c.

Below, we describe how to process a request from u to v.

a) Both nodes u and v are small: we add an edge between them (if it does not exists) and we transfer a packet along this edge.

b) One node is small and the other one is large: if nodes are not connected then we add the small node to the ego-tree of the large node and build a route via this Splay Tree.

c) Both nodes are large: if nodes are not connected then the algorithm chooses some small node h which we add to both ego trees. Later such a node is called *helper* node. You can see an example of helper nodes on Figure 10. Then, the route between large nodes is built through h. However, we have to note that sometimes one node lies in the ego-tree of another node and we can process the request like in the small-large case.

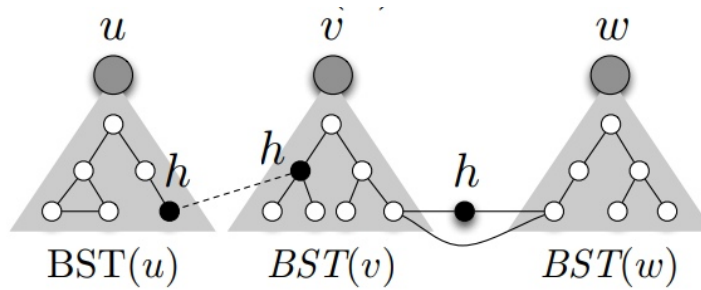

Figure 10 – Example of helper nodes from ReNet paper [3]

On the Figure 11 we can see an example of ReNet topology. u and v are large nodes and h is the helper node, which helps to build a route between u and v. x and y are small and are connected directly.

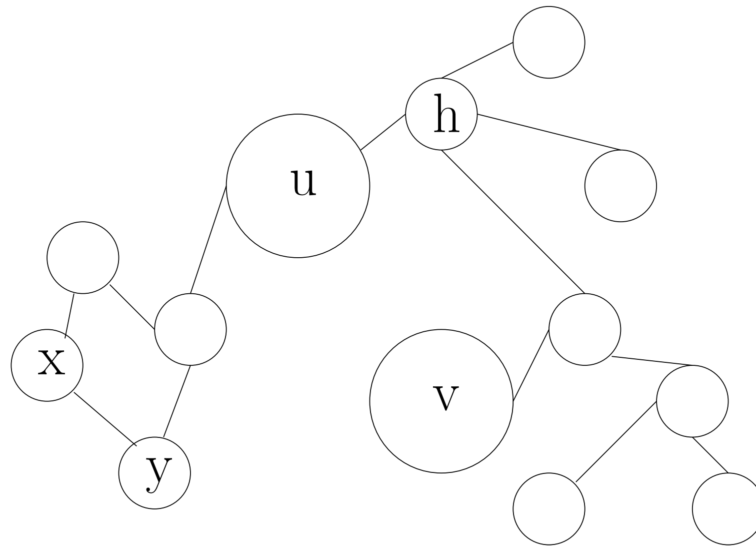

Figure 11 – Example of ReNet network.

1.6. Goals and Objectives

In this work, we consider all the algorithms described above in MM. In Matching Model the change of one edge costs α. This means, that if we take the algorithms above as they are presented in SM, the algorithms will have the static optimality complexity to be $O(\alpha)$ in MM which is too high.

Theorem 15 (Motivation for the work). Complexity of static optimality for presented algorithms in Standard Model is $O(\alpha)$ times higher in Matching Model.

Proof. For MTF we can get a lower bound on the complexity of static optimality. Suppose we start with the list $\{x_1, x_2, x_3, \ldots, x_n\}$. We create a sequence of requests $\sigma = \{x_2, x_1, x_2, x_1 ....\}$. This sequence always searches for the second element of

the list. In OPT list we perform $m = |\sigma|$ operations and in MTF we pay $O(\alpha m)$ cost. So, our complexity of static optimality is at least α.

We can apply the same idea to Splay Tree and SplayNet algorithms. The cost of routing will be one or two, while the cost of the adjustment is α. Because ReNet is based on Splay Tree, the complexity of static optimality of ReNet becomes also α times more. □

Our goal is to adapt these algorithms for Matching Model such that they have complexity of static-optimality smaller than $O(\alpha)$.

Conclusions on Chapter 1

In this chapter, we introduced the main definitions used in our work. Firstly, we described two notions from the title: Self-Adjusting Network algotithm and Matching Model. Then, we defined property by which we compare the algorithms — the complexity of static optimality. After that we overviewed the state-of-the-art self-adjusting algorithms for three cases of the network topology: List, Binary Tree and Graph with bounded degree. Finally, we show that naive approaches give awful complexity bound and we set a goal to reduce it.

CHAPTER 2. LAZY NETS

In the previous chapter we defined Matching Model and overviewed the state-of-the-art algorithms in the Standard Model. However, as mentioned, the naive usage of the state-of-the-art algorithms give us awful complexity bounds in Matching Model. In this chapter, we adapt these algorithms for MM using our "Idea of Lazyness" and prove their complexities of static optimality.

2.1. List Topology with search requests

We start with the networks that satisfy List Topology. In Standard Model we are provided with $O(1)$-statically optimal Move To Front algorithm [10]. We note that in Matching Model *"move-to-front"* operation during the request after an access of an element costs α. We want to amortize this cost by adjusting the network not at each search request.

Surprisingly, we found that quite a straightforward optimization of the MTF algorithm gives a desired theoretical bound:

— Maintain a counter for each node, being zero at initialization.
— On each request of a value, we increase the counter of the corresponding node by one.
— If the counter becomes α, we perform move-to-front operation on this node.

Such a modification allows us to get $O(1)$-statically optimal algorithm in MM. We name it "Lazy Move-To-Front". For the proof of the complexity we use the following theorems from the original paper [10].

Theorem 16. Let $p_t(x)$ be the position of element σ_t in the MTF list at time t and j be the position of element σ_t in the OPT list. Then, $j \geqslant p_t(x) - \phi_t(x)$, where $\phi_t(x)$ is the MTF potential function of element x at the request t, which is the number of inversions in MTF list with respect to the static optimal list (OPT).

Theorem 17. $\text{cost}(\sigma_t) + \Delta\phi_t(x) = 2 \cdot (p_t(x) - \phi_t(x)) - 1$

We re-prove these theorems in the proof of the following theorem.

Theorem 18. "Lazy Move-To-Front" algorithm is $O(1)$-statically optimal in Matching Model if $|\sigma| \geqslant a \cdot \frac{n(n+1)}{2}$.

Proof. We prove this Theorem using a standard amortization argument. For that we introduce a potential function for the Lazy MTF after t requests: $\Phi_t = \sum \alpha \cdot \phi_t(x)$, where $\phi_t(x)$ is the MTF potential function of element x, which is the number of inversions in MTF list with respect to the static optimal list (OPT). An inversion is

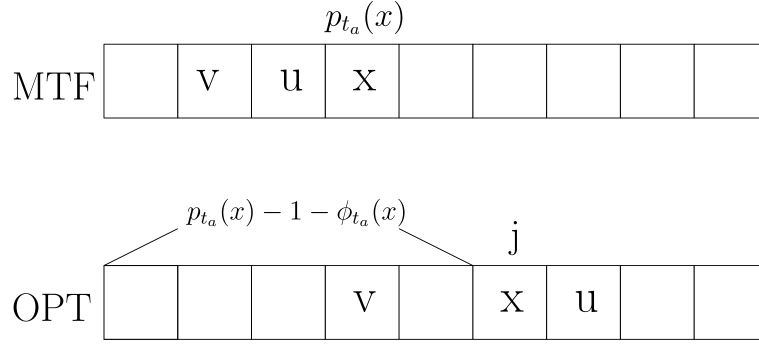

Figure 12 – Illustration for the Theorem 16 and 17. u increases $\phi_{t_a}(x)$ by 1 and v does not affects into $\phi_{t_a}(x)$ (number of nodes like v is not more than $(p_{t_a}(x) - 1 - \phi_{t_a}(x))$). After we move to front $\phi_{t_a+1}(x) = 0$, x starts increase $\phi_{t_a+1}(v)$ and do not affect on u node

a pair (x, y) of items such that x occurs before y in Move-To-Front list and it occurs after y in OPT list. We assume without loss of generality that Move-To-Front and OPT start with the same list, so, the initial potential is zero.

Our potential function changes only on the α-th request per element. So, we split the requests to the fixed element x into the blocks of length α. Now, we want to compare the total cost of requests in each block in OPT and Lazy MTF list.

a) The cost on OPT list is straightforward. It is equal to $\alpha \cdot j$, where j is the position of x in OPT list.

b) Consider α requests to the element x. Let $p_{t_i}(x)$ be the position of x in Lazy MTF after t_i-th request where i is the index of the request to x and t_i is the index of the request in σ. Thus, the total cost of the first $\alpha - 1$ requests that access x is equal to $\sum\limits_{i=1}^{\alpha-1} p_{t_i}(x)$.

Let us perform the α-th access to x. We briefly revise the proof of Theorem 17 and use the same amortization idea here.

The change in the MTF potential due to moving x to the front is equal to the sum of two contributions (you can see these contributions on Figure 12): 1) x no longer contributes anything to the potential, as it no longer has any elements in front of it, so, we have a decrease of the potential by $\phi_{t_a}(x)$; 2) all $p_{t_a}(x){-}1$ elements that were in front of x now have an additional element in front of them, x itself, and thus their potential might be increased by one. The total number of these elements is $p_{t_a}(x) - 1 - \phi_{t_a}(x)$, because all nodes, which did not affect on $\phi_{t_a}(x)$ before, are affected now. So, a total change of MTF potential of x is at most $-\phi_{t_a}(x) + (p_{t_a}(x) - 1 - \phi_{t_a}(x)) = p_{t_a}(x) - 1 - 2 \cdot \phi_{t_a}(x)$.

By that, we measures how the potential changes after the move-to-front adjustment. Thus, the amortized cost of the α-th operation is equal to:

$$p_{t_\alpha}(x) + \Delta\Phi = p_{t_\alpha}(x) + \alpha \cdot (p_{t_\alpha}(x) - 1 - 2 \cdot \phi_{t_\alpha}(x))$$

Then the total cost of all α operations in a block is equal to:

$$\begin{aligned}
&\sum_{i=0}^{\alpha-1} p_{t_i}(x) + p_{t_\alpha}(x) + \Delta\Phi = \\
&= p_{t_0}(x) + p_{t_1}(x) + ... + p_{t_\alpha}(x) - \alpha \cdot \phi_{t_\alpha}(x) + \alpha \cdot (p_{t_\alpha}(x) - 1 - \phi_{t_\alpha}(x)) \\
&= (p_{t_0}(x) - \phi_{t_\alpha}(x)) + (p_{t_1}(x) - \phi_{t_\alpha}(x)) + ...+ \\
&+ (p_{t_\alpha}(x) - \phi_{t_\alpha}(x)) + \alpha \cdot (p_{t_\alpha}(x) - \phi_{t_\alpha}(x) - 1)
\end{aligned} \tag{1}$$

At first, we know that $p_{t_0}(x) \leqslant p_{t_1}(x) \leqslant ... \leqslant p_{t_\alpha}(x)$ since the node can be only moved further the list due to other move-to-front nodes. Secondly, from Figure 12, we can see that $(p_{t_a}(x) - 1 - \phi_{t_a}(x)) \leqslant j - 1$, so $(p_{t_a}(x) - \phi_{t_a}(x)) \leqslant j$. We remind that j is the position of x in OPT. By these two statements we can conclude that $(p_{t_i}(x) - \phi_{t_\alpha}(x)) \leqslant (p_{t_\alpha}(x) - \phi_{t_\alpha}(x)) \leqslant j$. By making a substitution into equation (1) we get

$$(1) \leqslant j + j + ... + j + \alpha \cdot (j - 1) \leqslant \alpha \cdot (2 \cdot j - 1) < 2 \cdot \alpha \cdot j.$$

So, the total cost is strictly less than twice the cost of OPT list.

In the statement of the theorem we have a restriction on the number of requests σ. If $|\sigma| \geqslant a \cdot \frac{n(n+1)}{2}$ we do not have to think about requests in the "tail" of σ (i.e., the requests which belong to non-finished block and do not have move-to-front operation afterwards). It is due to the fact that $\text{sumCost}(\text{Lazy MTF}, G_0, \sigma) = \omega(an^2)$ and all requests from the tail, which are not going to be amortized by move-to-front, can be amortized over all $|\sigma|$ requests: at each node we can set counter to $a - 1$ and the potential is $O(an^2)$. □

2.2. Tree Networks

2.2.1. Idea of Lazyness

Suppose for a moment that we are provided with a self-adjusting data structure in SM and we want to adapt it to MM with the static-optimality complexity better than $O(\alpha)$. The general idea is to perform adjustments only once in a while — similarly to what we did for MTF. Obviously, there are a lot of different variants of how to use this idea. In this subsection, we give the details of our idea and provide the proof of its bound on the static-optimality complexity.

Let *ALG* be the algorithm in SM and we have a sequence of requests σ. Denote our algorithm as *LAZY*. Let us divide the list of requests, σ, into *epochs*. During one epoch the data structure maintained by *LAZY* algorithm remains unmodified and the data structure maintained by *ALG* adjusts exactly as in SM. Epoch continues until the total cost of operations in *LAZY* modifications exceeds α. After that *LAZY* synchronizes its data structure with the data structure of *ALG* and *LAZY* moves to a new epoch.

In SAN, *LAZY* algorithm works on the structure of the network itself, while *ALG* emulates the network inside the coordinator. Of course, at that point, we make a reasonable assumption that computations of *ALG* inside the coordinator cost much less than transferring information between nodes. So, we simply ignore the cost of the maintenance of *ALG* on the coordinator.

We are going to find the static optimality complexity of our algorithm in Matching Model by formula: $\frac{\text{sumCost}_{MM}(ALG,G_0,\sigma)}{\text{sumCost}_{MM}(static,OPT,\sigma)}$, where G_0 is an arbitrary configuration before at the beginning and OPT is a static optimal configuration for requests σ. Let us multiply and divide this ratio by $\text{sumCost}_{SM}(ALG, G_0, \sigma)$. By that we obtain $\frac{\text{sumCost}_{MM}(LAZY,G_0,\sigma)}{\text{sumCost}_{SM}(ALG,G_0,\sigma)} \cdot \frac{\text{sumCost}_{SM}(ALG,G_0,\sigma)}{\text{sumCost}_{MM}(\text{static},OPT,\sigma)}$. We know that $\frac{\text{sumCost}_{SM}(ALG,G_0,\sigma)}{\text{sumCost}_{SM}(\text{static},OPT,\sigma)} = c_{ALG}$, i.e., static-optimality complexity of *ALG* in Standard Model, and $\text{sumCost}_{MM}(\text{static}, OPT, \sigma) = \text{sumCost}_{SM}(\text{static}, OPT, \sigma)$ since OPT structure does not change. Thus, we get the complexity equal to $\frac{\text{sumCost}_{MM}(LAZY,G_0,\sigma)}{\text{sumCost}_{SM}(ALG,G_0,\sigma)} \cdot c_{ALG}$. In Splay Tree and ReNet algorithm we know the complexity c_{ALG} of static optimality — it is equal to one. For SplayNet we do not know the value of c_{ALG}, so we provide a bound that depends on it.

Let us split now the dividend and the divider of the ratio into epochs. Let i be the index of an epoch and m be the number of epochs. Suppose that G_i is the graph right after the i-th epoch and $\sigma^{(i)}$ be the requests performed during i-the epoch. By

using the inequality $\frac{a_1+a_2+\ldots+a_m}{b_1+b_2+\ldots+b_m} \leqslant \frac{c\cdot b_1+c\cdot b_2+\ldots+c\cdot b_m}{b_1+b_2+\ldots+b_m} = c$, where $c = \max\limits_{i=1\ldots m} \frac{a_i}{b_i}$, we get that:

$$\frac{\text{sumCost}_{MM}(LAZY, G_0, \sigma)}{\text{sumCost}_{SM}(ALG, G_0, \sigma)} = \frac{\sum\limits_{i=1}^{m} \text{sumCost}_{MM}(LAZY, G_{i-1}, \sigma^{(i)})}{\sum\limits_{i=1}^{m} \text{sumCost}_{SM}(ALG, G_{i-1}, \sigma^{(i)})} \leqslant$$

$$\leqslant \max_{i=1\ldots m} \frac{\text{sumCost}_{MM}(LAZY, G_{i-1}, \sigma^{(i)})}{\text{sumCost}_{SM}(ALG, G_{i-1}, \sigma^{(i)})}$$

This transformation suggests us an idea for the proof: we find the lower bound for $\text{sumCost}_{SM}(ALG, G_{i-1}, \sigma^{(i)})$ and the upper bound for $\text{sumCost}_{MM}(LAZY, G_{i-1}, \sigma^{(i)})$, thus, we find the maximal possible ratio which becomes the static optimal complexity. From now on, we consider and bound only the ratios of the epochs, not the whole execution.

2.2.2. Search requests

We start with applying our “Idea of Lazyness” to Splay Tree algorithm. Our modification has $O(\min(\sqrt{\alpha}, \log n))$-ratio regarding to Splay Tree in Standard Model. Since Splay Tree is $O(1)$-static optimal then our modification is $O(\min(\sqrt{\alpha}, \log n))$-static optimal in Matching Model.

Lemma 19. Lazy Splay Tree algorithm is the $O(\sqrt{\alpha})$-statically optimal algorithm in Matching Model

Proof. At first, we introduce all the necessary notions. Let T_{ST} and T_{LST} be the trees maintained by Splay Tree (ST) and Lazy Splay Tree (LST) algorithms, respectively. Let us divide the requests into epochs $\sigma^{(i)}$. Let $\sigma_j^{(i)}$ be j-th request in i-th epoch. Let s_{ij}^{ST} and s_{ij}^{LST} be the cost of the corresponding request $\sigma_j^{(i)}$ in T_{ST} and T_{LST} respectively.

By the definition of the epochs, we have for any epoch i $\sum_{j=1}^{|s_i|-1} s_{ij}^{ST} < \alpha$ and $\sum_{j=1}^{|s_i|} s_{ij}^{ST} \geqslant \alpha$. At the end of the epoch, i.e., after $|s_i|$-th request, by “Idea of Lazyness” we synchronize the data structures and make T_{LST} to be exactly as T_{ST} structure. Without loss of generality we consider an epoch i. We have two cases.

a) If $s_{ij}^{LST} \leqslant \sqrt{\alpha}$ for all j then during the epoch we perform more than $\sqrt{\alpha}$ operations: each operation costs less than $\sqrt{\alpha}$ and we end the epoch when the total cost exceeds α. Then $\text{sumCost}_{MM}(LST, G_{i-1}, \sigma^{(i)}) \leqslant \alpha + \sqrt{\alpha}$

because the cost of the last operation does not exceed $\sqrt{\alpha}$, and $\text{sumCost}_{SM}(ST, G_{i-1}, \sigma^{(i)}) \geqslant \sqrt{\alpha}$ since we do more than $\sqrt{\alpha}$ operations of cost at least one. So, our complexity of static optimality cannot exceed $\frac{\alpha+\sqrt{\alpha}}{\sqrt{\alpha}} = O(\sqrt{\alpha})$.

b) If there exists j for which $s_{ij}^{LST} \geqslant \sqrt{\alpha}$ (if there are several of them we take the one with the maximum cost) we should consider two cases: $s_{ij}^{ST} \geqslant \sqrt{\alpha}$ and $s_{ij}^{ST} < \sqrt{\alpha}$. The second case is possible when before the request j in our epoch we have done splay operations, which decreased the depth of requested node v. This is the only explanation why the change of the depth happens since at the beginning of the epoch the trees ST and LST were the same.
To begin with, we want to proof that $\sum_{k=1}^{j} s_{ik}^{ST} \geqslant s_{ij}^{LST}$, i.e., the total cost of the requests on Splay Tree from the start of the epoch is greater than one operation in Lazy Splay Tree.
Let us name the rotation operations during the splay: zig, zig-zig and zig-zag, as "Step-of-splay". At first, we need to understand why and how the depth of our node can change in the progress of epoch. Consider Figure 13. We can see that one step-of-splay can change the depth of all nodes in the tree.
Look at all "step-of-splay" operations that affected the depth of node v. Let this set be SS, Δd be the difference between the depth of node v in T_{ST} and T_{LST}, Δd_s and c_s be the differences in depth before and after splay operation s and the cost of route finding and adjustment in current step-of-splay s, i.e., the depth of the node.

$$\Delta d = \sum_{s \in SS} \Delta d_s \underset{\text{Figure 13}}{\leqslant} \sum_{s \in SS} c_s \leqslant \sum_{k=0}^{j-1} s_{ik}^{ST}$$

$$d_v^{LST} = \Delta d + d_v^{ST} \leqslant \sum_{k=1}^{j-1} s_{ik}^{ST} + d_v^{ST} = \sum_{k=1}^{j} s_{ik}^{ST}$$

It means that $\sum\limits_{k=1}^{j} s_{ij}^{ST} \geqslant d_v^{LST} = s_{ij}^{LST} \geqslant \sqrt{\alpha}$. Let us substitute this into static optimal complexity estimation during the epoch:

$$\frac{\text{sumCost}_{MM}(LST, T_i, \sigma^{(i)})}{\text{sumCost}_{SM}(ST, T_i, \sigma^{(i)})} \leqslant \frac{\alpha + s_{ij}^{LST}}{\sum\limits_{k=0}^{j} s_{ik}^{ST}} =$$

$$= \frac{\alpha}{\sum\limits_{k=0}^{j} s_{ik}^{ST}} + \frac{s_{ij}^{LST}}{\sum\limits_{k=0}^{j} s_{ik}^{ST}} \leqslant \sqrt{\alpha} + O(1) = O(\sqrt{\alpha})$$

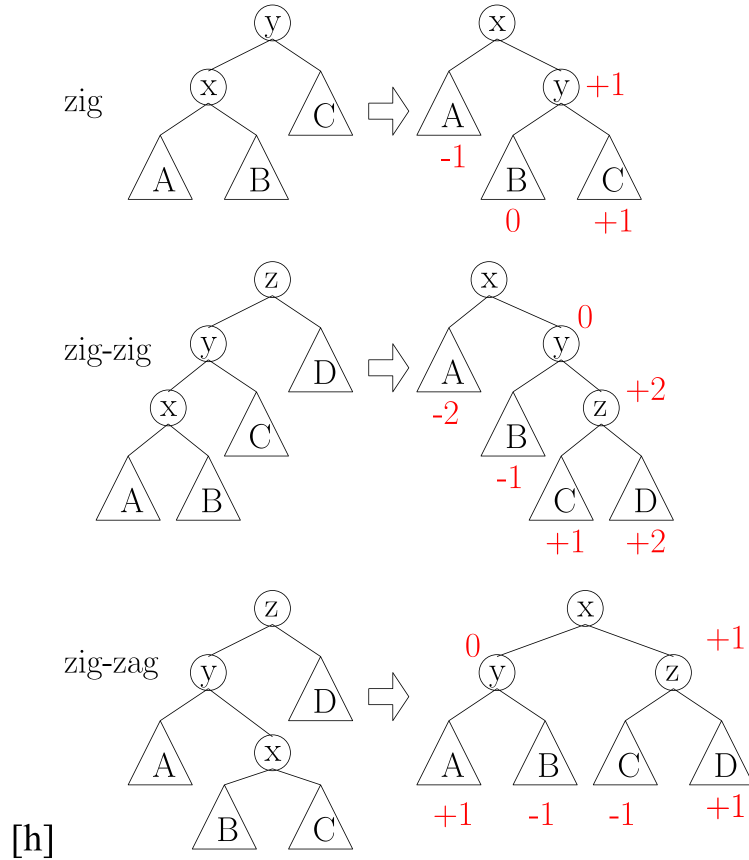

Figure 13 – Change depth in zig, zig-zig and zig-zag cases. In zig case cost for adjustment and routing cost are 3 and 1 respectively, maximum change of depth is 1. In zig-zig case this costs are 6, 2 and 2. In zig-zag - 4, 2 and 2.

□

In Lemma 19 we proved that Lazy Splay Tree is $O(\sqrt{\alpha})$-statically optimal. But what should we do, if our α is too big (for example, $O(n^4)$). We modify our algorithm when $\alpha > \log^2 n$: instead of Self-Adjusting algorithm we take static balanced tree of height $\log n$ instead of Lazy Splay Tree.

Theorem 20. Lazy Splay Tree algorithm is the $O(\min(\sqrt{\alpha}, \log n))$-statically optimal algorithm in Matching Model.

Proof. In Lemma 19 we proved that Lazy Splay Tree is $O(\sqrt{a})$-statically optimal (Lemma 19). And if $\alpha > \log^2 n$ we use the static balanced tree algorithm which gives us $O(\log n)$ complexity of static optimality. □

One could suggest that we can use different algorithm from Splay Tree and, probably, get the better complexity. However, as we show in the next theorem "Idea of Laziness" has the worst-case bound which coincides with the bound given by Splay Tree.

Theorem 21. "Idea of Laziness" can not achive better complexity of static optimality than $O(\min(\sqrt{\alpha}, \log n))$.

Proof. Since, we provided an algorithm with $O(\min(\sqrt{\alpha}, \log n))$ static optimality ratio, we want to prove that it is in fact the best ratio for the "Idea of Lazyness". For that we need to find a pattern of requests, that will always achieve $O(\sqrt{\alpha})$ ratio.

One of the worst cases for Lazy Splay Tree is when we access one element for the whole epoch. Let us consider the number of of operations in tree T_{ST}. Let d be the depth of the accessed node. In Lazy Splay Tree we will perform $\lceil \frac{\alpha}{d} \rceil$ requests. In Splay Tree the first request costs d, while the next requests cost one since we splayed the node to the root. So, the cost is equal to:

$$c(d) = d + \lceil \frac{\alpha - d}{d} \rceil = \begin{cases} d + (\frac{\alpha}{d} - 1) & \alpha \text{ divided by } d \\ d + \frac{\alpha}{d} & \alpha \text{ not divided by } d \end{cases}$$
$$c'(d) = 1 - \frac{\alpha}{d^2}$$

To find the extremum of the function c we have to equate $c'(d)$ with 0. Thus, we get $d = \sqrt{\alpha}$. If $d > \sqrt{\alpha}$ we get $c'(d) > 0$, which means that $d = \sqrt{\alpha}$ is the depth with the minimal cost for T_{ST}.

So, consider the following requests. We access a node at depth $\sqrt{\alpha}$. This gives us, that $\text{sumCost}_{SM}(ST, G_{i-1}, \sigma^{(i)}) = c(\sqrt{\alpha}) = O(\sqrt{\alpha})$ and $\text{sumCost}_{MM}(LST, G_{i-1}, \sigma^{(i)}) \leqslant a + \sqrt{a}$. It means, that in the worst case we have $O(\sqrt{\alpha})$ static optimality complexity.

But tree is not obliged to have a node at depth $\sqrt{\alpha}$. For example, when $a = n^4$ the height of the tree obviously does not exceed $n \leqslant \sqrt{\alpha} = n^2$. In this case, we are trying to get another bound that depends only of n.

We repeat our pattern to request one node the whole epoch. We know that $d_{max} \geqslant \log n$. To reach "no node at $\sqrt{\alpha}$ depth" case we need to assume that $a > \log^2(n)$. Previously we count a derivative for $c(d)$, which is monotonic and increasing function. Our ratio will be $\frac{\alpha}{c(d)}$. For the minimal ratio we need to request nodes with maximum depth which is $\geqslant \log(n)$.

$$\frac{\alpha}{c(d)} \geqslant \begin{cases} \frac{\alpha}{\log n + \frac{\alpha}{\log n} - 1} = \frac{1}{\frac{\log n}{a} + \frac{1}{\log n} - \frac{1}{a}} > \frac{1}{\frac{2}{\log n} - \frac{1}{\log^2 n}} = \frac{\log^2 n}{2 \cdot \log n - 1} > \frac{\log^2 n}{2 \cdot \log n} = \frac{\log n}{2} \\ \frac{\alpha}{\log n + \frac{\alpha}{\log n}} = \frac{1}{\frac{\log n}{a} + \frac{1}{\log n}} > \frac{1}{\frac{2}{\log n}} = \frac{\log n}{2} \end{cases}$$

It means that the optimality ratio lies between $\frac{\log n}{2} < c < \sqrt{\alpha}$. So, the lower bound for Idea of Laziness of static optimality complexity is $O(\min(\sqrt{\alpha}, \log n))$.

□

2.2.3. Routing requests

In the previous subsection, we applied "Idea of Lazyness" to Splay Tree algorithm. However, we can do the same to SplayNet algorithm. Instead of the depth we consider a distance of the route between two nodes. Lower bound for the lazy algorithm is the same: for each epoch, we ask for a pair with distance $\sqrt{\alpha}$ and then repeat this request $\sqrt{\alpha} - 1$ times.

Lemma 22.

$$\text{sumCost}(\text{Lazy SplayNet}, G_0, \sigma) = O(\sqrt{\alpha} \cdot \text{sumCost}(\text{SplayNet}, G_0, \sigma))$$

holds for any starting tree G_0 and any list of requests σ.

Proof. The proof for this algorithm is the same as the proof of Lemma 19 with one change: the cost of the request becomes the length of the path between two nodes.

We briefly overview the proof.We recommend to become familiar with the proof of Lemma 19. As previously, we consider two cases:

a) All requests in Lazy SplayNet cost not more than $\sqrt{\alpha}$. It means that we will do more than $\sqrt{\alpha}$ requests in SplayNet. So, our complexity of optimality cannot be more than $\frac{\alpha + \sqrt{\alpha}}{\sqrt{\alpha}} = O(\sqrt{\alpha})$.
b) We get a request that costs more than $\sqrt{\alpha}$ in Lazy SplayNet and, again, we want to prove that the sum of costs of the requests in SplayNet exceeds $\sqrt{\alpha}$.

At first, we say that in one step-of-splay the maximum difference of length between nodes in tree cannot not be more than the cost to perform the step-of-splay. You can check it using Table 1 and we can assume than in in each "Step of Splay" we change no more than three links.

Table 1 – Table of distance differences between nodes after each type step of splay

	x	y	A	B	C
x	0	0	0	+1	0
y	0	0	0	-1	0
A	0	0	0	+1	0
B	+1	-1	+1	0	-1
C	0	0	0	-1	0

(a) Zig

	x	y	z	A	B	C	D
x	0	0	0	0	+1	+1	0
y	0	0	0	0	-1	+1	0
z	0	0	0	0	-1	-1	0
A	0	0	0	0	+1	+1	0
B	+1	-1	-1	+1	0	0	-1
C	+1	+1	-1	+1	0	0	-1
D	0	0	0	0	-1	-1	0

(b) Zig-zig

	x	y	z	A	B	C	D
x	0	0	-1	0	+1	+1	-1
y	0	0	+1	0	-1	+1	=1
z	-1	+1	0	+1	0	-2	0
A	0	0	+1	0	-1	+1	+1
B	+1	-1	0	-1	0	+2	0
C	+1	+1	-2	+1	+2	0	-2
D	-1	+1	0	+1	0	-2	0

(c) Zig-zag

Then, let us sum all the costs of operations that affect the distance between the requested nodes. We get $\sum_{k=1}^{j} s_{ik}^{SN} \geqslant l_{st}^{LSN} = s_{ij}^{LST} \geqslant \sqrt{\alpha}$ and we use this property next:

$$\frac{\text{sumCost}_{MM}(LSN, T_0, \sigma^{(i)})}{\text{sumCost}_{SM}(SN, T_0, \sigma^{(i)})} \leqslant \frac{\alpha + s_{ij}^{LSN}}{\sum\limits_{k=0}^{j} s_{ik}^{SN}} =$$

$$= \frac{\alpha}{\sum\limits_{k=0}^{j} s_{ik}^{SN}} + \frac{s_{ij}^{LSN}}{\sum\limits_{k=0}^{j} s_{ik}^{SN}} \leqslant \sqrt{\alpha} + O(1) = O(\sqrt{\alpha})$$

□

In Lemma 2.2.3 we proved that Lazy SplayNet is $O(\sqrt{\alpha})$-statically optimal. But what should we do, if our α is too big (for example, $O(n^4)$). We modify our

algorithm when $\alpha > \log^2 n$: we use the static balanced tree of height $\log n$ instead of Self-Adjusting algorithm.

Theorem 23.

$$\text{sumCost}(\text{Lazy SplayNet}, G_0, \sigma) = O(\min(\sqrt{\alpha}, \log n) \cdot \text{sumCost}(\text{SplayNet}, G_0, \sigma))$$

holds for any starting tree G_0 and any list of requests σ.

Proof. In Lemma 19 we proved that Lazy SplayNet is $O(\sqrt{a})$-statically optimal (Lemma 19). And if $\alpha > \log^2 n$ static balanced tree algorithm is $O(\log n)$-statically optimal. □

Theorem 24. "Idea of Laziness" can not archive better complexity of static optimality than $O(\min(\sqrt{\alpha}, \log n))$.

Proof. We prove the sequence of request on which our "Idea of Laziness" achieves $O(\min(\sqrt{\alpha}, \log n))$ multiplier in complexity. At the beginning of an epoch, we choose two nodes on distance d and perform $c(d)$ such requests until the end of the epoch. Obviously, we get that the complexity cannot be smaller than $\sqrt{\alpha}$.

Finally, we find the lower bound, which does not depend on α. We repeat our pattern to request one pair of nodes for the whole epoch. We know that $d_{max} \geqslant 2\log n$. To reach "no route of length $2\log n$" case we need to assume that $a > 4\log^2(n)$. Previously we count a derivative for $c(d)$, which is monotonic and increasing function. Our ratio will be $\frac{\alpha}{c(d)}$. For the minimal ratio we need to request nodes with maximum distance which is $\geqslant 2\log(n)$.

$$\frac{\alpha}{c(d)} \geqslant \begin{cases} \frac{\alpha}{2\log n + \frac{\alpha}{2\log n} - 1} = \frac{1}{\frac{2\log n}{a} + \frac{1}{2\log n} - \frac{1}{a}} > \frac{1}{\frac{1}{2\log n} - \frac{1}{4\log^2 n}} = \frac{4\log^2 n}{2\cdot\log n - 1} > 2\log n \\ \frac{\alpha}{2\log n + \frac{\alpha}{2\log n}} = \frac{1}{\frac{2\log n}{a} + \frac{1}{2\log n}} > \frac{1}{\frac{1}{2\log n}} = 2\log n \end{cases}$$

It means that if the $2\log n < \sqrt{\alpha}$ our complexity lies between them. □

2.3. Network with bounded degree property

Finally, we apply our "Idea of Lazyness" to ReNet algorithm, which provides bounded degree property for the graph. If α is greater than $\log^2 n$, then for every epoch t, $G_t = T$, where T is an arbitrary balanced binary search tree, which gives

us $O(\log n)$ complexity of static-optimality. Thus, for the rest of the section we consider α less than $\log^2 n$.

From "Idea of Lazyness" we have ReNet structure, which maintains logically in the coordinator, and LazyReNet structure, which represents the physical network.

Theorem 25. $\mathrm{sumCost}(\mathrm{LazyReNet}, G_0, \sigma) = O(\sqrt{\alpha}\ \mathrm{sumCost}(\mathrm{ReNet}, G_0, \sigma))$ holds for every initial graph G_0 and the list of requests σ.

Proof. During reset in LazyReNet we delete all the edges, i.e., unlink all the nodes, by paying cost $O(\alpha)$. Thus, whenever a reset occurs in ReNet, we can apply an immediate synchronization (adjustment) step between LazyReNet and ReNet. Therefore, no reset can occur during an epoch, since an epoch is a segment of requests between two synchronizations. Thus, a large node cannot become small within an epoch.

Consider the requests $\sigma^{(t)} \sqsubseteq \sigma$ during the arbitrary epoch t, i.e. $\mathrm{sumCost}(\text{Lazy ReNet}, G_t, \sigma^{(t)}) \geqslant \alpha$, where G_t is the network topology at the beginning of epoch t. We assume that before the last request from $\sigma^{(t)}$ our sumCost was less than α. We split the requests of $\sigma^{(t)}$ into two sets: (i) A_1 is the set of all small-small requests for which a direct edge exists, the total cost of A_1 requests in LazyReNet is α_1; (ii) A_2 is the set of all small-large, large-small, and large-large requests for which a route already exists, the total cost of A_2 requests in LazyReNet is α_2. Sometimes nodes of the request are not connected and we have to add edge between them in ReNet. But what should we do at the same time in LazyReNet ? We can not add edge during an epoch, since we change network only during synchronization step. We let this edge to be a "phantom" edge. So, if we need to route through a "phantom edge" in LazyReNet we route this packet through coordinator paying cost c. To simplify the proof we make an assumption here that c also equals to the cost of adding edge in ReNet. Let the total cost of traversing through "phantom edges" be α_3. By the definition of epoch, $\alpha \leqslant \alpha_1 + \alpha_2 + \alpha_3$.

Small-small requests from A_1 in LazyReNet are not necessarily small-small requests in ReNet. It means that the cost of such request in ReNet is higher than the cost of that request in LazyReNet , which is one. Thus, we could suppose that all small-small requests in LazyReNet are also small-small in ReNet.

Further, we consider three cases depending on the values of α_1, α_2, and α_3. Consider the first case, when $\alpha_1 \geqslant \sqrt{\alpha}$. Since, small-small requests from A_1 cost the same in ReNet and LazyReNet, ReNet has to pay more than $\sqrt{\alpha}$. Let m be

the maximum cost of performing request from A_2 in LazyReNet and let this request be σ_m. Now we state that the total cost of requests from A_2 in ReNet is more than m. This can be proved in the same way, like in Lemma 19. Either the described cost m of operations in LazyReNet is hidden in splay steps of preceding splay oprations in the same epoch or in phantom edge insertions. Hence $\text{sumCost}(\text{ReNet}, G_t, \sigma') \geqslant \sqrt{\alpha} + m$ and $\text{sumCost}(\text{Lazy ReNet}, G_t, \sigma') \leqslant \alpha + m$. So, our complexity is:

$$\frac{\text{sumCost}(\text{LazyReNet}, G_t, \sigma')}{\text{sumCost}(\text{ReNet}, G_t, \sigma')} \leqslant \frac{\alpha + m}{\sqrt{\alpha} + m} \leqslant \frac{\alpha + \sqrt{\alpha} + m}{\sqrt{\alpha} + m} = \\ = \frac{\alpha}{\sqrt{\alpha} + m} + 1 \leqslant \sqrt{\alpha} + 1 = O(\sqrt{\alpha})$$

In the second case $\alpha_1 < \sqrt{\alpha}$ and $\alpha_2 = 0$. Thus, we can deduce that $\alpha_3 \geqslant \alpha - \alpha_1$. This means that LazyReNet performs $\frac{\alpha_3}{c}$ requests for which there an edge does not exists. In ReNet these requests cost at least $c + (\frac{\alpha_3}{c} - 1)$, i.e., we paid c for an edge insertion and at least 1 for the rest operations. By AM-GM, $c + (\frac{\alpha_3}{c} - 1) \geqslant 2\sqrt{\alpha_3} - 1 \geqslant \sqrt{\alpha_3}$. Thus, in case $\alpha_1 < \sqrt{\alpha}$ and $a_2 = 0$, $\text{sumCost}(\text{ReNet}, G_t, \sigma^{(t)}) \geqslant a_1 + \sqrt{\alpha_3} - 1 \geqslant \sqrt{\alpha_1} + \sqrt{\alpha_3} \geqslant \sqrt{\alpha_1 + \alpha_3} = \sqrt{\alpha}$. In the latter inequality, we used the following observation $\sqrt{x} + \sqrt{y} = \sqrt{(\sqrt{x} + \sqrt{y})^2} = \sqrt{x + y + 2\sqrt{xy}} \geqslant \sqrt{x + y}$, where x and y are positive integers. Note that by extension, $\sqrt{x} + \sqrt{y} + \sqrt{z} \geqslant \sqrt{x + y} + \sqrt{z} \geqslant \sqrt{x + y + z}$, where z is also a positive integer. Now, we count static optimality complexity for that case.

$$\frac{\text{sumCost}(\text{LazyReNet}, G_t, \sigma^{(t)})}{\text{sumCost}(\text{ReNet}, G_t, \sigma^{(t)})} \leqslant \frac{\alpha + c}{c + \frac{\alpha_3}{c} - 1 + \alpha_1} \leqslant \\ \leqslant \begin{cases} \frac{\alpha + c}{c + \alpha_1} \leqslant \frac{\alpha}{c + \alpha_1} + 1 \leqslant \frac{\alpha}{\sqrt{\alpha_3} + \alpha_1} + 1 = O(\sqrt{\alpha}) & c \geqslant \sqrt{\alpha_3} \\ \frac{\alpha + c}{\sqrt{\alpha} - 1} \leqslant \frac{\alpha + \sqrt{\alpha}}{\sqrt{\alpha} - 1} = O(\sqrt{\alpha}) & c < \sqrt{\alpha_3} \end{cases}$$

Now we consider the last case is when $\alpha_1 < \sqrt{\alpha}$ and $\alpha_2 > 0$. From the previous cases we know that in ReNet we pay at least $\alpha_1 + \sqrt{\alpha_3} - 1$. Further we want to prove that requests from A_2 costs at least $\sqrt{\alpha_2}$ in ReNet. We do it in the same way like in Lemma 19.

If all requests in A_2 cost at most $\sqrt{\alpha_2}$ in LazyReNet, then ReNet pays at least $|A_2| \geqslant \sqrt{\alpha_2}$.

Assume that there are less than $\sqrt{\alpha_2}$ requests in A_2, then there has to be at least one request that costs at least $\sqrt{\alpha_2}$ in LazyReNet. If there are more than one request, let us take the maximal one and let this request cost m. If that request costs at least $\sqrt{\alpha_2}$ in ReNet, then obviously ReNet pays at least $\sqrt{\alpha_2}$. Otherwise we consider several cases: small-large or large-large request in LazyReNet .

Suppose that this request is large-large. Let (u,v) be the request and h be the node through which the request is routed (h is either v or a helper node). For simplicity we assume that we have the route through the helper node. Let $d(u,h) + d(h,v) \geqslant \sqrt{\alpha_2}$ be the cost in LazyReNet and $d'(u,h) + d'(h,v) < \sqrt{\alpha_2}$ the cost in ReNet. Now we are using the same idea from Lemma 19: for the change of distance $d(x,y)$ to $d'(x,y)$ we pay by previous splay operations in this epoch. So, ReNet pays at least than $\sqrt{\alpha_2}$. Similarly, if the request was small-large in LazyReNet, then the analysis would be the same.

Finally, the cost of ReNet for $\sigma^{(t)}$ is

$$\begin{aligned} \text{cost}(\text{ReNet}, G_t, \sigma') \geqslant \alpha_1 + \sqrt{\alpha_2} + \sqrt{\alpha_3} - 1 \geqslant \sqrt{\alpha_1} + \sqrt{\alpha_2} + \sqrt{\alpha_3} - 1 \geqslant \\ \geqslant \sqrt{\alpha_1 + \alpha_2 + \alpha_3} - 1 = \sqrt{\alpha} - 1 \end{aligned}$$

Thus, static optimality complexity is:

$$\begin{aligned} \frac{\text{sumCost}(\text{LazyReNet}, G_t, \sigma^{(t)})}{\text{sumCost}(\text{ReNet}, G_t, \sigma^{(t)})} &\leqslant \begin{cases} \frac{\alpha+m}{\sqrt{\alpha_1+\alpha_3+m-1}} = \frac{\alpha}{\sqrt{\alpha_1+\alpha_3+\sqrt{\alpha_2}-1}} + O(1) & c \leqslant m \\ \frac{\alpha+c}{\sqrt{\alpha_1+\alpha_2+c-1}} = \frac{\alpha+c}{\sqrt{\alpha_1+\alpha_2+\sqrt{\alpha_3}-1}} + O(1) & c > m \end{cases} \\ &\leqslant \frac{\alpha}{\sqrt{\alpha}-1} + O(1) = O(\sqrt{\alpha}) \end{aligned}$$

Therefore, $\text{sumCost}(\text{LazyReNet}, G_0, \sigma) = O(\sqrt{\alpha} \cdot \text{sumCost}(\text{ReNet}, G_0, \sigma))$ holds in any of the cases above. □

Conclusions on Chapter 2

In this chapter we introduced lazy algorithms for Networks in Matching Model. Firstly, we introduced a counter modification for the Line Network algorithm Move-To-Front, which gives us $O(1)$-static optimal algorithm in MM. Then, we introduced "Idea of Laziness" and applied it to Binary Tree (Splay Tree and SplayNet) and Graph with bounded degree (ReNet) algorithms from Chapter 1. The

complexity of static-optimality of the new algorithms is $O(\min(\sqrt{\alpha}, \log n))$ regarding to their versions from Standard Model.

CONCLUSION

In this work, we looked into recently introduced model, Matching Model [1], of the cost of adjustments. We proposed new lazy versions of well-known Self-Adjusting algorithms tailored for Matching Model. Our lazy versions have a better complexity of static optimality than straightforward versions of algorithms from Standard Model. Please see the comparison in Table 2.

Table 2 – Comparison of algorithms in two models

	SM	MM Straightforward	MM Lazy
List	$O(1)$	$O(\alpha)$	$O(1)$
Tree (search)	$O(1)$	$O(\alpha)$	$O(\min(\sqrt{\alpha}, \log n))$
Tree (route)	c_{SN}	$O(\alpha \cdot c_{SN})$	$O(\min(\sqrt{\alpha}, \log n) \cdot c_{SN})$
Bound Deg Graph	$O(1)$	$O(\alpha)$	$O(\min(\sqrt{\alpha}, \log n))$

As a future work, it would be interesting to consider the following research questions. At first, we would like to design dynamic optimal algorithms in MM, However, for now, we do not know how dynamic optimal algorithms behave in MM. Nevertheless, we suppose that we can redesign Tango Tree for our adjusting operations. Secondly, we do not have algorithms for the routing requests in Line Topology that has good complexity in MM. Finally, we want to assign a cost of transferring a packet through an edge and recalculate our complexities of static optimality.

REFERENCES

1 An Online Matching Model for Self-Adjusting ToR-to-ToR Networks / C. Avin [et al.] // arXiv:submit/3235030. — 2020.

2 *Avin C.*, *Mondal K.*, *Schmid S.* Demand-aware network designs of bounded degree // Distributed Computing. — 2019. — P. 1–15.

3 *Avin C.*, *Schmid S.* ReNets: Toward Statically Optimal Self-Adjusting Networks // arXiv preprint arXiv:1904.03263. — 2019.

4 *Avin C.*, *Schmid S.* Toward demand-aware networking: A theory for self-adjusting networks // ACM SIGCOMM Computer Communication Review. — 2019. — Vol. 48, no. 5. — P. 31–40.

5 BCube: a high performance, server-centric network architecture for modular data centers / C. Guo [et al.] // Proceedings of the ACM SIGCOMM 2009 conference on Data communication. — 2009. — P. 63–74.

6 Beyond fat-trees without antennae, mirrors, and disco-balls / S. Kassing [et al.] // Proceedings of the Conference of the ACM Special Interest Group on Data Communication. — 2017. — P. 281–294.

7 *Al-Fares M.*, *Loukissas A.*, *Vahdat A.* A scalable, commodity data center network architecture // ACM SIGCOMM computer communication review. — 2008. — Vol. 38, no. 4. — P. 63–74.

8 *Knuth D. E.* Optimum binary search trees // Acta informatica. — 1971. — Vol. 1, no. 1. — P. 14–25.

9 Projector: Agile reconfigurable data center interconnect / M. Ghobadi [et al.] // Proceedings of the 2016 ACM SIGCOMM Conference. — 2016. — P. 216–229.

10 *Sleator D. D.*, *Tarjan R. E.* Amortized efficiency of list update and paging rules // Communications of the ACM. — 1985. — Vol. 28, no. 2. — P. 202–208.

11 *Sleator D. D.*, *Tarjan R. E.* Self-adjusting binary search trees // Journal of the ACM (JACM). — 1985. — Vol. 32, no. 3. — P. 652–686.

12 Splaynet: Towards locally self-adjusting networks / S. Schmid [et al.] // IEEE/ACM Transactions on Networking. — 2015. — Vol. 24, no. 3. — P. 1421–1433.