Brief Announcement: Toward Self-Adjusting Networks for the Matching Model

Evgeniy Feder
efeder@itmo.ru
ITMO University
St. Petersburg, Russia

Ichha Rathod
mt1180755@iitd.ac.in
Indian Institute of Technology Delhi
New Delhi, India

Punit Shyamsukha
punit99iitd@gmail.com
Indian Institute of Technology Delhi
New Delhi, India

Robert Sama
robert.sama@univie.ac.at
Faculty of Computer Science
University of Vienna
Vienna, Austria

Vitaly Aksenov
aksenov@itmo.ru
ITMO University
St. Petersburg, Russia

Iosif Salem
iosif.salem@univie.ac.at
Faculty of Computer Science
University of Vienna
Vienna, Austria

Stefan Schmid
stefan_schmid@univie.ac.at
Faculty of Computer Science
University of Vienna
Vienna, Austria

ABSTRACT

Self-adjusting networks (SANs) utilize novel optical switching technologies to support dynamic physical network topology reconfiguration. SANs rely on online algorithms to exploit this topological flexibility to reduce the cost of serving network traffic, leveraging locality in the demand. Models in prior work assign uniform cost for traversing and adjusting a single link (e.g. both cost 1). In this paper, we initiate the study of online algorithms for SANs in a more realistic cost model, the *Matching Model* (MM), in which the network topology is given by the union of a constant number of bipartite matchings (realized by optical switches), and in which changing an entire matching incurs a fixed cost α. The cost of routing is given by the number of hops packets need to traverse. We present online SAN algorithms in the MM with cost $O(\sqrt{\alpha})$ times the cost of reference algorithms in the uniform cost model.

CCS CONCEPTS

• **Theory of computation → Online algorithms**; • **Networks → Network algorithms**.

KEYWORDS

self-adjusting networks; matching model; online algorithms

ACM Reference Format:

Evgeniy Feder, Ichha Rathod, Punit Shyamsukha, Robert Sama, Vitaly Aksenov, Iosif Salem, and Stefan Schmid. 2021. Brief Announcement: Toward Self-Adjusting Networks for the Matching Model. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '21), July 6–8, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3409964.3461824

SPAA '21, July 6–8, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8070-6/21/07.
https://doi.org/10.1145/3409964.3461824

ACKNOWLEDGMENTS

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (864228 - AdjustNet). Vitaly Aksenov is supported by JetBrains Research.

1 INTRODUCTION

In this work, we study Self-Adjusting Networks (SANs) from an algorithmic point of view. SAN algorithms dictate how the network topology should change when there are shifts in the traffic demand, and especially, in the set of large "elephant flows" [1, 5]. In particular, in this paper we consider a model where the network needs to serve routing requests which arrive over time, in an online manner. Existing SAN algorithms are based on a uniform cost model where both traversing and changing a link have unit cost [4, 8]. This is a useful basic model that enabled the first algorithmic results. In practice, however, switching hardware usually allows to reconfigure the topology on a per-matching granularity, and changing a matching in a demand-aware manner is more costly than traversing a link (e.g., in terms of time) [2, 5].

The Matching Model (MM) proposed in [2] addresses this discrepancy, by assuming that traversing a single link has unit cost and changing the whole topology G to a new one G' comes at a fixed cost. Any topology can be defined as a union of matchings over the set of nodes and the MM assumes that rearranging the edges (links) of a single matching comes at a fixed cost (e.g., time), say α. Thus the total cost for adjusting the whole topology to a new one is the product of α and the number of matchings needed to construct the topology. In this paper we focus on scalable topologies where the maximum degree Δ is a constant and, thus, the topology reconfiguration cost in the MM is $O(\alpha)$, as the number of matchings needed is constant as well. This model better fits systems and hardware properties and early work has shown its relevance [6]. However, so far, we lack algorithmic and analytical techniques for this model.

This paper presents a first analysis of the Matching Model and describes efficient online algorithms for this model. We present our results for the MM in three steps; we start with line topologies, then we move to tree topologies, and we finally reach our main goal—bounded-degree networks. Our main contribution is a method for designing efficient online SAN algorithms in the MM, when compared to reference SANs in the uniform cost model. We cache a constant amount of topology adjustments and then *lazily* apply them by switching to a topology that is a result of all cached adjustments when it is most beneficial to pay the cost α of topology reconfiguration. Our method of *lazy* topology reconfiguration transforms a self-adjusting algorithm from the uniform-cost model to one in the MM. We show that in the three bounded-degree topology families we studied, the SANs in the MM cost $O(\sqrt{\alpha})$ times the algorithm cost in the uniform cost model, which is a clear improvement from the naive α factor that we mentioned earlier.

2 LAZY BOUNDED-DEGREE SANS

We start with presenting SANs and their optimality properties, before we present our SAN algorithms for the Matching Model.

Self-Adjusting Networks (SANs). Let $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_m) = ((u_1, v_1), (u_2, v_2), \dots, (u_m, v_m))$, where $u_i, v_i \in V$, be a sequence of *routing requests* to forward a packet from node u_i to v_i over a network topology $G = (V, E)$. If we assume that our topology has a distinguished node S, e.g., head for Lists and root for Trees, then instead of routing requests we perform *search requests* from node S when $u_i = S$ for all i and the notation of these requests will be $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_m)$, where $\sigma_i \in V$. After serving request σ_t in G_{t-1}, a SAN algorithm can change G_{t-1} to G_t. For a request σ_t, we denote by $routingCost(G_{t-1}, \sigma_t)$ and $adjustmentCost(G_{t-1}, G_t)$ the routing (in terms of packet hops) and adjustment (in terms of adjusting G_{t-1} to G_t) costs, respectively.

In the Standard Model (SM) the cost of traversing or adjusting a single edge (link) is equal to 1. Thus, $routingCost(G_{i-1}, \sigma_i)$ is the length of the route in G_{i-1} and $adjustmentCost(G_{i-1}, G_i)$ is the number of edges that change between G_{i-1} and G_i (single edge addition or deletion costs 1). In the MM, the routing cost is defined as in the SM and the adjustment cost per request, $adjustmentCost(G_{i-1}, G_i)$, is $c_{\sigma_i} \cdot \alpha$, where c_{σ_i} is the number of matchings that the SAN algorithm changed between G_{i-1} and G_i on i-th request and α is the cost of changing a matching.

Optimality of SAN algorithms. Two desirable optimality properties of online SAN algorithms are static and dynamic optimality [3]. Let $\text{sumCost}(\text{static}, G, \sigma)$ be the cost of the algorithm that computes a fixed network topology that minimizes the cost of serving a given sequence of communication requests, when no adjustments are allowed. A SAN algorithm $\mathcal{A}$ is called *statically optimal* if for every sequence of requests σ and for every starting configuration G_0, $\text{sumCost}(\mathcal{A}, G_0, \sigma) = O(\text{sumCost}(\text{static}, G_{static}, \sigma))$, where G_{static} is the offline (optimal) static topology. Similarly, a SAN algorithm $\mathcal{A}$ is called *dynamically optimal* if for every sequence of requests σ and for every starting configuration G_0, $\text{sumCost}(\mathcal{A}, G_0, \sigma) = O(\text{sumCost}(OPT, G_0, \sigma))$, where OPT is optimal online algorithm with perfect knowledge over σ.

2.1 Lazy Line Networks

We first expose our *lazy* topology adjustment method in line network topologies. We start with single-source communication sequences (search requests). In the Standard Model (SM) we are provided with a dynamically optimal Move-To-Front (MTF) algorithm [9]. We note that in the Matching Model (MM) the *"move-to-front"* operation costs α. Thus, we amortize this cost increase by not adjusting the network at each search request, but when a threshold of routing cost has been reached. The following straightforward optimization of the MTF algorithm for the MM gives an improved theoretical bound:

- Maintain a counter for each node, being zero at initialization.
- On each request for a node, we increase the node's counter by one.
- If the counter becomes α, we perform a move-to-front operation on this node (thus, the network adjustment cost will be amortized over α operations).

We refer to this algorithm as "Lazy Move-To-Front". It is not surprising that "Lazy MTF" is statically optimal in the MM: "Lazy MTF" is exactly the deterministic version of the randomized COUNTER algorithm in P^d from [7, Section 3.3] which is shown to be constant competitive (hence also statically optimal).

Theorem 1. *The "Lazy Move-To-Front" algorithm is statically optimal in the Matching Model if* $|\sigma| \geq \alpha \cdot \frac{n(n+1)}{2}$.

2.2 Lazy Tree Networks

We now turn to apply our lazy topology reconfiguration method in tree networks. Consider a self-adjusting algorithm ALG over a graph (which can be a search data structure or a network topology) in the SM, which we want to adapt in the MM. We will denote the adapted version of ALG in MM by $LazyALG$. If we simply run ALG in the MM ($LazyALG = ALG$), then we get that $cost_{MM}(LazyALG, G_0, \sigma) = \alpha \cdot cost_{SM}(ALG, G_0, \sigma)$, where G_0 is the initial graph, σ is a sequence of (search or routing) requests, and $cost_X(A, G_0, \sigma)$ is the cost of algorithm A in model $X \in \{SM, MM\}$ with initial topology G_0 and sequence σ. To improve the factor of α, we simply perform adjustments less often, by introducing our lazy topology reconfiguration method.

We design $LazyALG$, given ALG, as follows. Let us divide the list of requests σ into *epochs*. During one epoch the graph maintained by $LazyALG$ remains unmodified and the graph maintained by ALG adjusts exactly as in the SM. An epoch continues until the total cost of operations in $LazyALG$ exceeds α. After that $LazyALG$ synchronizes (copies) its graph with the graph maintained by ALG, resets the epoch cost counter to zero, and moves to a new epoch.

In SANs, $LazyALG$ adjusts the physical network topology, while ALG is a local computation running at the network coordinator, emulating the network. In our context, we are interested in the cost of routing and network reconfiguration, thus local computations as the ones done by the coordinator running ALG are ignored in the cost calculation.

We aim to calculate the ratio $\frac{\text{sumCost}_{MM}(LazyALG, G_0, \sigma)}{\text{sumCost}_{MM}(statOPT, G_{static}, \sigma)}$, where $statOPT$ is the statically optimal algorithm, i.e., the algorithm that has perfect knowledge of σ, but can only compute a static graph and perform no adjustments (the cost notation does not require G_0 in this case). This ratio measures how close $LazyALG$ is to static optimality; in case the ratio is a constant $LazyALG$ is statically optimal. Let us multiply and divide this cost ratio by $\text{sumCost}_{SM}(ALG, G_0, \sigma)$. By that we obtain:

$$\frac{\text{sumCost}_{MM}(LazyALG, G_0, \sigma)}{\text{sumCost}_{SM}(ALG, G_0, \sigma)} \cdot \frac{\text{sumCost}_{SM}(ALG, G_0, \sigma)}{\text{sumCost}_{MM}(statOPT, G_{static}, \sigma)}$$

We know that $\text{sumCost}_{MM}(statOPT, \sigma) = \text{sumCost}_{SM}(statOPT, G_{static}, \sigma)$, since $statOPT$ outputs a fixed graph. Also, if ALG is statically optimal in the SM, then $\frac{\text{sumCost}_{SM}(ALG, G_0, \sigma)}{\text{sumCost}_{SM}(statOPT, G_{static}, \sigma)}$ is equal to some value c_{ALG}. Thus, $\frac{\text{sumCost}_{MM}(LazyALG, G_0, \sigma)}{\text{sumCost}_{SM}(ALG, G_0, \sigma)} \cdot c_{ALG}$ is the resulting the cost ratio. As splay trees are statically optimal [10], $c_{SplayTree} = O(1)$, but we are not aware of c_{SPLAYNET} [8].

Let us split now the numerator and the denominator of the ratio (without c_{ALG}) 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-th epoch. By using the inequality $\frac{a_1+a_2+\ldots+a_m}{b_1+b_2+\ldots+b_m} \leq \frac{c \cdot b_1 + c \cdot b_2 + \ldots + c \cdot b_m}{b_1+b_2+\ldots+b_m} = c$, where $c = \max\limits_{i=1\ldots m} \frac{a_i}{b_i}$, we get that:

$$\frac{\text{sumCost}_{MM}(LazyALG, G_0, \sigma)}{\text{sumCost}_{SM}(ALG, G_0, \sigma)} = \frac{\sum_{i=1}^{m} \text{sumCost}_{MM}(LazyALG, G_{i-1}, \sigma^{(i)})}{\sum_{i=1}^{m} \text{sumCost}_{SM}(ALG, G_{i-1}, \sigma^{(i)})} \leq$$
$$\max_{i=1\ldots m} \frac{\text{sumCost}_{MM}(LazyALG, G_{i-1}, \sigma^{(i)})}{\text{sumCost}_{SM}(ALG, G_{i-1}, \sigma^{(i)})}$$

Thus, we focus on finding a lower bound for $\text{sumCost}_{SM}(ALG, G_{i-1}, \sigma^{(i)})$ and an upper bound for $\text{sumCost}_{MM}(LazyALG, G_{i-1}, \sigma^{(i)})$ for each epoch. In the following, we consider and bound only the ratios of the epochs, not the whole execution.

2.2.1 Search requests. We start with LAZYSPLAYTREE, which is the outcome of applying our lazy topology reconfiguration method to the splay tree algorithm. LAZYSPLAYTREE achieves a $O(\min(\sqrt{\alpha}, \log n))$-ratio with respect to splay tree in the SM. If α is regarded as a constant, then LAZYSPLAYTREE is statically optimal in the MM and our analysis is tight.

Theorem 2. *LAZYSPLAYTREE is a $O(\min(\sqrt{\alpha}, \log n))$-statically optimal algorithm in the MM.*

Theorem 3. *The complexity bound of LAZYSPLAYTREE is tight: a lazy algorithm can achieve at most $O(\min(\sqrt{\alpha}, \log n))$-static optimality.*

2.2.2 Routing requests. We show how to extend our methods to the SplayNet algorithm [8] and obtain a SAN for the MM. The key difference in our analysis is that we consider the distance of the route between two nodes, instead of the depth from the root, as we did in the previous section.

Theorem 4. *For any starting tree G_0 and any list of requests σ, it holds that* $\text{sumCost}(\textit{Lazy SplayNet}, G_0, \sigma) = O(\min(\sqrt{\alpha}, \log n) \cdot \text{sumCost}(\textit{SplayNet}, G_0, \sigma))$.

Theorem 5. *LAZYSPLAYNET cannot archive better complexity than $O(\min(\sqrt{\alpha}, \log n))$-static optimality.*

2.3 Lazy ReNet

We now study LAZYRENET in the Matching Model (MM), which is the product of applying lazy topology adjustment to RENET [4], As in Section 2.2, we show that the LAZYRENET complexity is asymptotically bounded by $\sqrt{\alpha}$ times the complexity of RENET.

A RENET is a union of *ego*, i.e., individual, views of each node. The ego view of a node is a star centered at a node and connected to recently communicated nodes, if they are less than the degree bound Δ, or a splay tree including these nodes, otherwise. A RENET is a SAN with node degree bounded by Δ that we can define as $G_t = (V, E_{coord} \cup E_t)$. The subgraph (V, E_{coord}) is used for contacting the network coordinator C and is static throughout the algorithm's execution (and has diameter c). The subgraph (V, E_t) is the dynamic part of the network and is subject to change at any time t.

Initially, E_0 is empty. Upon a request $\sigma_t = (s_t, d_t)$ if a route does not exist, s_t asks C to add a route. If both s_t and d_t are *small* nodes, i.e., if they have less than Δ edges, then C adds a direct edge between s_t and d_t. A node u becomes *large* when its degree becomes equal to Δ. In that instant, the coordinator deletes all direct links of u, creates a splay-tree (ego-tree) including all of u's former neighbors, and connects u to the splay-tree root. Communication from u to any node v in the ego-tree of u is done by following the route dictated by binary search and it is followed by splaying v to the root of the ego-tree. If a small node v is a part of an ego-tree, e.g., of u, when it becomes large, we pick a small *helper* node, add it in both ego-trees of u and v and use it as a relay when u and v communicate. If E_t becomes full (e.g. when there are no small nodes to pick or when $|E_t|$ reaches a threshold), the coordinator deletes all nodes in E_t.

Theorem 6. *For every initial graph G_0 and communication sequence σ,* $\text{sumCost}(\textit{LAZYRENET}, G_0, \sigma) = O(\sqrt{\alpha} \cdot \text{sumCost}(\textit{RENET}, G_0, \sigma))$.

REFERENCES

[1] Chen Avin, Manya Ghobadi, Chen Griner, and Stefan Schmid. 2020. On the complexity of traffic traces and implications. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 1 (2020), 1–29.

[2] Chen Avin, Chen Griner, Iosif Salem, and Stefan Schmid. 2020. An Online Matching Model for Self-Adjusting ToR-to-ToR Networks. *CoRR* abs/2006.11148 (2020). arXiv:2006.11148 https://arxiv.org/abs/2006.11148

[3] Chen Avin and Stefan Schmid. 2019. Toward demand-aware networking: A theory for self-adjusting networks. *ACM SIGCOMM Computer Communication Review* 48, 5 (2019), 31–40.

[4] Chen Avin and Stefan Schmid. 2021. ReNets: Statically-Optimal Demand-Aware Networks. In *2nd Symposium on Algorithmic Principles of Computer Systems, APOCS 2020, Virtual Conference, January 13, 2021*, Michael Schapira (Ed.). SIAM, 25–39. https://doi.org/10.1137/1.9781611976489.3

[5] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel Kilper. 2016. Projector: Agile reconfigurable data center interconnect. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 216–229.

[6] William M. Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C. Snoeren, and George Porter. 2017. RotorNet: A Scalable, Low-complexity, Optical Datacenter Network. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017*. ACM, 267–280. https://doi.org/10.1145/3098822.3098838

[7] Nick Reingold, Jeffery Westbrook, and Daniel D Sleator. 1994. Randomized competitive algorithms for the list update problem. *Algorithmica* 11, 1 (1994), 15–32.

[8] Stefan Schmid, Chen Avin, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, and Zvi Lotker. 2015. Splaynet: Towards locally self-adjusting networks. *IEEE/ACM Transactions on Networking* 24, 3 (2015), 1421–1433.

[9] Daniel D Sleator and Robert E Tarjan. 1985. Amortized efficiency of list update and paging rules. *Commun. ACM* 28, 2 (1985), 202–208.

[10] Daniel Dominic Sleator and Robert Endre Tarjan. 1985. Self-adjusting binary search trees. *Journal of the ACM (JACM)* 32, 3 (1985), 652–686.