Online Flow Computation on Unit-Vertex-Capacitated Networks*

Makis Arsenis† Robert Kleinberg†

Abstract

In many networking scenarios, long-lived flows can be rerouted to free up resources and accommodate new flows, but doing so comes at a cost in terms of disruption. An archetypical example is the transmission of live streams in a content delivery network: audio and video encoders (clients) generate live streams and connect to a server which rebroadcasts their stream to the rest of the network. Reconnecting a client to a different server mid-stream is very disruptive. We abstract these scenarios in the setting of a capacitated network where clients arrive one by one and request to send a unit of flow to a designated set of servers subject to edge/vertex capacity constraints. An online algorithm maintains a sequence of flows that route the clients present so far to the set of servers. The cost of a sequence of flows is defined as the net switching cost, i.e. total length of all augmenting paths used to transform each flow into its successor. We prove that for unit-vertex-capacitated networks, the algorithm that successively updates the flow using the shortest augmenting path from the new client to a free server incurs a total switching cost of $O(n \log^2 n)$, where n is the number of vertices in the network. This result is obtained by reducing to the online bipartite matching problem studied in prior work and applying their result. Finally, we identify a slightly more general class of networks for which essentially the same reduction idea can be applied to get the same bound.

1 Introduction

In routing, there is a fundamental tension between efficiency and stability: to utilize network resources as efficiently as possible in the face of changing conditions, it sometimes becomes necessary to change the routing paths used to send traffic between certain sources and destinations. For example, admitting new flows into a capacitated network may require rerouting existing flows to free up bandwidth.

A simple scenario in which this tension arises is the transmission of live streaming media in a content delivery network (CDN) [1]. When an audio or video encoder generates a new stream to be broadcast into the network, the first step is to connect the encoder to a server which receives the encoder's stream and retransmits it to one or more other hosts in the network. To do so while respecting bandwidth and server-capacity constraints, it may be necessary to disconnect other encoders from the selected server and reconnect them elsewhere, leading to undesirable disruptions in those other streams. Thus, the challenge of admitting as many streams as possible into the network while minimizing disruption to existing streams entails solving an interesting algorithmic challenge.

This example is emblematic of a broader category of algorithmic problems involving dynamic routing decisions in a network whose traffic demands, topology, or both are evolving over time, when the objective is to minimize "churn" — the costly process of shifting traffic from one routing path to another. The theme has become prominent in the design and analysis of wide-area traffic engineering schemes, such as Google's B4 [2] or Microsoft's SWAN [3]. In these systems a centralized software-defined networking controller uses a solver to periodically recompute an optimal multicommodity flow as the traffic matrix (the matrix specifying the amount of flow for each source-destination pair) varies over time. In the literature evaluating such systems, churn has been identified as a quantity to be minimized [4, 5] and traffic engineering frameworks such as SOL [6] and SMORE [7] are explicitly designed to minimize churn.

Despite this emphasis on churn minimization in the recent literature on traffic engineering, there is a lack of theoretical work analyzing worst-case bounds for churn when solving a sequence of flow problems. The only special case that has previously been studied is the case when each routing path consists of a single edge of a bipartite graph whose nodes all have capacity 1. If one side of the graph consists of static vertices ("servers") that are always present, and the other side consists of dynamic vertices ("clients") that arrive over time, each seeking to transmit one unit of flow to any server, then one obtains the *online bipartite matching* problem. Churn, in this setting, is quantified by the *switching cost* or *replacement cost* studied in prior work on online bipartite matching such as [8, 9, 10, 11, 12, 13].

In this paper we take the natural next step beyond online bipartite matching by allowing for multi-hop routes between sources and sinks, while retaining the other key features of online bipartite matching: sinks are always present in the network, sources arrive over time and seek to send a unit of flow which persists permanently after their arrival, and this unit of flow may be routed to any sink. These features match essential

*Both authors were supported by NSF grants CCF-1512964 and CCF-1637532.

†Department of Computer Science, Cornell University, Ithaca, NY, USA.

characteristics of the live streaming application presented earlier, with sinks representing servers that are part of the CDN's infrastructure and sources representing audio/video encoders that join the network over time and transmit a stream continually after their arrival, e.g. an Internet radio station.

We quantify the cost of a sequence of flows by summing the lengths of the augmenting paths used to modify each flow to its successor, a quantity that we call the *flow switching cost*. This cost measure — which matches (up to a factor-of-two rescaling) the switching cost used in the analysis of online bipartite matching — is motivated by two considerations. First, shifting flow from one routing path to another imposes overhead on the network: messages must be sent to each node whose next-hop changes and those nodes must change their internal state accordingly. The sum of augmenting path lengths quantifies the total number of such control messages and state changes. Second, in the live streaming application it is particularly costly for a source node (an audio/video encoder) to change its next hop, as this may entail a disruption in the stream itself. The number of such disruptions is bounded above by the *total* number of nodes in the network that change their next hop, and hence upper bounds on the sum of augmenting path lengths imply upper bounds on the number of such disruptions.

1.1 Our results and techniques The most natural algorithm for online matching, and more generally for online flow computation, is the greedy algorithm that modifies the flow in each time step so as to minimize the cost of switching from the preceding flow. Specialized to online matching, this corresponds to the Shortest Augmenting Path (SAP) algorithm analyzed in [8, 10, 11, 13]. In this paper we analyze the aforementioned greedy algorithm that generalizes SAP. To distinguish between these two algorithms, we henceforth refer to the greedy algorithm for online flow computation as Network-SAP and its specialization to online bipartite matching as Bipartite-SAP.

Our main result, Theorem 3.2, pertains to a special case of online flow computation in which there is a (directed or undirected) network consisting of static vertices present at time zero, and dynamic vertices which arrive one by one. Edges between static vertices are also present at time zero, edges from a dynamic vertex to a static vertex arrive at the same time as the arrival of their dynamic endpoint, and edges between dynamic vertices are assumed not to exist in our model. There is a fixed set of static vertices known as *sinks*; each dynamic vertex is a *source* that requests, upon arriving into the network, to send one unit of flow which may be routed to any sink. We prove a $\mathcal{O}(n \log^2 n)$ bound on the total flow switching cost in the unit-capacitated case when all vertices and edges have capacity 1, where n is the total number of vertices in the network.

The main tool that enables us to obtain this bound is a reduction from online flow computation in unit-vertex-capacitated graphs to online bipartite matching. Bernstein et al. [8] prove a $\mathcal{O}(n_c \log^2 n_c)$ bound for the online bipartite matching, where n_c is the number of dynamic vertices (clients). This bound translates into a similar bound (up to a constant factor) for flow switching cost in our model. The reduction consists of creating a sequence of auxiliary bipartite graphs and matchings, one per time step, such that flow-augmenting paths in the online flow computation problem are in one-to-one correspondence with matching-augmenting paths in the auxiliary graphs. This correspondence, which preserves augmenting path length up to a factor-of-two rescaling, implies that there is a one-to-one correspondence between executions of the Network-SAP algorithm for our problem and executions of the Bipartite-SAP algorithm for the auxiliary problem. One wrinkle that arises in the reduction is that in order for the correspondence to hold at time zero, the auxiliary bipartite graph sequence must be initialized with a non-empty matching. We observe in Section 3 below that the results of [8] continue to hold when one runs the SAP algorithm on a bipartite graph initialized with a non-empty matching.

This $\mathcal{O}(n \log^2 n)$ bound constitutes the first non-trivial worst-case bound on flow switching cost for an online flow computation algorithm in multi-hop networks. As noted earlier, the flow switching cost is motivated by at least two considerations. One can interpret it as quantifying the message complexity of updating the forwarding state in a software-defined network (SDN) of switches governed by a centralized controller, or one can interpret it as an upper bound on the number of times that flow sources must disrupt their stream of packets by changing the next-hop address to which they are forwarding the stream. In the latter interpretation, this "source switching cost" may be strictly less than the flow switching cost (which counts changes of next-hop at all nodes, not just source nodes) but a $O(n \log^2 n)$ upper bound on source switching cost is still quite non-trivial. To see this, specialize to directed bipartite networks with edges directly connecting sources to sinks, and observe that it becomes equivalent to the best-known bound on switching cost for the online bipartite matching problem.

We hope the results in this paper, which are primarily inspired by the streaming media delivery application, may prefigure future worst-case bounds on flow switching cost (or other measures of churn) for

other important applications such as traffic engineering. With an eye toward extending our result beyond unit-vertex-capacitated networks, we devote Section 4 to presenting a generalization to a class of bipartite directed networks that includes the bipartite networks obtained by applying the gadget reduction in Section 3 to unit-vertex-capacitated networks. The generalization involves introducing structures that we call *semi-matchings* which generalize matchings, and showing that the $O(n_c \log^2 n_c)$ bound of Bernstein et al. [8] generalizes to executions of the SAP algorithm initialized with a semi-matching rather than a matching.

1.2 Related work Worst-case bounds for the switching cost of the SAP algorithm for online bipartite matching were derived in many prior works [8, 10, 11, 13]. Obtaining a tight analysis for this algorithm in general bipartite graphs is still an important open problem. Letting n_c denote the number of dynamic vertices (clients) in the graph, it was shown in [13] that any algorithm must incur a total switching cost of at least $\Omega(n_c \log n_c)$ in the worst case, even when the graph is a path. A matching upper bound of $\mathcal{O}(n_c \log n_c)$ is known for trees [11], for bipartite graphs whose dynamic vertices have maximum degree 2 [13], and for general bipartite graphs whose dynamic vertices arrive in random order [12]. For general bipartite graphs the best known upper bound is $\mathcal{O}(n_c \log^2 n_c)$, obtained in a breakthrough result by Bernstein et al. [8]. It is an open question whether this bound can be improved to $\mathcal{O}(n_c \log n_c)$.

Reductions from flow problems to matching problems have appeared in prior works [14, 15] on offline algorithms for maximum flow and maximum matching. Our reduction, like Lin's reduction [14], uses a simple gadget that replaces each vertex with two vertices joined by an edge and initializes a bipartite matching problem with this set of "gadget edges". Our analysis of this reduction in the online setting, which involves coupling the executions of the online flow and matching algorithms and proving a suitable invariant of the coupling which ensures that augmenting paths remain in one-to-one correspondence as nodes join the network, is novel to the best of our knowledge.

Our work falls under the general framework known as online algorithms with *recourse* (or *replacement*) bounds. In this model one needs to maintain optimal (or near-optimal) solutions at each step while keeping the total number of changes needed to transform the solutions from one step to the next as low as possible. Problems that have been studied in this framework include Job Scheduling [16, 17, 18], Maximal Independent Set [19, 20, 21] and Minimum Set Cover [21]. In the context of flows and matchings, [22] consider similar problems and study the trade-offs between the approximation ratio and the recourse bound. [16] also consider problems involving flow computation on bipartite graphs in a decremental model where edges and nodes disappear over time.

Our theoretical investigation of minimizing reconfiguration changes in network routing is inspired by, and intended to complement, the investigation of the same topic in the systems literature. Following the introduction of systems such as B4 [2] and SWAN [3] that use a centralized controller in conjunction with a solver to repeatedly re-optimize the distribution of traffic over routing paths in a network, a few papers have investigated methods for reducing the rate of "churn" in such a system. SMORE [7] proposes to do so by restricting flows to a statically selected set of routing paths chosen using Räcke's [23] oblivious routing algorithm, and only using the solver to re-optimize the distribution of traffic over this limited set of paths. It is known that in the worst case, no such "semi-oblivious" routing scheme can achieve a constant-factor approximation to the optimum congestion unless it uses exponentially many paths [24]. SOL [6] supports generating new routing paths each time the solver runs, but it allows the user to specify constraints that bound (or minimize) the logical distance between the new configuration and its predecessor, to ensure that churn remains at a tolerable level. The same idea of limiting churn by bounding the logical distance between two consecutive configurations is used by Niagara [5] but at the level of entries in an individual switch's rule-table, rather than at the level of paths in a network.

2 Notation/Preliminaries

In this section we explain some basic assumptions and notations.

2.1 Online Flow Computation A network is a directed[1] graph $G = (V, E)$ with two special disjoint sets of vertices $S, T \subseteq V$ called *sources* and *sinks* respectively along with a capacity function $c : (V \cup E) \to \mathbb{R}_+$ on vertices and edges. Vertices in $V \backslash (S \cup T)$ will sometimes be referred to as *internal*. The sets V, E will sometimes be denoted as $V(G), E(G)$. In what follows we'll only consider *simple* networks which contain no parallel edges or self-loops.

A flow on a network G is a function $f : E \to \mathbb{R}_{\geq 0}$. Given f, we define $f_{\text{in}}, f_{\text{out}} : V \to \mathbb{R}_{\geq 0}$ as $f_{\text{in}}(v) =$

[1] Our model can also handle undirected graphs as follows: Replace each undirected edge with two directed edges of the same capacity, one for each direction. Then delete all in-going edges to sources and all out-going edges from sinks.

$\sum_{u:(u,v)\in E} f(u,v)$ and $f_{\text{out}}(v) = \sum_{u:(v,u)\in E} f(v,u)$. Abusing notation we'll denote by $f(v)$ for a vertex v the net flow imbalance at v, $f(v) = f_{\text{in}}(v) - f_{\text{out}}(v)$. We call f a *valid flow* if:

1. $0 \leq f(e) \leq c(e)$ for every edge $e \in E$.
2. $f_{\text{in}}(v) \leq c(v)$ and $f_{\text{out}}(v) \leq c(v)$ for every vertex $v \in V$.
3. $f(v) = 0$ for all $v \in V \backslash \{S, T\}$
4. $-1 \leq f(v) \leq 0$ for $v \in S$ and $0 \leq f(v) \leq 1$ for $v \in T$.

The *value* of a flow f is defined as $|f| = \sum_{v\in T} f(v)$. We'll call a network G *flow-admissible* if there exists a valid flow of value $|S|$ i.e. where every source sends one unit of flow. We'll denote by G_f the *residual graph* of G based on flow f defined the standard way. A vertex $v \in S \cup T$ is free if $f(v) = 0$. An augmenting path is a path in G_f from a free client to a free server.

A simple directed network $G = (V, E)$ for which $c(x) = 1$ for every $x \in V \cup E$ will be called *unit-vertex-capacitated*. For those networks, we'll make the additional assumption that sources have 0 in-degree and sinks have 0 out-degree. The assumption is without loss of generality, in the sense that edges into sources and out of sinks can be deleted from the network without changing the maximum flow value. That is because every flow path that passes through a source (respectively, sink) can be truncated to begin at that source (respectively, end at that sink) without changing the flow value.

Online Setting We are going to work on an *incremental online setting* where sources arrive one at a time, requesting to send a unit amount of flow through the network to an arbitrary sink. The part of the graph containing the vertices $V\backslash S$ and the edges between them is known in advance. The new information that becomes available when a source arrives is the set of edges originating from that source.

More formally, denote by $S = \{x_1, x_2, \ldots, x_{|S|}\}$ the set of sources. Then, for any integer $\tau \in [0..|S|] = \{0, 1, \ldots, |S|\}$, let G^τ be the induced subgraph of G on the vertex set $S^\tau \cup (V\backslash S)$ where $S^\tau = \{x_j \mid 1 \leq j \leq \tau\}$ is the set of the sources that have arrived by time τ. At any point, we have to maintain a valid flow of maximum value (*max flow*) in G^τ — assuming G is flow-admissible — while minimizing the total *flow-switching cost*, defined as follows.

Definition 2.1. (Flow-switching cost) *Let f_τ be the maximum flow maintained after the first $\tau < |S|$ sources have arrived and $f_{\tau+1}$ the flow maintained on the next iteration. We define the cost of switching from flow f_τ to flow $f_{\tau+1}$ as:*

$$SC^G(f_\tau, f_{\tau+1}) = \sum_{e\in E(G)} |f_\tau(e) - f_{\tau+1}(e)|$$

The total flow-switching cost *of an algorithm $\mathcal{A}$ which maintains flows $f_0, f_1, \ldots, f_{|S|}$ is defined as:*

$$FSC^G(\mathcal{A}) = \sum_{\tau=0}^{|S|-1} SC^G(f_\tau, f_{\tau+1})$$

A path in G is represented as a sequence of edges. In the case of simple graphs (e.g. unit-vertex-capacitated ones), sequences of vertices uniquely identify a path. We'll sometimes assume P is represented as a sequence of vertices instead and we'll try to make it clear, based on the context, which definition of path we adopt at each point. In either case, the *length* of a path is defined as the number of *edges* in the edge-based representation.

Network-SAP We are interested in analyzing the performance of the *Shortest Augmenting Path* algorithm (denoted as Network-SAP to distinguish it from Bipartite-SAP to be introduced later). The algorithm works as follows: when the τ-th source arrives, choose a shortest augmenting path in the residual graph $G^\tau_{f_{\tau-1}}$ from x_τ to a free sink (breaking ties arbitrarily) and update f by pushing 1 unit of flow along that path.

2.2 Online Bipartite Matching In their paper, Bernstein et al. [8] consider the following setting for the problem of *online bipartite matching*: a bipartite undirected graph $G = (L \cup R, E)$ where R is a set of *servers* (right side) and L is a set of *clients* (left side)[2], arriving one at a time along with the edges incident to them. We'll sometimes denote the number of clients by $n_c = |L|$. In this context, we'll call a graph *matching-admissible* if a matching of size $|L|$ exists. The goal is to come up with an algorithm $\mathcal{A}$ which maintains at each point a perfect matching between the set of clients that have arrived so far and a subset of the servers while minimizing the total *vertex-switching cost* (denoted by $\text{VSC}^G(\mathcal{A})$) which is defined as the total number of times a client gets reassigned (under the matching) to a different server.

Given a matching, an *alternating path* is a sequence of edges alternating between not belonging to the matching and belonging to it. A vertex is *free* under a matching M if it's not an endpoint of any edges in M. An *augmenting path* is an alternating path from a free client to a free server.

The algorithm considered in [8] is what we'll call *Bipartite Shortest Augmenting Path* (Bipartite-SAP):

[2]The sets L and R are not necessarily of the same size

when a client $l \in L$ arrives, update the matching by switching the state (membership in the matching) of every edge along a shortest augmenting path from l to a free server (ties broken arbitrarily). Notice that augmenting paths in bipartite graphs are of odd length and an augmenting path of length $2k+1$ contributes exactly k to the switching cost. Consequently, the total vertex-switching cost is at most half the total length of the augmenting paths used during the run of the algorithm and thus in what follows we focus on bounding the length of the augmenting paths used.

The main result in [8] is that the total vertex-switching cost of the Bipartite-SAP algorithm on a matching-admissible graph G is at most $\mathcal{O}(n_c \log^2 n_c)$. A crucial property of their technique is that reasoning about the length of the augmenting path chosen when the τ-th client arrives is independent of the specific algorithm used to match the clients which arrived before τ. This allows us to generalize in the following way.

THEOREM 2.1. (GENERALIZATION OF [8, THEOREM 1, LEMMA 6]) *Let $G = (L \cup R, E)$ be a matching-admissible bipartite graph and let M_0 be a matching covering every vertex of some set $L_0 \subseteq L$. Suppose we run Bipartite-SAP on G with an initial matching M_0 and the vertices in $L \backslash L_0$ arriving in an online fashion one at a time. The total vertex-switching cost of the algorithm is at most $\mathcal{O}(n_c \log^2 n_c)$ where $n_c = |L|$.*

For completeness, we provide the proof of this theorem — in fact an even further generalization of it needed in Section 4 — in the Appendix (Theorem 6.1).

3 From Matchings to Flows

We now describe a reduction of the online flow computation problem on unit-vertex-capacitated networks to the online bipartite matching problem which will allow us to transfer the bound on the vertex-switching cost of the Bipartite-SAP algorithm to the flow-switching cost of the Network-SAP algorithm. This is achieved by proving an equivalence between augmenting paths in the two settings, meaning that Network-SAP and Bipartite-SAP are essentially the same algorithm on slightly different graphs.

The reduction is achieved in two steps. First, we apply a standard gadget reduction which replaces every internal vertex with a pair of vertices connected with an edge effectively replacing vertex capacities with edge capacities. A consequence of this step is that the resulting graph H has a bipartite structure in which all sources are on one side and all sinks on the other. The second step is to notice that paths in H correspond to alternating paths in the undirected version $\overline{H}$ of H under an appropriate initial matching M_0, a correspondence that has also been noted by Lin in the context of reducing between *offline* flow and matching problems [14]. Furthermore this correspondence continues to hold at any time τ (as new sources arrive) between the residual graph of the modified network and the undirected version of the modified network under an appropriate matching M_τ.

REMARK 3.1. *For the rest of the section, we annotate some properties of the graph H with labels in the range (P1)—(P6). The annotations are irrelevant to this section and can be ignored by the reader on a first reading. They become relevant in Section 4 when we generalize the reduction to any graph satisfying properties labeled (P1)—(P6); we will then use the annotations to cross-reference arguments presented in this section.*

Let G be a flow-admissible unit-vertex-capacitated network with $S, T \subseteq V(G)$ its sets of sources and sinks respectively. For the first step of the reduction, we define $H = (V, E)$ by substituting each vertex u with a "left" and a "right" counterpart (u, l) and (u, r)[3]. Edges are either "internal" (those within the gadget) or "external", corresponding to edges already existing in G. If (u, v) was an edge in G then $((u, l), (v, r))$ becomes an edge of H. The reduction is illustrated in Figure 1 and formalized as follows:

$$S' = (S \times \{l\}), T' = (T \times \{r\}) \tag{3.1}$$

$$V(H) = S' \cup T' \cup ((V(G) \backslash (S \cup T)) \times \{l, r\}) \tag{3.2}$$

$$E_i = \{((u, r), (u, l)) : u \in V(G) \backslash (S \cup T)\} \tag{3.3}$$

$$E_o = \{((u, l), (v, r)) : (u, v) \in E(G)\} \tag{3.4}$$

$$E(H) = E_i \cup E_o \tag{3.5}$$

$$c(x) = \begin{cases} 1, & x \in S' \cup T' \cup E \\ +\infty, & \text{otherwise} \end{cases} \tag{3.6}$$

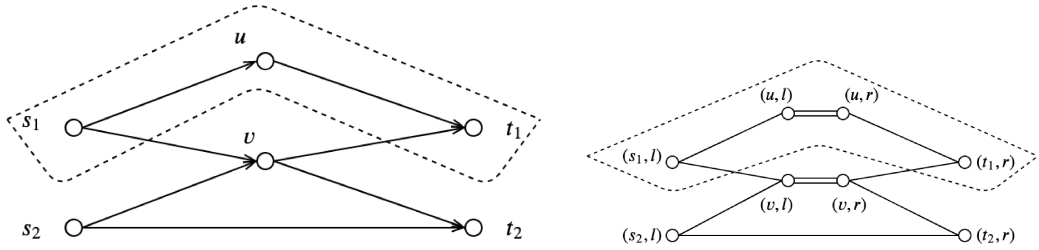
For the second step, we define the undirected analogue of a graph as follows.

DEFINITION 3.1. *Given a directed network $H = (V, E)$, define $\overline{H} = (V, \overline{E})$ where $\overline{E} = \{\{u, v\} \mid (u, v) \in E\}$*

For the particular H we just defined, notice the following properties on which the rest of the section crucially relies.

(H1) There is a bijection between edges of $\overline{H}$ and edges of H since H contains no 2-cycles **(P4)** and no parallel edges (by assumption that G is simple). Indeed, suppose there exist vertices $(u, l), (v, r)$ in H that

[3] l, r in this context are just labels we introduce to make the construction easier to define.



(a) G (b) $\overline{H}$

Figure 1: Here's an example of the gadget reduction. (1a) G is a directed flow network (edge and vertex capacities are assumed to be 1). $S = \{s_1, s_2\}, T = \{t_1, t_2\}$. An augmenting path in G is shown within the dotted set. (1b) $\overline{H}$ is the resulting undirected graph after we applied the gadget reduction and discarded the directions of the edges. Edges in M_0 are shown with double lines. The corresponding augmenting path is shown in a dotted set.

form a 2-cycle. Then $((u,r),(v,l))$ must have been introduced in E_i. But that means $u = v$ and since the reverse edge is introduced in E_o it must be that $(u,u) \in E$ which contradicts the assumption that G is simple.

(H2) The residual graph of H under any valid, integral flow f has a special structure as a consequence of the unit edge capacities **(P3)** and the absence of parallel edges and 2-cycles **(P4)**. Namely, for every $e = (u,v) \in E(H)$, either $f(e) = 1$ in which case in place of e there is a backwards edge (v,u) in H_f or $f(e) = 0$ in which case e is in H_f. In either case there is only one edge between u, v in H_f at any time.

(H3) $\overline{H}$ is bipartite under the partition $V = L \cup R$ where $L = V(H) \cap (V(G) \times \{l\}), R = V(H) \cap (V(G) \times \{r\})$ and $S' \subseteq L, T' \subseteq R$ **(P1), (P2)**. In the context of bipartite matchings, vertices in L play the role of clients, vertices in R the role of servers and S' is the set of clients arriving online.

As a reminder, we use the notation H^τ and $\overline{H}^\tau$ to denote the respective subgraphs at the time when the first τ sources have arrived. The next step is to define the initial matching:

$$M_0 = \{\{u,v\} \mid u \in L, v \in R, (v,u) \in E(H^0)\} \tag{3.7}$$

To verify that M_0 is indeed a matching, notice that $M_0 = \overline{E}_i$ (the undirected analogue of the set E_i) which is a matching by construction.

An example of the transformations described so far is shown in Figure 1.

Consider running Network-SAP starting on H^0 with an empty flow f_0 (i.e. $f_0(e) = 0$ for every $e \in E(H^0)$) and denote by f_τ the flow maintained by Network-SAP on H^τ. At the same time, consider running Bipartite-SAP starting on $\overline{H}^0$ with the initial matching M_0. Denote by M_τ the matching maintained by the algorithm on $\overline{H}^\tau$ at the end of iteration τ.

Before we proceed any further, let us introduce some helpful notation.

Definition 3.2. *Let $\phi_\tau, \psi_\tau, \chi_\tau, f_\tau : V \times V \to \{0,1\}$ be defined as follows on pairs (u,v) of adjacent vertices at time τ.*

$$\phi_\tau(u,v) = \mathbb{I}[\{u,v\} \in M_\tau]$$
$$\psi_\tau(u,v) = \mathbb{I}[(u \in L) \text{ and } (v \in R)]$$
$$\chi_\tau(u,v) = \mathbb{I}[(u,v) \in E(H^\tau)]$$

where $\mathbb{I}[P]$ is the indicator function of property P.

For f_τ, abusing notation, we define it as the symmetric extension of the flow function $f_\tau : E(H^\tau) \to \{0,1\}$, i.e. if $e = (u,v) \in E(H^\tau)$, define $f_\tau(u,v) = f_\tau(v,u) = f_\tau(e)$.

Definition 3.3. (Invariant (I)) *We say that invariant (I) holds between a flow f_τ and matching M_τ at time τ if:*

$$\phi_\tau(u,v) \oplus \psi_\tau(u,v) \oplus \chi_\tau(u,v) \oplus f_\tau(u,v) = 0$$

for all $u, v \in V$ such that $\{u,v\} \in E(\overline{H}^\tau)$.

The significance of the above invariant will become apparent from the following lemma which establishes an equivalence between augmenting paths in the network H and augmenting paths in the bipartite graph $\overline{H}$.

Lemma 3.1. *At any point τ, if invariant (I) holds between f_τ and M_τ then any sequence of vertices $P = v_0, \ldots, v_k$ that forms an augmenting path in $H^\tau_{f_\tau}$ also forms an augmenting path in $\overline{H}^\tau$ under M^τ and vice versa.*

Proof. Observe that a sequence of vertices $v_0, v_1, \ldots, v_k$ with $v_0 \in S'$ and $v_k \in T'$ constitutes an augmenting path in $\overline{H}^\tau$ if and only if

(i) v_0 and v_k are free with respect to M_τ,

(ii) for every consecutive pair $(u, v) = (v_i, v_{i+1})$ in the sequence, either $(u \in L, v \in R, \{u, v\} \notin M_\tau)$ or $(v \in L, u \in R, \{u, v\} \in M_\tau)$.

Similarly, $v_0, \ldots, v_k$ constitutes an augmenting path in $H^\tau_{f_\tau}$ if and only if

(iii) v_0 and v_k are free with respect to f_τ,

(iv) for every consecutive pair $(u, v) = (v_i, v_{i+1})$ in the sequence, either $(e = (u, v) \in E(H^\tau)$ and $f_\tau(e) = 0)$ or $(e = (v, u) \in E(H^\tau)$ and $f_\tau(e) = 1)$.

Property (ii) is equivalent to $\phi_\tau(u, v) \oplus \psi_\tau(u, v) = 1$, while property (iv) is equivalent to $\chi_\tau(u, v) \oplus f_\tau(u, v) = 1$. Thus, assuming invariant (I), an ordered pair of vertices satisfies (ii) if and only if it satisfies (iv).

To conclude the proof we must show that a vertex v is free with respect to f_τ if and only if it is free with respect to M_τ. This, too, follows from invariant (I). We will present the argument assuming $v \in T'$, omitting the symmetric argument that pertains when $v \in S'$. If $v \in T'$ is not free in M_τ it means that M_τ contains an edge $\{u, v\}$. Observe that $u \in L$ and $v \in R$. This means $\phi_\tau(u, v) = \psi_\tau(u, v) = \chi_\tau(u, v) = 1$. Invariant (I) then implies $f_\tau(u, v) = 1$, from which it follows that v is not free in f_τ. Conversely, if v is not free in f_τ then $f_\tau(v) \neq 0$, hence there exists an edge $(u, v) \in H^\tau$ such that $f_\tau(u, v) = 1$. Observing that $u \in L$ and $v \in R$ we have $\psi_\tau(u, v) = \chi_\tau(u, v) = 1$. Invariant (I) then implies $\phi_\tau(u, v) = 1$, hence v is not free in M_τ. □

Now we just need to prove that (I) indeed holds for every time $\tau \in [0..|S|]$ and the equivalence we claimed will follow.

First, let's notice some useful properties of the functions defined earlier. ϕ_τ and f_τ are symmetric on their arguments: $\phi_\tau(u, v) = \phi_\tau(v, u), f_\tau(u, v) = f_\tau(v, u)$ and ψ_τ, χ_τ are skew-symmetric in the boolean sense: $\psi_\tau(u, v) = 1 - \psi_\tau(v, u), \chi_\tau(u, v) = 1 - \chi_\tau(v, u)$ for every pair u, v of adjacent vertices at time τ. Hence when arguing that (I) holds, it suffices to prove it for only one of the pairs $(u, v), (v, u)$. Furthermore, for every τ_0, τ_1 such that $u, v \in V(H^{\min(\tau_0, \tau_1)}) : \psi_{\tau_0}(u, v) = \psi_{\tau_1}(u, v)$ and $\chi_{\tau_0}(u, v) = \chi_{\tau_1}(u, v)$, i.e. the direction of the edges as well as which endpoints belong to L vs R remains a constant throughout the run.

We are now ready to prove (I).

THEOREM 3.1. *Let f_τ, M_τ be the flow and matching maintained by the two algorithms. Then invariant (I) holds between them for all $\tau \in [0..|S|]$.*

Proof. We proceed by induction on τ. For $\tau = 0$, let u, v be arbitrary adjacent vertices in $\overline{H}^0$. Initially the flow is empty so $f_0(u, v) = 0$. By definition of $M_0, \{u, v\} \in M_0$ (or equivalently $\phi_0(u, v) = 1$) if and only if $\psi_0(u, v) \oplus \chi_0(u, v) = 1$. Hence (I) holds.

Suppose the invariant holds at time $\tau < |S|$. Now after the arrival of source $x_{\tau+1}$, a shortest augmenting path is chosen by the Network-SAP algorithm leading to a free sink — such a path is guaranteed to exist by the flow-admissibility assumption. By the inductive hypothesis and Lemma 3.1, we can assume without loss of generality that the Bipartite-SAP algorithm chooses the corresponding augmenting path on $\overline{H}^\tau$.

Consider two adjacent vertices $u, v \in E(\overline{H}^\tau)$. If $u, v \neq x_{\tau+1}$ then those vertices were already present in H^τ and as noted $\psi_{\tau+1}(u, v) = \psi_\tau(u, v), \chi_{\tau+1}(u, v) = \chi_\tau(u, v)$. Now either $\{u, v\}$ was an edge of the augmenting path in which case Network-SAP modified its flow value and Bipartite-SAP modified its presence in the matching resulting in $\phi_{\tau+1}(u, v) = 1 - \phi_\tau(u, v), f_{\tau+1}(u, v) = 1 - f_\tau(u, v)$, or it was not on the augmenting path in which case the values of ϕ, f remain the same. In either case it's easy to see that (I) holds at $\tau + 1$.

It remains to show the invariant in the case that either u or v is the new source. As mentioned before, it suffices to show it for the case $u = x_{\tau+1}$ as the other one is symmetric. So in this case $\{u, v\}$ is a new edge and $\psi_{\tau+1}(u, v) = 1$ because all sources are in L **(P1)** and $\overline{H}$ is bipartite **(P2)** and $\chi_{\tau+1}(u, v) = 1$ because sources only have outgoing edges. If (u, v) was an edge of the augmenting path then $\phi_{\tau+1}(u, v) = f_{\tau+1}(u, v) = 1$. Otherwise $\phi_{\tau+1}(u, v) = f_{\tau+1}(u, v) = 0$. So in either case (I) holds. □

To complete the reduction, we need to associate the flow-switching cost in G with the vertex-switching cost in $\overline{H}$. Lemma 3.1 and Theorem 3.1 guarantee that the two algorithms follow corresponding paths on graphs $G, \overline{H}$. It's also easy to see that there is a one-to-one correspondence between paths in G^τ and H^τ under which paths of length k become paths of length $2k - 1$. Summing up over all those contributions for every source:

$$\text{FSC}^G(\text{Network-SAP}) = \frac{1}{2}(\text{FSC}^H(\text{Network-SAP}) + |S|) \tag{3.8}$$

Now, a path of length k in H^τ contributes $\frac{1}{2}(k - 1)$ to the vertex switching cost in $\overline{H}^\tau$. Summing again over

all sources:

(3.9)
$$\text{FSC}^H(\text{Network-SAP}) = 2\text{VSC}^{\overline{H}}(\text{Bipartite-SAP}) + |S|$$

Combining the above and noticing that $|S| \leq n$:

(3.10)
$$\text{FSC}^G(\text{Network-SAP}) \leq \text{VSC}^{\overline{H}}(\text{Bipartite-SAP}) + n$$

The final step is to argue that Theorem 2.1 applies on $\overline{H}$ which amounts to showing that $\overline{H}$ is matching-admissible. The existence of M_0 is not enough since it doesn't cover all vertices in L. The assumption that G is flow-admissible allows us to prove the matching-admissibility of $\overline{H}$ as follows.

LEMMA 3.2. *If G is a flow-admissible unit-vertex-capacitated network then $\overline{H}$ is matching-admissible.*

Proof. Suppose f is a max flow in G which accommodates all sources. Due to the vertex capacity constrains, this flow must be decomposable into vertex-disjoint paths $\mathcal{Q} = \{Q_1, \ldots, Q_{|S|}\}$ in G. Let $\{P_1, \ldots, P_{|S|}\}$ denote the corresponding set of paths in H, obtained by inflating each vertex u in the interior of one of the paths into the pair of "gadget vertices" $(u, r), (u, l)$. We claim that the symmetric difference $M = \left(\bigcup_{\tau=1}^{|S|} \overline{P_\tau}\right) \oplus M_0$ is a matching which covers all vertices in L: M_0 is a matching and $\left(\bigcup_{\tau=1}^{|S|} \overline{P_\tau}\right)$ is a set of vertex-disjoint M_0-augmenting paths, one for each source in S. □

Theorem 2.1 thus implies a bound of $\mathcal{O}(|L| \log^2 |L|)$ to $\text{VSC}^{\overline{H}}(\text{Bipartite-SAP})$ and $|L| \leq |V(G)| = n$. Combining with equation 3.10 we get the following theorem.

THEOREM 3.2. *For any flow-admissible unit-vertex-capacitated network $G = (V, E)$, the flow-switching cost of the Network-SAP algorithm on G when sources arrive in an online manner is at most $\mathcal{O}(n \log^2 n)$ where $n = |V|$.*

4 More general networks

The reduction described in the previous section can in fact be generalized. The unit vertex capacities were easy to handle since the gadget of the first step transformed the network in a way that not only obviated the vertex capacity constraints, but also provided the network with the special bipartite structure that enabled us to argue that a certain equivalence holds between augmenting paths in the directed and undirected case. In this section we identify the properties that a directed graph needs to possess in order for the reduction to go through. These properties turn out to define a more general family of directed graphs than the ones produced by the construction (3.1)-(3.6) specified in the preceding section.

Let $H = (V, E)$ be a directed network with a set $S \subseteq V$ of sources and a set $T \subseteq V$ of sinks. Let L, R be a partition of its vertices. We claim that the essential properties that H must have in order for the reduction to go through are the following.

(P1) $S \subseteq L,\ T \subseteq R$.

(P2) **[Bipartite]** $E \subseteq (L \times R) \cup (R \times L)$

(P3) **[Unit capacities]** $\forall x \in (E \cup S \cup T) : c(x) = 1$ and $c(x) = +\infty$ otherwise.

(P4) **[Oriented graph]** The graph is *simple* (no loops or parallel edges) and contains *no* 2-*cycles* (for every $u, v \in V$ either $(u, v) \notin E$ or $(v, u) \notin E$).

(P5) **[Contains a matching]** The undirected analogue $\overline{H}$ (see Definition 3.1) of H contains a matching that covers all vertices in L.

(P6) The set $\{\{u, v\} \mid u \in R, v \in L, (u, v) \in E(H)\}$ is a matching in H.

Before proving it, notice that the graph H defined in the preceding section by (3.1)-(3.6) indeed satisfies all these properties. (P1)—(P4) hold by construction as noted in the previous section when defining $\overline{H}$. Property (P5) is what Lemma 3.2 asserts and (P6) holds by definition of H.

In fact, we can generalize (P6) to the following:

(P6′) **[Bounded in-degree]** $\forall l \in L : |E \cap (R \times \{l\})| \leq 1$, i.e. vertices in L have in-degree at most 1.

For a network satisfying (P6′), the set M_0 as defined in 3.7 is now a *semi-matching*, i.e. vertices in R are allowed to be matched to multiple vertices in L but each vertex in L is matched to at most one vertex in R (see Definition 6.1). As we shall see shortly, Theorem 3.1 still holds when M_τ is a semi-matching and furthermore, as we prove in the Appendix, the bound of [8] extends to when the graph is initialized with a semi-matching.

THEOREM 4.1. *For any flow-admissible network $H = (V, E)$ for which there exists a partition $V = L \cup R$ of its vertices that satisfies (P1)—(P5) and (P6′), the flow-switching cost of the Network-SAP algorithm on H when sources arrive in an online manner is at most $\mathcal{O}(n \log^2 n)$ where $n = |V|$.*

Proof. The proof is presented in Section 3. That proof relied on three properties of H that were denoted (H1)—(H3); the justifications for those properties presented in Section 3 are annotated with labels indicating how the justifications can be derived from properties (P1)—(P4). The proofs of Lemma 3.1 and Theorem 3.1 rely only on (H1)—(H3) and on the fact that M_τ is a matching. As discussed, (H1)—(H3) follow from (P1)—(P4), and M_0 is a semi-matching due to (P6′). The fact that M_τ is a semi-matching for $\tau > 0$ is proven in Lemma 6.1. Thus, the proofs of Lemma 3.1 and Theorem 3.1 remain valid if we replace "matching" with "semi-matching" throughout. Property (P5) replaces Lemma 3.2: $\overline{H}$ is matching-admissible according to the original definition of that term (not modified to incorporate semi-matchings), and the property of matching-admissibility is needed for the application of Theorem 6.1 to come.

Equation (3.9) still holds and expresses the flow-switching cost in H in terms of the vertex-switching cost in $\overline{H}$:

$$\text{FSC}^G(\text{Network-SAP}) \leq 2\text{VSC}^{\overline{G}}(\text{Bipartite-SAP}) + n$$

Theorem 6.1 now applies on $\overline{H}$ because of (P5) and (P6′) hence $\text{FSC}^H(\text{Network-SAP}) = \mathcal{O}(|L| \log^2 |L|) = \mathcal{O}(n \log^2 n)$. □

5 Discussion and open problems

We showed an $\mathcal{O}(n \log^2 n)$ bound on the switching cost of the Network-SAP algorithm in networks with unit edge and vertex capacities. Throughout the paper we assumed admissibility of the underlying network (i.e. it has the capacity to simultaneously accommodate the requests of all the sources). As [8, Observation 25] observe in the bipartite matching case, this assumption can be removed by effectively ignoring sources whose insertion doesn't increase the value of the flow: by the nature of augmenting paths, sources that never contributed to the flow have no incoming edges in the residual graph and cannot be part of any future augmenting path, hence they can be safely ignored.

The reduction of our model to the online bipartite matching problem depended crucially on the ability to generalize the results of the latter to a setting where the graph can be initialized with an arbitrary semi-matching. Our reduction is thus quite general in the sense that it can transfer switching cost bounds from one model to the other as long as these bounds hold under the aforementioned generalization to semi-matchings. We believe that the $\mathcal{O}(n \log n)$ bound of [11] on the switching cost of the Bipartite-SAP algorithm on trees might be a setting generalizable to semi-matching in which case such a generalization would entail a similar bound on the flow-switching cost of networks whose undirected counterpart is a tree.

Closing the gap between our $\mathcal{O}(n \log^2 n)$ upper bound and the $\Omega(n \log n)$ lower bound of the Network-SAP in the unit-vertex-capacitated case remains open — as is the case for the online bipartite matching problem.

In our view, the most appealing and consequential future direction is to design algorithms and prove non-trivial switching cost bounds for broader families of dynamic routing problems. For example, one could aim to prove non-trivial bounds on the switching cost of the Network-SAP (or other algorithms) when applied to:

1. networks with heterogeneous capacities on edges and/or vertices;
2. sources that may depart from the network some time after their arrival (rather than the arrival-only model assumed in this work), e.g. modeling live streams of finite duration;
3. multi-commodity versions of the problem where clients target their requests to specific servers, as in the traffic engineering application presented in Section 1;
4. combinations of the foregoing assumptions.

One can also consider algorithms that make both routing and admission control decisions. In other words, the algorithm may have the option not to accommodate the demands of every source, even if they can be accommodated. This line of investigation would aim to quantify the trade-off between the number of requests denied and the switching cost incurred while accommodating the requests served.

6 Appendix

Here we present and slightly generalize the results of [8]. The framework and relevant notation is explained in Section 2.2. We begin by generalizing the definition of a matching to that of *semi-matching* which allows vertices on the server-side of the bipartite graph to be matched to more than one client.

DEFINITION 6.1. (SEMI-MATCHING) *A subset $M \subseteq E$ of the edges of $G = (L \cup R, E)$ is a* semi-matching *if every vertex $l \in L$ has exactly one incident edge in M.*

It's important to note that semi-matchings are only used when initializing an assignment of clients to servers and we're *not* altering the definitions of free servers and augmenting paths nor the behavior of the Bipartite-SAP algorithm. When a fresh client arrives and requests to be matched, the algorithm will still try to find an

alternating path from that client to a free server and augment down that path updating the semi-matching. The following lemma guarantees that the algorithm remains well-defined. As a reminder, Bipartite-SAP is applied on a graph where a set L_0 of clients are present at time $\tau = 0$ and clients in $D = L \backslash L_0$ arrive one at a time.

LEMMA 6.1. *If M_0 is an initial semi-matching then the set M_τ maintained at the end of the τ-th iteration of the Bipartite-SAP algorithm remains a semi-matching for any $\tau \in [|D|]$.*

Proof. We proceed by induction on τ. Our assumption on M_0 covers the base case. Suppose $M_{\tau-1}$ is a semi-matching. Let u be a client on the augmenting path P_τ used to update $M_{\tau-1}$ to M_τ.

If u is the vertex x_τ that just arrived then no edge in $M_{\tau-1}$ is incident to it and exactly one edge incident to u enters M_τ.

If u is any other client then there are exactly two edges of P_τ incident on u, one belonging in $M_{\tau-1}$ and the other not. The former is in fact the only edge of $M_{\tau-1}$ incident to u because of the inductive hypothesis. Augmenting down P_τ means switching the state of those two edges so in M_τ there will still be exactly one edge incident to u. ∎

In what follows we'll denote by $N(v)$ the *neighborhood* of a vertex v and for a subset $A \subseteq L \cup R$ of the vertices, denote $N(A) = \bigcup_{v \in A} N(v)$. To prove their result, [8] introduce the notion of balanced flow. To avoid confusion with the notion of flow on a directed network we'll use the term *balanced function* instead to refer to the same concept.

DEFINITION 6.2. (BALANCED FUNCTION) *A function $\alpha : R \to \mathbb{R}_{\geq 0}$ is called* balanced *if there exist non-negative $(w_e)_{e \in E}$ such that:*

$$\begin{array}{ll} \sum_{r \in N(l)} w_{lr} = 1 & \forall l \in L \\ \sum_{l \in N(r)} w_{lr} = \alpha(r) & \forall r \in R \\ w_{lr} = 0 & \forall l \in L, r \in N(l) \backslash Act(l), \\ & Act(l) = argmin_{r \in N(l)} \alpha(r) \end{array}$$

The set $Act(l)$ is called the active neighborhood *of a client l. An edge $\{l, r\}$ where $r \in Act(l)$ is an* active edge.

Essentially we can think of each element of $l \in L$ as having one unit of some quantity it must entirely distribute to its neighbors in a "balanced" way in the sense that it cannot redistribute its quantity in a way that strictly reduces the maximum load among its neighbors.

The following lemma — the proof of which we omit and can be found in [8] — guarantees that, under a reasonable condition, a balanced function exists and is uniquely defined:

LEMMA 6.2. ([8, LEMMA 14]) *A* unique *balanced function exists for a graph $G = (L \cup R, E)$ if and only if $|N(l)| \geq 1$ for all $l \in L$.*

In what follows we'll be working with graphs where each client has at least one neighbor (a consequence of the matching-admissibility assumption) and we'll denote by $\alpha(\cdot)$ the unique balanced function associated with that graph. Balanced functions exhibit the following useful properties which we provide without proof:

LEMMA 6.3. ([8, LEMMA 21 AND 22]) *Denote by $\alpha_\tau(\cdot)$ the balanced function of the graph after the τ-th dynamic client has arrived for $\tau \in [0..|D|]$.*[4] *The change $\Delta^\tau \alpha(r) = \alpha_\tau(r) - \alpha_{\tau-1}(r)$ of each server $r \in R$ obeys the following properties for all τ:*

(a) $\Delta^\tau \alpha(r) \geq 0$ for all $r \in R$.

(b) $\Delta^\tau \alpha(r) = 0$ for all $r \in R$ such that $\alpha_{\tau-1}(r) < \mu(x_\tau) = \min_{v \in N(x_\tau)} \alpha_{\tau-1}(v)$

LEMMA 6.4. ([8, LEMMA 23]) *The bipartite graph $G = (L \cup R, E)$ contains a matching of size $|L|$ if and only if $\alpha(r) \leq 1$ for all $r \in R$.*

In order to bound the length of augmenting paths originating from a client, it is convenient to bound the length of alternating paths originating from servers in the neighborhood of that client and thus the following definition introduced in [8] is useful: an *augmenting tail* is an alternating path from a server to a free server. Notice that every augmenting path has a unique maximal augmenting tail. Similarly, an *active augmenting tail* (under some semi-matching) is an augmenting tail for which the edges not belonging to the semi-matching are active.

We now proceed to prove a generalization of an essential lemma in [8]. This lemma allows us to assert the existence of an augmenting tail of bounded length from each server r based on how far $\alpha(r)$ is from 1. Later, using the fact that each client contributes 1 to the sum of the balanced function values and Lemma 6.4 we'll be able to bound the total length of the augmenting paths used by the Bipartite-SAP.

LEMMA 6.5. (GENERALIZATION OF [8, EXPANSION LEMMA]) *Let M be an semi-matching of a matching-admissible bipartite graph $G = (L \cup R, E)$ and $r \in R$*

[4] For $\tau = 0$, $\alpha_0(\cdot)$ is the balanced function of the graph where the only clients present are the ones in L_0.

with $\alpha(r) \leq 1-\epsilon$ for some $\epsilon > 0$. Then there exists an active augmenting tail from r to a free server of length at most $\frac{2}{\epsilon}\ln(|L|)$.

Proof. Notice that any server r' reachable from r by an active augmenting tail will have $\alpha(r') \leq \alpha(r) \leq 1-\epsilon$ by the definition of active edges.

Let K_i for $i \geq 1$ be the set of all clients reachable from r by an active augmenting tail of length at most $2i-1$, thus $K_1 \subseteq K_2 \subseteq \ldots \subseteq K_i$, and let $k_i = |K_i|$. Denote $\bigcup_{l \in K_i} \mathrm{Act}(l)$ by $\mathrm{Act}(K_i)$.

We have:

$$\begin{aligned} k_i = |K_i| &\leq \sum_{r' \in \mathrm{Act}(K_i)} \alpha(r') \\ &\leq \sum_{r' \in \mathrm{Act}(K_i)} (1-\epsilon) \\ &= |\mathrm{Act}(K_i)|(1-\epsilon) \end{aligned}$$

Now suppose there is no active augmenting tail of length $\leq 2i$. This means that all servers in $\mathrm{Act}(K_i)$ are matched under M and furthermore, since M is a semi-matching, the function that maps each element of K_{i+1} that belongs to an edge of M to the opposite endpoint of that edge is a surjection from a subset of K_{i+1} to $\mathrm{Act}(K_i)$, so $k_{i+1} \geq |\mathrm{Act}(K_i)|$.

Thus, $k_{i+1} \geq \frac{k_i}{1-\epsilon}$ i.e. the set of clients reachable by an active augmenting tail of length $2i-1$ expands by a factor of at least $\frac{1}{1-\epsilon}$ at each increment of i. Consequently, $|L| \geq k_{i+1} \geq \left(\frac{1}{1-\epsilon}\right)^i k_1 \geq \left(\frac{1}{1-\epsilon}\right)^i$, and thus $i \leq \frac{\ln|L|}{\ln\frac{1}{1-\epsilon}}$. Using the inequality $1-\epsilon < e^{-\epsilon}$ we get $i < \frac{1}{\epsilon}\ln|L|$.

We have shown that the hypothesis that no active augmenting tail has length $\leq 2i$ implies that $i < \frac{1}{\epsilon}\ln|L|$. Hence, there must exist a free server reachable by an active augmenting tail of length at most $\frac{2}{\epsilon}\ln|L|$. □

We are now ready to state and prove the main theorem which closely follows the proof techniques in [8].

THEOREM 6.1. (GENERALIZATION OF [8, THEOREM 1, LEMMA 6]) *Let $G = (L \cup R, E)$ be a matching-admissible bipartite graph and let M_0 be a semi-matching covering every vertex of some set $L_0 \subseteq L$. Suppose we run Bipartite-SAP on G with an initial semi-matching M_0 and the vertices in $D = L \backslash L_0$ arriving in an online fashion one at a time. The total vertex-switching cost of the algorithm is at most $\mathcal{O}(n_c \log^2 n_c)$ where $n_c = |L|$.*

Proof. Recall that $\mathrm{VSC}^G(\text{Bipartite-SAP}) \leq \frac{1}{2}\sum_{\tau=1}^{|D|}|P_\tau|$ where $|P_\tau|$ is the length of the τ-th augmenting path. Denoting by $m(h)$ the number of augmenting paths among the P_τ whose length is at least h, the sum can alternatively be computed as $\sum_{\tau=1}^{2n_c} m(h)$ — an augmenting path cannot have more than $2n_c$ edges. We're going to bound $m(h+2)$ from above by $\frac{4n_c \ln n_c}{h}$ from which then the result follows:

$$\begin{aligned} \mathrm{VSC}^G(\text{Bipartite-SAP}) &\leq \frac{1}{2}\sum_{h=1}^{2n_c} m(h) \\ &= \frac{1}{2}\left(m(1) + m(2) + \sum_{h=1}^{2n_c-2} m(h+2)\right) \\ &\leq \frac{1}{2}\left(|D| + |D| + 4n_c \ln n_c \sum_{h=1}^{2n_c}\frac{1}{h}\right) \\ &= \mathcal{O}(n_c \log^2 n_c) \end{aligned}$$

For what follows fix an h and assume $h > 4\ln n_c$; otherwise the bound $m(h+2) \leq n_c$ is trivial.

Let x_τ be a client whose insertion resulted in an SAP of length $|P_\tau| \geq h+2$ and denote by $\alpha_{\tau-1}$ the balanced function before x_τ arrives. It follows that every server $r \in N(x_\tau)$ must have $\alpha_{\tau-1}(r) > 1 - \frac{2\ln n_c}{h}$ and thus $\mu(x_\tau) > 1 - \frac{2\ln n_c}{h}$. (Recall $\mu(x_\tau) = \min_{v \in N(x_\tau)} \alpha_{\tau-1}(v)$.) For supposing $\alpha_{\tau-1}(v) \leq 1 - \frac{2\ln n_c}{h}$ for some $v \in N(x_\tau)$, by Lemma 6.5 there must exist an active augmenting tail of length at most h and so an augmenting path from x_τ of length at most $h+1$ which is a contradiction.

Consider the sequence of sets $S_\tau = \{r \in R \mid \alpha_\tau(r) > 1 - \frac{2\ln n_c}{h}\}$ of clients for $\tau \in [0..|D|]$. By part (a) of Lemma 6.3, those sets are nested: $S_0 \subseteq S_1 \subseteq \ldots \subseteq S_{|D|}$. Define $\tau_0(r) = \min\{\tau \in [0..|D|] \mid r \in S_\tau\}$. Now we can bound $m(h+2)$ as follows:

$$m(h+2) = \sum_{\tau:|P_\tau|\geq h+2} 1 \tag{6.11}$$

$$= \sum_{\tau:|P_\tau|\geq h+2} \sum_{r\in R} \Delta^\tau \alpha(r) \tag{6.12}$$

$$= \sum_{\tau:|P_\tau|\geq h+2} \sum_{r\in S_{\tau-1}} \Delta^\tau \alpha(r) \tag{6.13}$$

$$\leq \sum_{1\leq\tau\leq|D|} \sum_{r\in S_{\tau-1}} \Delta^\tau \alpha(r) \tag{6.14}$$

$$= \sum_{r\in S_{|D|-1}} \sum_{\tau_0(r)<\tau\leq|D|} \Delta^\tau \alpha(r) \tag{6.15}$$

$$= \sum_{r\in S_{|D|-1}} \left(\alpha_{|D|}(r) - \alpha_{\tau_0(r)}(r)\right) \tag{6.16}$$

$$< \sum_{r\in S_{|D|-1}} \left(1 - \left(1 - \frac{2\ln n_c}{h}\right)\right) \tag{6.17}$$

$$\leq |S_{|D|}| \cdot \frac{2\ln n_c}{h} \tag{6.18}$$

Equation (6.12) holds because every new client contributes a total of 1 unit to the sum of the balanced functions. Equation (6.13) follows from part (b) of Lemma 6.3. Equation (6.15) is reversing the order of summation. Equation (6.16) is expanding the telescoping sum and equation (6.17) follows from the observation that $\alpha_{|D|}(r) \leq 1$ for all servers r and $\alpha_{\tau_0(r)}(r) > 1 - \frac{2\ln n_c}{h}$ as observed in a previous paragraph. The rest of the equations and inequalities follow easily using part (a) of Lemma 6.3 that all $\Delta^\tau \alpha(r) \geq 0$.

The final step of the proof is to bound $|S_{|L\setminus L_0|}|$. Notice that there are n_c total clients which can contribute to the values $\alpha_{|D|}(r)$ so $n_c \geq |S_{|D|}| \cdot \left(1 - \frac{2\ln n_c}{h}\right) \geq \frac{|S_{|D|}|}{2}$ where the second inequality follows from the assumption that $h > 4\ln n_c$.

Hence we can conclude that $m(h+2) \leq \frac{4n_c \ln n_c}{h}$.

□

References

[1] Leonidas Kontothanassis, Ramesh Sitaraman, Joel Wein, Duke Hong, Robert Kleinberg, Brian Mancuso, David Shaw, and Daniel Stodolsky. A transport layer for live streaming in a content delivery network. *Proceedings of the IEEE*, 92(9):1408–1419, 2004. Special issue on evolution of Internet technologies.

[2] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jonathan Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a Globally Deployed Software Defined WAN. In *Proceedings of ACM SIGCOMM 2013*, 2013.

[3] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving High Utilization with Software-Driven WAN. In *Proceedings of ACM SIGCOMM 2013*, 2013.

[4] Xin Jin, Hongqiang Liu, Rohan Gandhi, Srikanth Kandula, Ratul Mahajan, Jennifer Rexford, Roger Wattenhofer, and Ming Zhang. Dionysus: Dynamic Scheduling of Network Updates. In *Proceedings of ACM SIGCOMM 2014*, 2014.

[5] Nanxi Kang, Monia Ghobadi, John Reumann, Alexander Shraer, and Jennifer Rexford. Efficient traffic splitting on commodity switches. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, pages 6:1–6:13, New York, NY, USA, 2015. ACM.

[6] Victor Heorhiadi, Michael K. Reiter, and Vyas Sekar. Simplifying software-defined network optimization using SOL. In *13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016, Santa Clara, CA, USA, March 16-18, 2016*, pages 223–237, 2016.

[7] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. Semi-oblivious traffic engineering: The road not taken. In *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.

[8] Aaron Bernstein, Jacob Holm, and Eva Rotenberg. Online bipartite matching with amortized o(log2 n) replacements. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 947–959, Philadelphia, PA, USA, 2018. Society for Industrial and Applied Mathematics.

[9] B. Bosek, D. Leniowski, P. Sankowski, and A. Zych. Online bipartite matching in offline time. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 384–393, Oct 2014.

[10] Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych-Pawlewicz. Shortest augmenting paths for online matchings on trees. *Theory of Computing Systems*, 62(2):337–348, Feb 2018.

[11] Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych-Pawlewicz. A tight bound for shortest augmenting paths on trees. In Michael A. Bender, Martín Farach-Colton, and Miguel A. Mosteiro, editors, *LATIN 2018: Theoretical Informatics*, pages 201–216, Cham, 2018. Springer International Publishing.

[12] K. Chaudhuri, C. Daskalakis, R. D. Kleinberg, and H. Lin. Online bipartite perfect matching with augmentations. In *IEEE INFOCOM 2009*, pages 1044–1052, April 2009.

[13] Edward F. Grove, Ming-Yang Kao, P. Krishnan, and Jeffrey Scott Vitter. Online perfect matching and mobile computing. In Selim G. Akl, Frank Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures*, pages 194–205, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[14] Henry Lin. Reducing directed max flow to undirected max flow, 2009. unpublished manuscript.

[15] Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 253–262. IEEE, 2013.

[16] Steven Phillips and Jeffery Westbrook. Online load balancing and network flow. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 402–411, New York, NY, USA, 1993. ACM.

[17] Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Math. Oper. Res.*, 34(2):481–498, May 2009.

[18] Jeffery Westbrook. Load balancing for response time. *J. Algorithms*, 35(1):1–16, April 2000.

[19] Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 815–826, New York, NY, USA, 2018. ACM.

[20] Keren Censor-Hillel, Elad Haramaty, and Zohar Karnin. Optimal dynamic distributed mis. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, pages 217–226, New York, NY, USA, 2016. ACM.

[21] Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 537–550, New York, NY, USA, 2017. ACM.

[22] Anupam Gupta, Amit Kumar, and Cliff Stein. Maintaining assignments online: Matching, scheduling, and flows. In *Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 468–479, Philadelphia, PA, USA, 2014. Society for Industrial and Applied Mathematics.

[23] Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 255–264, New York, NY, USA, 2008. ACM.

[24] MohammadTaghi Hajiaghayi, Robert Kleinberg, and Tom Leighton. Semi-oblivious routing: Lower bounds. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 929–938, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.