


# Shortest Augmenting Paths for Online Matchings on Trees

Bartłomiej Bosek<sup>1</sup>, Dariusz Leniowski<sup>2</sup>, Piotr Sankowski<sup>2</sup>, and Anna Zych<sup>2</sup>

<sup>1</sup> Theoretical Computer Science Department,  
Faculty of Mathematics and Computer Science,  
Jagiellonian University, Kraków, Poland

`bosek@tcs.uj.edu.pl`

<sup>2</sup> University of Warsaw,  
Warsaw, Poland

**Abstract.** The shortest augmenting path (SAP) algorithm is one of the most classical approaches to the maximum matching and maximum flow problems, e.g., using it Edmonds and Karp in 1972 have shown the first strongly polynomial time algorithm for the maximum flow problem. Quite astonishingly, although it has been studied for many years already, this approach is far from being fully understood. This is exemplified by the online bipartite matching problem. In this problem a bipartite graph  $G = (W \uplus B, E)$  is being revealed online, i.e., in each round one vertex from  $B$  with its incident edges arrives. After arrival of this vertex we augment the current matching by using shortest augmenting path. It was conjectured by Chaudhuri et al. (INFOCOM'09) that the total length of all augmenting paths found by SAP is  $O(n \log n)$ . However, no better bound than  $O(n^2)$  is known even for trees. In this paper we prove an  $O(n \log^2 n)$  upper bound for the total length of augmenting paths for trees.

## 1 Introduction

The shortest augmenting path (SAP) algorithm is one of the most classical approaches to the maximum matching and maximum flow problems. Using this idea Edmonds and Karp in 1972 have shown the first strongly polynomial time algorithm for the maximum flow problem [5]. Quite astonishingly, although this idea is one of the most basic algorithmic techniques, it is far from being fully understood. It is easier to talk about it by introducing the online bipartite matching problem. In this problem a bipartite graph  $G = (W \uplus B, E)$  is being revealed online, i.e., in each round one vertex from  $B$  with its incident edges arrives. After arrival of this vertex we augment this matching by using shortest augmenting path. It was conjectured by Chaudhuri et al. [4] that the total length of augmenting paths found by SAP is  $O(n \log n)$ . However, no better bound than  $O(n^2)$  is known even for trees. Proving this conjecture would have quite striking

---

This work was supported by NCN Grant 2013/11/D/ST6/03100, ERC StG project PAA1 259515 and FET IP project MULTIPLEX 317532.

consequences even for maximum flow problem, as it would show that the total length of augmenting paths in unit capacity networks in Edmonds-Karp algorithm is  $O(m \log n)$ . This consequence is obtained via the bipartite line graph construction that is used to reduce the max-flow problem to maximum matching problem [10]. The obtained bipartite line graph has  $2m$  vertices.

Our paper contributes to the study of SAP algorithm by showing that in the case of trees the total length of all augmenting paths is bounded by  $O(n \log^2 n)$ . This result is obtained via the application of the heavy-light decomposition of trees [15] combined with charging technique that carefully assigns shortest augmenting paths to the structure of the tree. Although, this result seems to be restricted only to trees we believe that it constitutes the first nontrivial progress towards resolving the above conjecture. Moreover, we actually conjecture here that trees are the worst-case examples for this problem. It seems that adding more edges can only help the SAP algorithm. In addition to that we explain why SAP is harder to analyze than other augmenting path algorithms, even though it seems way more natural.

## 2 Related Work

The online bipartite matching problem with augmentations has recently received increasing research attention [3, 4, 6, 7]. There are several reasons to study this problem. First of all, it provides a simple solution to the online bipartite matching algorithms used in many modern applications such as online advertising (e.g. Google Ads) [11] or client-server assignment [4]. Secondly, they could give rise to new effective offline bipartite matching algorithms as in [3]. This new algorithm provides new insights to the old problem that was studied for decades.

In this paper we concentrate on bounding the total length of augmenting paths and not on the running time. With this respect, it was shown that if the vertices of  $B$  appear in a random order, the expected total paths' length for SAP is  $O(n \log n)$  [4]. The worst-case total length of paths remains an open question even for trees. In the class of trees the authors of [4] proposed a different augmenting path algorithm that achieves total paths' length of  $O(n \log n)$ . On the other hand, for general bipartite graphs greedy ranking algorithm [3] guarantees  $O(n\sqrt{n})$  total length of paths.

First of all, the above study of online bipartite matching with augmentations should be related to the work of Gupta et al. [7] which shows an  $O(n)$  bound on the total length of paths, but allows to exceed the capacity of each server by a constant factor.

Another point of view is given by the dynamic matching algorithms. Most papers in this area consider edge updates in a general fully-dynamic model which allows for both insertions and deletions intermixed with each other. We note, however, that the exact results in this model [9, 14] do not imply any bound on the number of changes to the matching. Much faster update times can be achieved by constant approximate algorithms, for example [1, 13], which achieve polylogarithmic and logarithmic update times. Yet, the 2-approximation can be obtained in our setting by trivial greedy algorithm that performs no changes at all.

Better approximation factor of  $\frac{3}{2}$  was achieved by [12] in  $O(\sqrt{m})$  update time, and then improved by Gupta and Peng to  $(1 + \varepsilon)$  in  $O(\sqrt{m}\varepsilon^{-2})$  [8]. The  $O(\sqrt{m})$  barrier was broken by Bernstein and Stein who gave a  $(\frac{3}{2} + \varepsilon)$ -approximation algorithm that achieves  $O(m^{1/4}\varepsilon^{-2.5})$  update time [2]. The same paper proposes an  $(1 + \varepsilon)$ -approximation algorithm in very fast  $O(\alpha(\alpha + \log n) + \varepsilon^{-4}(\alpha + \log n) + \varepsilon^{-6})$  update time for the special case of bipartite graphs with constant arboricity. However, when allowing approximation in our model a much better results are possible. An  $(1 + \varepsilon)$  approximation in  $O(m\varepsilon^{-1})$  total time and with  $O(n\varepsilon^{-1})$  total length of paths was shown in [3].

### 3 Preliminaries

Let us define the matching problem we consider more formally. Let  $W$  and  $B$  be two sets of vertices over which the bipartite graph will be formed. The set  $W$  (called white vertices) is given up front to the algorithm, whereas the vertices in  $B$  (black vertices) arrive online. We denote by  $G_t = \langle W \uplus B_t, E_t \rangle$  the bipartite graph after the  $t$ 'th black vertex has arrived. The graph  $G_t$  is constructed online in the following manner. We start with  $G_0 = \langle W \uplus B_0, E_0 \rangle = \langle W \uplus \emptyset, \emptyset \rangle$ . In turn  $t$  a new vertex  $b_t \in B$  together with all its incident edges  $E(b_t)$  is revealed and  $G_t$  is defined as:

$$\begin{cases} E_t = E_{t-1} \cup E(b_t), \\ B_t = B_{t-1} \cup \{b_t\}; \end{cases}$$

The goal of our algorithm is to compute for each  $G_t$  the maximum size matching  $M_t$ . For simplicity we assume that we add in total  $|W|$  black vertices. The final graph  $G_{|W|}$  which is obtained in this process will be denoted by  $G = \langle W \uplus B, E \rangle$ . We denote  $n = |W| = |B|$  and  $m = |E|$ .

For every  $t \in [n]$ , we add orientation to edges of the graph  $G_t$ . This orientation is induced by matching  $M_t$ : the matched edges are oriented towards black vertices, while the unmatched edges are oriented towards white vertices. When a new vertex  $b_t$  arrives, we get an intermediate orientation  $G_t^{\text{int}} = (E_t^{\text{int}}, B_t)$ , where the edges of  $b_t$  are oriented towards its neighbors, and the rest of the edges is oriented according to  $M_{t-1}$ . Note that  $G_t^{\text{int}}$  and  $G_{t-1}$  differ only by one vertex  $b_t$ . Any simple directed path in  $G_t^{\text{int}}$  from  $b_t$  to some unmatched white vertex is an augmenting path. In turn  $t$ , if  $b_t$  can be matched, the edges of  $G_t^{\text{int}}$  are reoriented along augmenting path  $\pi_t$  chosen by the algorithm, and the resulting orientation is  $G_t$ . The unmatched white vertices are called *seeds*. We denote the set of seeds after turn  $t$  as

$$S_t = \{w \in W : wb \notin M_t \text{ for any } b \in B\}.$$

So in turn  $t$  the augmenting paths in  $G_t^{\text{int}}$  are the directed paths from  $b_t$  to some  $s \in S_{t-1}$ . We refer to the seed of the path  $\pi_t$  from turn  $t$  as  $s_t$ , where  $s_t \in S_{t-1}$ . We represent a path as a graph consisting of path vertices and path edges. We use the notation  $v \xrightarrow{\pi} v'$  to denote that a (directed) path  $\pi$  starts in  $v$  and ends in  $v'$ , and  $v \rightarrow v'$  to denote a connection via a directed edge. We use the notation

$v \in \pi$  and  $\rho \subseteq \pi$  to state that a vertex  $v \in V(\pi)$  and that a path  $\rho$  is a subgraph of  $\pi$ , respectively. We also denote the length of a path  $\pi$  as  $|\pi|$ . Throughout the paper, when we write “at time  $t$ ”, what we formally mean is “in  $G_t^{\text{int}}$ ”.

The next thing we define is a set of vertices  $\mathcal{D}_t$  called dead at time  $t$ . The set  $\mathcal{D}_t$  is defined as the set of vertices in  $G_t^{\text{int}}$  that cannot reach  $S_{t-1}$  via a directed path in  $G_t^{\text{int}}$ . Observe, that if at some point there is no directed path from a vertex to a seed, never again there will be such a path. If a vertex is dead, all vertices reachable from it are dead as well. Hence, no alternating path can enter such a dead region and reorient its edges to make some vertices alive. In other words,  $\mathcal{D}_t \subseteq \mathcal{D}_{t+1}$  for every time moment  $t$ . The vertices of  $\mathcal{D}_t$  are called dead, while the remaining vertices are called alive.

We now define the effective degree of a black vertex  $b$  in turn  $t$  as the number of it's non-dead out-neighbors:

$$\text{degeff}_t(b) = |\Gamma_t(b) \setminus \mathcal{D}_t|$$

where  $\Gamma_t(b)$  is the set of vertices  $v$  such that  $b \rightarrow v$  in  $G_t^{\text{int}}$ , referred to sometimes as out-neighbours of  $b$ . In particular  $\text{degeff}_t(b_t)$  is the number of all non-dead neighbors (in the undirected sense) of  $b_t$ , as all the edges adjacent to  $b_t$  are directed towards its neighbors.

Since we consider in this paper the special case when  $G_t$  is a tree at any time  $t$ , we will refer to  $G$  as  $T$ , and to  $G_t$  as  $T_t$  from now on.

## 4 Run-away-from-the-root Algorithm

In this section we present the algorithm for trees given in [4], whose total augmenting paths' length amounts to  $O(n \log n)$ . We refer to this algorithm as RAFR or as the run-away algorithm. We briefly explain why their analysis does not apply to the SAP algorithm.

The RAFR algorithm maintains a forest  $\mathcal{F}$ , which is exactly the set of trees composed of the edges and vertices already revealed. Each tree of the forest is rooted. Initially, the trees are all singleton vertices of  $W$ . In turn  $t$ , vertex  $b_t$  connects to some trees of  $\mathcal{F}$ . Three cases are distinguished:

1.  $\text{degeff}_t(b_t) > 1$ : in this case  $b_t$  connects at least two trees, in which there are two disjoint directed paths connecting  $b_t$  with a seed. We pick the smallest such tree and route the alternating path over there. The root of the newly connected tree is the root of the largest such tree.
2.  $\text{degeff}_t(b_t) = 1$ : we pick a path that minimizes the number of edges traversed towards the root. In other words, we choose a path that runs away from the root as soon as possible.
3.  $\text{degeff}_t(b_t) = 0$ : no path is possible,  $b_t$  immediately becomes a member of a dead region.

The analysis of the above algorithm in terms of the total length of the augmenting paths is as follows. We count, for every edge, how many times this edge

is traversed via augmenting paths. The edge can be either traversed when Case 1 applies (we refer to such traversal as connecting) or when Case 2 applies (we refer to such traversal as non-connecting). The edge can be connecting-traversed no more than  $\log n$  times, as each augmenting path applied in Case 1 implies that the tree containing this edge doubles its size. We now observe, that between two connecting traversals or after/before the last/first connecting traversal of an edge there can be at most two non-connecting traversals. The edge, before it changes its root (Case 1 applies again), can be traversed towards the root (and reversed the opposite direction) only once. This is because after such reversal the endpoint of the traversed edge becomes dead. As a consequence, every edge is reversed  $O(\log n)$  number of times and the total length  $O(n \log n)$  of all applied augmenting paths follows.

This algorithm cleverly plans the uniform distribution of work between the edges. By running away from the root, it distributes the work to the edges furthest from the root, and does not do unnecessarily pass through the edges that are closer to the root. In the following sections we analyze a shortest augmenting path algorithm, which is not as clever. In particular, there are examples where a single edge can be traversed  $\Omega(\sqrt{n})$  times. Hence, the simple charging techniques for RAFR do not apply to SAP.

## 5 Shortest Paths on Trees

In this section we study the shortest augmenting path (SAP) algorithm, which in each turn chooses the shortest among all available augmenting paths. We start by giving an easy argument, that the total length of augmenting paths for SAP is  $O(n \log n)$  if all vertices  $b_t$  satisfy  $\text{degeff}_t(b_t) > 1$ . This shows that the difficult case is to deal with vertices of effective degree 1.

**Lemma 1.** *If for each  $t \in [n]$  it holds that  $\text{degeff}_t(b_t) > 1$ , then the total length of all augmenting paths applied by SAP is  $O(n \log n)$ .*

*Proof.* Due to the definition of effective degree, every vertex  $b_t$  connects at least two trees  $T_1$  and  $T_2$  that contain a directed path connecting  $b_t$  with a seed. Let  $T_1$  be a smaller of the two trees. The length of the shortest path  $\pi_t$  from  $b_t$  to a seed is at most the size of  $T_1$ . We charge the cost of  $\pi_t$  to  $|\pi_t|$  arbitrary vertices of  $T_1$ . During the course of the SAP algorithm, every vertex can be charged at most  $\log n$  times, as each time it is charged, the size of its tree doubles. The total charge is hence  $O(n \log n)$ .  $\square$

The main result of this paper and the subject of the remainder of this section is the bound for the general case, stated in the following theorem.

**Theorem 1.** *The total length of augmenting paths applied by SAP is  $O(n \log^2 n)$ .*

In order to prove Theorem 1 we introduce a few definitions and observations. The core of our proof is the concept of a *dispatching vertex*.

**Definition 1.** A black vertex  $b$  is called *dispatching* at time  $t$  if  $\text{degeff}_t(b) > 1$  and  $b$  is the first from  $b_t$  such vertex on  $\pi_t$ . In such case we write  $b = \text{dis}(\pi_t)$ . If there is no such black vertex on  $\pi_t$ , we define  $s_t$  to be the dispatching vertex at time  $t$ . We also define, for every dispatching black vertex  $b$ , the time moment  $\text{tlast}(b)$ , when  $b$  is dispatching for the last time.

So every path  $\pi_t$  applied by SAP is assigned a uniquely defined dispatching vertex  $\text{dis}(\pi_t)$ . The first observation we make is that we only have to care about suffixes of  $\pi_t$ 's starting with  $\text{dis}(\pi_t)$ .

**Definition 2.** We define the split of  $\pi_t = \mu_t \rho_t$ , where  $\rho_t$  is the suffix of  $\pi_t$  such that  $\text{dis}(\pi_t) \xrightarrow{\rho_t} s_t$ . Path  $\mu_t = \pi_t \setminus \rho_t$  is the remaining part of  $\pi_t$  (a possibly empty prefix that ends in a vertex preceding  $\text{dis}(\pi_t)$ ). We sometimes refer to the above defined suffixes as *dispatching paths*.

**Lemma 2.** The total length of paths  $\mu_t$  is linear in the size of the tree  $T$ , i.e.,

$$\sum_{t \in [n]} |\mu_t| \in O(n)$$

*Proof.* The lemma holds due to Observation 2, proven below, which states that vertices of  $\mu_t$  die at the time  $t$  when  $\pi_t$  is applied. With this observation it is clear that the time  $\mu_t$  passes through a vertex is the last time SAP visits that vertex. So every vertex in the tree is visited by  $\mu_t$  for any  $t$  at most once.  $\square$

**Observation 2.** Vertices of  $\mu_t$  die at the time  $t$  when  $\pi_t$  is applied.

*Proof.* At the time when  $\pi_t$  is applied, all vertices on  $\mu_t$  have effective degree equal to 1, i.e., they have only one alive directed out-neighbour — their successor on  $\mu_t$ . If we reverse the edges, the only chance for the vertices of  $\mu_t$  to be alive is the last vertex  $b_t$ . This vertex however becomes dead, because its only alive out-neighbour is removed. As a consequence the whole path dies.  $\square$

To bound the total length of augmenting paths  $\pi_t$ , it remains to bound the total length of dispatching paths:  $\sum_{t \in [n]} |\rho_t|$ . Observe, that  $\rho_t = s_t$  if  $s_t = \text{dis}(\pi_t)$ , so there is no need to worry about such paths. It is enough to consider the sum over all dispatching paths  $\rho_t$  that start in a black dispatching vertex, so from now on we focus our attention on those. As a consequence, our goal is to bound the following sum.

**Lemma 3.** The total length of non-trivial dispatching paths is  $O(n \log^2 n)$ , i.e.,

$$\sum_{\substack{t \in [n]: \\ \text{dis}(\pi_t) \in B}} |\rho_t| \in O(n \log^2 n)$$

The proof of Lemma 3 constitutes of two steps presented by the following two lemmas. We first bound the total length of dispatching paths which start with a dispatching vertex  $b \in B$  at a time before  $b$  is dispatching for the last time (such paths are called non-final):

**Lemma 4.** *The total length of non-final dispatching paths is  $O(n \log n)$ , i.e.,*

$$\sum_{\substack{t \in [n]: \\ b = \text{dis}(\pi_t) \in B \\ t < \text{tlast}(b)}} |\rho_t| \in O(n \log n)$$

Then we move on to bounding the sum of dispatching paths starting in a dispatching vertex  $b \in B$  at a time when  $b$  is dispatching for the last time (such paths are called final):

**Lemma 5.** *The total length of final dispatching paths is  $O(n \log^2 n)$ , i.e.,*

$$\sum_{\substack{t \in [n]: \\ b = \text{dis}(\pi_t) \in B \\ t = \text{tlast}(b)}} |\rho_t| \in O(n \log^2 n)$$

The distinction between final and non-final dispatching paths is made for the sake of clarity of our proofs. We now continue with the proof of Lemma 4, stated again below:

*Lemma 4* The total length of non-final dispatching paths is  $O(n \log n)$ , i.e.,

$$\sum_{\substack{t \in [n]: \\ b = \text{dis}(\pi_t) \in B \\ t < \text{tlast}(b)}} |\rho_t| \in O(n \log n)$$

*Proof.* We first observe that every time some vertex  $b \in B$  is dispatching not for the first time, one of its neighbours dies. To be more specific, if  $b = \text{dis}(\pi_t)$  and  $\pi_t$  does not start in  $b$  (what happens every but the first time  $b$  is dispatching), then  $w \rightarrow b \subseteq \mu_t$  for some neighbour  $w$  of  $b$ . Based on Observation 2, the vertex  $w$  dies.

Hence, if  $b$  is a dispatching vertex for the  $k$ -th out of  $l$  times at some time moment, then it has at least  $l - k + 2$  alive white out-neighbours at that time. We say that a subtree hangs in the neighbour  $w$  of  $b$ , if it is obtained by the removal of  $b$  from  $T$  and it contains  $w$ . Suppose that we discard two neighbors of  $b$  with the heaviest trees hanging in them, i.e., two heaviest neighbours. Then for  $k = l - 1$  we have at least one alive neighbor, for  $k = l - 2$  we have at least two alive neighbors, that is, at least one alive neighbor other than the neighbor used at  $k = l - 1$ , and so on. In other words, for any  $k < l$  we can find a distinct, not already assigned, alive neighbor  $w$  different than the two heaviest neighbors of  $b$ . However, the size of the subtree hanging in that neighbour bounds the length of the shortest augmenting path starting at  $b$ . Therefore, we can bound the total length of non-final paths dispatching at  $b$  by the total size of all subtrees of  $b$  except the two heaviest. Summing that up over the whole tree gives us a  $O(n \log n)$  upper bound, as shown by the next lemma.  $\square$

**Lemma 6.** *Let  $T$  be any unrooted tree of size  $n$ . For any vertex  $v$  let  $S^v = \langle S_0^v, S_1^v, \dots \rangle$  be the sequence of subtrees of  $v$  (i.e., the connected components of  $T \setminus \{v\}$ ) ordered descending by their size, that is,  $|V(S_i^v)| \geq |V(S_{i+1}^v)|$ . Then for*

$$\Psi(v) = \sum_{i=2}^{|S^v|-1} |V(S_i^v)|$$

we have  $\sum_{v \in V(T)} \Psi(v) \in O(n \log n)$ .

*Proof.* Let  $r$  be a centroid point of  $T$ , that is, a vertex such that  $|V(S_0^r)| \leq \frac{1}{2}|V(T)|$ . We root  $T$  at  $r$ , and perform the heavy-light decomposition of  $T$  (see Definition 3). Observe that for all vertices  $v \neq r$  we have that  $S_0^v$  contains  $r$  (it corresponds to the parent of  $v$ ) and  $S_1^v$  corresponds to the biggest child of  $v$ . In other words, at most  $S_0^v$  and  $S_1^v$  can be connected by heavy edges, all the other subtrees  $S_2^v, S_3^v, \dots$  are connected by light edges.

Now we take an arbitrary vertex  $w$  and calculate how many times it can appear in  $\sum_{v \in V(T)} \Psi(v)$ . Suppose  $v$  is a vertex that counts  $w$  in  $\Psi(v)$ , then the first edge on the path from  $v$  to  $w$  has to be light, moreover,  $S_0^v$  is not counted in  $\Psi(v)$ , so that path cannot pass through the parent of  $v$ . Because of that  $v$  has to be an ancestor of  $w$ , however, there are at most  $O(\log n)$  light edges on any path from  $w$  to the root  $r$  for any  $w$ . In other words, there can be at most  $O(\log n)$  vertices that count  $w$  in its sum of  $\Psi$ . Summing that for all vertices of  $T$  we get the desired bound of  $O(n \log n)$ .  $\square$

We continue with the proof of Lemma 5, stated again below.

*Lemma 5* The total length of final dispatching paths is  $O(n \log^2 n)$ , i.e.,

$$\sum_{\substack{t \in [n]: \\ b = \text{dis}(\pi_t) \in B \\ t = \text{tlast}(b)}} |\rho_t| \in O(n \log^2 n)$$

*Proof.* In order to bound the sum as claimed, we introduce some additional structure on  $T$ . We decompose  $T$  into paths which cover  $T$ . We pick an arbitrary vertex of  $T$  as a root. We adopt the heavy-light decomposition defined below.

**Definition 3.** *In the heavy-light decomposition each non-leaf node selects one heavy edge - the edge to the child that has the greatest number of descendants (breaking ties arbitrarily). The selected edges form the paths of the decomposition (called heavy paths). These heavy paths partition the vertices of  $T$ . Let  $\text{pheavy}(v)$  denote the heavy path containing  $v$ . A light edge is an edge of  $T$  that is not heavy.*

By construction, every path in  $T$  contains at most  $O(\log n)$  light edges. In Fig. 1, the heavy paths in the tree are marked bold.

Now fix a black dispatching vertex  $b$  and the last time  $t = \text{tlast}(b)$  when  $b$  is dispatching. We bound the length of  $\rho_t$  by the length of  $\lambda_t$ , which is a path from  $b$  to a seed in  $S_{t-1}$ , that leaves each heavy path as soon as possible.

To be more precise, we define  $\text{closest}_t(v)$  as the closest vertex reachable from  $v$  at time  $t$  (in  $T_t^{\text{int}}$ ), which belongs to  $\text{pheavy}(v)$  and has a light directed edge to an



alive child at time  $t$ . Note that such a vertex exists if  $v$  is black and dispatching. Let  $\text{light-child}_t(v)$  be the alive light child of  $v$  such that  $v \rightarrow \text{light-child}_t(v)$  at time  $t$  if such child exists. We now define a sequence of vertices:

$$\begin{cases} f_0 = b \\ e_i = \text{closest}_t(f_{i-1}) \text{ for } i = 1 \dots k \\ f_i = \text{light-child}_t(e_i) \text{ for } i = 1 \dots k \end{cases}$$

where  $k$  is the index when we reach a seed, i.e., either  $e_i \in S_{t-1}$  or  $f_i \in S_{t-1}$ . We define  $\lambda_t = f_0 \rightarrow e_1 \rightarrow f_1 \rightarrow \dots \rightarrow e_k/f_k$ , see Fig. 1 for an illustration. Note, that  $\lambda_t$  is only defined for such  $t$ , that  $t = \text{tlast}(b)$  for some black dispatching vertex  $b$ . We introduce a useful observation before we proceed.

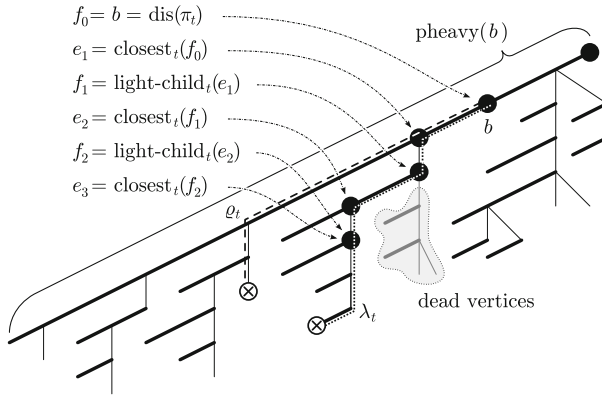


Fig. 1. The heavy-light decomposition and the definition of  $\lambda_t$

**Observation 3.** Any  $\lambda_t$  move towards the root only via heavy edges.

As mentioned before, as  $\rho_t$  is the shortest path to a seed, we charge the cost of  $\rho_t$  onto the vertices of  $\lambda_t$ , which is certainly at least as long. The argument we are pursuing is going to be completed by the claim that every vertex is charged at most  $O(\log^2 n)$  times during the runtime of SAP.

To that end we introduce the last definitions and observations. For any vertex  $v$  we define  $\text{heavy-charge}(v)$  to be the set of black dispatching vertices  $b \in \text{pheavy}(v)$  such that at time  $t = \text{tlast}(b)$  paths  $\lambda_t$  charge onto  $v$ :

$$\text{heavy-charge}(v) = \{b \in \text{pheavy}(v) \cap B : b = \text{dis}(\pi_t) \text{ and } t = \text{tlast}(b) \text{ and } v \in \lambda_t\}$$

We emphasize here that a black dispatching vertex  $b = \text{dis}(\pi_t)$  of  $\text{pheavy}(v)$  can charge  $\lambda_t$  onto  $v$  at most once, and hence  $\text{heavy-charge}(v)$  is not a multiset.

Now fix a vertex  $w$ . We count how many times  $w$  is charged. Let  $\text{charge}(w)$  be the set of all black dispatching vertices that charge onto  $w$  the last time when they are dispatching:

$$\text{charge}(w) = \{b \in B : b = \text{dis}(\pi_t) \text{ and } t = \text{tlast}(b) \text{ and } w \in \lambda_t\}$$

Clearly,  $|\text{charge}(w)|$  is the total number of times the vertex  $w$  is charged and that is what we want to bound. To complete the argument, we introduce one more definition.

**Definition 4.** *The head of a heavy path  $\pi$ , denoted as  $\text{head}(\pi)$ , is the closest to the root vertex of  $\pi$  (closest in the undirected sense). A light ancestor of a vertex  $v$ , denoted as  $\text{light-ancestor}(v)$ , is the parent in the tree  $T$  of the head of the heavy path containing  $v$ , i.e.,  $\text{light-ancestor}(v)$  is a parent of  $\text{head}(\text{pheavy}(v))$ .*

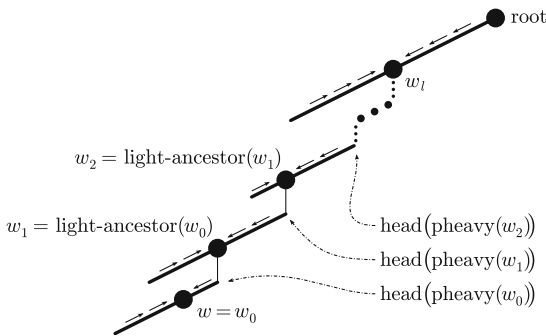
We now define a sequence of vertices starting with  $w_0 = w$ , such that  $w_i = \text{light-ancestor}(w_{i-1})$ , for  $i = 1 \dots l$ , where  $l$  is such that  $\text{head}(\text{pheavy}(w_l))$  is the root of  $T$ . By the definition of the heavy-light decomposition,  $l \in O(\log n)$ . We observe that the black dispatching vertices that can potentially charge onto  $w$  are the vertices in  $V(\text{pheavy}(w_0)) \cup \dots \cup V(\text{pheavy}(w_l))$ . Moreover,

$$\text{charge}(w) \subseteq \bigcup_{i=0}^l \text{heavy-charge}(w_i)$$

since every black dispatching vertex that charges onto  $w$  that is in  $V(\text{pheavy}(w_i))$  charges also onto  $w_i$ . Since sets  $V(\text{pheavy}(w_i))$  are pairwise disjoint, this implies

$$|\text{charge}(w)| \leq \sum_{i=0}^l |\text{heavy-charge}(w_i)|$$

For the illustration of our construction of the charging scheme see Fig. 2. The black arrows mark the heavy charges of vertices  $w_i$ , which sum up to the total charge of  $w$ .

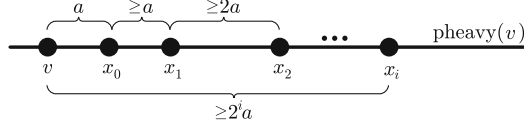


**Fig. 2.** The charging scheme

Below we prove Lemma 7, which states that for all  $v \in V(T)$  it holds that  $|\text{heavy-charge}(v)| \in O(\log(|\text{pheavy}(v)|))$ . Having Lemma 7 at our disposal,

we have  $|\text{charge}(w)| \leq \sum_{i=0}^l |\text{heavy-charge}(w_i)| \in O(\log^2 n)$ . This completes the proof of Lemma 5.  $\square$

**Lemma 7.** *For all  $v \in V(T)$  it holds that  $|\text{heavy-charge}(v)| \in O(\log(|\text{pheavy}(v)|))$ .*



**Fig. 3.** The illustration to the proof of Lemma 7

*Proof.* We partition vertices of  $\text{heavy-charge}(v) = X \cup Y \cup Z$  into pairwise disjoint sets  $X, Y, Z$ . We let  $X = \{x_0 \dots x_p\}$  be the ancestors of  $v$  in  $\text{heavy-charge}(v)$  ordered in a way that  $x_{i+1}$  is the ancestor of  $x_i$ . Similarly, let  $Y = \{y_0, \dots, y_q\}$  be the descendants of  $v$  in  $\text{heavy-charge}(v)$  ordered in a way that  $y_{i+1}$  is a descendant of  $y_i$ . Finally we set  $Z = \{v\}$  if  $v$  is a black dispatching vertex, otherwise  $Z = \emptyset$ . We focus on the number of vertices in  $X$  first. We use here  $d(\bullet, \bullet)$  to denote the distance between two vertices in  $T$  in the undirected sense. Let  $a = d(v, x_0)$ . We prove inductively that  $d(x_i, v) \geq 2^i a$ , see Fig. 3 for an illustration. The claim clearly holds for  $i = 0$ . Now assume it holds for  $j \leq i$  for some  $i$ . Consider  $x_i$  and  $x_{i+1}$ . Let  $t_i = \text{tlast}(x_i)$  and  $t_{i+1} = \text{tlast}(x_{i+1})$ . We distinguish two cases:

1.  $t_i < t_{i+1}$ . By definition of  $\lambda_{t_i}$ , it holds that  $d(\text{closest}_{t_i}(x_i), x_i) \geq d(x_i, v) \geq 2^i a$ . Because  $x_i$  is dispatching at time  $t_i$ , there is at time  $t$  an alternative path  $\lambda'_{t_i}$  (going up the tree towards  $x_{i+1}$ ) from  $x_i$  to a seed. Consider again two cases:

- (a)  $\lambda'_{t_i}$  does not cross  $x_{i+1}$ . This means that  $\lambda'_{t_i}$  leaves  $\text{pheavy}(v)$  in a vertex  $u$  that has a directed edge to a light alive child at time  $t_i$ . Hence,

$$d(x_i, x_{i+1}) \geq d(x_i, u) \geq d(\text{closest}_{t_i}(x_i), x_i) \geq 2^i a$$

so  $d(x_{i+1}, v) \geq 2^{i+1} a$ .

- (b)  $\lambda'_{t_i}$  crosses  $x_{i+1}$ . Then,  $x_{i+1}$  at time  $t_i$  has a directed edge to an alive light child. This holds because if all reachable light children of  $x_{i+1}$  at time  $t_i$  are dead, then  $\text{degeff}_t(x_{i+1}) \leq 1$  remains for  $t \geq t_i$ , so  $x_{i+1}$  cannot be dispatching at time  $t_{i+1}$ . So, since  $x_{i+1}$  does have a directed edge to an alive light child at time  $t_i$ , we get  $d(x_{i+1}, x_i) \geq d(x_i, v)$  and thus  $d(x_{i+1}, v) \geq 2^{i+1} a$ .
2.  $t_{i+1} < t_i$ . By definition,  $\lambda_{t_{i+1}}$  crosses  $x_i$ . By a similar argument as above, at time  $t_{i+1}$  vertex  $x_i$  has a directed edge to a light alive child. This is a contradiction, as in such case  $\lambda_{t_{i+1}}$  leaves  $\text{pheavy}(v)$  in  $x_i$ .

The claim that we proved implies that  $|X| \in O(\log |\text{pheavy}(v)|)$ . We analogously show that  $|Y| \in O(\log |\text{pheavy}(v)|)$ . Since  $|Z| \leq 1$ , we obtain  $|\text{heavy-charge}(v)| = |X \cup Y \cup Z| \in O(\log |\text{pheavy}(v)|)$ . This completes the proof of Lemma 7 and the proof of Theorem 1.  $\square$

## References

1. Baswana, S., Gupta, M., Sen, S.: Fully dynamic maximal matching in  $O(\log n)$  update time. In: Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, pp. 383–392. IEEE Computer Society, Washington, DC, USA (2011)
2. Bernstein, A., Stein, C.: Fully dynamic matching in bipartite graphs. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9134, pp. 167–179. Springer, Heidelberg (2015)
3. Bosek, B., Leniowski, D., Sankowski, P., Zych, A.: Online bipartite matching in offline time. In: 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, 18–21 October 2014, pp. 384–393. IEEE Computer Society (2014)
4. Chaudhuri, K., Daskalakis, C., Kleinberg, R.D., Lin, H.: Online bipartite perfect matching with augmentations. In: INFOCOM 2009, 28th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 19–25 April 2009, Rio de Janeiro, Brazil, pp. 1044–1052. IEEE (2009)
5. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM* **19**(2), 248–264 (1972)
6. Grove, E.F., Kao, M.Y., Krishnan, P., Vitter, J.S.: Online perfect matching and mobile computing. In: Akl, S.G., Dehne, F., Sack, J.-R., Santoro, N. (eds.) Algorithms and Data Structures. Lecture Notes in Computer Science, vol. 955, pp. 194–205. Springer, Heidelberg (1995)
7. Gupta, A., Kumar, A., Stein, C.: Maintaining assignments online: matching, scheduling, and flows. In: Chekuri, C., (ed.) Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, 5–7 January 2014, pp. 468–479. SIAM (2014)
8. Gupta, M., Peng, R.: Fully dynamic  $(1+e)$ -approximate matchings. In: IEEE 54th Annual Symposium on Foundations of Computer Science, pp 548–557 (2013)
9. Ivković, Z., Lloyd, E.L.: Fully dynamic maintenance of vertex cover. In: Leeuwen, J. (ed.) Graph-Theoretic Concepts in Computer Science. Lecture Notes in Computer Science, vol. 790, pp. 99–111. Springer, Heidelberg (1994)
10. Karp, R.M., Upfal, E., Wigderson, A.: Constructing a perfect matching is in random NC. *Combinatorica* **6**(1), 35–48 (1986)
11. Mehta, A., Saberi, A., Vazirani, U., Vazirani, V.: Adwords and generalized on-line matching. In: 46th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2005, pp. 264–273, October 2005
12. Neiman, O., Solomon, S.: Simple deterministic algorithms for fully dynamic maximal matching. In: Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing, STOC 2013, pp. 745–754. ACM, New York, NY, USA (2013)
13. Onak, K., Rubinfeld, R.: Dynamic approximate vertex cover and maximum matching. In: Goldreich, O. (ed.) Property Testing, vol. 6390, pp. 341–345. Springer, Heidelberg (2010)

14. Sankowski, P.: Faster dynamic matchings and vertex connectivity. In: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, pp. 118–126. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2007)
15. Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. *J. Comput. Syst. Sci.* **26**(3), 362–391 (1983)