Local Dynamic Weighted Matching

Liat Atsmon Guz

Local Dynamic Weighted Matching

Research Thesis

Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Electrical Engineering

LIAT ATSMON GUZ

Submitted to the Senate of the Technion – Israel Institute of Technology
AV 5772 HAIFA JULY 2012

The Research Thesis Was Done Under the Supervision of Prof. Idit Keidar in the Faculty of Electrical Engineering.

THE GENEROUS FINANCIAL HELP OF THE TECHNION IS GRATEFULLY ACKNOWLEDGED.

Acknowledgments

I would like to thank the many people who made this work possible and made the experience truly remarkable.

I am deeply indebted to my adviser, Prof. Idit Keidar, for her continuous guidance and unwavering confidence in where we were going. Working and thinking with you has been humbling, and I thank you for always being available and keen to help when I needed it.

I would like to thank Prof. Yoram Moses, for his friendship and advice over the years. I thank my fellow students at the Technion, who struggled and succeeded with me, encouraged and motivated me. Thank you Ayelet, Kirill and Daniel, and my friends in Idit's group– Dima, Ittai, Nathaniel, Alex and Oved. I want to thank the staff of the EE faculty for their help and kindness, and I owe special thanks to Keren Seker-Gafni, who with patience and care made all the administrative fuss painless.

I want to thank my family who supported me and pampered me endlessly during stressful times; my two incredible parents who are an inspiration to me. I would also like to thank the Guz family who were drawn into this with me and were ever positive and encouraging.

With all my heart, I dedicate this work to my dear husband Zvika for never doubting me or losing patience. I was no match for your quiet determination, and here I am.

Contents

List of Figures

List of Algorithms

Abstract

We define and solve the distributed *dynamic weighted matching* problem. Namely, unlike most previous work, we consider a scenario in which the network is *asynchronous* and *dynamic*, experiencing churn, failures, topology changes, and link weight changes. An algorithm that solves the dynamic weighted matching problem should adapt to such changes in the network and constantly output a matching. We develop a new algorithm that solves this problem and guarantees a 2-approximation of the optimal solution. Moreover, we show that following local changes, the algorithm converges back to a 2-approximation after $O(1)$ rounds.

Chapter 1

Introduction

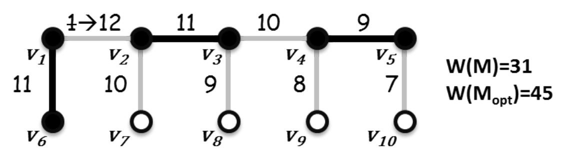
In the *distributed weighted matching* problem, nodes in a communication graph (V, E) have to collectively find a subset $M \subseteq E$ in which no two links share a common node and the sum of the link weights is maximized. The matching problem naturally maps to scheduling transmissions in multihop wireless networks under primary interference constraints [21, 17, 15]. Primary interference constraints in such networks imply that transmissions of adjacent nodes should be scheduled such that at any given time each node communicates with at most a single neighbor. Since at any given time, some links may be more important than others (e.g., due to packet queue length, priority, etc.), a weight is attached to each link, and the goal of the scheduling algorithm is to maximize the overall matching weight.

While distributed weighted matching has received a lot of attention lately [20, 7, 19, 18, 22, 29, 1, 28, 13, 16], most of the previous work has focused on solving the *one-shot* problem in *static*, *synchronous*, *fault-free* networks. Unfortunately, these settings are not a good fit for real world networks such as wireless networks, which are *asynchronous*, *dynamic*, and *fault-prone*. First, synchronous algorithms rely on a clock synchronization framework which is a very hard problem that is not necessarily local [10]. Second, wireless networks change over time as nodes join, leave, or move around, and as communication links appear and disappear. Moreover, link weights continuously change as packets are accumulated or transmitted. Therefore, when utilized in dynamic networks (e.g., for wireless transmission scheduling), these algorithms must be periodically re-run afresh. This is a suboptimal approach due to the inherent tradeoff it induces between the staleness of the matching in use and communication and computation costs.

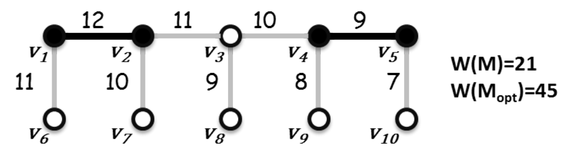
To remedy these shortcomings, we introduce in Chapter 3 the *dynamic weighted matching* problem, which is appropriate for asynchronous, fault prone, dynamic networks. Solution to this problem reacts immediately to changes and hence is never stale. For example, when used for wireless scheduling, it allows nodes to transmit packets over links that are part of the current matching while the algorithm converges, without consideration of parts of the network that might have changed or where the solution might not have converged to the final (stable) matching yet. In this context, we are interested in the matching weights at all times, including when the algorithm is running. When the network changes do cease, the matching should also eventually stabilize.

The only other asynchronous matching algorithm we are aware of is *greedy maximal matching*, originally presented as a centralized algorithm in [26], later distributed by Hoepman [13], and adapted to be self stabilizing in [20] (see Chapter 2). Under this greedy approach, an edge is added to the matching if it has maximum weight among all its adjacent edges. This solution converges to a 2-approximation of the optimal. It works well in a static network, when starting from an empty matching, but is not ideal in a dynamic setting. Specifically, it suffers from two undesirable properties: First, it can take linear time to converge back to the approximation, even after a single change. Second, this greedy approach can *reduce* a stable matching's weight, even following a change that increases the weight of links in the graph, and when no links in the matching are affected by the change. An example of both drawbacks can be seen in Figure 1.1.

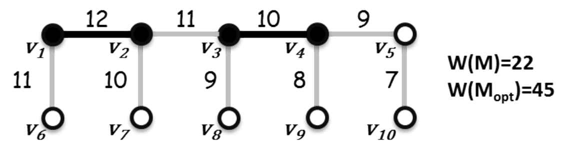
In Chapter 4, we develop an alternative approach that does not suffer from these limitations; we present a novel distributed dynamic weighted matching algorithm, which emits a matching throughout its operation and copes with any combination of network changes. In our approach, links are added to the matching only if they increase its weight. In Chapter 5, we informally argue that our algorithm stabilizes, quiesces, and provides a 2-approximation when changes cease; detailed formal proofs are deferred to the appendices. Furthermore, we show that after stabilization, our algorithm handles new changes *locally*, in the sense that it converges back to a 2-approximation in constant time following any single change – node addition or removal, or link addition, weight change,



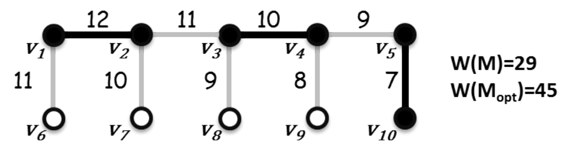
(a) (v_1,v_2) is locally heaviest following the change.



(b) (v_1,v_2) is added; (v_3,v_4) becomes locally heaviest.



(c) (v_3,v_4) is added; (v_5,v_{10}) becomes locally heaviest.



(d) (v_5,v_{10}) is added; finally back to 2-approximation.

Figure 1.1: An example of the drawbacks of using greedy maximal matching in a dynamic setting. Dark lines and full circles denote links and nodes in the matching, respectively. We see that: (i) convergence to the approximation takes linear time following a single link change; and (ii) the matching weight decreases following a change that increases the weight of a link and does not affect any of the links in the previous stable matching.

or removal. Moreover, when used in a static setting, our algorithm has the same linear time and message complexity as the greedy maximal matching algorithm distributed by Hoepman [13].

To summarize, the main contribution of this thesis is addressing the problem of maximum weight matching in an asynchronous, fault-prone, dynamic network. We define the dynamic weighted matching problem, and develop a local and stable algorithm that solves the problem.

Chapter 2

Background

The problem of graph matching has been around for over a century. Up until the late 1950s only solutions for the bipartite problem were known. Claude Berge [3] was the first to come up with a solution for general graphs through a Lemma that came to be known as Berges Augmenting Path Theorem. Nearly a decade later, Jack Edmonds' [9] showed that both maximum matching and maximum weighted matching can be solved in polynomial time. Over the years, many papers improved on Edmonds result. The best known result to date is due to Gabow [12] which solved the problem of maximum weighted matching in $O(|V| \cdot |E| + |V|^2 \cdot log(|V|))$. Since algorithms for maximum weighted matching have super linear running times, approximation algorithms for the problem have attracted more and more attention. Preis [26] was the first that suggested a centralized solution for the 2-approximation weighted matching in $O(|E|)$ running time. Drake et al. [8] built on the work of Preis [26] and derived a simple algorithm that achieves the same result.

Several recent works concentrate on synchronized weighted matching algorithms in a distributed environment. Wattenhofer et al. [29] presented the first distributed algorithms for the approximated maximum weighted matching. For general graphs, they gave a randomized solution that runs in $O(log^2(|V|))$ time and yields a 5-approximation, with high probability. Lotker et al. [19], improved on this result and designed a randomized $(4+\epsilon)$-approximation distributed algorithm whose running time is $O(log(|V|))$ with high probability. In a later paper, Lotker et al. [18] further improved on this result and provided a randomized algorithm with similar running time that converges to a $(2+\epsilon)$-approximation of the maximum weighted matching. Panconesi et al. [25] devised a 6-approximation algorithm which is the first deterministic algorithm for weighted matching with poly-logarithmic running time. For the special case of bounded-degree and bounded-weight graphs, Banerjee et al. [1] offered a distributed algorithm with an approximation factor of $\frac{2}{3}$ while reducing the round complexity to $O(log(\frac{1}{\epsilon}) + log^*(|V|)$. Although these works provide performance guarantees, and in some cases have relatively low complexities, the proposed algorithms are not a good fit even for one-shot matching in our setting. In particular, using these algorithms would require a synchronizer to be employed to close the gap to asynchronous communication, resulting in message overhead penalties. Moreover, such an approach is only applicable to fault-free networks.

To the best of our knowledge, only two asynchronous distributed algorithms have been developed for the weighted matching problem. First, Hoepman [13] presented a distributed, asynchronous, one-shot algorithm. The algorithm yields a 2-approximation in $O(|V|)$ running time. Our algorithm has identical performance in the static case while also supporting dynamic changes. Second, Manne et al. [20] developed an asynchronous self-stabilizing weighted matching algorithm that also computes a 2-approximation. Being self-stabilizing, it deals with dynamic topology changes. Unfortunately, its self stabilization guarantees come at a very high message overhead cost – a node where the algorithm has not yet converged may need to exchange messages in every round. Moreover, following changes, the previous matching quickly destabilizes, precluding continuous use of the old matching during convergence time. In contrast, our algorithm drops links from the previous matching only when a conflicting link is added to the new matching, and converges back to a 2-approximation within a constant number of rounds.

We also note *centralized* non-weighted matching algorithms that have been developed for a dynamic setting. The following algorithms all maintain a matching under insertion or deletion of edges. Ivkovic et al. [14] designed a solution that maintains a 2-approximation for maximum matching with an amortized update time that is polynomial in n. Onak et al. [24] improved on this result and designed a data structure that achieves a constant approximation factor in amortized $O(log^2(|V|))$ time. Most recently, Baswana et al. [2] designed a randomized data structure that takes $O(log(|V|))$ expected amortized time for each update. Last, the best known non-approximation solution is due to Sankowski [27] who shows that a maximum matching can be maintained with $O(|V|^{1.495})$ computation per update.

Chapter 3

Model and Problem Definitions

3.1 Model

The network is comprised of a dynamic set of nodes, partially connected by dynamic, time-varying, and undirected weighted communication links.

Nodes and links can be dynamically added to the network, and may fail or be removed from the network. The sets of nodes and communication links at time t are denoted by V_t and E_t, respectively. A time-varying weight function $w_t : E_t \to \mathbb{R}$ is defined over the links in the network; weights in w_t are assumed to be unique, as node identities can be used to break ties. For a set of links $S \subseteq E$, the *weight* of S, denoted by $w_t(S)$ is the sum of all link weights in S at time t. Two nodes connected by a communication link at time t, are called *neighbors* at time t, and can send messages to each other. Two links are *incident* at time t if they share a common node. Note that we sometimes omit t in the notation when it is not important for the context.

Communication links are reliable (if they persist), FIFO, and asynchronous, i.e., there is no bound on message delay. Each *run* progresses in steps, where in each step some node is notified of an event (as detailed below) and is allowed to react. In a step, a node may change its internal state and send messages to its current neighbors. We informally refer to the time after the t^{th} step in a run as “time t”.

Events There are two types of events – external ones triggered by the environment, and message send events triggered by nodes. The external events are: node start, node stop, weight change, link removal, and link addition. Message send events lead to **receive** notifications, whereas external events lead to $\langle$"nbrs-update"$, \Gamma, w\rangle$ notifications, Γ and w being the notified node's current neighbors and weights of links connecting the node to its neighbors, respectively. The system starts at time 0 when an initial network (V_0, E_0) is created and each node in V_0 is started. Whenever a node v is started at any time in the run, its first step is triggered by a $\langle$"nbrs-update"$, \Gamma, w\rangle$ notification.

Nodes are notified of events in FIFO order per link exactly once. (Only the receiving node is notified of a "message send" event, and both end nodes, if exist, are notified of an external event on the link.) Once a node fails or stops it does not take further steps, and its neighbors are notified of the link removal, as noted above. When a link is removed, all pending messages on the links are lost forever; if the same link is recreated, it is empty when added.

For analysis purposes it will be useful to divide a run into non–overlapping subsequences in which all pending events are processed. Formally, we define a *round* as a non–empty subsequence of a run starting at some time t, in which each node is notified of at least all events that occurred before time t. For example, a round starting at time 0 completes once every node in V_0 receives a "nbrs-update" notification. An additional round completes once all messages sent in response to the "nbrs-update" notification are received by their destinations, in addition to notifications of other events that occur in the first round (if any).

3.2 Problem Definition

In an undirected weighted network $G = (V, E)$, a *matching* $M \subset E$ is a set of links s.t. no two links in M are incident to one another. If a link $e = (u, v) \in M$ then u is considered to be the *match* of v, and vice versa. A matching that has maximum weight among all matchings in G is called a *Maximum Weighted Matching* of G, and is denoted by M_{opt}.

In the classical distributed one-shot matching problem, the run starts at time 0, the network does not change after this time, and when the run ends – each node outputs its match. A link $(u, v) \in M$ if both nodes u and v output each other as their match. We now generalize it and define the problem of *Dynamic Weighted Matching* as follows:

Definition 1 (Dynamic Weighted Matching)**.** *(i) At any time t, every node $u \in V_t$ outputs either $\perp$ or a neighbor v as its match. (ii) If there is a time t after which the network does not change, i.e., $\forall t' > t$ $(V_{t'}, E_{t'}, w_{t'}) = (V_t, E_t, w_t)$, then eventually, there is a time when the output does not change and every node u outputs node v if and only if v outputs u.*

Another desirable property of a converging algorithm is *quiescence*, namely that after stabilization, the algorithm eventually stops sending messages. We say that the system is *quiescent* if there are no notifications or messages in the queues.

In static networks, dynamic matching reduces to the regular matching problem. In addition, at any point in a dynamic run (even before convergence), we can discuss the current matching. We denote by m the output of node u, and we define the matching at time t, M_t, as $M_t = \{(u, v) \mid m_u = v$ and $m_v = u$ at time $t\}$.

Chapter 4

Dynamic Algorithm

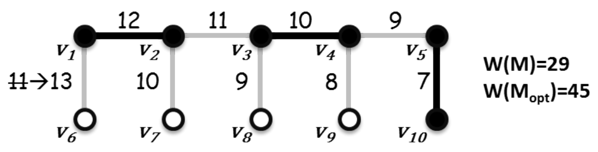
As noted above, the popular approach of greedily selecting locally heaviest links for the matching does not work well when we already have an existing matching at hand. Instead, our approach is to select links that increase the weight of the current matching. To this end, we introduce (in Section 4.1) the notion of the *gain* of a link w.r.t a matching M, and use it (first in a centralized way) to select links with positive gain. In Section 4.2 we outline the elements we need for a distributed implementation of such an approach in a dynamic network. Finally, in Section 4.3 we give a detailed description of our distributed algorithm.

4.1 Centralized Weighted Matching Using Gain

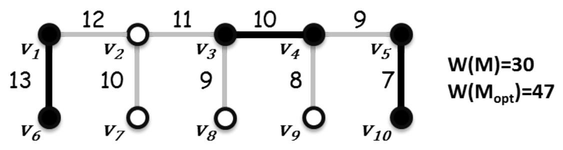
In the following we denote by *incident(e)* the set of links that are incident to e.

Given a matching M, the set $M \cup \{e\}$ is a matching only if no link incident to e is in M. Therefore, when a link is added to a matching M, all of its incident links need to be removed from M; the link's net contribution to the matching is captured by the notion of gain: $gain_M(e) \triangleq w(e) - w(M \cap incident(e))$.

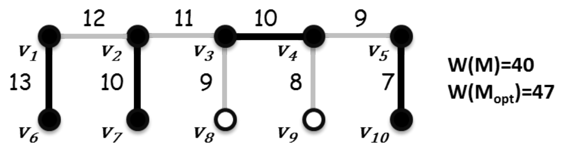
We are specifically interested in links that have a positive gain, which we call *augmenting* links. Formally, a link e is *augmenting* w.r.t M if $gain_M(e) > 0$. The following lemma establishes that when no augmenting links exist, the matching at hand provides a 2-approximation.



(a) Link ($v1$,$v6$) becomes augmenting.



(b) Link ($v2$,$v7$) becomes augmenting.



(c) The matching stabilizes.

Figure 4.1: Example run of the centralized matching algorithm.

Lemma 1. *If no augmenting links exist w.r.t a matching M, then M is a 2-approximation to M_{opt}.*

Proof. Since there are no augmenting links, for each link $e \in M_{opt} \setminus M$, $w(e) \leq (M \cap incident(e))$. Also, since M is a matching, each $f \in M$ can be incident to at most two links in M_{opt}. Therefore, $w(M_{opt} \setminus M) \leq 2 \cdot w(M \setminus M_{opt})$ and we can conclude $w(M_{opt}) \geq 2 \cdot w(M)$. $\square$

A centralized algorithm using gain is outlined in Algorithm 1. The algorithm non-deterministically chooses which augmenting link to add in each iteration. It continues to add links as long as there are augmenting ones. We could refine the algorithm, for example, to add links in a greedy way by choosing *locally maximal augmenting* ones in each iteration (i.e., augmenting links with a higher gain than all incident links to them). In the example of Figure 1.1 above, there are no augmenting links following the change, and therefore, Algorithm 1 would take no steps. Figure 4.1 shows an example run where a network change does introduce augmenting links.

Algorithm 1 Centralized Weighted Matching Using Gain

```
1: $M \subset E$, initially any matching
2: while an augmenting link exists do
3:     $a \leftarrow$ some link $(u, v)$ s.t. $gain(u, v) > 0$
4:     $M \leftarrow M \backslash \text{incident}(a)$
5:     $M \leftarrow M \cup \{a\}$
6: gain(u,v):
7:     return $w(u, v) - w(\ \{incident(u, v) \cap M\}\ )$
```

It is easy to see that if Algorithm 1 is initiated with a valid matching M, M remains a valid matching throughout the run. We show next that the algorithm terminates.

Claim 1. *Algorithm 1 terminates for any network $G = (V, E)$ and initial matching M.*

Proof. In each iteration, the weight of M increases. Since the weight of M_{opt} is bounded, the algorithm eventually terminates. □

In Lemma 2, we use the notion of gain to bound the distance of a matching M from a 2-approximation. We define the set of augmenting links w.r.t M that are in M_{opt}: $A(M) \triangleq \{a \in M_{opt} \mid gain_M(a) > 0\}$.

Lemma 2.

$$w(M) + \frac{1}{2} \sum_{a \in A(M)} gain_M(a) \geq \frac{1}{2} M_{opt}.$$

Proof. Each link in M_{opt} can belong to one of three sets: (1) $M \cap M_{opt}$, or (2) $M_{opt} \setminus (M \cup A(M))$ (non-augmenting), or (3) $A(M)$ (augmenting). From the definition of augmenting we get that for every e in (2), $w(e) < w(M \cap incident(e))$. From the definition of gain we get that for e in (3), $w(e) = w(M \cap incident(e)) + gain_M(e)$. Note that since M is a matching, each link in M can be incident to at most two links in M_{opt}, and only if it is not in M_{opt} already. This means that any link f in $M \setminus M_{opt}$, can belong to at most two $w(M \cap incident(e))$ clauses. The next step is to sum these inequalities over all edges in (2) and (3):

$$\sum_{e \in M_{opt} \setminus M} w(e) \ \leq \ 2 \cdot \sum_{f \in M \setminus M_{opt}} w(f) + \sum_{a \in A(M)} gain_M(a).$$

Finally,

$$w(M_{opt}) = \sum_{e \in M_{opt} \cap M} w(e) + \sum_{e \in M_{opt} \setminus M} w(e) \ \leq$$

$$\sum_{e \in M_{opt} \cap M} w(e) + 2 \cdot \sum_{f \in M \setminus M_{opt}} w(f) + \sum_{a \in A(M)} gain_M(a) \ \leq \ 2 \cdot w(M) + \sum_{a \in A(M)} gain_M(a).$$

□

4.2 Distributed Algorithm Overview

In a distributed implementation, the match is handled by individual nodes. Each node has a view of its neighbors Γ and a mapping w to the weights of the links connecting the node to them that is updated via "nbrs-update" messages when network changes occur. We use a subscript to mark a variable of a specific node where needed, e.g., Γ_u for the neighbors of node u. Nodes need to recognize which of their adjacent links are augmenting, which means that they need to know their neighbors' match weights. This information is sent in "match-weight" messages and kept in *nbrs-mw*.

Another challenge for the distributed algorithm is that links cannot be added atomically to the matching. Nodes must coordinate through messages. Each node *courts* a neighbor by sending a "preference" message to it; the target neighbor is saved in the variable pm. A node stores in $suitors$ the neighbors from which it receives a "preference". A node changes its match stored in m once it has both sent and received a "preference" message to and from the same node. Once a match change occurs, a "match-drop" message is sent to the node's previous match, if exists, and a "match-weight" message is sent to all other neighbors. An example of such a message exchange is given in Figure 4.2. The node outputs its match through the variables m and pm; typically m, but when m is unset it outputs pm.

Note that if u and v match, then before u can send another "preference" to v, the match must be dropped. This happens after both nodes have responded to their corresponding "preference" messages from the previous time they matched with "match weight", and hence, no "preference" messages can be pending at this point. Thus, we do not need to worry about old "preference" messages getting confused with new ones. Similarly, we do

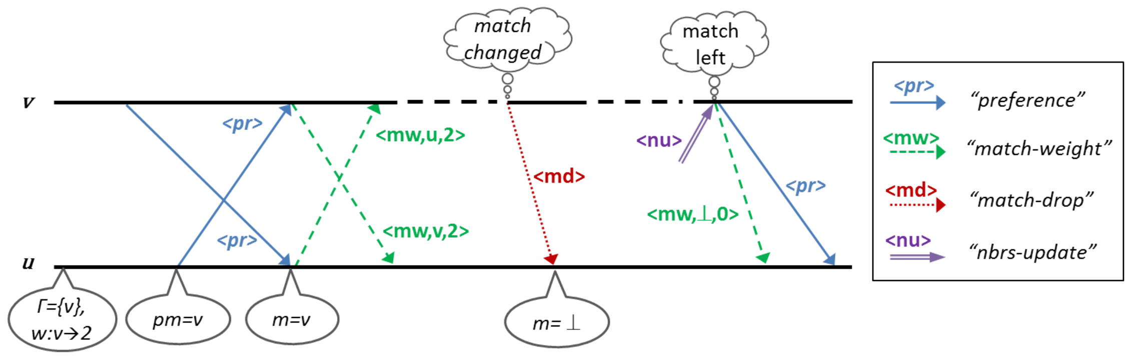


Figure 4.2: Node coordination in the distributed implementation. In this diagram, u and v match, then v matches with some other node. That node later leaves the network, which makes v update u and try to re-match with it.

not need to worry about other types of messages (e.g., “match drop”) being mixed across matching attempts.

We use a greedy approach for choosing which nodes to court. Using m,w and *nbrs-mw*, a node can tell which of the links next to it contributes the maximum gain to the matching. This is called a *maximal augmenting link*, and the neighbor at the other end of this link is called the *maximal gain neighbor*. If every node courts its maximum gain neighbor, then all nodes on maximal augmenting links exchange “preference” messages.

One difficulty that arises, however, is that the network and the matching keep changing, and with them the maximal gain augmenting links. Moreover, since communication is asynchronous, nodes might temporarily see different maximal augmenting links depending on the timing in which they receive “nbrs-update” and “match-weight” messages. One possible solution to this would be to keep sending “preference” messages. The problem with this approach is that a “preference” message is an invitation to match. However, a node only matches with the last node it sent a “preference” message to. This makes previously sent “preference” messages outdated, which may cause nodes to give up their current match for no gain. We therefore rule out this approach.

Instead, we overcome this difficulty by allowing each node to court at most one neighbor at a time. When a node’s maximal gain neighbor changes, it first tries to recall its previous preference by sending a “recall-preference” to its pm. This can lead to two possible scenarios depending on whether v is matched with u by the time the message is received

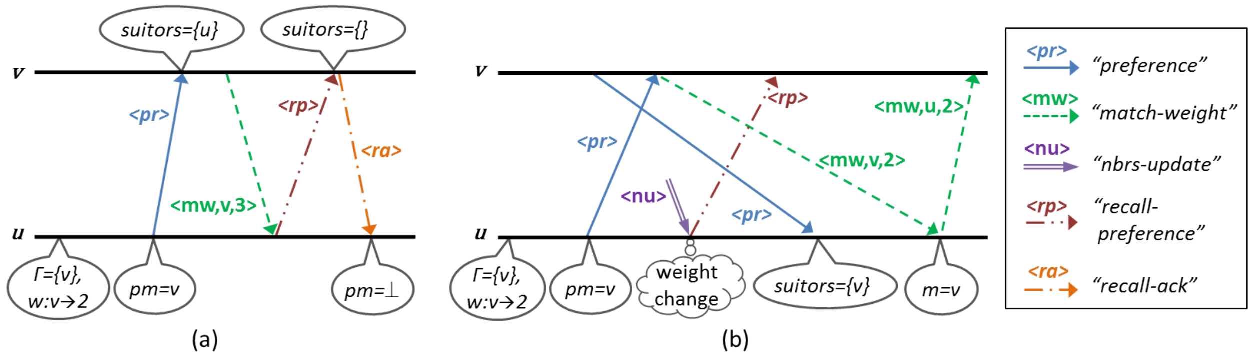


Figure 4.3: Two scenarios for "recall-preference" messages. (a) Node u's "recall-preference" is responded by "recall-ack" (and u is deleted from $suitors_v$). (b) Node u's "recall-preference" is ignored by v since v is matched with u.

by v. If it is not, then v responds with a "recall-ack" and removes u from *suitors*$_v$. When u receives the "recall-ack", pm at u is cleared, and u is free to court another neighbor. Otherwise, v is matched with u and simply ignores the "recall-preference". In this case, v has already sent a "match-weight" message to u, and when u receives it, it will match with v. Figure 4.3 shows examples of both scenarios. Either way, after a node sends a "recall-preference" message to its pm, it no longer changes its match to pm upon receiving a "preference" from pm, but does so only upon receiving a "match-weight" from its pm.

Note that he algorithm can be initiated with any matching, including an empty one.

Algorithm 2 Dynamic Weighted Matching Algorithm: code for node v

```
1:  $\Gamma \subseteq V$, initially $\emptyset$
2:  $w : \Gamma \cup \{\perp\} \rightarrow \mathbb{R}$
3:  $suitors \leftarrow \emptyset$
4:  $nbrs\text{-}mw : \Gamma \rightarrow \in \mathbb{R} \cup \{0, \perp\}$,
        initially $\perp \ \forall u \in \Gamma$
5:  $pm \in \Gamma$, initially $\perp$
6:  $pm\text{-}status$ $\in${Prefer, Recall}
7:  $m \in \Gamma$, initially $\perp$
8:  $nbrs\text{-}ack : \Gamma \rightarrow$ {True,False}, initially True

9:  **output:**
10:     **if** $m \neq \perp$ **output** $m$
11:     **else output** $pm$

12: // Initial match info was received from all neighbors
13: **ready():** $\nexists u \in \Gamma$ s.t. $nbrs\text{-}ack(u)$ =False

14: // Finds maximum-gain augmenting link
15: **best-match():**
16:     **if** $\nexists u \in \Gamma$ s.t. $gain(u) > 0$ **then**
17:         return $\perp$
18:     **else return** $\arg\max_{u \in \Gamma} gain(u)$
19: **gain(u):**
20:     **if** $u = m$ **return** 0
21:     **return** $w(u) - (nbrs\text{-}mw(u) + w(m))$

22: // Clears affected states and notifies neighbors
23: **upon** ⟨"nbrs-update", $\Gamma'$, $w'$⟩
24:     $suitors \leftarrow suitors \setminus (\Gamma \setminus \Gamma')$
25:     **if** $pm \in \Gamma \setminus \Gamma'$ **then** $pm \leftarrow \perp$
26:     **if** $m \neq \perp$ **then**
27:         // check whether my match dropped
28:         **if** $m \in \Gamma \setminus \Gamma'$ **then**
29:             $m \leftarrow \perp$
30:             $nbrs\text{-}ack(x)$ ←False $\forall x \in \Gamma' \cap \Gamma$
31:         **if** $m = \perp$ **or** $w(m) \neq w'(m)$ **then**
32:             // update old neighbors of new weight
33:             **send** ⟨"match-weight", $m, w'(m)$⟩ **to**
                    $\Gamma' \cap \Gamma \setminus \{m'\}$
34:     // update new neighbors of match weight
35:     **send** ⟨"match-weight", $m, w'(m)$⟩ **to** $\Gamma' \setminus \Gamma$
36:     $nbrs\text{-}ack(x)$ ←False $\forall x \in \Gamma' \setminus \Gamma$
37:     $\Gamma \leftarrow \Gamma'$,$w \leftarrow w'$, $m \leftarrow m'$
38:     check-pm()

39: **upon receive** ⟨"preference"⟩ **from** $u$
40:     **if** $m = u$ **return**
41:     $suitors \leftarrow suitors \cup \{u\}$
42:     **if** $pm_v = u$ **and** $pm\text{-}status$ ≠Recall **then**
43:         update-match()

44: **upon receive** ⟨"recall-preference"⟩ **from** $u$
45:     **if** $u = m$ **then return**
46:     $suitors \leftarrow suitors \setminus \{u\}$
47:     **send** ⟨"recall-ack"⟩ **to** $u$

48: **upon receive** ⟨"recall-ack"⟩ **from** $u$
49:     $pm \leftarrow \perp$
50:     check-pm()

51: **upon receive** ⟨"match-weight", $mn, mw$⟩
    **from** $u$
52:     $nbrs\text{-}mw(u) \leftarrow mw$
53:     **if** $mn = \perp$ **then**
54:         **send** ⟨"nbr-ack"⟩ **to** $u$
55:     **if** $mn = v$ **then**
56:         **if** $m \neq u$ **then**
57:             // pm must be u in this scenario
58:             update-match()
59:         // pm can be cleared: match is acknowledged
60:         $pm \leftarrow \perp$
61:     check-pm()

62: **upon receive** ⟨"nbr-ack"⟩ **from** $u$
63:     $nbr\text{-}ack(u)$ ←True
64:     check-pm()

65: **upon receive** ⟨"match-drop",mw⟩ **from** $u$
66:     **if** $u \neq m$ **return**
67:     $nbrs\text{-}mw(u) \leftarrow mw$
68:     **send** ⟨"match-weight", $\perp, 0$⟩ **to** $\Gamma$
69:     $nbrs\text{-}ack(x)$ ←False $\forall x \in \Gamma$
70:     $m \leftarrow \perp$

71: // Tries to make pm be best-match()
72: **check-pm():**
73:     **if** (**not** ready()) **or** $pm$ = best-match() **return**
74:     // Can only have one pending preference
75:     **if** $pm = \perp$ **then**
76:         $pm \leftarrow$ best-match()
77:         $pm\text{-}status$ ←Prefer
78:         **send** ⟨"preference"⟩ **to** $pm$
79:         **if** $pm \in suitors$ **then**
80:             update-match()
81:     // At most one recall, when nodes aren't matched
82:     **else if** $pm \neq m$ **and** $pm\text{-}status$ =Prefer
83:         $pm\text{-}status$ =Recall
84:         **send** ⟨"recall-preference"⟩ **to** $pm$

85: **update-match():**
86:     **if** $m \neq \perp$ **then**
87:         **send** ⟨"match-drop",w(pm)⟩ **to** $m$
88:     **send** ⟨"match-weight", $pm, w(pm)$⟩ **to** $\Gamma \setminus \{m\}$
89:     $m \leftarrow pm$
90:     $suitors \leftarrow suitors \setminus \{pm\}$
```

4.3 Distributed Algorithm

The pseudo code of our algorithm can be found in Algorithm 2. The algorithm is driven by event handlers and a node only takes steps when it receives an event notification.

The code consists of event handlers – one for each message and five helper functions. We describe the functions bottom up. The implementation of *gain()* computes the gain based on the node's local information. The *best-match()* function returns the maximal gain neighbor, if an augmenting link exists, or otherwise $\perp$. A node is *ready()*, if it received match weight information from each of its neighbors.

The *update-match()* function is called when a node matches with a neighbor. It updates the state and sends notifications about the match change to the neighbors: "match-drop" to the previous match (if one exists), and "match-weight" to all other neighbors. The variable m changes to reflect the new match, and pm is deleted from $suitors$ since we allow matching only once per "preference" message. Note that pm remains m which means that the node cannot send a "preference" yet to a new neighbor. The node waits for a "match-weight" from m confirming the node matched back before releasing pm and thus enabling a new match. This ensures that there is a time at which both are matched to each other. (i.e., assuming that u and v match, there is always a time were both $m_u = v$ and $m_v = u$.)

The *check-pm()* function tries to make pm be *best-match()*, in order to later match with it. As discussed in Section 4.2, each node has at most one "preference" out at any time. We therefore consider two cases: if $pm = \perp$ (lines 75 – 80), then any previously sent "preference" messages were cleared, and so pm can become *best-match()*. A "preference" message is sent to the new pm. If a "preference" message from pm was already received (lines 79 – 80), the node can go ahead and change its match to pm. In case $pm \neq \perp$ (lines 82 – 84), the node first tries to get pm to become $\perp$, which is done via a "recall-preference" message. We send a "recall-preference" at most once per pm and only if v and pm are not matched yet.

Next, we describe the notification handlers.

The *"nbrs-update"* handler does a couple of things. First, if pm, m or *suitors* hold a node that is no longer a neighbor, it clears their value. Second, if the match weight of the node changed either because m was cleared or because the link weight of the match changed, it sends a "match-weight" to the remaining neighbors. Third, regardless of whether the node's match has changed or not, a "match-weight" message is sent to all new neighbors since they have no knowledge of the node's match weight. Last, since *best-match()* might have changed, *check-pm* is called.

All received *"preference"* messages are recorded in *suitors*, as they are sent only once. If no "recall-preference" was sent and v receives a "preference" from pm, v changes its match to pm. However, if v had sent a "recall-preference" to pm, then a "preference" from pm is not enough for v to match with pm.

If a node receives a *"recall-preference"* message from $u \neq m$, then it removes u from *suitors* and responds with a "recall-ack", which will clear pm_u and allow u to send a "preference" to another node. If a "recall-preference" is received from $u = m$, then it is not relevant, since the node had already sent a "match-weight" indicating u as its match,.

A node can only receive a *"recall-ack"* message from its pm, as it can only send one "recall-preference" and only to pm. Upon receipt of this message, pm is cleared, and *check-pm* is called to send a new "preference" message if needed.

A node sends a *"match-weight"* message for two purposes: to notify neighbors of the change to its match weight, and to inform its new match that it has matched with it. The matched node waits for such a notification before clearing pm. This is done in order to guarantee a time when the pair of matched nodes both output each other as their match before they can move on. Upon receiving such a message v first updates *nbrs-mw*. Then, if v is indicated as the new match and m_v is not yet the message sender, then *update-match()* is called. (This would be the case if v sent a "recall-preference" to pm, and then waits to see whether it will receive a "recall-ack" or a "match-weight".) Either way *check-pm()* is called since pm might have been cleared or *best-match()* might have changed.

A match between nodes can either be broken by one of the nodes or independently broken by both. When it is broken by one node, the *"match-drop"* is needed to let the other node know to clear its match state. Since this causes a match weight change, a "match-weight" message is also sent to all neighbors. In addition, *check-pm()* is called since *best-match()* could have changed. If the match is broken independently by both nodes, m already changed from u and the message is redundant by the time it is received and is simply ignored.

Chapter 5

Algorithm Analysis

In this chapter we show that Algorithm 2 is correct, quiescent, converges to a 2-approximation, and satisfies a locality property. All our results apply after a global stabilization time (GST), which is a time after which no more network changes occur. Formally, $\forall t \geq GST$ no "nbrs-update" is received.

We begin with a few basic lemmas in Section 5.1. Then, in Section 5.2, we prove that when no notifications are pending between two nodes, a node u outputs v as its match if and only if v outputs u as its match. In Section 5.3 we build the case for the algorithm becoming eventually quiescent. We conclude Section 5.3 by proving the following results:

Theorem 1. *(Quiescence)*

If GST exists, then $\exists t > GST$ such that Algorithm 2 is quiescent from time t onwards.

Theorem 2. *(Dynamic Weighted Matching Correctness) If GST exists, then $\exists t > GST$ such that the output of Algorithm 2 does not change from time t onwards and every node u outputs node v if and only if v outputs u.*

Theorem 3. *(Approximation) If GST exists, then $\exists t > GST$ such that $w(M_t) \geq \frac{1}{2}w(M_{t,opt})$ from t onwards.*

We conclude the chapter in Section 5.4 where we show the locality property of the algorithm:

Theorem 4. *(Locality) If Algorithm 2 is quiescent at time t, and then one of the following* *events occurs: node start, node stop, weight change, link removal, or link addition, then after O(1) rounds, $\exists t'$ s.t. $w(M_{t'}) \geq \frac{1}{2}w(M_{t',opt})$.*

In the proofs, we use the following notations. We denote a local variable of node u with subscript u, for example, m_u is u's current value of the variable m (its selected match). We use the notation Q_u to discuss "nbrs-update" notifications not yet received by node u, and the notation $Q_{u,v}$ to discuss notifications sent from u but not yet received by node v. We use the following abbreviations for some of the messages: $\hat{v}$ describes a ⟨"nbrs-update",Γ,*⟩ notification s.t. $v \in \Gamma_u \setminus \Gamma$ where u is the target of the notification. (i.e., u receives a notification that v is no longer its neighbor), and $mw_{=v}$ and $mw_{\neq v}$ describe a ⟨"match-weight",m,*⟩ message s.t. $m = v$ and $m \neq v$, respectively.

Also, for simplicity, when u or v are $\perp$, the values $w_u(v)$ and $w(u, v)$ are interpreted as equal to zero.

5.1 Simple Invariants

We observe that when a node has no potentially weight-changing notifications in its queues, its weight state variables must be up-to-date:

Observation 1. *(Correct view)*

1. *At time t, if $\nexists$"nbrs-update"$\in Q_u$, then $\forall v \in \Gamma_u$, $w_u(v) = w_t(u, v)$.*
2. *If $\nexists mw \in Q_{v,u}$ and $\nexists$"match-drop"$\in Q_{v,u}$ at time t, then $nbrs\text{-}mw_u(v) = w_t(m_v)$.*

We say in such cases that a node u has a *correct view*.

The following five simple invariants link the different messages that can be in transit to the corresponding node states. They follow immediately from the structure of the protocol and the fact that messages from different matching attempts do not get "mixed up" as subsequent attempts are separated by either a "match-drop" or a "recall-ack" exchange before any attempt to re-match.

Invariant 1. *If $\exists$"preference"$\in Q_{u,v}$ then $pm_u = v$, $u \notin suitors_v$ and $\nexists mw_{=u} \in Q_{v,u}$.*

Invariant 2. *If $\exists mw_{=v} \in Q_{u,v}$ then $pm_v = u$ and either $m_v = u$ or $m_u = v$.*

Invariant 3. *If* $\exists$*"recall-preference"*$\in Q_{u,v}$ *then either* $m_v = u$ *or* $m_u \neq v$, $pm_u = v$, *and* $pm\text{-}status_u$*=Recall.*

Invariant 4. *If* $\exists$*"recall-ack"*$\in Q_{u,v}$ *then* $v \notin suitors_u$ $pm_v = u$, $pm\text{-}status_v = Recall$, $m_u \neq v$ *and* $m_v \neq u$.

Invariant 5. *If* $\exists$*"match-drop"*$\in Q_{u,v}$ *then* $\nexists mw_{=u}$ *in* $Q_{v,u}$ *and either* $m_u \neq v$ *or* $m_v \neq u$.

Invariant 6. *Exactly one of the following claims is true at any given time* t*:*

1. $nbrs\text{-}ack_u(v)$ *=True*

2. $\exists mw_{=\perp} \in Q_{u,v}$

3. $\exists$*"nbr-ack"*$\in Q_{v,u}$

5.2 Agreement Lemmas

This section shows that when m_u becomes v and the link (u, v) does not drop from the network, then eventually m_v becomes u.

The next claim shows that while $pm = m$, m cannot change. It only considers a certain sub-case that is useful to Invariant 7.

Claim 2. *If* $m_v = u$ *and* $pm_u = m_u = v$ *at time* t*, and* pm_u *does not change in step* $t+1$*, then* m_u *does not change in step* $t+1$.

Proof. The variable m_u changes in Lines 29, 70, 89. Line 29 is executed together with Line 25, which means that pm_v becomes $\perp$ at step $t + 1$ in contradiction to the Claim's assumptions. Line 70 is not possible since both $m_u = v$ and $m_v = u$, which from Invariant 5 means no "match-drop" can exist. Last, Line 89 means that *update-match()* is called. The function *update-match()* is called from Lines 43, 58, and 80. Line 43 cannot be executed since $m_u = v$ and the check in Line 40 fails. Line 58 is not executed since $pm_u = m_u$, and finally, Line 80 is not possible since $pm_u \neq \perp$. $\square$

The next two claims show the intuitive property that if $u \in suitors_v$ then $pm_u = v$:

Claim 3. *If* $\exists mw_{=u} \in Q_{v,u}$ *them* $u \notin suitors_v$

Proof. When v sends a $mw_{=u}$ message to u, it also deletes u from $suitors_v$ in Line 90. From Invariant 1, while a $mw_{=u}$ is pending in $Q_{v,u}$, no "preference" can exist in $Q_{u,v}$. It follows that u cannot be re-added to $suitors_v$ while $\exists mw_{=u} \in Q_{v,u}$. □

Claim 4. *If at time t, $pm_u \neq v$ and $(u, v) \in E_t$, then $u \notin suitors_v$*

Note, that if link (u, v) is removed from the network, then a $\hat{v}$ and $\hat{u}$ is sent to u and v, respectively. When u and v receive such a message then $suitors$, pm, and m are cleared in Lines 24- 29. It is as if the communication between them did not happen.

Proof. Let's assume by contradiction that $pm_u \neq v$ but $u \in suitors_v$. A node is inserted to $suitors_v$ only in Line 41 when v receives a "preference" message from u. We denote this time as t_0. Since there is a "preference"$\in Q_{u,v}$ at t_0, then from Invariant 1, it follows that $pm_u = v$ at t_0. Let $t_1 > t_0$ be the time pm_u changes from v – the lines that could potentially change pm_u are 25, 49, 60 and 76. Line 25 is not possible since $(u, v) \in E_t$. Line 49 is not possible from Invariant 4, since $u \in suitors_v$. Line 60 is not possible from Claim 3, since $\nexists mw_{=u} \in Q_{v,u}$ while $u \in suitors_v$. Finally, Line 76 is not possible since pm_u must be $\perp$ in order for it to run. □

Invariant 7 specifies all the different variable and queue states that nodes u and v can be in when $m_u = v$, and how the nodes move between them.

Invariant 7. *(Match-state invariant) If at time t, $(u, v) \in E_t$ and $m_u = v$, then:*

A. *if $pm_u = v$ then:*

1. *$m_v \neq u$, $pm_v = u$, and $\exists mw_{=v} \in Q_{u,v}$ or*
2. *$m_v = pm_v = u$, $\exists mw_{=v} \in Q_{u,v}$, and $\exists mw_{=u} \in Q_{v,u}$ or*
3. *$m_v = u$, $pm_v \neq u$, and $\exists mw_{=u} \in Q_{v,u}$ or*
4. *$m_v \neq u$ and $\exists mw_{=u}$ followed by a "match-drop" in $Q_{v,u}$.*

B. *if $pm_u \neq v$, then:*

1. *$m_v = pm_v = u$ and $\exists mw_{=v} \in Q_{u,v}$ or*
2. *$m_v = u$ and $pm_v \neq u$ or*

3. $m_v \neq u$ and $\exists$"match-drop" in $Q_{v,u}$.

Proof. We prove the invariant by induction on the number of received notifications at either u or v. Base: let t_0 be the time m_u changes to v. The variable m changes to a non $\perp$ value only in Line 89 which means that $update\text{-}match_u()$ is executed. As a result, $m_u = pm_u = v$ and $\exists mw_{=v} \in Q_{u,v}$. From Invariant 2, since $\exists mw_{=v} \in Q_{u,v}$, then $pm_v = u$. If $m_v \neq u$, Case A(1) is true and the invariant holds. If $m_v = u$, then we need to show that either Case A(2) or Case B(1) are true. Let's consider which line led to $update\text{-}match_u()$ being executed. If it was Line 58 then u received a $mw_{=u}$ from v. It follows that Lines 55 – 60 are executed which means that pm_u becomes $\perp$ and Case B(1) is true. Otherwise, we need to show that $\exists mw_{=u} \in Q_{v,u}$ at t_0. Let $t_1 < t_0$ be the last time m_v changed to u. Then, v sent a $mw_{=u}$ to u at t_1. We considered earlier the case where this $mw_{=u}$ was received at t_0. Now, we assume by contradiction that it was received at some time $t_2 < t_0$. (Note that $t_1 < t_2$ since the message was only sent at t_1). When the $mw_{=u}$ is received, it follows from Invariant 2 that $pm_u = v$ which guarantees that Lines 55 – 60 are executed. As a result, a $mw_{=v}$ is sent to v. Since $m_v = u$ both at t_1 and at t_0, and we assumed that the last time m_v becomes u is at t_1, it follows that m_v cannot change between t_1 and t_0. When the $mw_{=v}$ sent at t_2 is received, pm_v becomes $\perp$ and since $gain(m_v) = 0$, it cannot become u while $m_v = u$. This means $pm_v \neq u$ at t_0. However, from Invariant 2, when u sends the $mw_{=v}$ at t_0, pm_v must be u and therefore we reached a contradiction.

Step: we assume that the invariant is true at $t - 1 \geq t_0$, and prove that it is still true at t by considering any subsequent notification that can affect each case.

Case A(1) : either m_v changes to u, pm_v changes, or $mw_{=v}$ is received by v. From Invariant 2, pm_v cannot change while $\exists mw_{=v}$. The variable m_v can become u only in Line 89. Line 89 is executed with Line 88, which means that v sends a $mw_{=u}$ to u, and Case A(2) becomes true. If $mw_{=v}$ is received, since $pm_v = u$, Lines 55 – 60 are executed. This means that v sends a $mw_{=u}$ to u, m_v becomes u and pm_v becomes $\perp$, which make Case A(3) become true.

Case A(2) : either m_v or pm_v change or one of the "match-weight" messages are received. From Invariant 2, pm_v cannot change until $mw_{=v}$ is received, and from

Claim 2, m_v cannot change until pm_v changes. If $mw_{=u}$ is received then pm_u becomes $\perp$ in Line 60 and Case B(1) becomes true. Last, if $mw_{=v}$ is received, then pm_v becomes $\perp$ after Line 60 executes, and then Case A(3) becomes true.

Case A(3) : either m_v changes, pm_v becomes u, or the notification is received. The variable pm_v cannot become u, since this happens only in Line 76, but Line 76 does not execute because $gain(m_v) = 0$, which means that $m_v \neq best\text{-}match()$. If the $mw_{=u}$ is received then pm_u becomes $\perp$ in Line 60, which means that Case B(2) becomes true. This leaves the possibility of m_v changing. The variable m_v can change in Lines 29, 70, 89. Line 29 is not possible since $(u, v) \in E_t$. Line 70 is not possible since from Invariant 5 no "match-drop" can exist in $Q_{u,v}$ while $\exists mw_{=u} \in Q_{v,u}$. Last, if Line 89 is executed, then in Line 87 a "match-drop" is sent to u and Case A(4) is true.

Case A(4) : either m_v becomes u, or the $mw_{=u}$ is received. Due to the FIFO assumption, the "match-drop" notification cannot be received. For m_v to become u, $update\text{-}match_v()$ needs to be called. This can happen in Lines 43, 58, and 80. Line 43: not possible since it would mean that there is "preference" from u at time t which from Invariant 1 is not possible when $\exists mw_{=u} \in Q_{v,u}$. Line 58: From Invariant 5 no $mw_{=v}$ can exist while $\exists$"match-drop"$\in Q_{v,u}$. Line 80: not possible since from Claim 3, $u \notin suitors_v$ when $\exists mw_{=u} \in Q_{v,u}$. This leaves $mw_{=u}$ being received which would make Case B(3) become true.

Case B(1) : either m_v changes, pm_v changes, or $mw_{=v}$ is received. From Invariant 2, pm_v cannot change while the $mw_{=v}$ is pending. Subsequently, from Claim 2, m_v cannot change. If $mw_{=v}$ is received then pm_v becomes $\perp$ in Line 60 and Case B(2) becomes true.

Case B(2) : either m_v changes or pm_v becomes u. The function $gain()$ always returns 0 for m, which means that $best\text{-}match_v()$ never returns m_v. Since pm_v can become u only in Line 76 where $best\text{-}match_v()$ is assigned to it, it is not possible that u is assigned to it while $m_v = u$. The variable m_v can change in Lines 29, 70, 89. Line 29 is not possible since $(u, v) \in E_t$. Line 70 is not possible since both $m_u = v$

and $m_v = u$ which from Invariant 5 means no "match-drop" can exist. If Line 89 is executed then in Line 87 a "match-drop" is sent to u and Case B(3) becomes true.

Case B(3) : either m_v becomes u, or the "match-drop" is received. For m_v to become u, $\mathit{update\text{-}match}_v()$ needs to be called. This can happen in Lines 43, 58, and 80. Both Lines 43 and Line 58 are not possible from Invariants 1 and 2 since $pm_u \neq v$. Line 80 is not possible from Claim 4 which states that $u \notin \mathit{suitors}_v$ when $pm_u \neq v$. If the "match-drop" is received by u, then m_u becomes $\perp$ in Line 70 and the invariant vacuously holds.

□

Corollary 1. *If at time t, m_u becomes v and $m_v \neq u$ and for the next round no "nbrs-update" are sent to u or v then exists t' after at most 1 s.t. $m_u = v$ and $m_v = u$.*

Corollary 2. *If $m_u = v$, and both $Q_{u,v}$ and $Q_{v,u}$ are empty, then $m_v = u$.*

5.3 Termination and Quiescence

We show that the algorithm terminates by proving that after GST the algorithm only adds augmenting links to the matching. In general, there is a very limited number of scenarios that might cause a non-augmenting link to be added to the matching. These arise because a link is not added to the matching atomically. It takes a step by each of the endpoints to add the link. Once the first endpoint makes the first step, the other endpoint will always follow through with its own step. One scenario where non-augmenting links are added, is if between those steps a network change occurs which makes the link become non augmenting. The second scenario occurs when a node mistakes a link for positive gain when it has an incorrect match weight for some neighbor. This is due to pending "match-weight" messages. This scenario is quite rare since both endpoints of the link need to make the same mistake – they both need to send "preference" messages in order to match. On top of this, most occurrences of this scenario do not lead to link additions thanks to "recall-preference" messages. Nodes only match in such a case if they exchange "preference" messages and at least one of them does not send a "recall-preference". Next, we claim this formally.

We denote by $mw_>$ a $\langle$"match-weight"$, m \to m \neq u, mw\rangle \in Q_{v,u}$ message sent by v to u s.t. $mw > nbrs\text{-}mw_u(v)$. i.e., a "match-weight" message that notifies of a weight increase of the current match (as opposed to a new match).

Observation 2. *A $mw_>$ notification is sent by node v only when node v receives a $\langle$"nbrs-update",Γ'_v,$w'_v\rangle$ s.t. $w'_v(m_v) > w_v(m_v)$*

We are interested in the case where a non-augmenting link is chosen because *best-match()* is outdated relative to the variable pm. The next claim outlines when these two values are equal.

Claim 5. *If for some node u at time t, $m_u \neq v$ and pm-$status$ =Prefer, then pm_u = best-$match_u()$.*

Proof. The variable pm is either set to $\perp$ or to *best-match()*. The function *check-pm()* is called whenever pm is set to $\perp$ and whenever *best-match()* might change. In *check-pm()*, if $pm_u = \perp$ then $best\text{-}match_u()$ is assigned to it (Line 76). Otherwise, if $m_u \neq pm_u$ then $pm\text{-}status$ receives Recall in Line 83. □

Next, we want to show that if $v \in suitors_u$, then v is committed to u and cannot send another neighbor a "preference" message. We prove a slightly weaker claim, where we assume $m_u \neq v$ and $m_v \neq u$.

Claim 6. *If $m_v \neq u$, $m_u \neq v$, and $v \in suitors_u$, then $pm_v = u$.*

Proof. If $v \in suitors_u$ at some time t then $\exists t' \leq t$ s.t. u received a "preference" message from v. From Invariant 1, at t', $pm_v = u$. Let us consider all the cases in which pm_v can change - Lines 25, 49, 60, and 76. Line 25: node v receives a $\hat{u}$, then node u also receives a $\hat{v}$ and v is deleted from $suitors_u$. Line 49 is not possible since $v \in suitors_u$ and if a "recall-ack" exists in $Q_{u,v}$ it is a contradiction to Invariant 4. Line 60 is not possible since both $m_v \neq u$ and $m_u \neq v$, and if a $mw_{=v}$ exists it would contradict Invariant 2. Finally, Line 76 is not possible since $pm_v \neq \perp$. □

Next, we show that if a node u initiates a match with node v, and u does not know v's up-to-date weight, then there must be a pending $mw_>$ notification.

Claim 7. *If at time t, m_u becomes v while $m_v \neq u$ and $nbrs\text{-}mw_u(v) < w(v, m_v)$, then $\exists mw_> \in Q_{v,u}$.*

Proof. First, from Invariant 5 there can be no "match-drop" message since $m_u \neq v$ and $m_v \neq u$. Hence, by Observation 1, because $nbrs\text{-}mw_u(v) \neq w(v, m_v)$, there must be a $\langle$"match-weight"$, mo \to mn \neq u, mw\rangle \in Q_{v,u}$. It remains to show that in the notification, $mw > nbrs\text{-}mw_u(v)$ and $mo = mn$, meaning that the match of v did not change, and the weight of that match has increased. Observe that, if m_u becomes v, *update-match()* is called. This can potentially happen in Lines 43, 58, or 80. However, since $m_v \neq u$ and $m_u \neq v$ at t, then from Invariant 2 there can be no pending $mw_{=u}$ to receive and Line 58 is not possible. On both lines 43 and 80, $v \in suitors_u$. Let t' be the time u last received a "preference" from v, then $nbrs\text{-}mw_u$ is updated on the match weight of v at the time in which v sent the "preference" message to u. From Claim 6 since $v \in suitors_u$ between t' and t, then $pm_v = u$ at this interval. A node sends a "preference" only when pm changes so it is not possible that v matched with another node since sending the "preference" to u. This means that the increase in the weight of v's match occurred for an existing match, and $\exists mw_> \in Q_{v,u}$. □

The next Lemma shows that except for the case of network changes that directly drop links, whenever a link is dropped, a new link is added in its place.

Lemma 3. *Let (u, x) be removed from the matching at time t s.t. m_u changes at time t, and no "nbrs-update" is received by u at time t. Then, m_u becomes $v \neq \perp$, and within at most one round, $m_v = u$.*

Proof. First, it is not possible that m_u changed to $\perp$. The variable m is assigned $\perp$ only in line 29 or 70. Line 29 cannot be executed at time t since no "nbrs-update" was received at time t. It is not possible that Line 70 executes since there is no "match-drop" by Invariant 5. The second part follows immediately from Corollary 1. □

Lemma 4 shows that under certain conditions only augmenting links are added (i.e., the weight of an added link exceeds those of the edges it supplants). Remember that a link is added to the matching in two steps - both endpoints need to change their match. Whenever there are no network changes, either none of these steps occur or both of them do. We consider an augmenting link with respect to the matching at the time the first step occurred. Let node u output v as its match at time t, then we define the *match weight* of node u as $w_t(u, v)$ when v also outputs u as its match, or zero otherwise.

Lemma 4. *(Augmenting-additions) Consider an edge (u, v) that is added to the matching, where m_u becomes v at time t_1, and m_v becomes u at time $t_2 > t_1$. Let w_1 and w_2 denote u and v's match weights until times t_1 and t_2, respectively. If no "nbrs-update" is received by either u or v between t_1 and t_2 and $\nexists mw_> \in Q_{v,u}$ at t_1, then $w(u, v) > w1 + w2$ at t_2.*

Proof. Since $t_1 < t_2$, m_u changes first. The variable m only changes in *update-match()*, and *update-match()* is called from 3 places in the code: Lines 43, 58, and 80. Line 58 is not possible since from Invariant 2 there cannot be a pending $mw_{=u}$ while $m_v \neq u$ and $m_u \neq v$. Therefore, *update-match()* is called from either Line 43 or Line 80, which means that $pm\text{-}status$=Prefer and $v \in suitors_u$. From Claim 5, $pm_u = best\text{-}match_u()$. Since *best-match()=v*$\neq \perp$ only if $gain_u(v) > 0$, then $w_u(v) > nbrs\text{-}mw_u(v) + w_u(x)$. Since there are no "nbrs-update" messages received by u or v between t_1 and t_2, we can replace $w_u(v)$ and $w_u(x)$ by $w_{t_2}(u, v)$ and w_1. From Observation 2, there is no $mw_>$ between t_1 and t_2, and from Claim 7, $nbrs\text{-}mw_u(v) \geq w_2$. Together, this gives us $w(u, v) > w1 + w2$. □

For the following lemma, we define the *match weight* of node u at time t as $w_t(u, m_u)$ if $(u, m_u) \in M_t$, or zero otherwise. Lemma 5 shows that the weight of the added link exceeds those of the edges it supplants:

Lemma 5. *Consider an edge (u, v) that is added to the matching at least one round after GST, where m_u becomes v at time t_1 and m_v becomes u at time $t_2 > t_1$. Let w_1 and w_2 denote u and v's match weights until times t_1 and t_2, respectively. Then, $w(u, v) > w1 + w2$.*

Proof. From GST onwards no more "nbrs-update" message are received, but for one round following GST there can still be $mw_>$ messages as a result of "nbrs-update" messages received just prior to GST. After this round all conditions exist for Lemma 4, and it follows that $w(u, v) > w1 + w2$. □

Everything is set now to prove that the algorithm terminates:

Lemma 6. *(Termination) If GST exists, then $\exists t > GST$ such that M_t does not change from t onwards.*

Proof. From Lemma 5, a round after GST only augmenting links are added to the matching. From Lemma 3 no link is removed without a link being added so the weight of M can only increase. Finally, since $W(M)$ is bounded by $W(M_{opt})$, $\exists t$ s.t. M_t does not change from t onwards. $\square$

We have shown that the matching eventually stops changing. In the remainder of this section we concentrate on showing that when it does, no augmenting links exist and that the algorithm becomes quiescent shortly after the matching stops changing.

The next two claims show progress when a node is at a certain state. The first claim shows progress when nodes send "preference" messages to each other and do not recall them:

Claim 8. *If at time t $pm_u = v$, $pm\text{-}status_u$=Prefer and $pm_v = u$, $pm\text{-}status_v$=Prefer, and $m_u \neq v$, then the following statements are all true:*

- $\exists$*"preference"*$\in Q_{v,u}$.
- $\nexists$*"recall-preference"*$\in Q_{v,u}$.
- $\nexists$*"recall-ack"*$\in Q_{u,v}$.

Proof. First, since $pm\text{-}status_v$=Prefer and $m_u \neq v$ then the second and third statements are straightforward from Invariants 3 and 4. We concentrate on proving that $\exists$"preference"$\in Q_{v,u}$. We denote by t_1 and t_2 the latest time prior to t at which pm_u becomes v and pm_v becomes u, respectively. When pm_v changes to u, a "preference" message is sent to u. We further denote by t_3 the time at which this "preference" is received by u. Note that at t_1 (t_2), $pm\text{-}status_u$=Prefer ($pm\text{-}status_v$=Prefer). Both $pm\text{-}status$ variables are Prefer at time t as well. It is not possible that in between either variable changes to Recall, since it can only change back to Prefer if the respective pm changes and that does not happen again before t. Similarly, $m_u \neq v$ both at time t_1 and at time t. It cannot become v between t_1 and t since in order to become $m_u \neq v$ again, pm_u must first change to $\perp$, in contradiction to t_1 being the latest time prior to t that pm_u becomes v. It suffices to show that $t_3 > t$ and therefore at t, $\exists$"preference"$\in Q_{v,u}$. Assume by way of contradiction that $t_3 \leq t$ and consider the relation between t_3 and t_1. If $t \geq t_3 > t_1$, then at t_3 Line 43 would subsequently cause m_u to become v in contradiction to m_u not changing between t_1 and t. If $t_3 \leq t_1$, then at t_3 $v \in suitors_u$. If at t_1 v is still in $suitors_u$ then Line 80 would subsequently cause m_u to become v in contradiction to m_u not changing between t_1 and t. Hence, both $t_3 \leq t_1$ and $v \notin suitors_u$ at time t_1. This means that somewhere between t_3 and t_1, v is removed from $suitors_u$. We show next that this is not possible by considering all the lines in which a node is removed from $suitors$. Line 24: node u receives a $\hat{v}$ notification, then symmetrically v receives a $\hat{u}$. This means that in Line 25, pm_v becomes $\perp$ in contradiction to t_1 being the latest time prior to t in which pm_u becomes v. Line 46: not possible since between time t_3 and t_1, $pm\text{-}status_v$=Prefer and $m_u \neq v$ so according to Invariant 3, $\nexists$"recall-preference"$\in Q_{v,u}$. Line 90: then m_u becomes v in contradiction to m_u not changing between t_1 and t. We exhausted all cases in which $t_3 \leq t$, which means that at t, the "preference" message must still be at $Q_{v,u}$. □

The second claim shows progress when a node sends a "recall-preference" message:

Claim 9. *If at time t, $pm_u = v$ and $pm\text{-}status_u$=Recall, and $m_v \neq u$, then either $\exists$"recall-preference"$\in Q_{u,v}$ or $\exists$"recall-ack"$\in Q_{v,u}$.*

Proof. Let $t_1 < t$ be the latest time prior to t that $pm\text{-}status_u$ changes to $Recall$. The change can only occur in Line 83. Line 83 always executes together with Line 84, which means that a "recall-preference" message is sent to v at t_1. Notice that $pm_u = v$ both at t_1 and at t. It cannot change in between since pm_u would need to be assigned v again before t, and when pm is assigned a node, $pm\text{-}status$ is Prefer. This means that $pm\text{-}status_u$ would need to change again before t in contradiction to t_1 being the last time. If at t the "recall-preference" has still not been received, the claim holds. Otherwise let $t_2 < t$ be the time the "recall-preference" is received by v. It is not possible that at t_2, $m_u = v$ since for m_u to become $\neq v$ again at time t, pm_u must become $\perp$ and as mentioned before that is a contradiction to our assumption. Therefore, when v receives the "recall-preference", $m_v \neq u$ and v sends a "recall-ack" to u. The "recall-ack" must still be in $Q_{v,u}$ at time t, since otherwise when u receives the "recall-ack", pm_u becomes $\perp$ which again would be a contradiction. □

The next invariant shows that when there are no messages in the queues, the values in $nbrs\text{-}ack$ map are all true. This guarantees that $ready()$ must return true.

The next claim shows that the value of *best-match()* eventually determines the value of pm.

Claim 10. *If at time $t > GST$, no "match-weight", "match-drop", or "nbr-ack" notifications exist in any queue and $best\text{-}match_u() = v$, then either $pm_u = v$ or $pm\text{-}status_u$=Recall.*

Proof. Let $t_1 < t$ be the last time that $check\text{-}pm_u()$ is run prior to t. Since $check\text{-}pm_u()$ is run after every received "nbrs-update","match-weight", and "match-drop" notification, and at t none of these notifications exist in the queues, then no "nbrs-update","match-weight" and "match-drop" notification can exist in u's queues between t_1 and t. It follows that u has a correct view from t_1, which means that $best\text{-}match_u()$ did not change since t_1. Therefore, at t_1, $best\text{-}match_u() = v$. Also, since no "match-weight" or "nbr-ack" notifications exists in u's queues since t_1, and no "nbrs-update" exists after GST, then $ready_u()$ is true at t_1. When $check\text{-}pm_u()$ is run, either $pm_u = best\text{-}match_u()$, or $pm_u = \perp$, or $pm_u \neq \perp$ and $pm_u \neq best\text{-}match_u()$. If $pm_u = best\text{-}match_u()$ then the function immediately exits. If $pm_u = \perp$ then $best\text{-}match_u()$ is assigned to it which means v is assigned to it. Otherwise, $pm_u \neq \perp$ and $pm_u \neq best\text{-}match_u()$. If $pm_u \neq m_u$, then the check in Line 82 is true and $pm\text{-}status_u$ becomes Recall. Since no "match-weight" or "match-drop" notifications exist, it follows from Invariant 7 that $pm_u \neq m_u$, and we conclude that at t_1, $pm_u = v$ or $pm\text{-}status_u$=Recall. Since each change made to pm_u and $pm\text{-}status_u$ is either at $check\text{-}pm_u()$ or subsequently calls $check\text{-}pm_u()$, then pm_u and $pm\text{-}status_u$ do not change again prior to t. □

The next Lemma is key to proving that the algorithm converges to the desired approximation and becomes quiescent. We show that the matching can stop changing only if no augmenting link exists.

Lemma 7. *If there is an augmenting link at time $t > GST$ then after at most six rounds $\exists t'$ s.t. $M_{t'} \neq M_t$.*

Proof. Assume by way of contradiction that an augmenting link exists but M does not change after six rounds. Note that "match-weight" and "match-drop" messages are sent only in response to "nbrs-update" and to changes in the matching. After GST no "nbrs-update" messages are received, which means that at most a round after the last change in M or in the network (whichever is later) prior to t, no "match-weight" or "match-drop" messages exist in the queues. A round after, no "nbr-ack" messages are in the queues. Let $t_1 > t$ be a time s.t. no "match-weight", "match-drop" and "nbrs-ack" messages exist in the queues. For the purpose of counting rounds, t_1 is at most two rounds after t. Let (u, v) be the augmenting link with maximum gain w.r.t M_{t_1}. Then, since at t_1 the view of all the nodes is correct, then from Observation 1, $best\text{-}match_u() = v$ and $best\text{-}match_v() = u$. We show that either u or v (or both) change their match within at most seven rounds from t.

We prove the lemma by considering all possible values of pm_u and pm_v at t_1.

Case 1 $pm_u = \perp$ or $pm_v = \perp$. Not possible from Claim 10.

Case 2 $pm_u = v$ and $pm_v = u$.

We show that u and v match. Since no "match-weight" or "match-drop" exist at t_1, then from Invariant 7 it is not possible that $m_u = v$ and $m_v \neq u$ or vice versa. In that case, nodes u and v sent preference messages to each other but they did not match yet. We further separate to sub-cases based on the value of the variable $pm\text{-}status$ in each of them.

Case 2.A $pm\text{-}status_u$=Prefer and $pm\text{-}status_v$=Prefer.

From Claim 8, since $pm_u = v$, $pm_v = u$, and $m_u \neq v$, there must exists a "preference" in both $Q_{u,v}$ and $Q_{v,u}$. Also from Claim 8 there can be no "recall-preference" or "recall-ack" messages in $Q_{u,v}$ and $Q_{v,u}$. This means that when each receives the "preference" message from the other, Line 43 is executed and they match. Note that in this case, the stall is just waiting for the "preference" message to arrive to both sides which can take at most one round. It follows that at most three rounds after t, M changes.

Case 2.B $pm\text{-}status_u$=Prefer, and $pm\text{-}status_v$=Recall.

Since $m_u \neq v$ at t_1 then from Claim 9, either $\exists$"recall-preference" in $Q_{v,u}$ or $\exists$"recall-ack" in $Q_{u,v}$. If $\exists$"recall-preference" in $Q_{v,u}$ then let $t_2 > t_1$ be the time the "recall-preference" is received. At t_2 either $m_u = v$ or $m_u \neq v$. If $m_u = v$ then from Invariant 7, m_v becomes u one round later, and u and v match. If $m_u \neq v$, then u responds to v with a "recall-ack" message. From Invariant 4, while a "recall-ack"$\in Q_{u,v}$, $m_u \neq v$ and $m_v \neq u$. When v receives the "recall-ack", it clears pm_v but on the subsequent call to $check$-$pm_v()$, since $best$-$match_v = u$, pm_v becomes u and pm-$status_v$ =Prefer. Since $pm_u = v$ and pm-$status_u$=Prefer, we can now follow Case 2.A which shows that u and v match. Note that in this case, at worst, we pay two extra rounds (one round to receive the "recall-preference" and one round to receive the "recall-ack") plus the round Case 2.A takes. It follows that at most five rounds after t, M changes.

The case that pm-$status_u$ =Recall and pm-$status_v$ =Prefer is symmetric to Case 2.B.

Case 2.C pm-$status_u$=Recall and pm-$status_v$=Recall.

From Claim 9, either $\exists$"recall-preference"$\in Q_{u,v}$, or $\exists$"recall-ack"$\in Q_{v,u}$. Note that in case a "recall-preference" exists, it is not possible for m_u to become v before the "recall–preference" is received. This is because $pm_u = v$ and no "match-weight" exists at time t_1, which means that *update-match()* can only be called by Line 43 but would not since pm-$status_u$=Recall. Therefore, if $\exists$"recall-preference"$\in Q_{v,u}$ then once it is received, u responds with a "recall-ack" to v. When v receives the "recall-ack", it clears pm_v. On the subsequent call to $check$-$pm_v()$, since $best$-$match_v = u$, pm_v becomes u and pm-$status_v$=Prefer. Now we are at either Case 2.A or Case 2.B depending on the value of pm-$status_v$ at the time. Note that in this case, at worst, we pay two extra rounds (one round to receive the "recall-preference" and one round to receive the "recall-ack") plus the rounds Case 2.A or Case 2.B take. It follows that at most seven rounds after t, M changes.

Case 3 either $pm_u = x \neq v$ or $pm_v = y \neq u$.

In this case it is possible that u or v match with x or y respectively instead of matching with each other. Without loss of generality, $pm_u = x \neq v$. From Claim 10, since *best-match*$_u() = v$, it follows that *pm-status*=Recall. Furthermore, from Invariant 7 it is not possible that $m_u = x$ and $m_x \neq u$ or vice versa, or that $pm_u = m_u = v$ and $m_x = u$. ("match-weight" or "match-drop" exist at t_1.) Since $m_x \neq u$ at t_1 then from Claim 9, $\exists$"recall-preference" in $Q_{u,x}$ or $\exists$"recall-ack" in $Q_{x,u}$. If $\exists$"recall-preference" in $Q_{u,x}$, then let $t_2 > t_1$ be the time the "recall-preference" is received. At t_2 either $m_x = u$ or $m_x \neq u$. If $m_x = u$ then from Invariant 7, m_u becomes x one round later, and u and x match. If $m_x \neq u$, then x responds to u with a "recall-ack" message. From Invariant 4, while a "recall-ack"$\in Q_{x,u}$, $m_u \neq x$ and $m_x \neq u$. When u receives the "recall-ack" from x, it clears pm_u and on the subsequent call to *check-pm*$_u()$, since *best-match*$_u = v$, pm_u becomes v and *pm-status*$_u$=Prefer. If $pm_v = u$, we can now follow Case 2.A or Case 2.B depending on *pm-status*$_v$. If $pm_v = y \neq u$, then the communication between v and y is similar to u and x. Either v and y match, or pm_v becomes u and *pm-status*$_v$ =Prefer. In the latter option we can apply Case 2.A. Note that in this case, at worst, we pay two extra rounds (one round to receive the "recall-preference" and one round to receive the "recall-ack") plus the rounds Case 2.A or Case 2.B take. It follows that at most seven rounds after t, M changes.

□

Finally, Lemma 8 shows that when no links are added to the matching, Algorithm 2 becomes quiescent within a bounded number of rounds:

Lemma 8. *If $\exists t > GST$ after which M does not change, then after at most $O(1)$ rounds the algorithm is quiescent and for all $u \in V$, $pm_u = \perp$.*

Proof. We need to prove that after $O(1)$ rounds all the message queues are empty and for all $u \in V$, $pm_u = \perp$. The matching does not change after t, which from Lemma 7 means that at t no augmenting links exist. Remember that it takes at most one round for a sent message to be received. Let t_1 be one round after t, then all "match-weight" and "match-drop" message are received. Since no "nbrs-update" notifications are sent after GST and the matching does not change, after t_1 no "match-weight" or "match-drop" exist in the queues. From Observation 1, the view of every node is correct and since there are no augmenting links, then from t_1 onwards $\forall u$ $best\text{-}match_u() = \bot$. Since $best\text{-}match_u()$ is $\bot$ at t_1 and $check\text{-}pm_u()$ is called whenever $best\text{-}match_u()$ changes, then also at t_1, either $pm_u = \bot$, or $pm_u = v \neq \bot$ and $pm\text{-}status_u$ =Recall. Since pm_u already reacted to $best\text{-}match_u() = \bot$ then from t_1 onwards, no “preference” or “recall-preference” messages are sent. Let t_2 be one round after t_1, then all previous “preference” and “recall-preference” messages are received, and no more “recall-ack” messages are sent after t_2. This means that after three rounds no more messages are in the queues. As mentioned earlier, from t_1 onwards, for each $u \in V$ either $pm_u = \bot$, or $pm_u = v \neq \bot$ and $pm\text{-}status_u$ =Recall. Since a round after t_2 no more messages exist in the queues, then from Claim 9, for every $u \in V$ it follows that $pm_u = \bot$. □

Theorem 1 (restated). (Quiescence) If GST exists, then $\exists t > GST$ such that Algorithm 2 is quiescent from time t onwards.

Proof. Immediate from Lemma 6 and Lemma 8. □

Theorem 2 (restated). (Dynamic Weighted Matching Correctness) If GST exists, then $\exists t > GST$ such that the output of Algorithm 2 does not change from time t onwards and every node u outputs node v if and only if v outputs u.

Proof. Immediate from Theorem 1 and Corollary 2. □

Theorem 3 (restated). (Approximation) If GST exists, then $\exists t > GST$ such that $w(M_t) \geq \frac{1}{2} w(M_{t,opt})$ from t onwards.

Proof. Follows from Lemma 1 in Section 4.1 and Lemma 7 above. □

5.4 Locality

We divide network events into four categories – *weight increase*, *weight decrease*, *node addition*, and *node deletion*. We show Theorem 4 by proving a separate lemma for each category. The same approach is employed in each

We have already shown that shortly after GST there is a time after which the weight of M can only increase. We next show how to refine this for some network changes and show that it quickly increases to a 2-approximation.

Lemma 2 is used to bound the matching weight's distance from the approximation following the change. Then, we consider the nature of the change and show that the algorithm adds at least that missing weight within a constant number of rounds. Since M_{opt} does not change, then from Lemma 2, this is sufficient for M to become a 2-approximation of M_{opt}.

In the changes we are looking at, the nodes neighboring the change have a correct view of the network. We use this to prove some properties:

Claim 11. *If node u has a correct view when it sends a "preference" message to node v at time t then*

1. $gain_{M_t}(u, v) > 0$
2. $\nexists (u, x) \in E_t$ *s.t.* $gain_{M_t}(u, x) > gain_{M_t}(u, v)$

Proof. Node u sends a "preference" message to the node $best\text{-}match_u()$ returns. The function $best\text{-}match()$ calculates the maximum gain neighbor from u's $nbrs\text{-}mw_u$ and w_u variables. From Observation 1, since node u has a correct view, $w_u(v) = w_t(u, v)$ and $nbrs\text{-}mw_u(v) = w_t(m_v)$. The latter gives us part 2, and part 1 is immediate from line 16. $\square$

The first change we consider is an increase in weight of some link in the network. The addition of a new link also falls under this category, as we can consider the addition as a weight increase from weight 0.

Reminder: $A(M)$ is the set of augmenting links w.r.t M that are in M_{opt}: $A(M) \triangleq \{a \in M_{opt} \mid gain_M(a) > 0\}$.

Lemma 9. *(Weight increase) If Algorithm 2 is quiescent at time t, and then the weight of any one link $e = (u, v)$ is increased, then after O(1) rounds, $\exists t'$ s.t. $w(M_{t'}) \geq \frac{1}{2} w(M_{t',opt})$.*

Proof. Since no link other than e changed and $w(e)$ has increased, the only possible augmenting link at time t is e. The algorithm is quiescent at time t, which means that all the nodes are idle ($\forall x \in V\ pm_x = \mathit{best\text{-}match}_x() = \perp$). As a result of the change "nbrs-update" notifications are sent to u and v. Scenario 1: e is not augmenting and therefore from Lemma 1, M_t is still a 2-approximation of M_{opt}. We show next that the matching does not change. The algorithm was quiescent prior to the change which guarantees that when node u and v receive the "nbrs-update" their view is correct and therefore from Lemma 11 neither will send a "preference". If $(u, v) \in M_t$ both u and v subsequently send a "match-weight" to their neighbors. When any neighbor receives the "match-weight", its view is correct and therefore will not send a "preference" message. After two rounds the algorithm becomes quiescent again. Scenario 2, e is augmenting. When node u receives the "nbrs-update", its view is correct and $\mathit{best\text{-}match}_u() = v$. As a result, and since $pm_u = \perp$ when $\mathit{check\text{-}pm}_u()$ is run, a "preference" message is sent to v. The same logic applies to node v, and u and v exchange "preference" messages. When both u and v receive the "preference" messages from each other. $m_u = v$ and $m_v = u$. This takes at most one round (the only message it takes a round to receive is the "preference"). Next, we show that the addition of e to the matching is enough for $M_{t'}$ to be a $\frac{1}{2}$-approximation of $M_{t',opt}$. At t, $A(M_t)$ can only include e which means from Lemma 2 that $w(M_t) \geq \frac{1}{2}w(M_{t,opt}) + \frac{1}{2}w(e)$. Since there are no changes between t and t', M_{opt} does not change, and when e is added to $M_{t'}$, $w(M_{t'}) \geq \frac{1}{2}w(M_{t',opt})$.

□

The second change we consider is a decrease in the weight of a link in the network. A deletion of a link also falls under this category - it is similar to a weight decrease to 0. Note that if the changed link is not in M, there can be no augmenting links after the change as the measured augmentation weight is with respect to links in M.

Lemma 10. *(Weight decrease) If Algorithm 2 is quiescent at time t, and then the weight of one link $e = (u, v)$ is decreased, then after O(1) rounds, $\exists t'$ s.t. $w(M_{t'}) \geq \frac{1}{2}w(M_{t',opt})$.*

Proof. The change triggers "nbrs-update" notifications to be sent to u and v. If no link became augmenting as a result of the change, M_t is still a 2-approximation of M_{opt}. The algorithm was quiescent prior to the change which guarantees that when node u and

v receive the "nbrs-update" their view is correct and therefore from Lemma 11 neither will send a "preference". If $(u, v) \in M_t$ both u and v subsequently send a "match-weight" to their neighbors. When any of their neighbors receives the "match-weight", its view is correct and therefore will not send a "preference" message. After two rounds the algorithm becomes quiescent again. Otherwise, there is some augmenting link. Note that all augmenting links must be adjacent to e since the algorithm was quiescent before t and only e changed. Without loss of generality, let the maximum gain augmenting link be incident to u and we denote it by (u, x). We show next that (u, x) is added to the matching after a constant number of rounds, and that adding (u, x) suffices for us to be at the approximation. When node u receives the "nbrs-update", its view is correct, and after sending "match-weight" to all its neighbors it also sends a "preference" to x which is its maximum gain neighbor. Node x is idle until it receives a message. After at most one round it receives the "match-weight" from u and possibly a "match-weight" from v if x is a neighbor of v as well. If node x receives the message from u first or (v, x) is not augmenting, it sends a "preference" message to node u. If node x receives the message from v first and (x, v) is augmenting, then x sends the "preference" to v. When x receives the "match-weight" from u, it will send a "recall-preference" to v. If x is not v's maximum gain neighbor, v will respond with a "recall-ack" and once x receives it, x will send a "preference" to u. Otherwise, since v receives the "preference" from x before the "recall-preference", it will match to x, subsequently send a "match-weight", and ignore the "recall-preference" message from x. When x receives the "match-weight" from v, it will match to v but since $w(u, x) > w(u, v)$, it will send a "preference" to u. In either case, x is delayed by only two rounds. One round in which its "preference" and "recall-preference" are received by v, and another round in which v responds with either a "match-weight" or a "recall-ack". Regardless of the specific scenario, let t' be the earliest time that both u and x receive each other's "preference" message, then at time t', $(u, x) \in M_{t'}$. From Lemma 2, at t, $w(M_t) \geq w(M_{t,opt}) + \frac{1}{2} \sum_{a \in A(M_t)} gain_{M_t}(a)$. Let's consider $A(M_t)$ at time t. Since all augmenting links are adjacent to e and M_{opt} is a matching, $A(M_t)$ can only include two links. Link (u, x) has at least as much gain as either link in $A(M_t)$, which means that $w(u, x) \geq \frac{1}{2} \sum_{a \in A(M_t)} gain_{M_t}(a)$. $\square$

The next change we consider is a deletion of some node u. When a node is deleted all its incident links are also deleted. Since at most one of these links can be in M, this change is almost identical to a weight-decrease of link (u, m_u). The only difference is that all of u's neighbors receive a "nbrs-update" for the change.

Corollary 3. *(Node deletion) If Algorithm 2 is quiescent at time t, and then one node u is deleted from V_t, then after O(1) rounds, $\exists t'$ s.t. $w(M_{t'}) \geq \frac{1}{2}w(M_{t',opt})$.*

The last change to consider is a node addition. Here it is vital that the new node does not send "preference" messages until it has a correct view, and this is why we need the *ready()* function in the protocol.

Lemma 11. *(Node addition) If Algorithm 2 is quiescent at time t, and one node u is added to V_t, then after O(1) rounds, $\exists t'$ s.t. $w(M_{t'}) \geq \frac{1}{2}w(M_{t',opt})$.*

Proof. The change triggers "nbrs-update" notifications to be sent to u and its neighbors. Each of u's neighbors subsequently sends a "match-weight" to u. Since the algorithm was quiescent prior to the change, the only possible augmenting links after the change are the links incident to u. If $(u, v) \in E_t$ is augmenting, then v sends a "preference" message to u. The view of u's neighbors is correct once they receive the "nbrs-update". From Lemma 11, none of them send any "preference" on a non-augmenting link. Node u will have a correct view once it receives all the "match-weight" messages from its neighbors. It will not send any "preference" message before that (Line 54). Therefore, from Lemma 1, if no augmenting links exist then $w(M_t) > \frac{1}{2}w(M_{opt})$ and the algorithm becomes quiescent again once the "match-weight" messages are received. Otherwise, at least one augmenting link exists. Node u sends a "preference" message to its maximum gain neighbor (denoted by v). Let t' be the earliest time in which both u and v receive the "preference" messages from each other. Then at t', (u, v) is added to the matching. Since all the potential augmenting links are incident to u, $A(M_t)$ can only include one link (denoted by a). From Lemma 2, $w(M_t) \geq \frac{1}{2}w(M_{t,opt}) + \frac{1}{2}w(a)$. The view of node u is correct when it sends the "preference", which from Lemma 11 guarantees that $gain_{M_t}(u, x) \geq gain_{M_{t_1}}(a)$. The matching M_{opt} does not change from t onwards and therefore at t', $w(M_{t'}) \geq \frac{1}{2}w(M_{t',opt})$.

□

Finally, we show that the matching weight increases monotonically from t' onwards. From Lemma 4, only augmenting links can be added, and from Lemma 3, a link is removed only if an adjacent link is added. This means that with every added link the matching weight increases, but since links are not added atomically, i.e., when (u, v) is added m_u and m_v change at different times, it is possible that in between the weight of the matching temporarily decreases. We show next that under the assumptions of the Locality Theorem, the weight of the matching does not temporarily decrease.

First, we note that a link can change its augmenting status only as a result of a network change event or if an incident link is added or removed from the matching:

Observation 3. *(augmenting-status)*

1. *If link (u, v) becomes augmenting at time t and no new "nbrs-update" was introduced, then an incident link was removed from M_t at t.*

2. *If link (u, v) becomes not augmenting at time t and no new "nbrs-update" was introduced, then an incident link was added to M_t at t.*

We show that under the locality property's assumptions and after the first link is added (following the network change), a link can only be augmenting if one of its end nodes is unmatched:

Lemma 12. *Assume Algorithm 2 is quiescent at time t_0, and then one of the following events occurs: node start, node stop, weight change, link removal, or link addition. Let t' denote the time the first link is added after t_0, then (u, v) is augmenting at $t'' > t'$ only if $\nexists x, y$ s.t. $(u, x) \in M_{t''}$ and $(v, y) \in M_{t''}$*

Proof. Following Observation 3, since the only "nbrs-update" were introduced at t_0, a link changes its augmenting status after t_0 only when an incident link is removed or added to the matching. Furthermore, from Lemma 3, a link is removed only if an adjacent link is added. Therefore, the augmenting status of links only changes when links are added to the matching. We prove the lemma by induction on the added links to the matching.

Base: we show that the lemma holds after the first link (which we denote by (u, v)) is added. We denote by u' and v' the values of m_u and m_v at time t_0. Since the algorithm was quiescent at t_0, no augmenting link exists before the network change and we can consider only links that become augmenting after the change.

Weight increase: following the change only one link can become augmenting which means it must be (u, v). After (u, v) is added, u' and v' lose their match, and links incident to u' and v' can become augmenting (which correlates with the lemma). Links incident to (u, v) were not augmenting previously and have even less cause to become augmenting now since $w(u, v) > w(u, u') + w(v, v')$. All other links were not augmenting before and their augmenting status does not change since there were no changes to their incident links.

Weight decrease: only links incident to the changed link can become augmenting and subsequently added to the matching. This means that either the weight of (u, u') or (v, v') decreased. Without lost of generality, we assume that the weight of (u, u') decreased. After (u, v) is added, links next to u' can remain augmenting, since u' now lost its match. Links next to v' can become augmenting since v' lost its match. As for u, it is shown by Lemma 10 that u matches with its maximum gain neighbor. This means that $\forall x \in \Gamma_u \, w(u, v) > w(u, x)$, which guarantees that after (u, v) is added, $\nexists x \in \Gamma_u$ s.t. (u, x) is augmenting. A link next to v, unless it is next to u', was not augmenting before, and will not become augmenting now since $w(u, v) > w(v, v')$. All other links were not augmenting before and their augmenting status does not change since there were no changes to their incident links.

Node deletion: either u' or v' was deleted, and without loss of generality we assume that u' is deleted. Node u lost its match, and any number of u's links might become augmenting. As shown by Lemma 3, u matches with its maximum gain neighbor, which means that $\forall x \in \Gamma_u \; w(u, v) > w(u, x)$, and therefore after (u, v) is added, $\nexists x \in \Gamma_u$ s.t. (u, x) is augmenting. As for links next to v, they were not augmenting before, and since $w(u, v) > w(v, v')$ they will not become augmenting now. Links next to v' might become augmenting since v' lost its match. All other links were not augmenting before and their augmenting status does not change since there were no changes to their incident links.

Node addition: either u or v is added, and without loss of generality we assume that u is added. Then, any number of u's links might become augmenting. As shown by Lemma 11, u matches with its maximum gain neighbor, which means that $\forall x \in \Gamma_u \; w(u,v) > w(u,x)$, and therefore after (u,v) is added, $\nexists x \in \Gamma_u$ s.t. (u,x) is augmenting. As for links next to v, they weren't augmenting before, and since $w(u,v) > w(v,v')$ they will not become augmenting now. Links next to v' might become augmenting since v' lost its match. All other links were not augmenting before and their augmenting status does not change since there were no changes to their incident links.

Step: we assume by way of contradiction that a link is added at t, and subsequently, a link (u,v) is augmenting even though $\exists x, y$ s.t. $(u,x) \in M_t$ and $(v,y) \in M_t$. Since link (u,v) is augmenting, (1) $w(u,v) > w(u,x) + w(v,y)$.

Case 1: link (u,v) was not augmenting at $t-1$. Since there are no more "nbrs-update" messages introduced, the only way (u,v) can become augmenting at t is if u or v lose their match which means they cannot both be matched at t.

Case 2: link (u,v) was augmenting at $t-1$. Then, since the lemma holds at $t-1$ it must be that either (u,x) or (v,y) were added at t. Without loss of generality, we assume that (u,x) was added at t. Since there are no network changes after t_0, and each node receives its "nbrs-update" message before making any other steps, we can both omit that time when we consider the weight of a link and interchange $w(u,v)$ for $w_u(v)$ for every node. Node (u,x) was added at t, which means that $pm_u = x$ at t and we denote by t_1 the time pm_u becomes x. Then, at t_1, u prefers x over v which means that (2) $w(u,x) > w(u,v) - nbrs\text{-}mw_v(u)$. It must be that $nbrs\text{-}mw_v(u) > w(v,y)$, or otherwise (2) and (1) contradict each other. Let y' be the the match of v at time t_1 which correlates to $nbrs\text{-}mw_v(u)$. Either $y = y'$ and $w(v,y)$ changed which we consider later, or $y \neq y'$, and v changed its match. Since $w(v,y) < w(v,y')$ and only augmenting links are added, it must be that v lost its match before matching with y. Before v matches with y, pm_v must first change to y and we denote by t_2 the time that pm_v becomes y. When a node loses its match,

$ready()$ is false until the node receives a "nbr-ack" from each of its neighbors. At t_1, node u did not know yet that v lost its match, which means that pm_v can only change after t_1 ($t_2 > t_1$). At t_2, (3) $w(v,y) > w(u,v) - nbrs\text{-}mw_v(u)$, and since v received all notifications from u up until t_1 and at t_1, $pm_u = x$, it must be that (4) $nbrs\text{-}mw_v(u) \leq w(u,x)$. From (3) and (4) we get (5) $w(v,y) > w(u,v) - w(u,x)$ which contradicts (1). Finally, we consider the case that $y = y'$ and yet $nbrs\text{-}mw_u(v)$ at t_1 is bigger than $w(v,y)$ at t. This would be possible only if exists a $\langle$"match-weight"$, m \to m \neq u, mw\rangle \in Q_{v,u}$ s.t. $mw < nbrs\text{-}mw_u(v)$. (i.e., a "match-weight" message that notifies of a weight decrease of the current match of v.). However, this kind of message is only sent after a network change of weight decrease and it is the first message u receives after t_0. This is before any message that makes (u,x) augmenting and it cannot still exist at t_1.

□

In the final step we show that the matching weight strictly increases when a link with an unmatched node is added:

Claim 12. *Consider an edge (u,v) that is added to the matching, where m_u becomes v at time t_1, and m_v becomes u at time $t_2 > t_1$. Let w_1 and w_2 denote u and v's match weights at time t_1, respectively. If no "nbrs-update" is received by either u or v between t_1 and t_2 and $\nexists mw_> \in Q_{v,u}$ at t_1, and $w_1 = 0$ or $w_2 = 0$ at t_1, then $w(M)_{t_1} \geq w(M)_{t_1-1}$ and $w(M)_{t_2} \geq w(M)_{t_2-1}$.*

Proof. We separate to the two cases.

$w_1 = 0$**:** since m_u has no match, then no weight is lost at t_1 and $w(M)_{t_1} = w(M)_{t_1-1}$. It follows from Lemma 4 that $w(M)_{t_2} \geq w(M)_{t_2-1}$, since $w(u,v) > w_1 + w_2$.

$w_2 = 0$**:** since $pm_v = u$ at t_1, then (u,v) is added to M at t_1. It follows from Lemma 4 that $w(M)_{t_1} \geq w(M)_{t_1-1}$ since $w(u,v) > w_1 + w_2$. When m_v becomes u at t_2, no weight is lost since $w_2 = 0$ and $w(M)_{t_2} = w(M)_{t_2-1}$.

□

Corollary 4. *Assume Algorithm 2 is quiescent at time t, and then one of the following events occurs: node start, node stop, weight change, link removal, or link addition. If t' is the time that the first link is added following the change, then from t' onwards the matching weight never decreases.*

Theorem 4 (restated). (Locality) If Algorithm 2 is quiescent at time t, and then one of the following events occurs – node start, node stop, weight change, link removal, or link addition, then after O(1) rounds, $\exists t'$ s.t. $w(M_{t'}) \geq \frac{1}{2}w(M_{t',opt})$.

Proof. Immediate from Lemmas 9, 10, 11, Corollary 3, and Corollary 4. □

Chapter 6

Conclusions

In this thesis we studied the *dynamic weighted matching* problem. Namely, we considered the weighted matching problem in an asynchronous, fault prone, and dynamic network, where the topology and link weights may change. We developed a distributed algorithm that can cope with these conditions and is resistant to any combination of network changes. We showed that the algorithm stabilizes, quiesces, and provides a 2-approximation when changes cease. Moreover, we proved that following a single change, the algorithm converges back to the approximation in $O(1)$ time, thereby not suffering from the drawbacks of other distributed matching algorithms in a dynamic environment. Lastly, in the static case, the algorithm converges to a 2-approximation in $O(|V|)$ time and $O(|E|)$ message complexity similar to [13].

Our focus on the model and algorithm is motivated by applications to wireless networks which have many of the characteristics described above. In particular, due to the Local Pooling conditions [4], in many such networks, *the 2-approximation algorithm will actually achieve 100% throughput*. Yet, in many realistic cases, the interference constraints are more complex than primary interference constraints that imply that the set of active links should be a matching in the network graph [6, 30]. For example, under secondary interference constraints, each pair of simultaneously active links must be separated by at least two hops (these constraints are usually used to model IEEE 802.11 networks [6]). In general, an interference graph is used to model the various interference constraints, and a specific transmission schedule corresponds to an independent set in the interference graph. When taking into account the physical layer characteristics, the schedule and the power allocations should adhere to the Signal to Interference and Noise Ratio (SINR) constraints [23, 11, 5].

In all those cases, there is a need for distributed algorithms for an asynchronous and dynamic environment (e.g., when nodes leave, join, or even change their location, not only link weights change but also there is a significant effect on the SINR). However, most of the previous work has focused either on the throughput and stability region implications or on efficient distributed implementations. It has usually been assumed that the environment is synchronous and static. A potential direction for future work is to build on the results of this thesis and develop distributed approximation algorithms for general interference constraints as well as for the SINR model that will be tailored for asynchronous and dynamic environments.

Bibliography

[1] S. Banerjee, A. Chowdhury, and S. Ghosh. Distributed approximation for maximum weight matching on bounded degree bounded integer weight graphs. *Inf. Process. Lett.*, 109:790–794, June 2009.

[2] S. Baswana, M. Gupta, and S. Sen. Fully dynamic maximal matching in O(log n) update time. In *FOCS*, pages 383–392, 2011.

[3] C. Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences of the United States of America*, 43(9):842–844, 1957.

[4] B. Birand, M. Chudnovsky, B. Ries, P. Seymour, G. Zussman, and Y. Zwols. Analyzing the performance of greedy maximal scheduling via local pooling and graph theory. *IEEE/ACM Trans. Netw.*, 20(1):163 –176, Feb. 2012.

[5] D. Chafekar, V. Kumar, M. Marathe, S. Parthasarathy, and A. Srinivasan. Capacity of wireless networks under SINR interference constraints. *Wirel. Netw.*, 17:1605–1624, Oct. 2011.

[6] P. Chaporkar, K. Kar, X. Luo, and S. Sarkar. Throughput and fairness guarantees through maximal scheduling in wireless networks. *IEEE Trans. Inform. Th.*, 54(2):572–594, Feb. 2008.

[7] A. Czygrinow and M. Hańćkowiak. Distributed algorithms for weighted problems in sparse graphs. *J. of Discrete Algorithms*, 4:588–607, Dec. 2006.

[8] D. E. Drake and S. Hougardy. A simple approximation algorithm for the weighted matching problem. *Inf. Process. Lett.*, 85:211–213, 2003.

[9] J. Edmonds. Paths, trees, and flowers. *Canad. J. Math.*, 17:449–467, 1965.

[10] R. Fan and N. Lynch. Gradient clock synchronization. In *Proc. ACM PODC'04*, 2004.

[11] A. Fanghänel, T. Kesselheim, H. Räcke, and B. Vöcking. Oblivious interference scheduling. In *Proc. ACM PODC'09*, 2009.

[12] H. N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, SODA '90, pages 434–443, 1990.

[13] J.-H. Hoepman. Simple distributed weighted matchings. In *cs.DC/0410047*, 2004.

[14] Z. Ivkovic and E. L. Lloyd. Fully dynamic maintenance of vertex cover. In *Proceedings of the 19th International Workshop on Graph-Theoretic Concepts in Computer Science*, WG '93, pages 99–111. Springer-Verlag, 1994.

[15] C. Joo. A local greedy scheduling scheme with provable performance guarantee. In *Proc. ACM MobiHoc'08*, 2008.

[16] C. Koufogiannakis and N. Young. Distributed algorithms for covering, packing and maximum weighted matching. *Distrib. Comput.*, 24:45–63, 2011.

[17] X. Lin and N. B. Shroff. The impact of imperfect scheduling on cross-layer rate control in wireless networks. *IEEE/ACM Trans. Netw.*, 14(2):302–315, Apr. 2006.

[18] Z. Lotker, B. Patt-Shamir, and S. Pettie. Improved distributed approximate matching. In *Proc. ACM SPAA'08*, 2008.

[19] Z. Lotker, B. Patt-Shamir, and A. Rosen. Distributed approximate matching. In *Proc. ACM PODC'07*, 2007.

[20] F. Manne and M. Mjelde. A self-stabilizing weighted matching algorithm. In *Proc. SSS'07*. Springer-Verlag, 2007.

[21] E. Modiano, D. Shah, and G. Zussman. Maximizing throughput in wireless networks via gossiping. In *Proc. ACM SIGMETRICS'06*, June 2006.

[22] T. Nieberg. Local, distributed weighted matching on general and wireless topologies. In *Proc. ACM DIALM-POMC'08*, 2008.

[23] Y. P. O. Goussevskaia and R. Wattenhofer. Efficiency of wireless networks: Approximation algorithms for the physical interference model. *Foundations and Trends in Networking*, 4:303–420, 2010.

[24] K. Onakand and R. Rubinfeld. Dynamic approximate vertex cover and maximum matching. In O. Goldreich, editor, *Property Testing*, volume 6390 of *LNCS*, pages 341–345. Springer, 2011.

[25] A. Panconesi and M. Sozio. Fast primal-dual distributed algorithms for scheduling and matching problems. *Distributed Computing*, 22:269–283, 2010.

[26] R. Preis. Linear time 1/2 -approximation algorithm for maximum weighted matching in general graphs. In *Proc. STACS'99*. Springer-Verlag, 1999.

[27] P. Sankowski. Faster dynamic matchings and vertex connectivity. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, pages 118–126, 2007.

[28] V. Turau and B. Hauck. A new analysis of a self-stabilizing maximum weight matching algorithm with approximation ratio 2. *Theor. Comput. Sci.*, 412(40):5527 – 5540, 2011.

[29] M. Wattenhofer and R. Wattenhofer. Distributed weighted matching. In R. Guerraoui, editor, *Distributed Computing*, volume 3274 of *Lecture Notes in Computer Science*, pages 335–348. Springer Berlin / Heidelberg, 2004.

[30] X. Wu, R. Srikant, and J. R. Perkins. Queue-length stability of maximal greedy schedules in wireless networks. *IEEE Trans. Mobile Comput.*, 6(6):595–605, June 2007.

פתרון לוקאלי לשידוך במשקל מכסימלי ברשתות דינמיות

ליאת עצמון גוז

פתרון לוקאלי לשידוך במשקל מכסימלי ברשתות דינמיות

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים בהנדסת חשמל

ליאת עצמון גוז

הוגש לסנט הטכניון – מכון טכנולוגי לישראל
אב תשע"ב חיפה יולי 2012

המחקר נעשה בהנחיית פרופסור עדית קידר בפקולטה להנדסת חשמל.

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי.

תודות

ברצוני להודות לכל האנשים אשר ליוו אותי בשנותי בטכניון והפכו את החוויה לבלתי נשכחת.

תודה לפרופ' עדית קידר, שבראיה חיובית מופלאה לא ויתרה עלי או על העבודה הזו למרות כל המהמורות שנקלענו אליהן בדרך. זכיתי במנחה מיוחדת ואני חבה לך תודה אדירה על ההנחיה וההשקעה בי.

תודה לפרופ' יורם מוזס על הידידות וההכוונה - למדתי רבות מהעבודה יחד. תודה לכל חברי בטכניון, על התמיכה ברגעים הקשים וההשתתפות ברגעי ההצלחה - חיבקתם, הצחקתם, העשרתם ועוררתם בי השראה. אם הייתי יכולה הייתי לוקחת את כולכם איתי. תודה לאנשי צוות הפקולטה על האדיבות, העזרה, והסבלנות תמיד.

תודה לאבי שהשריש בי את האהבה ללימודים, לאימי, אחותי ואחי על התמיכה וההתעניינות. תודה למשפחת גוז היקרה, למודי הנסיון, על ההבנה וההקשבה.

אני מקדישה את התזה לאהובי צביקה, הסלע האיתן שלי, הסוחף אותי להאמין שאין דבר שאיני יכולה לו.

תקציר

בתורת הגרפים, *שידוך* הינו תת-קבוצה של קשתות כך שאף זוג קשתות אינו חולק צומת משותף. *שידוך במשקל מכסימלי* הינו שידוך עבורו סכום משקלי הקשתות בשידוך הינו מכסימלי מבין כל השידוכים האפשריים. בבעית השידוך במשקל מכסימלי המבוזרת, נדרשים הצמתים בגרף לשתף פעולה על מנת למצוא שידוך במשקל מכסימלי.

בעית השידוך מתמפה בצורה טבעית לבעית תזמון תשדורות ברשתות אלחוטיות בהן אין חיבור ישיר מנקודה לנקודה (multi-hop), ואשר עובדות תחת אילוצי התנגשות מדרגה ראשונה (primary interference constraints). ברשתות אלה, אילוצי ההתנגשות מדרגה ראשונה מחייבים שתזמון השידור של שני צמתים סמוכים יעשה כך שבכל רגע נתון צומת מתקשר לכל היותר עם צומת בודד. כיוון שבכל רגע נתון קשתות מסוימות יכולות להיות חשובות יותר מקשתות מאחרות (לדוגמא עקב אורך תורים, עדיפויות, וכו') מקובל לייחס משקל לקשתות השונות, ומטרת אלגוריתם התזמון הינה למכסם את משקל השידוך הכולל.

רוב האלגוריתמים הקיימים למציאת שידוך במשקל מכסימלי בסביבה מבוזרת מספקים פתרון חד-פעמי (one-shot) ומניחים מערכת נטולת שינויים, סינכרונית, וללא תקלות. הנחות אילו אינן מתאימות לרשתות אלחוטיות, שהינן דינמיות, אסינכרוניות, ומועדות לתקלות. ראשית, סנכרון במערכות אלחוטיות הינו בעיה סבוכה, שאינה בהכרח לוקאלית. שנית, רשתות אלחוטיות משתנות לאורך זמן– צמתים מתווספים, עוזבים או משנים מקום, וקווי תקשורת מופיעים ונעלמים. יתר על כן, משקלן של קשתות משתנה ללא הרף עקב הצטברות ושידור חבילות תקשורת. שימוש באלגוריתם שאינו מסתגל לשינוים (משמע ללא ייצוב עצמי) מחייב הרצות חוזרות של האלגוריתם מדי פעם בכדי להתאים את הפתרון לשינויים התכופים. גישה זו הינה בעייתית עכב ההכרח לאזן בין התקורה שבהרצות חוזרות ונשנות של האלגוריתמים לבין אורך פרקי הזמן שבהם התזמון אינו אופטימלי למצב הנוכחי ברשת. (קרי: פשרה בין מחיר החישוב למחיר השידור.)

בכדי להתגבר על חסרונות אלה ועל מנת לספק פתרון התואם לתנאי הסביבה ברשתות אלחוטיות, אנו מגדירים בעבודת מחקר זו בעיה חדשה- *בעית השידוך במשקל מכסימלי*

ברשתות דינמיות. בעיה זו עוסקת בשידוך במשקל מכסימלי תחת הנחות סביבה מבוזרת, אסינכורנית, דינמית, ומועדת לתקלות. פתרון לבעיה זו נדרש להגיב מיידית לשינויים ברשת ולכן לעולם תואם למצב הרשת הנוכחי. תזמון שידור ברשת אלחוטית אשר מתבסס על פתרון שכזה, מאפשר משלוח הודעות על קשתות שהינן חלק מהשידוך הנוכחי כבר במהלך שלב ההתכנסות של האלגוריתם, מבלי להיות מושפע מחלקי הרשת בהם התרחשו שינויים או מחלקי הרשת בהם האלגוריתם עדיין לא התכנס למצב יציב. בהקשר זה, אנו מעוניינים במשקל השידוך לאורך זמן, כולל זמנים בהם האלגוריתם עדיין לא התייצב. כאשר הרשת מתייצבת ולא מתרחשים בה שינויים נוספים, נדרש כי השידוך יתייצב אף הוא.

האלגוריתם האסינכרוני היחיד לבעית השידוך במשקל מכסימלי נקרא *greedy maximal matching*, והוא מספק קירוב 2 לפתרון האופטימלי. במהלך ריצת אלגוריתם זה, קשת נוספת לשידוך אם משקלה הינו מכסימלי מבין כל הקשתות הסמוכות לה. האלגוריתם עובד היטב ברשות שאינן משתנות, וכאשר לא קיים שידוך התחלתי, אבל הוא סובל משתי מגרעות מרכזיות אשר פוגמות בהתאמתו לסביבה דינמית. ראשית, זמן ההתכנסות חזרה לשידוך בקירוב 2 לאחר שינוי יכול להיות לינארי בגודל הרשת, אפילו במקרים של שינוי בודד. שנית, משקל השידוך לאחר שינוי יכול לקטון, אפילו במקרים קיצוניים בהם התרחש שינוי בודד שהגדיל את משקלו של צומת בגרף, או במקרים שבהם אף צומת בשידוך הקיים לא השתנה.

בעבודת מחקר זו אנו מציעים גישת חלופית אשר אינה סובלת מהמגרעות האמורות. אנו מציגים אלגוריתם דינמי לשידוך במשקל מכסימלי אשר מספק שידוך קביל בצורה רציפה לאורך כל פעולתו. באלגוריתם המוצע, קשתות נוספות לשידוך רק אם הן מגדילות את משקל השידוך. אנו מוכיחים שלאחר הפסקת השינויים ברשת, האלגוריתם מתייצב, חדל משליחת הודעות, ומספק קירוב 2. יתר על כן, לאחר התייצבותו, האלגוריתם מטפל בשינויים חדשים בצורה *לוקאלית*, כלומר הוא מתכנס חזרה לקירוב 2 בזמן קבוע. אנו מראים שלאחר שינוי יחיד– הוספת צומת, הסרת צומת, הוספת קשת או הסרת קשת– האלגוריתם מתכנס חזרה אל הקירוב בסיבוכיות זמן $O(1)$. בנוסף לפתרון המגרעות האמורות במקרה הדינמי, ביצועי האלגוריתם המוצע בסביבה סטטית זהים לביצועי אלגוריתם ה-greedy maximal matching.

לסיכום, תרומתה העיקרית של עבודת מחקר זו היא הטיפול בבעית השידוך במשקל

מכסימלי ברשתות אסינכרוניות, דינמיות, ומועדות לתקלות. אנו מגדירים את *בעית השידוך במשקל מכסימלי ברשתות דינמיות*, ומציגים אלגוריתם לוקאלי ויציב, אשר מספק קירוב 2 לבעיה.