Algorithms **2013**, *6*, 824-856; doi:10.3390/a6040824

OPEN ACCESS

algorithms

ISSN 1999-4893

www.mdpi.com/journal/algorithms

Article

Overlays with Preferences: Distributed, Adaptive Approximation Algorithms for Matching with Preference Lists

Giorgos Georgiadis * and Marina Papatriantafilou

Department of Computer Science and Engineering, Chalmers University of Technology, S-412 96 Göteborg, Sweden; E-Mail: ptrianta@chalmers.se

* Author to whom correspondence should be addressed; E-Mail: georgiog@chalmers.se; Fax: +46-31-7723663.

Received: 21 June 2013; in revised form: 7 November 2013 / Accepted: 8 November 2013 / Published: 19 November 2013

Abstract: A key property of overlay networks is the overlay nodes' ability to establish connections (or be matched) to other nodes by preference, based on some suitability metric related to, e.g., the node's distance, interests, recommendations, transaction history or available resources. When there are no preference cycles among the nodes, a stable matching exists in which nodes have maximized individual satisfaction, due to their choices, however no such guarantees are currently being given in the generic case. In this work, we employ the notion of node satisfaction to suggest a novel modeling for matching problems, suitable for overlay networks. We start by presenting a simple, yet powerful, distributed algorithm that solves the many-to-many matching problem with preferences. It achieves that by using local information and aggregate satisfaction as an optimization metric, while providing a guaranteed convergence and approximation ratio. Subsequently, we show how to extend the algorithm in order to support and adapt to changes in the nodes' connectivity and preferences. In addition, we provide a detailed experimental study that focuses on the levels of achieved satisfaction, as well as convergence and reconvergence speed.

Keywords: matching; overlay networks; distributed; adaptive

1. Introduction

Overlays represent a significant puzzle piece in contemporary and future networking infrastructure, whether they are intended to support resource sharing or other collaborative applications, such as

searching, *ad hoc* connectivity and persistent services. The common scenario is that peers are able to know part of the overlay network (in terms of potential neighbors), but want to connect only to a small number of other peers in order to conserve resources. Connection decisions are not to be taken blindly, but should rather be based on some suitability metric related to, e.g., the peer's distance, interests, recommendations, transaction history or available resources. In the fully distributed scenario, every peer may follow an individually chosen metric (that it may even not want to disclose to other peers), but still wants to be able to coordinate with others in order to improve the quality of its connections. We present algorithms that enable peers that follow them to achieve a guaranteed level of collective quality in their connections. They achieve that by disclosing a limited amount of metric information to their immediate neighbors, but not the metric itself and are, in fact, independent of any individual metric choices. At the same time, the algorithms proposed here are able to adapt and tolerate the high dynamicity commonly found in these resource sharing overlays, with peers leaving, joining or changing the metrics about their neighbors at any time.

Observe that finding good methods to help peers establish connections and accommodate such needs or preferences relates to some form of *matching problem* in a graph. Moreover, in the context of overlay networks, an important goal is to enable peers to satisfy their preferences during overlay construction: note that these "preference lists" point towards a form of stable marriage/roommates problem [1–3]. The first to observe the relation between overlay construction and stable marriage problems were Gai *et al.* [4], followed by Mathieu [5], in which overlay construction is regarded as a *generalized* stable roommates matching problem [6] (also referred to as a *b-matching* problem [5,7]). In such a problem, the agents under consideration have more than one opportunity to connect to each other, in a similar way to, e.g., attendees in a conference exchanging the limited amount of calling cards they brought with them. The goal for each agent in this setting is to be able to form the desired amount of connections with the highest quality (most preferred) neighbors. However, the existence of cycles among the preference lists of the peers may turn overlay construction into a challenging task: matchings between peers on a cycle always create opportunities for improvement at other sites in the network, leading to infinite loops and unstable configurations. Gai *et al.* in [4] proved that, in the case of an acyclic preference system, there is always a stable configuration and also supplied examples of preference systems based on global or symmetric metrics. Furthermore, in order to give a qualitative measure of the stabilization process, Mathieu in [5] defined the notion of *satisfaction* gleaned by a peer out of its current matching choices.

With the research focused on preference systems that are known to have a stable configuration, such as acyclic ones, there is a relative scarcity of results concerning *arbitrary* preference systems and practical algorithms for achieving stable configurations (if existing) or good approximations thereof. Can we look at this problem from a different perspective? How can a solution be measured? What are the characteristics of a practical algorithm in a fully distributed scenario? Can we adapt to and tolerate changes in the overlay, such as preference change or peer joining/leaving?

To propose answers to these questions, we focus on the distributed b-matching problem: we suggest a novel modeling, present algorithms for both its adaptive and non-adaptive forms, show convergence and guaranteed approximation bounds and complement our results with an extensive experimental study.

- **Novel modeling:**We start by reflecting further on the evaluation of different matching choices of peers (as proposed in [5]). By considering satisfaction about the peer's connection choices

(compared to the optimal choices) as an optimization metric, we focus on distributed algorithms that try to maximize the satisfaction of peers that follow them (either a group or the whole overlay). We then proceed to propose a method of constructing an acyclic preference system from any arbitrary one, and we use this helper acyclic system to reduce the initial problem to a many-to-many maximum weighted matching problem.

- **Algorithmic solutions:**Generalizing the approximation algorithm for the one-to-one matching problem found in [8,9], we present a simple, fully distributed, $\frac{1}{2}$-approximation algorithm for the many-to-many maximum weighted matching problem using only communication among immediate neighbors. In addition, we complement our distributed algorithm with a variant that can adapt to and tolerate changes in the overlay, such as peer joining/leaving and preference changes. We prove that both algorithms converge and that their computed solution is at least $\frac{1}{2}\left(1+b_{\max}^{-1}\left(1+\frac{1-b_{\max}^{-1}}{2s+b_{\max}^{-1}}\right)\right)$ of the maximum total satisfaction of the initial system, where $b_{\max}$ is the maximum connection quota in the graph and $s = \frac{L_{\max}}{L_{\min}}$ is the ratio of the maximum and minimum neighbor list sizes in the graph, $L_{\max}$ and $L_{\min}$, respectively, resulting in a $\frac{1}{4}\left(1+b_{\max}^{-1}\left(1+\frac{1-b_{\max}^{-1}}{2s+b_{\max}^{-1}}\right)\right)$-approximation algorithm in total.
- **Experimental study:** We also provide an extensive experimental study of the behavior of the algorithm under a variety of scenarios, including normal operation, but also operation under high stress. Under normal operation, we focus on the levels of achieved satisfaction, as well as convergence time (in the case of our non-adaptive algorithm) and reconvergence time (in the case of our adaptive algorithm). Specifically, we show that the resulting satisfaction is always significantly higher than the worst-case theoretical bound after convergence, but also remains at high levels during and after reconvergence, while reconvergence is achieved in an efficient way under a variety of changes. Besides, motivated by [10,11], we conducted experiments that focus on the stability of the network under join/leave attacks, by exposing it to high churn rates, and observed that the algorithm withstands the attacks while maintaining graceful satisfaction values throughout them.

The rest of the paper is organized as follows. In Section 2, we introduce the necessary notation and define our problem, while in Section 3, we present, separately, the notion of node satisfaction, along with an extensive commentary, due to its importance in our modeling. In Section 4, we show how our original b-matching problem can be reduced into a many-to-many maximum weighted matching problem that is guaranteed to have a stable solution, and we prove that this can be done with constant factor approximation. We present our distributed algorithm for the many-to-many maximum weighted matching problem in two variations, adaptive and non-adaptive, in Section 5, along with analytical proofs of their approximation ratio for our original problem. Our extensive experimental study can be found in Section 6, while discussion on the presented, as well as related work can be found in Section 7. Finally, Section 8 concludes the paper.

2. Problem Model

We represent a connectivity network as an undirected graph $G(V, E)$, with $|V| = n$, $|E| = m$, where V is the set of overlay peers and E the set of potential connections. Each node i has degree d_i and

keeps a preference list L_i of all nodes in its neighborhood Γ_i (in the rest of the paper and when it is clear from the context, we will use notation L_i to denote both the list and its length). Let $R_i(j)$ denote the rank of node j in node i's preference list, with $R_i(\cdot) \in \{0, 1, \ldots, |L_i| - 1\}$, attributing zero to its most desirable neighbor. Each node i wants to maintain at most b_i connections (in order to form the overlay) to the best possible nodes according to its preference list and rank function, and at no point can it exceed this number. Clearly, it must be $b_i \leq |L_i|$. In the following sections, we will refer to two nodes as *neighboring nodes* when they are connected by an edge in graph G and *connected nodes* when they are matched by a matching algorithm. The problem of trying to find a many-to-many matching that respects the individual preferences and connection quotas b_i is a form of a generalized stable roommates problem called the *stable fixtures problem* [6] or *b-matching* [5]. We call *adaptive b-matching* the dynamic form of b-matching, where nodes can join, leave or change preferences at any time. In the remainder of this paper, we will refer to these events simply as *changes*. We will also consider an asynchronous model for messages and will not consider link or node failures, *i.e.*, messages arrive asynchronously, but do not get lost, and nodes depart gracefully or their absence can be detected by other means (for example, special periodic "alive" messages).

In order to measure the success of node i's efforts in establishing its b_i connections, we make use of the notion of *satisfaction*, S_i, which is defined in [5] by the following formula:

$$S_i = \frac{c_i}{b_i} + \frac{c_i\left(c_i - 1\right)}{2b_iL_i} - \frac{\sum\limits_{j \in C_i} R_i\left(j\right)}{b_iL_i} \tag{1}$$

where C_i (with $|C_i| = c_i \leq b_i$) is an *ordered list of node i's connections* in decreasing preference. Satisfaction, due to its significance in the context of this paper, is discussed and analyzed separately in Section 3.

We define an optimization variation of the b-matching problem, which we call the *maximizing satisfaction b-matching problem*, where the objective is to find a b-matching that maximizes the total sum of the nodes' satisfaction. Later on, we also define a *truncatedSmaximizing satisfaction b-matching problem*, which is based on the same basic b-matching problem, but tries to maximize a different (truncated) satisfaction function (see Section 4) (although we may refer to them simply as the b-matching problem and the truncatedS b-matching problem, respectively, it will be clear from the context that we are referring to the optimization versions).

Consider that edges $e = (i, j) \in E$ in the previously defined graph $G(V, E)$ have assigned weights $w(i, j) = w_{ij}$. A *weighted matching problem* on this graph is the problem of finding a set of edges, such that their weight sum is maximized and there are no common endpoints between them. The many-to-many variant that we will use replaces the last constraint on no common endpoints with node capacities that need to be respected, which, in this case, are the connection quotas b_i per node i.

During the analysis of the distributed algorithm, we will use the notion of a *locally heaviest edge* [9]: if we define the set E_{ij} as the set of edges that have either node i or node j as an endpoint (but not both):

$$E_{ij} = \left\{(i, n_i)\,|n_i \in \Gamma_i \backslash j\right\} \cup \left\{(j, n_j)\,|n_j \in \Gamma_j \backslash i\right\} \tag{2}$$

an edge (i, j) is called locally heaviest if it has the greatest weight among all edges $e \in E_{ij}$:

$$w\left(i, j\right) > w\left(e\right), e \in E_{ij} \tag{3}$$

In the following sections, we will assume that the nodes following the algorithms proposed here, whether a group or the whole network, cooperate willfully, and we will provide guarantees about the maximization of the total satisfaction in this group or network.

3. Node Satisfaction: A Metric for Optimization

In this section, we discuss the previous definition of satisfaction in depth, providing examples that motivate its usage and clarify the defining formula. Furthermore, by digging into satisfaction, we get a derivative form of Equation (1) to use directly in the algorithms of the following sections.

3.1. Interpreting Satisfaction

It is easy to see from Equation (1) that satisfaction S_i of node i takes values in the range $[0, 1]$, with a maximum value when all b_i connections are established with node i's top b_i ranked neighbors. In the general case of $c_i \leq b_i$ connections, S_i takes the value of $\frac{c_i}{b_i}$ minus a penalty if the connected nodes are not the top choices in the preference list. More specifically, using the previously defined connection list C_i of node i, observe that for each connected node j the penalty is proportional to the difference between its rank $Q_i(j)$ in the connection list of node i and its rank $R_i(j)$ in the preference list of node i:

$$\begin{aligned} S_i &= \frac{c_i}{b_i} - \frac{\sum\limits_{j \in C_i} \left(R_i\left(j\right) - Q_i\left(j\right)\right)}{b_i L_i} = \\ &= \frac{c_i}{b_i} - \frac{\sum\limits_{j \in C_i} R_i\left(j\right) - \frac{c_i(c_i-1)}{2}}{b_i L_i} = \\ &= \frac{c_i}{b_i} + \frac{c_i\left(c_i - 1\right)}{2 b_i L_i} - \frac{\sum\limits_{j \in C_i} R_i\left(j\right)}{b_i L_i} \end{aligned}$$

As defined in Equation (1), satisfaction S_i also measures the deviation of the connection list C_i from the optimal case. The following example (Figure 1) illustrates this notion of deviation:

Figure 1. Example of satisfaction computation for node i with $b_i = 4$.

Rank $R_i(j)$	0	1	2	3	4	5
L_i	15	31	5	7	11	22

Rank $Q_i(j)$	0	1	2	3
C_i	31	7	11	22

The optimal case for connections in C_i would be to occupy the top c_i slots in node i's preference list (*i.e.*, for node 31 to have $R_i(31) = 0$). For each node j that deviates from this optimal case a penalty must be paid, giving to node i a satisfaction of:

$$\begin{aligned} S_i = \frac{c_i}{b_i} &- \frac{R_i\left(31\right) - Q_i\left(31\right)}{b_i L_i} - \frac{R_i\left(7\right) - Q_i\left(7\right)}{b_i L_i} \\ &- \frac{R_i\left(11\right) - Q_i\left(11\right)}{b_i L_i} - \frac{R_i\left(22\right) - Q_i\left(22\right)}{b_i L_i} = 0.708 \end{aligned}$$

In short, the interpretation of satisfaction (Equation (1)) we consider here can be described simply as follows: a node i is totally satisfied when it connects to its b_i "most wanted" preferences with its b_i degrees of freedom; otherwise, each "opening" in the list and/or each degree of freedom that remains unused (not matched) is reflected in the node's lower satisfaction.

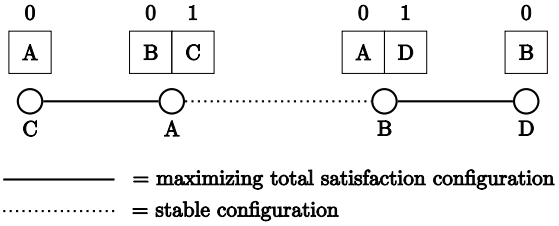
3.2. Satisfaction and Stability

The most common way to describe stability in stable roommate problems is through the notion of *blocking pairs* [12]. For example, when indifference is not considered in the preference lists, two nodes form a blocking pair when they both strictly prefer each other over some of their current matches, and a matching configuration is called *stable* when there are no blocking pairs in the network. The satisfaction definition of Equation (1), together with the interpretation in Section 3.1, can assist in moving beyond the classic stable matchings and towards the overlay network context. Although work in this area has been fairly limited, the notion of satisfaction does seem to stand out as a natural extension to the blocking pairs notion: instead of nodes forming a strict, stable matching, satisfaction gives them the flexibility to make choices of measurable "goodness", by assigning real values to their choices and applying algorithms that, e.g., maximize the total satisfaction on a network level. This *maximizing total satisfaction* criterion introduces a social element to the traditional notion of stability and can be considered stricter than the above definition of blocking pairs, as the following observation suggests.

Observation 1 *If a stable configuration exists, it can differ from a maximizing total satisfaction configuration, but the total satisfaction achieved by the latter is greater than or equal to the total satisfaction of the first.*

An example network that illustrates this observation can be found in Figure 2.

Figure 2. Example network exhibiting different stable and maximizing total satisfaction configurations.



In this toy example, we assume that every node has only one free connection slot, and the preference lists are shown above the respective nodes. Here, a maximizing total satisfaction configuration is depicted by solid lines connecting nodes A and B with C and D, respectively. However, this configuration is not stable, since nodes A and B constitute a blocking pair (each one prefers the other over their current partners). By resolving this blocking pair, we get the stable configuration depicted by a dotted line between nodes A and B (see Figure 2). However, using Equation (1) to compute the total satisfaction of the network before (maximizing total satisfaction configuration S) and after (stable configuration S')

the blocking pair resolution, we observe that the loss of satisfaction from nodes C and D offsets the gain from the resolved blocking pair, A,B:

$$S = S_A + S_B + S_C + S_D = 0.5 + 0.5 + 1 + 1 = 3$$
$$S' = S'_A + S'_B + S'_C + S'_D = 1 + 1 + 0 + 0 = 2$$

It follows that the selfish choices of nodes A and B resulted in a network with decreased total satisfaction. This behavior also suggests that total satisfaction maximization may be preferable to traditional stability in overlay collaborative applications, where nodes pool together resources towards a common goal,*i.e.*, construct an overlay network, and social characteristics are inherent. Furthermore, in the above example, we showed that it is possible to achieve equal or greater satisfaction to the one achieved by the stable configuration, whose existence cannot be always guaranteed. This is in contrast to the maximizing total satisfaction criterion, which can always be applied, becoming a natural and sensible target goal to strive for in an overlay collaborative environment. Note here that the maximizing total satisfaction criterion need not be applied necessarily to the whole network: any amount of nodes, working closely within the network, can apply it and benefit from it as a group.

3.3. Optimizing through the Use of Satisfaction Increase

In the following sections, we will make frequent use of the *satisfaction increase* ΔS_i^j of node i (where $S_i = \sum_{j \in C_i} \Delta S_i^j$), due to its choosing of node j as its $(Q_i(j)+1)$-th highest ranked connection, which can be derived from Equation (1) by considering only the contribution (and possible penalty) of node j in node i's satisfaction, where $Q_i(j) \leq b_i$:

$$\begin{aligned} \Delta S_i^j &= \frac{1}{b_i} - \frac{R_i(j) - Q_i(j)}{b_i L_i} = \\ &= \left(\frac{1 - R_i(j)/L_i}{b_i} \right) + \left(\frac{Q_i(j)/b_i}{L_i} \right) = S_{i,j}^s + S_{i,j}^d \end{aligned} \tag{4}$$

Obviously, we can rewrite Equation (1) as:

$$S_i = \sum_{j \in C_i} \Delta S_i^j = \sum_{j \in C_i} S_{i,j}^s + \sum_{j \in C_i} S_{i,j}^d = S_i^s + S_i^d \tag{5}$$

It is easy to see that ΔS_i^j breaks down into two parts $S_{i,j}^s$, $S_{i,j}^d$, which we call the *a priori* part and the *a posteriori* part, respectively, since the former is computable for each j regardless of whether it is currently matched with i or not, while the latter is computable only for those j that are currently matched with i by the solution. Alternatively, we will refer to these parts as *static* and *dynamic*, respectively. The static part depends only on the rank of node j in preference list L_i, but the dynamic part depends on the rank of node j among node i's chosen connections at any given time. This rank among the nodes connected to i may depend on various factors in an execution, possibly also choices of nodes that are relatively remote to node i. Knowing that even simple matching is not a locally solvable problem [13], this is no surprise. Note that an algorithm that makes connection decisions based on the static part of ΔS_i^j has all the necessary information available from the beginning of its execution (*i.e.*, does not use any runtime information), and therefore, a node's valuation of other nodes (that depends only on that

static part) does not change over time, due to its runtime connection decisions. As a result, oscillations, due to cyclic preferences are avoided.

In the next section, we prove that it is possible to define a variation of the maximizing satisfaction b-matching problem (based on a modified definition of ΔS_i^j) that approximates the original problem and can lead to a simple greedy algorithm.

4. Approximating B-Matchings with Weighted Matchings

In this section, we show how a simple modification connects the maximizing satisfaction b-matching problem with well known optimization problems, such as the maximum weighted matching.

4.1. Discarding the A Posteriori *Term*

As the first step to approximate the maximizing satisfaction b-matching problem, we define a modified problem based on the same basic b-matching problem, but computing satisfaction using a modified version of Equation (4) (and subsequently Equation (1)):

$$\Delta\overline{S}_i^j = S_{i,j}^s = \frac{1}{b_i} - \frac{R_i(j)}{b_i L_i}, \tag{6}$$

$$\overline{S}_i = S_i^s = \frac{c_i}{b_i} - \frac{\sum_{j\in C_i} R_i(j)}{b_i L_i}, c_i \leq b_i \tag{7}$$

Using the above definition, we effectively disregard the dynamic part of Equation (4), making the prospective satisfaction increase ΔS_i^j of node i independent of the number of connections. We refer to the node satisfaction defined by Equation (7) as *truncated satisfaction* and the corresponding maximizing satisfaction b-matching problem as a *truncatedS maximizing satisfaction b-matching problem*. In the following lemma, we prove that by solving the truncatedS maximizing satisfaction b-matching problem, we get an approximation of the original problem.

Lemma 1 *The truncatedS maximizing satisfaction b-matching problem is a* $\frac{1}{2}\left(1 + b_{\max}^{-1}\left(1 + \frac{1-b_{\max}^{-1}}{2s+b_{\max}^{-1}}\right)\right)$*-approximation of the maximizing satisfaction b-matching problem, where* $b_{\max}$ *is the maximum connection quota in the graph and* $s = \frac{L_{\max}}{L_{\min}}$ *is the ratio of the maximum and minimum neighbor list sizes in the graph,* $L_{\max}$ *and* $L_{\min}$*, respectively.*

Proof: Since only S_i^s is used in the modified maximizing satisfaction b-matching problem, we are interested in studying the ratio:

$$r_i = \frac{S_i^s}{S_i^s + S_i^d} \tag{8}$$

and, specifically, its minimum value, $r_{\min}$. Knowing this minimum value, we can conclude that the total satisfaction of the modified problem, S_{mod}, is a $r_{\min}$-approximation of the total satisfaction of original b-matching problem S_{orig}, since:

$$S_{mod} = \sum_{i=1}^{n} S_i^s = \sum_{i=1}^{n} r_i\left(S_i^s + S_i^d\right) \geq r_{\min}\sum_{i=1}^{n}\left(S_i^s + S_i^d\right) = r_{\min} S_{orig} \tag{9}$$

where $n = |V|$ is the number of nodes in the graph. Note that we are looking for the minimum value $r_{\min}$, and therefore, we can assume that a given node i has connected with the desired amount of neighbors b_i since then the difference between S_i^s and S_i^d is maximized, as we show below.

Initially, consider the scenario in which every node i is connected with its bottom-most b_i neighbors. It is easy to see that S_i^d gets its maximum value when the length of the list is maximized ($|C_i| = b_i$), since the dynamic part of the satisfaction increase ΔS_i^j is due to the connection list C_i of node i. In this case of $|C_i| = b_i$ connections, S_i^s achieves its minimum value when the connections $j \in C_i$ are drawn from the bottom of the preference list L_i. Since both of these conditions are met in the scenario considered here, the relative value of S_i^d is maximized or, equivalently, the relative value of S_i^s is minimized and the sums of dynamic and static parts, S_i^d and S_i^s, acquire the values of s_i^d and s_i^s, respectively. For these values s_i^s and s_i^d, we have:

$$\begin{aligned} s_i^s &= \frac{1-\frac{(L_i-b_i)}{L_i}}{b_i} + \frac{1-\frac{(L_i-b_i+1)}{L_i}}{b_i} + \ldots + \frac{1-\frac{(L_i-1)}{L_i}}{b_i} = \frac{b_i+1}{2L_i} \\ s_i^d &= \frac{0}{b_i L_i} + \frac{1}{b_i L_i} + \ldots + \frac{b_i-1}{b_i L_i} = \frac{b_i-1}{2L_i} \end{aligned}$$

meaning that the relative ratio of the sum of the static parts S_i^s is at least:

$$\frac{S_i^s}{S_i^s + S_i^d} \geq \frac{s_i^s}{s_i^s + s_i^d} = \frac{\frac{b_i+1}{2L_i}}{\frac{b_i}{L_i}} = \frac{1}{2}\left(1 + \frac{1}{b_i}\right) \tag{10}$$

This is clearly the worst possible case for all nodes and a lower bound for the $\frac{S_{\text{mod}}}{S_{orig}}$ ratio, since:

$$S_{\text{mod}} = \sum_{i=1}^{n} S_i^s = \sum_{i=1}^{n} \frac{b_i+1}{2L_i} \tag{11}$$

$$S_{orig} = \sum_{i=1}^{n} \left(S_i^s + S_i^d\right) = \sum_{i=1}^{n} \frac{b_i}{L_i} \tag{12}$$

$$S_{\text{mod}} \geq \frac{1}{2}\left(1 + \frac{1}{b_{\max}}\right) S_{orig}. \tag{13}$$

However, note that each node i that connects with its b_i bottom-most neighbors must have been rejected by its $L_i - b_i > 0$ top-most ones. This means that these neighbors j cannot connect to their b_j bottom-most neighbors, since in the worst case the last place must be occupied by node i. In this case, the worst possible scenario connects them to b_j neighbors starting at the previous to last position and moving without gaps towards more preferable neighbors. We write B for the set of nodes that connect to their bottom-most neighbors, and we call these nodes *bottom-choosers*. Note here that we discuss a worst case scenario and therefore we do not include nodes with $L_i - b_i = 0$ in set B; in such a case the b_i bottom-most neighbors are the same as the b_i top-most ones, which is the best possible outcome for node i.

Since both bottom-choosers and non-bottom-choosers are connected to their desirable number of neighbors, but the former are connected to their bottom-most neighbors, while the later are connected to their bottom-most-but-one neighbors, for each node $bc_i \in B$ we have:

$$S^s_{bc_i} = \frac{1-\frac{L_{bc_i}-b_{bc_i}}{L_{bc_i}}}{b_{bc_i}} + \ldots + \frac{1-\frac{L_{bc_i}-1}{L_{bc_i}}}{b_{bc_i}} = \frac{b_{bc_i}+1}{2L_{bc_i}} \tag{14}$$

$$S^d_{bc_i} = \frac{0}{b_{bc_i}L_{bc_i}} + \ldots + \frac{b_{bc_i}-1}{b_{bc_i}L_{bc_i}} = \frac{b_{bc_i}-1}{2L_{bc_i}} \tag{15}$$

while for each node $nb_i \in V/B$ we have:

$$S^s_{nb_i} = \frac{1-\frac{L_{nb_i}-b_{nb_i}-1}{L_{nb_i}}}{b_{nb_i}} + \ldots + \frac{1-\frac{L_{nb_i}-2}{L_{nb_i}}}{b_{nb_i}} = \frac{b_{nb_i}+3}{2L_{nb_i}} \tag{16}$$

$$S^d_{nb_i} = \frac{0}{b_{nb_i}L_{nb_i}} + \ldots + \frac{b_{nb_i}-1}{b_{nb_i}L_{nb_i}} = \frac{b_{nb_i}-1}{2L_{nb_i}} \tag{17}$$

Using Equations (11) to (17) we get:

$$\frac{S'_{\text{mod}}}{S'_{orig}} = \frac{\sum_{i=1}^{n} S^s_i + \sum_{i\in V/B}\frac{1}{L_i}}{\sum_{i=1}^{n}\left(S^s_i + S^d_i\right) + \sum_{i\in V/B}\frac{1}{L_i}} = \frac{S_{\text{mod}} + \sum_{i\in V/B}\frac{1}{L_i}}{S_{orig} + \sum_{i\in V/B}\frac{1}{L_i}} \tag{18}$$

For reasons of clarity, in the remaining proof we are going to refer to quantities $\sum_{i\in V/B}\frac{1}{L_i}$ and $\sum_{i\in V/B}\frac{1}{L_i}\Big/ S_{orig}$ with Q and q, respectively, as well as $q_{\min}$ for the minimum value of the latter. Using the above notation and Equations (13) and (18), we get:

$$\frac{S'_{\text{mod}}}{S'_{orig}} \geq \frac{\frac{1}{2}\left(1+\frac{1}{b_{\max}}\right)S_{orig}+Q}{S_{orig}+Q} \geq \frac{\frac{1}{2}\left(1+\frac{1}{b_{\max}}\right)S_{orig}+q_{\min}\cdot S_{orig}}{S_{orig}+q_{\min}\cdot S_{orig}} = \frac{\frac{1}{2}\left(1+\frac{1}{b_{\max}}\right)+q_{\min}}{1+q_{\min}} \tag{19}$$

In order to calculate $q_{\min}$, we bound the sum $\sum_{i\in V/B}\frac{1}{L_i}$ as follows:

$$\sum_{i\in V/B}\frac{1}{L_i} \geq |V/B|\frac{1}{L_{\max}} \geq \min\left(|V/B|\right)\frac{1}{L_{\max}} = \left(n-\max|B|\right)\frac{1}{L_{\max}} \tag{20}$$

where $L_{\max}$ is the maximum neighbor list size in the graph. For $|B|$, we observe that each bottom-chooser corresponds to $L_i - b_i$ non-bottom-choosers (its $L_i - b_i$ top-most neighbors), and summing up for the whole graph, we have:

$$n = |B| + \sum_{i\in B} L_i - b_i \geq |B| + |B|\min_{i\in B}\left(L_i - b_i\right) = |B| + |B| \Rightarrow |B| \leq \frac{n}{2} \tag{21}$$

since $\min_{i\in B}\left(L_i - b_i\right) = 1$.

For every node i we define $L_i - b_i = c_i L_i$, where c_i is the percentage of list L_i that is covered by $L_i - b_i$. By the same definition, we have $\frac{b_i}{L_i} = 1 - c_i$. By Equations (20) and (21) and the above definition, we get:

$$q = \frac{\sum_{i\in V/B}\frac{1}{L_i}}{S_{orig}} = \frac{\sum_{i\in V/B}\frac{1}{L_i}}{\sum_{i=1}^{n}\frac{b_i}{L_i}} = \frac{\sum_{i\in V/B}\frac{1}{L_i}}{\sum_{i=1}^{n}\left(1-c_i\right)} \geq \frac{\left(n-\frac{n}{2}\right)\frac{1}{L_{\max}}}{n\left(1-c_{\min}\right)} = \frac{1}{2L_{\max}\left(1-c_{\min}\right)} \tag{22}$$

From Equations (19) and (22) and using the observation that $1 - c_{\min} = \max \frac{b_i}{L_i} \leq \frac{b_{\max}}{L_{\min}}$, we get:

$$\begin{aligned}
\frac{S'_{\text{mod}}}{S'_{orig}} &\geq \frac{\left(\frac{b_{\max}+1}{2b_{\max}}\right) + \frac{1}{2(1-c_{\min})L_{\max}}}{1 + \frac{1}{2(1-c_{\min})L_{\max}}} \\
&= \frac{(b_{\max}+1)(1-c_{\min})L_{\max} + b_{\max}}{2b_{\max}(1-c_{\min})L_{\max} + b_{\max}} \\
&= \frac{1}{2}\frac{(b_{\max}+1)(1-c_{\min})L_{\max} + b_{\max}}{b_{\max}(1-c_{\min})L_{\max} + \frac{1}{2}b_{\max}} \\
&= \frac{1}{2}\left(1 + \frac{(1-c_{\min})L_{\max} + \frac{1}{2}b_{\max}}{b_{\max}(1-c_{\min})L_{\max} + \frac{1}{2}b_{\max}}\right) \\
&= \frac{1}{2}\left(1 + \frac{1}{b_{\max}}\frac{(1-c_{\min})L_{\max} + \frac{1}{2}b_{\max}}{(1-c_{\min})L_{\max} + \frac{1}{2}}\right) \\
&= \frac{1}{2}\left(1 + \frac{1}{b_{\max}}\left(1 + \frac{b_{\max}-1}{2(1-c_{\min})L_{\max}+1}\right)\right) \\
&\geq \frac{1}{2}\left(1 + \frac{1}{b_{\max}}\left(1 + \frac{b_{\max}-1}{2\frac{b_{\max}}{L_{\min}}L_{\max}+1}\right)\right) \\
&= \frac{1}{2}\left(1 + \frac{1}{b_{\max}}\left(1 + \frac{1-\frac{1}{b_{\max}}}{2\frac{L_{\max}}{L_{\min}} + \frac{1}{b_{\max}}}\right)\right) \\
&= \frac{1}{2}\left(1 + b_{\max}^{-1}\left(1 + \frac{1-b_{\max}^{-1}}{2s + b_{\max}^{-1}}\right)\right)
\end{aligned} \tag{23}$$

where $s = \frac{L_{\max}}{L_{\min}}$ is the ratio of the maximum and minimum neighbor list sizes in the graph, $L_{\max}$ and $L_{\min}$ respectively, which proves the desired approximation ratio. □

4.2. Converting to a Maximum Weighted Matching

The truncatedS b-matching problem, as defined above, assumes privately kept preference lists and cannot be considered a maximum weighted matching problem, which needs the weights associated with edges to be known and common to both endpoints. In order to convert it, we will create edge weights by adding the satisfaction gleaned by the two endpoints for the specific link. For an edge $e = (i,j) \in E$, the weight should be:

$$\begin{aligned}
w(i,j) &= \Delta\overline{S}_i^j + \Delta\overline{S}_j^i = S_{i,j}^s + S_{j,i}^s = \\
&= \left(\frac{1 - R_i(j)/L_i}{b_i}\right) + \left(\frac{1 - R_j(i)/L_j}{b_j}\right)
\end{aligned} \tag{24}$$

By using these weights to construct and solve a many-to-many weighted matching problem, we also get a solution for the truncatedS b-matching problem, as the following lemma suggests.

Lemma 2 *We consider the truncatedS b-matching problem that uses Equation (6) for satisfaction calculations. A solution derived from a many-to-many maximum weighted matching on the same graph, with edge weights given by Equation (24), is also a solution for the truncatedS b-matching problem, and vice versa.*

Proof: Let $A \subseteq E$ be the edge set that is the solution of a many-to-many maximum weighted matching on a graph $G(V, E)$ with edge weights defined by Equation (24). This set A corresponds to a collection C of connection lists $C_i \; \forall i \in V$, and maximizes the expression $\sum_{(i,j)\in A} w\,(i,j)$ to a value $w(A)$:

$$w\,(A) = \sum_{(i,j)\in A} w\,(i,j) = \sum_{(i,j)\in A} \left(\Delta\overline{S}_i^j + \Delta\overline{S}_j^i\right) \tag{25}$$

Let also $A^{'} \subseteq E$ be the corresponding edge set for the truncatedS b-matching problem (using Equation (6)) on the same graph. This set corresponds to a collection $C^{'}$ of connection lists $C_i^{'} \; \forall i \in V$, and maximizes the expression, $\sum_{i\in V} \sum_{j\in C_i^{'}} \Delta\overline{S}_i^j$, to a value $w(A^{'})$ where $\sum_{j\in C_i^{'}} \Delta\overline{S}_i^j$ is the satisfaction gleaned by node i for the connections of list $C_i^{'}$:

$$w\left(A^{'}\right) = \sum_{i\in V} \sum_{j\in C_i^{'}} \Delta\overline{S}_i^j \tag{26}$$

We will prove that these solutions $w\,(A)$ and $w(A^{'})$ are equal. Assume that $w(A) > w(A^{'})$. If we group the satisfaction increases $\Delta\overline{S}_i^j$ in Equation (25) for a node i, we can write:

$$w\,(A) = \sum_{(i,j)\in A} \left(\Delta\overline{S}_i^j + \Delta\overline{S}_j^i\right) = \sum_{i\in V} \sum_{j\in C_i} \Delta\overline{S}_i^j \tag{27}$$

By the assumption and Equation (27), we have:

$$w\,(A) > w\left(A^{'}\right) \Rightarrow \sum_{i\in V} \sum_{j\in C_i} \Delta\overline{S}_i^j > \sum_{i\in V} \sum_{j\in C_i^{'}} \Delta\overline{S}_i^j$$

implying that the collection C achieves a greater value than $C^{'}$ for the maximizing expression of the truncatedS b-matching problem. However, this contradicts the definition of $C^{'}$.

Symmetrically, assuming that $w(A) < w(A^{'})$ also leads to a contradiction, meaning that any solution for the many-to-many maximum weighted matching we are considering is also a solution for the corresponding truncatedS b-matching problem, and *vice versa*. □

The following theorem follows directly from Lemmas 1 and 2.

Theorem 1 *We consider the maximizing satisfaction b-matching problem that uses Equation (4) to maximize the total satisfaction. A solution derived from a many-to-many maximum weighted matching on the same graph, with edge weights given by Equation (24), is a* $\frac{1}{2}\left(1 + b_{\max}^{-1}\left(1 + \frac{1-b_{\max}^{-1}}{2s+b_{\max}^{-1}}\right)\right)$*-approximation of the b-matching problem, where* $b_{\max}$ *is the maximum connection quota in the graph and* $s = \frac{L_{\max}}{L_{\min}}$ *is the ratio of the maximum and minimum neighbor list sizes in the graph,* $L_{\max}$ *and* $L_{\min}$*, respectively.*

Having approximated the original b-matching problem by a many-to-many maximum weighted matching, we need only a distributed algorithm for that problem. The rest of this paper presents a simple, distributed algorithm in two variations, an adaptive and a non-adaptive, that solve the many-to-many maximum weighted matching problem using a $\frac{1}{2}$-approximation guarantee. We also present and utilize a centralized helper algorithm that we employ in order to prove the approximation ratio of the distributed algorithm.

5. Distributed Algorithms for the Many-to-Many Maximum Weighted Matching Problem

In this section, we present a distributed algorithm, in adaptive and non-adaptive forms, that solves the many-to-many maximum weighted matching problem with an approximation ratio of $\frac{1}{2}$. The non-adaptive version of the algorithm is presented in Section 5.1, followed by the analysis of its correctness and convergence properties in Section 5.2. Sections 5.3 and 5.4 present the adaptive version and its analytical properties, respectively. Finally, Section 5.5 shows that the solutions of both variations are a $\frac{1}{2}$-approximation for the many-to-many maximum weighted matching problem.

5.1. LID *Algorithm*

The simple greedy algorithm we are proposing is fully distributed and operates by choosing the locally heaviest edges in every node's neighborhood, generalizing the one-to-one matching algorithm by Hoepman [8].

In the beginning, each node i reports to each neighboring node, j, only j's relative rank in i's preference list, in the form of $\Delta\overline{S}_i^j$, $\Delta\overline{S}_j^i$ exchange between i and j, and they both compute the weight $w(i,j)$ of their connecting edge. This ensures the flexibility of the ranking metric each node uses (it does not need to reveal the metric) at the cost of only local communication between nodes. Every node keeps these newly formed weights of its adjacent edges in a *weight list*, which is then used during the algorithm's execution to determine the desirability of a neighboring node. Note that these weight lists do not replace the individual preference lists of the nodes: they are auxiliary lists that are used in a similar way (to determine desirability), but only during the algorithm's execution, *i.e.*, not for measuring the final satisfaction.

The *Local Information-based Distributed* (LID) algorithm (*cf.* Algorithm 1 for pseudocode) uses, at each node i, four sets (U_i, P_i, A_i, K_i) and a function $(topRanked(\cdot))$ and sends two kinds of messages (PROP and REJ):

- A node i sends PROP messages to propose to its heaviest-weight neighbors the establishment of a connection with them. If an asked node also sends a PROP message to node i, then the connection is established (*locked* or *selected*); note that this will happen at both endpoints. Set P_i keeps the neighbors to which node i proposed with a PROP message; A_i keeps the neighbors which approached node i with a PROP message; K_i keeps the locked neighbors, and U_i the neighbors that did not send any message to node i or were not contacted yet. The algorithm terminates when $U_i = \emptyset$.
- A node sends a REJ message when it has locked as many neighbors as it could. When a node receives a REJ message, it sends a new PROP message to the next unproposed neighbor.
- The $topRanked(\cdot)$ function returns the top ranked node of its set argument, according to the weight list of the calling node. In Algorithm 1, this means that PROP messages are sent to neighbors in decreasing ranking order and there are at most b_i such unanswered messages originating from i at any time. A new PROP message is sent only if a previously asked node has explicitly declined.
- When the algorithm finishes in a node i the connected neighbors can be found in set K_i.

Algorithm 1 LID: Local Information-based Distributed algorithm for many-to-many maximum weighted matchings, run on node i

$K_i = \emptyset; A_i = \emptyset; P_i = \emptyset; U_i = \Gamma_i$
while $|P_i| < b_i$ **do**
  $P_i \leftarrow P_i \cup \texttt{topRanked}(U_i \backslash P_i)$
forall the $v \in P_i$ **do**
  send(PROP,v)
while $U_i \neq \emptyset$ **do**
  receive(m,u):
    if m = PROP **then**
      $A_i \leftarrow A_i \cup u$
    if m = REJ **then**
      $U_i \leftarrow U_i \backslash u$
      if $u \in P_i$ **then**
        $P_i \leftarrow P_i \backslash u$
        $v \leftarrow \texttt{topRanked}(U_i \backslash P_i)$
        $P_i \leftarrow P_i \cup v$
        send(PROP,v)
    if $\exists v \in (P_i \backslash K_i) \cap A_i$ **then**
      $U_i \leftarrow U_i \backslash v$
      $A_i \leftarrow A_i \backslash v$
      $K_i \leftarrow K_i \cup v$
    if $P_i \backslash K_i = \emptyset$ **then**
      forall the $v \in U_i$ **do**
        send(REJ,v)
      $U_i \leftarrow \emptyset$

5.2. Analysis of the LID *Algorithm*

At the center of the algorithm above is the notion of the locally heaviest edge [8,9], since the nodes send PROP and REJ messages in order to compare their heaviest edges and find the locally heaviest ones. Note here that we assume unique edge weights, since it is important for the greedy algorithms described here to be able to recognize the locally heaviest edges in an unambiguous way (ties can be broken using node identities). However, globally unique edge weights are not necessary: it is sufficient for edge weights to be unique in the local neighborhoods of their endpoint nodes.

By the definition in Section 2, at most, one locally heaviest edge can be attached to a node i at any specific point during the execution of the algorithm. However, once an edge is selected by node i, another one can possibly become locally heaviest in node i's neighborhood, and so on, until the algorithm selects enough of them. On the other hand, when nearby nodes fill in their quotas of possible connections, any

unselected edges they might have with node i become unavailable. In order to express this recursive property of locally heaviest edges, we continue to use condition Equation (3):

$$w(i,j) > w(e), e \in E_{ij}$$

but we adapt the definition of the set E_{ij} to include only edges of nodes i and j with unlocked neighbors k ($k \in U_{\{i,j\}} \backslash K_{\{i,j\}}$) that have not filled their connection quotas yet ($P_k \backslash K_k \neq \emptyset$):

$$\begin{aligned} E_{ij} = &\{(i,k) \,|\, (k \in U_i \backslash K_i) \wedge (P_k \backslash K_k \neq \emptyset) \wedge (k \neq j)\} \cup \\ &\{(j,k) \,|\, (k \in U_j \backslash K_j) \wedge (P_k \backslash K_k \neq \emptyset) \wedge (k \neq i)\} \end{aligned} \tag{28}$$

Note that for the initial conditions $K_i = P_i = \emptyset,\ U_i = \Gamma_i, \forall i \in V$, the definition above coincides with the one of Equation (2).

The following two lemmas address the dynamics arising from the aforementioned recursive definition by showing two important properties: the algorithm's execution at node i: (i) chooses only locally heaviest edges (although not in any particular order); and (ii) chooses edges in a way, such that any unselected locally heaviest edge adjacent to node i at the end of the algorithm's execution has lower absolute edge weight than the selected ones adjacent to the same node.

Lemma 3 *Every locked edge is locally heaviest at some point during the execution of the algorithm.*

Proof: As edge (i,j) is eventually locked, we assume without loss of generality that node i sent to node j a PROP message, inserted it in $P_i \backslash K_i$, later on received a confirmation from node j and eventually locked the edge. It is easy to see that sets $P_i \backslash K_i$ and $P_j \backslash K_j$ contain the heaviest available edges in the neighborhoods of node i and j, respectively; heaviest because Algorithm LID sends PROP messages to neighbors in decreasing edge weight order and available, since PROP messages were sent, but no reply came back yet, either positive or negative (a positive answer will move the answering node to K, and a negative answer will remove it from P). Since we only need to search for locally heaviest edges in sets $P_i \backslash K_i$ and $P_j \backslash K_j$, we can replace sets U_i, U_j with P_i, P_j respectively in Equation (28) and prove that the new condition holds at some point during the algorithm's execution:

$$\begin{aligned} E'_{ij} = &\{(i,k) \,|\, (k \in P_i \backslash K_i) \wedge (P_k \backslash K_k \neq \emptyset) \wedge (k \neq j)\} \cup \\ &\{(j,k) \,|\, (k \in P_j \backslash K_j) \wedge (P_k \backslash K_k \neq \emptyset) \wedge (k \neq i)\}, \\ &\text{and } w(i,j) > w(e), e \in E'_{ij} \end{aligned} \tag{29}$$

Assume that condition Equation (29) does not hold at time t_0, but edge (i,j) is selected (at that time t_0), and suppose a node k exists, such that $w(i,k) > w(i,j)$. We know that node i proposed to node k before proposing to node j, since PROP messages are being sent in decreasing ranking order. Node k will answer back at some point $t_1 > t_0$ during the execution of the algorithm, either positively or negatively, by which time it will be removed from set $P_i \backslash K_i$. At that point, condition Equation (29) will hold and edge (i,j) will be locally heaviest. The same is also true if a node l exists, such that $w(l,j) > w(i,j)$ or if both nodes k and l exist. □

Lemma 4 *For every node i Algorithm* LID *chooses all locally heaviest edges that are adjacent to it, if there is enough quota b_i available, or otherwise chooses the b_i of those that are heavier than any unchosen one.*

Proof: Based on Lemma 3, by the end of the algorithm, all chosen edges of node i have been locally heaviest. Furthermore, if there is enough quota b_i available, any unselected locally heaviest edges will be eventually selected. The only way some locally heaviest edge has been left unchosen is for node i to fill in its connection quota b_i with other locally heaviest edges. We just need to prove that in this case the unchosen edge is of lower weight than the chosen ones.

Note that the LID algorithm proposes to nodes of heavier edges first and proceeds only if it receives an explicit decline (meaning it was not a locally heaviest edge or the other node filled its quota), so the chosen edges are always heavier than any unchosen one. □

Calculating edge weights in the way previously described enables the conversion of the original maximizing satisfaction b-matching problem of Section 2 to a many-to-many maximum weighted matching problem, but as an added benefit it enables us to use the weight lists to make preference-based decisions, in a way similar to a b-matching problem. This new b-matching problem does not replace the original maximizing satisfaction b-matching: the nodes keep their original preference lists, but a new b-matching problem arises when they try to cooperate in order to collectively achieve a guaranteed level of connection quality. However, this new b-matching problem always converges regardless of the original problem, due to the symmetric nature of the edge weights [4]. The following lemma expresses exactly this property by showing that the algorithm terminates for all nodes.

Lemma 5 *Algorithm* LID *terminates for every node* $i \in V$.

Proof: From the code of Algorithm LID we can see that it terminates for node i when $U_i = \emptyset$ or $P_i \backslash K_i = \emptyset$, that is, when every neighbor replies (and node i gets less than b_i positive replies) or enough neighbors reply (for node i to get b_i positive replies), respectively. The only case where the algorithm would not terminate is if some node would wait indefinitely for a neighbor's answer, *i.e.*, if a communication cycle exists: each node $n_{i \bmod k}$ in a group of nodes, $\{n_0, n_1, \ldots, n_{k-1}\}$, sends a PROP message to node $n_{(i+1) \bmod k}$ and awaits for an answer in order to reply back to node $n_{(i-1) \bmod k}$. In order to prove that the algorithm terminates, we only need to prove that communication cycles cannot exist.

Assume there is a communication cycle $\{n_0, n_1, \ldots, n_{k-1}\}$. Since node $n_{i \bmod k}$ sent a PROP message to node $n_{(i+1) \bmod k}$ and not to node $n_{(i-1) \bmod k}$, we know that $w(n_{i \bmod k}, n_{(i+1) \bmod k}) > w(n_{i \bmod k}, n_{(i-1) \bmod k})$. If we add up the respective inequalities for all $i \in [0, k-1]$, we get:

$$\sum_{i=0}^{k-1} w\left(n_{i \bmod k}, n_{(i+1) \bmod k}\right) > \sum_{i=0}^{k-1} w\left(n_{i \bmod k}, n_{(i-1) \bmod k}\right) \tag{30}$$

Using properties of the modulo operator to change the sum limits and since edge weights are symmetric to their respective endpoints (see Equation (24)), we get the following:

$$\sum_{i=0}^{k-1} w\left(n_{i \bmod k}, n_{(i+1) \bmod k}\right) = \sum_{i=0}^{k-1} w\left(n_{(i+1) \bmod k}, n_{i \bmod k}\right) = \sum_{i=0}^{k-1} w\left(n_{i \bmod k}, n_{(i-1) \bmod k}\right) \tag{31}$$

which is a contradiction of Equation (32) and of the assumption on the existence of a communication cycle. □

5.3. ADAPTIVELID *Algorithm*

In a dynamic setting, where nodes join/leave the network or change preferences about their neighbors at any time, there is a partial or full solution that is disturbed by a specific operation. In this case, it is desirable to "repair" the solution locally instead of recomputing it globally. It would also be advantageous to limit the repairs to the neighborhood of the operation, so that far enough nodes would remain unaffected. Note that the locally-heaviest-edge property that we are using here seems ideal for this purpose: it only makes sense to preserve and use it further to support dynamicity.

In the adaptive algorithm, ADAPTIVELID presented here, all three cases of dynamicity mentioned above (join/leave/change) are supported. In the case of a joining (respectively departing) node, neighboring nodes add (respectively delete) it to (respectively from) their preference lists. On the other hand, when a node changes preferences, no change occurs to the neighboring nodes' preference lists, but edge weights may change radically. A common thread between these cases is that the nodes directly involved in the operations must recalculate their marginal satisfactions for their neighbors and exchange them so that their adjacent edges have the correct weights. Afterwards, they must re-evaluate their connections: if they are not locally heaviest any more, the nodes abandon the least weighted ones and try to get matched with the locally heaviest ones. Note here that this method avoids the recalculation of the solution over the whole network, instead limiting it to a neighborhood around the network area where the dynamic operation took place. An additional benefit is that the involved nodes maintain their current connections unless proven to be non-optimal, *i.e.*, they change them only if necessary.

The ADAPTIVELID algorithm uses at each node i five sets $(P_i, K_i, A_i, R_i, B_i)$ and an incoming message queue, $queue_i$, and sends three kinds of messages (PROP, REJ and WAKE):

- A node i sends PROP messages to propose to its heaviest-weight neighbors the establishment of a connection. If an asked node also sends a PROP message to node i, then the connection is established (*locked*): note that this will happen at both endpoints. Set P_i stores the neighbors to which node i proposed with a PROP message; A_i stores the neighbors that approached node i with a PROP message; K_i stores the locked neighbors; B_i stores the neighbors that rejected node i, and R_i the neighbors that node i rejected. Sets B_i^* and A_i^* are copies of sets B_i and A_i, respectively, that do not contain neighbors of edges heavier than the edge of the worst connected neighbor.
- A node sends a REJ message when it has locked as many neighbors as it could. Nodes can send additional PROP messages to available neighbors if they receive a REJ message. PROP messages are sent to neighbors in decreasing ranking order, and there are at most b_i such unanswered messages originating from i at any time.
- Node i is constantly checking if its PROP messages are addressed to heaviest-weight neighbors, as ranking can change due to a change in the network. If it detects a better available node than the currently proposed ones, it sends a REJ message to the worst connected neighbor and a PROP to the better candidate. However, if the better candidate has simultaneously rejected and been rejected by node i, node i sends only a WAKE message.

Algorithm 2 ADAPTIVELID()

```
ReceiveMsgs()
SendMsgs()
BookkeepingUpdates()
```

Procedure 1 ReceiveMsgs()

for $msg \in queue_i$ **do**
  if $msg.type = PROP$ **then**
    $A_i \leftarrow A_i \cup msg.sender$
    $B_i \leftarrow B_i - msg.sender$
  if $msg.type = REJ$ **then**
    $B_i \leftarrow B_i \cup msg.sender$
    $A_i \leftarrow A_i - msg.sender$
    $K_i \leftarrow K_i - msg.sender$
    $P_i \leftarrow P_i - msg.sender$
  if $msg.type = WAKE$ **then**
    $B_i \leftarrow B_i - msg.sender$

Procedure 2 SendMsgs()

while $(|\Gamma_i - P_i - (B_i - R_i)| \neq 0) \wedge (|P_i| < b_i)$ **do**
  find heaviest edge neighbor c that belongs in $(\Gamma_i - P_i - (B_i - R_i))$
  if $c \neq null$ **then**
    if $c \in B_i$ **then**
      send a WAKE msg to c
      $R_i \leftarrow R_i - c$
    else
      send a PROP msg to c
      $P_i \leftarrow P_i \cup c$
      $R_i \leftarrow R_i - c$

Function 1 GetWorstNode(node i)

return $\left\{ l : w(l,i) = \min_{j \in P_i} w(j,i) \right\}$

Function 2 GetBestNode(node i)

return $\left\{ h : w(h,i) = \max_{j \in (\Gamma_i - P_i - (B_i - R_i))} w(j,i) \right\}$

Procedure 3 BookkeepingUpdates()

$T_i \leftarrow (P_i - K_i) \cap A_i$
if $|T_i| \neq 0$ **then**
  $A_i \leftarrow A_i - T_i$
  $K_i \leftarrow K_i \cup T_i$
  match node i to all nodes in T_i
if $(|\Gamma_i - P_i - (B_i - R_i)| \neq 0) \wedge (|P_i| \neq 0) \wedge (|K_i| \neq 0)$ **then**
  $l \leftarrow$ GetWorstNode(i)
  $h \leftarrow$ GetBestNode(i)
  while $(l \neq null) \wedge (h \neq null)$ **do**
    if $w(h,i) > w(l,i)$ **then**
      if $h \in B_i$ **then**
        send a WAKE msg to h
        $R_i \leftarrow R_i - h$
      else
        send a REJ msg to l
        $A_i \leftarrow A_i - l$
        $R_i \leftarrow R_i \cup l$
        $P_i \leftarrow P_i - l$
        $K_i \leftarrow K_i - l$
        send a PROP msg to h
        $P_i \leftarrow P_i \cup h$
        $R_i \leftarrow R_i - h$
      $l \leftarrow$ GetWorstNode(i)
      $h \leftarrow$ GetBestNode(i)
    else if $P_i = K_i$ **then**
      for $j \in (\Gamma_i - R_i - B_i^* + A_i^* - P_i)$ **do**
        send a REJ msg to j
        $A_i \leftarrow A_i - j$
        $R_i \leftarrow R_i \cup j$
      break
    else
      break
unmatch node i from all nodes in B_i

5.4. Analysis of the ADAPTIVELID *Algorithm*

The following lemmas prove that the algorithm always converges after a finite amount of steps or, in the case of changes in the network, in a finite amount of steps after the changes stop. Although implied by the distributed nature of the algorithm, it is useful to note that the algorithm continues to run at all nodes regardless of any changes that are happening in the network. In fact, as we show in the experimental section, it manages to maintain a reduced but steady level of service while under extremely heavy stress or possibly a network attack. However, convergence can be guaranteed after all changes complete, since any changes that might occur require appropriate readjustment by the distributed algorithm.

Lemma 6 *In a failure-free execution, edge weight updates that are caused by node or preference changes complete in a finite amount of time.*

Proof: When a node joins (leaves) the network, it gets inserted to (deleted from) neighboring nodes' preference lists, causing changes that need to be communicated to their own neighbors. The same happens to the node itself when it changes its own preference list. Therefore, every change causes a weight update that propagates at a maximum distance of two and to a bounded amount of nodes (bounded by the size of a distance of two neighborhood from the originating node). Note that the neighbor's neighbors (or the immediate neighbors in the case of simple preference change) accept the weight update passively and do not propagate it further. Since we assumed that nodes do not fail and messages do not get lost, it is evident that all nodes are fully updated in a finite amount of time after the change. □

We define as *available*, with respect to node i, a node j in the neighborhood of node i that has neither been proposed by node i nor rejected node i.

A node j is the *locally heaviest node* in the neighborhood of node i at some point in time if there are no available nodes adjacent to node i that are endpoints of heavier edges. Note that edges with such a node as an endpoint are candidates for being locally heaviest edges (hence, the name "locally heaviest node"). In fact, when the endpoints of an edge consider simultaneously each other locally heaviest, the edge between them is the locally heaviest edge.

Lemma 7 *In a finite amount of time after a node or preference change, every node cancels all proposals towards neighbors that are no longer locally heaviest and issues an equal amount towards available neighbors that are locally heaviest.*

Proof: Every change triggers weight updates at a distance of one (nodes that the joining/leaving node connect to/disconnect from or direct neighbors of a node that changes preferences) and possibly at a distance of two (neighbor's neighbors of a joining/leaving node). By Lemma 6, the weight updates complete in a finite amount of time. When procedure 3 executes next in any node i at a distance of one or two from the change, it will repeatedly examine nodes that are either available or have simultaneously rejected and been rejected by node i, as long as they are heavier than the proposed node of the lowest weight. Every time it encounters a node of the latter category, it sends a WAKE message, prompting the node to revoke its rejection, whereas every time it encounters a node of the former category, it sends a PROP message, preceded by a REJ to the proposed node of the lowest weight (note that whether a

node is considered locally heaviest or not may change during a single execution of procedure 3.). In any case, at the end of procedure 3's execution, all proposals from node i to its neighbors that are no longer locally heaviest are canceled, and complementary proposals are sent to available neighbors that are locally heaviest; the same is true for every node in the network. □

Lemma 8 *The* ADAPTIVELID *algorithm terminates for every node $i \in V$ after changes complete.*

Proof: For the static case, Lemma 5 applies, and the algorithm terminates. For the dynamic case, a change may cause some proposals to be canceled and reissued on nodes at a distance of one or two (Lemma 7). The only cases in which the algorithm may not terminate on these nodes is when they wait indefinitely for a neighbor's answer or their preference list "oscillates", with proposals being canceled and reissued on the same neighbors in an alternating way over time. By Lemma 6, we can ignore weight updates, since they complete in a finite amount of time.

For the first case, a node can wait indefinitely only if a communication cycle exists: each node $n_{i \bmod k}$ in a group of nodes, $\{n_0, n_1, \ldots, n_{k-1}\}$, sends a PROP message to node $n_{(i+1) \bmod k}$ and awaits for an answer in order to reply back to node $n_{(i-1) \bmod k}$, that is $w(n_{i \bmod k}, n_{(i+1) \bmod k}) > w(n_{i \bmod k}, n_{(i-1) \bmod k})$. By adding all such equations on the cycle and using properties of the modulo operator, we get:

$$\sum_{i=0}^{k-1} w\left(n_{i \bmod k}, n_{(i+1) \bmod k}\right) > \sum_{i=0}^{k-1} w\left(n_{i \bmod k}, n_{(i-1) \bmod k}\right) = \\ = \sum_{i=0}^{k-1} w\left(n_{(i+1) \bmod k}, n_{i \bmod k}\right) = \sum_{i=0}^{k-1} w\left(n_{i \bmod k}, n_{(i+1) \bmod k}\right) \quad (32)$$

which is a contradiction.

For the second case, by Lemma 7 and since we assumed that the changes are completed, we have that any potential canceling and reissuing of proposals finishes in a finite amount of time and therefore no oscillations occur. □

Lemma 9 *For every node i Algorithm* ADAPTIVELID *chooses all locally heaviest edges that are adjacent to it, if there is enough quota b_i available, or otherwise chooses the b_i of those that are heavier than any unchosen one.*

Proof: For a static network, Lemma 4 is applicable. For a dynamic network, by Lemma 7, after a finite amount of time, every node i cancels all proposals to neighbors that are no longer locally heaviest after a change and proposes to the locally heaviest ones. Some of these proposals will result in a match if the receiving node also considers the originating node locally heaviest. The same will happen to all proposed nodes, as long as the originating and receiving nodes have available quotas. If some node lacks in available quota, we know that its matched incident edges are heavier than its unmatched locally heaviest, since by Lemma 7, it would have to cancel the appropriate proposals and issue new ones towards locally heaviest neighbors. □

Lemma 10 *The* ADAPTIVELID *algorithm, when run on a network with changes, produces the same matching with the* LID *algorithm that is run on the same network after the changes complete.*

Proof: By Lemma 8, we get that the ADAPTIVELID algorithm terminates for every node, and by Lemma 9, we know that at termination, it has chosen only the locally heaviest edges of maximum weight. Since by Lemma 4 the static algorithm also selects the locally heaviest edges of maximum weight at termination, it follows that the two algorithms make the same choices for the same networks. □

5.5. Approximation Ratio for the Distributed Algorithms

Following the methodology in [8], in order to further analyze the distributed algorithm in both variations, we present a centralized algorithm for many-to-many maximum weighted matchings, for which we show that it behaves in the same way with the distributed algorithm and both have the same approximation ratio of $\frac{1}{2}$.

Algorithm 3 LIC: Local Information-based Centralized algorithm for many-to-many maximum weighted matchings

$M \leftarrow \emptyset; P \leftarrow E$
forall the $v \in V$ **do**
  $counter(v) \leftarrow d_v$
while $P \neq \emptyset$ **do**
  `take a locally heaviest edge` $(a, b) \in P$
  $M \leftarrow M \cup (a, b)$
  $P \leftarrow P \backslash (a, b)$
  $counter(a) \leftarrow counter(a) - 1$
  $counter(b) \leftarrow counter(b) - 1$
  if $counter(a) = 0$ **then**
    $P \leftarrow P \backslash \{(a, n_a) \mid n_a \in \Gamma_a\}$
  if $counter(b) = 0$ **then**
    $P \leftarrow P \backslash \{(b, n_b) \mid n_b \in \Gamma_b\}$

Algorithm LIC is a simple greedy algorithm with the distinctive feature of using only locally available information, by selecting locally heaviest edges in a centralized way. Note that the comment of Section 5 about the recursive nature of the locally heaviest edges is still valid here: by systematically removing from the edge pool P the edges we select (line 6, Algorithm 3), along with any unselected edges of nodes with filled quotas (lines 8 and 9, Algorithm 3), we get the same dynamics as in the distributed case (*cf.* Lemma 11). In the following theorem, using a similar proof strategy to the one used by Preis [9] for his centralized one-to-one weighted matching algorithm, we prove that it achieves a $\frac{1}{2}$-approximation compared to the optimum algorithm (OPT) that selects edges with maximum weights over the whole graph.

Theorem 2 *The* LIC *algorithm produces a many-to-many maximum weighted matching M_C that is a $\frac{1}{2}$-approximation of the matching M_{OPT} produced by the optimal algorithm, OPT.*

Proof: Let V_C be the set of nodes matched at least once by the LIC algorithm. We will show that the condition:

$$w\left(M_C\right) \geq \frac{1}{2} w\left(\{(u,v) \in M_{OPT} | u \in V_C \vee v \in V_C\}\right) \tag{33}$$

holds for every step of the LIC algorithm, that is, the weight of matching M_C, $w(M_C)$, is at least $\frac{1}{2}$ of the weight $w(M_{OPT})$ of the optimal matching M_{OPT} when including only edges adjacent to nodes matched by LIC. Note that the algorithm can leave two or more nodes (completely) unmatched, but only if they are not connected in $G(V,E)$ (if they are connected, they will be matched, since the possibility exists and they have available connection quotas, leaving possibly one odd node unmatched). By the time LIC terminates, for every $(u,v) \in E$, it would be either $u \in V_C$ or $v \in V_C$, and Equation (33) will become $w\left(M_C\right) \geq \frac{1}{2} w\left(M_{OPT}\right)$.

For the initial condition $M_C = \emptyset$ Equation (33) holds. Let us assume that at some step of LIC, edge (a,b) is included in the matching M_C and the left-hand side of Equation (33) increases by $w(a,b)$. For the right-hand side, there are only two options: either (a,b) is part of the optimal matching, M_{OPT}, or two other edges, (a,c) and (b,d), occupy the respective slots of nodes a and b in the optimal matching. For these cases, we have the following:

- $(a,b) \in M_{OPT}$: The total weight of the right-hand side is increased by $\frac{1}{2} w(a,b)$ and Equation (33) holds.
- (a,c), $(b,d) \in M_{OPT}$: If both $c,d \in V_C$, then both edges are already taken into account in the right-hand side's weight sum, and Equation (33) holds. If one or both c and d are not included in V_C, then the total weight of the right-hand side is increased by $\frac{1}{2} w(a,c)$, $\frac{1}{2} w(b,d)$ or $\frac{1}{2}\left(w(a,c) + w(b,d)\right)$ by the addition of (a,c),(b,d) or both, respectively. However, edge (a,b) was selected as being locally heaviest, so Equation (33) holds even in the worst case, since:

$$\left.\begin{array}{l} w\left(a,b\right) \geq w\left(a,c\right) \\ w\left(a,b\right) \geq w\left(b,d\right) \end{array}\right\} \Rightarrow w\left(a,b\right) \geq \frac{1}{2}\left(w\left(a,c\right) + w\left(b,d\right)\right)$$

□

The following lemma mirrors Lemma 4 and is the key lemma in proving the equivalence of both centralized and distributed algorithms in the subsequent theorem.

Lemma 11 *For every node i Algorithm* LIC *chooses all locally heaviest edges that are adjacent to it, if there is enough quota b_i available, or otherwise chooses the b_i of those that are heavier than any unchosen one.*

Proof: If there are less than b_i locally heaviest edges, algorithm LIC will choose all of them, since, by construction, it will continue to select them until there are no quota slots available at either endpoint. Otherwise, again by construction, it will choose the b_i heaviest of them for node i. □

Theorem 3 *The* LID *algorithm is a $\frac{1}{4}\left(1 + b_{\max}^{-1}\left(1 + \frac{1 - b_{\max}^{-1}}{2s + b_{\max}^{-1}}\right)\right)$-approximation algorithm for the maximizing satisfaction b-matching problem, where $b_{\max}$ is the maximum connection quota in the graph and $s = \frac{L_{\max}}{L_{\min}}$ is the ratio of the maximum and minimum neighbor list sizes in the graph, $L_{\max}$ and $L_{\min}$, respectively.*

Proof: By Lemmas 11 and 4, we know that algorithms LIC and LID choose the same edges for each node and therefore produce the same solution. This means that the LID algorithm is also a $\frac{1}{2}$-approximation algorithm for the many-to-many maximum weighted matching (as Theorem 2 suggests for the LIC algorithm). Furthermore, the many-to-many maximum weighted matching we solve, by theorem 1, is a $\frac{1}{2}\left(1+b_{\max}^{-1}\left(1+\frac{1-b_{\max}^{-1}}{2s+b_{\max}^{-1}}\right)\right)$-approximation of the corresponding b-matching problem. These two approximations combined give a $\frac{1}{4}\left(1+b_{\max}^{-1}\left(1+\frac{1-b_{\max}^{-1}}{2s+b_{\max}^{-1}}\right)\right)$-approximation for the original maximizing satisfaction b-matching, where $b_{\max}$ is the maximum connection quota in the graph. □ From the above theorem and Lemma 10, we also get the following theorem about the approximation ratio of ADAPTIVELID:

Theorem 4 *The* ADAPTIVELID *algorithm solves the adaptive b-matching with preferences problem with* $\frac{1}{4}\left(1+b_{\max}^{-1}\left(1+\frac{1-b_{\max}^{-1}}{2s+b_{\max}^{-1}}\right)\right)$*-approximation.*

6. Experimental Study

The following extensive experimental study complements the preceding analytical part with useful observations and conclusions about the behavior of the LID and ADAPTIVELID algorithms in a variety of scenarios. Focus was given on the performance of the algorithms in regard to the following points:

- Convergence and reconvergence times
- Satisfaction levels, both in normal operation and under heavy stress (*i.e.*, during a network-level attack)
- Fairness properties of satisfaction-based optimization
- Behavior on different types of networks
- Behavior during different operations (joins, leaves, preference changes and churn)

6.1. Network Types

The networks used during the experiments were power-law and random networks, created with the Barabási–Albert (*BA*) [14] and Erdős–Rényi (*ER*) [15] procedures, respectively, having mean degree and initial (joining) degree of $0.05n$ and $0.05n$, respectively (where n is the network size). These networks were selected for their different node degree distributions: in power-law networks, the vast majority of nodes has a very low degree, and few nodes have a very high degree (power-law distribution), while in ER random networks, all nodes have comparable degrees (binomial distribution). In fact, high degree nodes in BA networks are connected mostly with many low degree ones, which leads to the creation of very different neighborhoods around individual nodes for these two network types. This difference, coupled with the ADAPTIVELID algorithm's ability to perform local repairing operations, leads to the varying behaviors that can be seen in the experiments below.

On the other hand, nodes of both network types had full preference lists (consisting of all their neighbors), but ranked uniformly at random and with a desired number of connections equal to half their own degree. Focus was given on random preference lists, since previous research [4,5] showed that: (a) a strict matching solution cannot always be found when they are used; and (b) the measured satisfaction

of unconverged instances can be relatively low. These characteristics make random preferences a challenging test case to evaluate the performance of the algorithms.

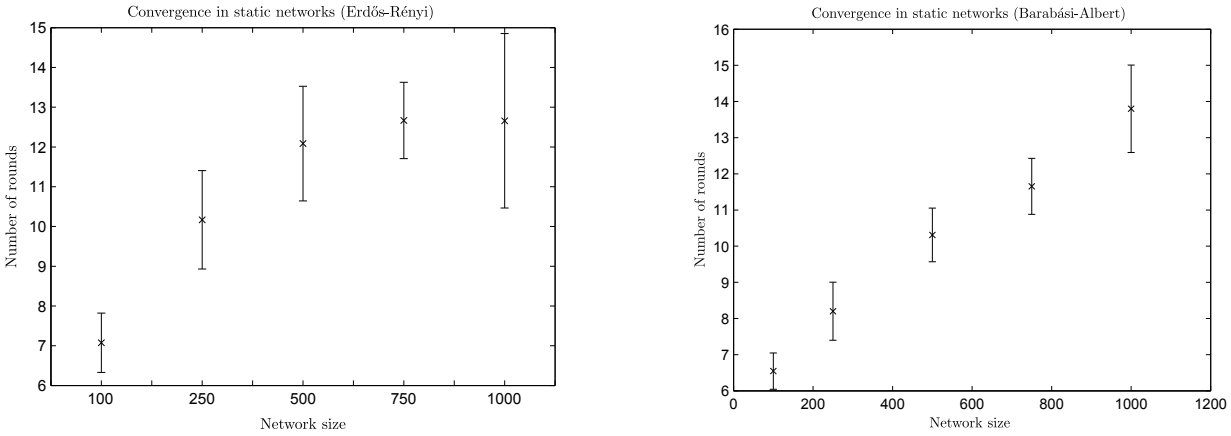
6.2. Experimental Procedure

The following experiments were conducted using the PeerSim [16] platform in a synchronous way, *i.e.*, execution proceeded in rounds, where each node in each round made a receive-respond-process step, unless it had nothing to execute. This synchronous execution mode is not necessary for the algorithm, but it is used here to measure the time needed by both algorithms to converge. For each of the following experiments, networks of size $n = 100, 250, 500, 750$ and $1,000$ nodes were used, considering 30 network instances for each size, and the results presented here are the mean values over these instances. For every network instance, a matching was calculated and, in the case of the AdaptiveLID algorithm, the following operations were performed on a varying amount of nodes (1% to 50% of the network size, in increments of 1%): *join/leave*, where nodes enter/exit the network simultaneously; *preference change*, where existing nodes change the ranking of their neighbors in their preference lists simultaneously; and *churn*, where existing nodes exit and an equal amount of new nodes enter the network simultaneously. For the first three cases, the network was left to reconverge after one operation, while in the case of churn, the operation was repeated for several rounds before the network was left to reconverge.

6.3. Convergence and Reconvergence

The mean value and standard deviation of convergence speed for a variety of network sizes can be found in Figure 3. It is easy to see that the convergence speed depends on the type of the network. For example, BA networks of a size of 1,000 take almost twice the amount of time to converge than networks of a size of 100, while ER networks of a size of 1,000 need less than twice the amount of time needed by networks of a size of 100 and only a slightly higher amount of time than the networks of a size of 500 and 750.

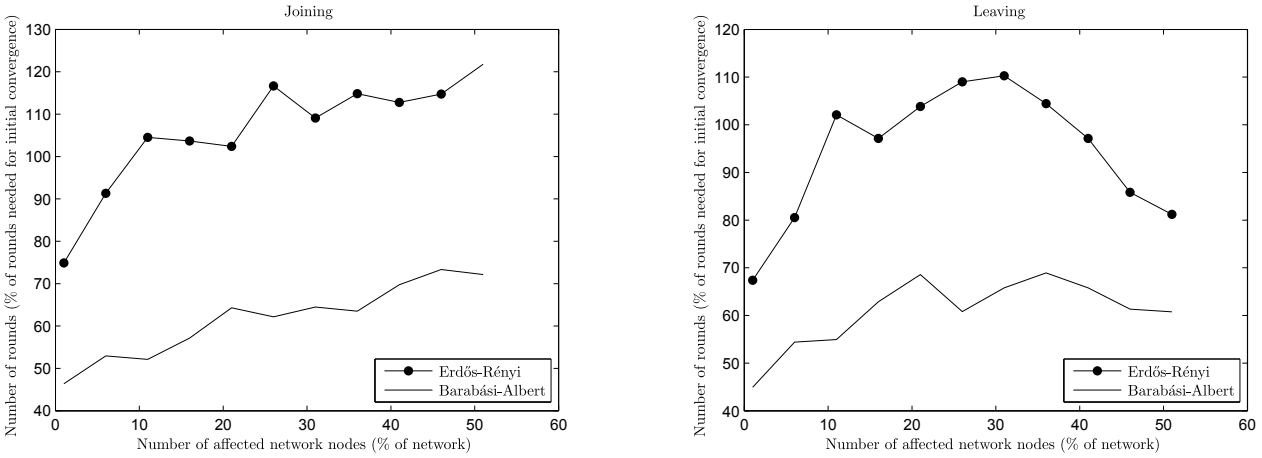
Figure 3. Convergence speed per network size.



Likewise, the time needed for reconvergence can be seen in Figures 4 and 5, for networks of a size of 1,000 of both types and for the four types of operations under consideration. By focusing on low percentages of affected nodes (*i.e.*, up to 20% of the network size, which is a high volume of change), it

is easy to see that reconvergence is obtained in most cases for a *fraction* of the rounds needed for initial convergence. For join and leave operations, reconvergence is expressed not in rounds, but in relation to the convergence time, since network sizes change significantly. Note here that this extreme change in network sizes leads in some cases to percentages greater than 100%, *i.e.*, more rounds are needed for the reconvergence than for the initial convergence. For the preference change and churn operations, this is not the case: the network size remains the same, either because no node joins or leaves (preference change case) or the amount of nodes joining and leaving is the same (churn case), and the reconvergence time is expressed in rounds.

Figure 4. Reconvergence speed per operation, joins/leaves, $n = 1,000$.



In the case of join operations, nodes arrive at the network and want to join the already established equilibrium of connections by being more attractive choices for some of the nodes with whom they are neighbors. This creates cascade effects of nodes rejecting old connections in favor of the newcomers, the rejected nodes trying to repair their lost connections, and so on. Naturally, the more nodes wanting to join the network, the bigger upheaval is created. A similar effect is generated during leave operations, where previously rejected nodes suddenly become attractive choices for nodes that were left behind by departing nodes. Note here that the BA networks reconverge much faster than the corresponding ER networks in the case of join operations. This happens because new nodes (being of a low degree) connect preferentially to relatively few high degree nodes, limiting the extension of the upheaval in the network. In the case of leave operations, the same behavior poses a challenge, since departing nodes may happen to be of a high degree themselves, leaving behind a lot of low degree nodes to repair their connections. Notice though that in both cases (ER or BA networks), when a substantial percentage of the network departs (*i.e.*, above 35%), the remaining nodes repair their connections much more easily, since they have more unformed connections than established ones.

Preference change affects both network types in the same way: a node that changes preferences destroys some connections, creating waves of changes in its neighborhood. For the case of BA networks, it may happen that a node changing preferences is a high degree one, causing a lot of nodes to repair their connections. However, this effect dies off quickly, since most of its neighbors are of a low degree, leading to an overall performance similar to the ER case.

The two network types show their differences more prominently under churn (Figure 5). For the ER networks, a joining node under churn can be seen as a "reincarnation" of a leaving node with all its previous connections dropped and its preferences changed, since both of them have comparable node degrees. However, the churn operation is detrimental for the BA network, since the joining nodes are of a low degree and the departing ones of a potentially much higher degree. As a result, the degree distribution itself is changing, leading to higher reconvergence times (*cf.* join operation). This phenomenon can be seen more clearly when looking at networks of different sizes (Figure 6): high churn for small networks is not particularly problematic (small slope on the graph for 100 nodes), since degree variation is limited, but for networks of a size of 1,000 it is quite detrimental, leading to higher reconvergence times (high slope on the graph for 1,000 nodes).

Figure 5. Reconvergence speed per operation, preference change/churn, $n = 1,000$.

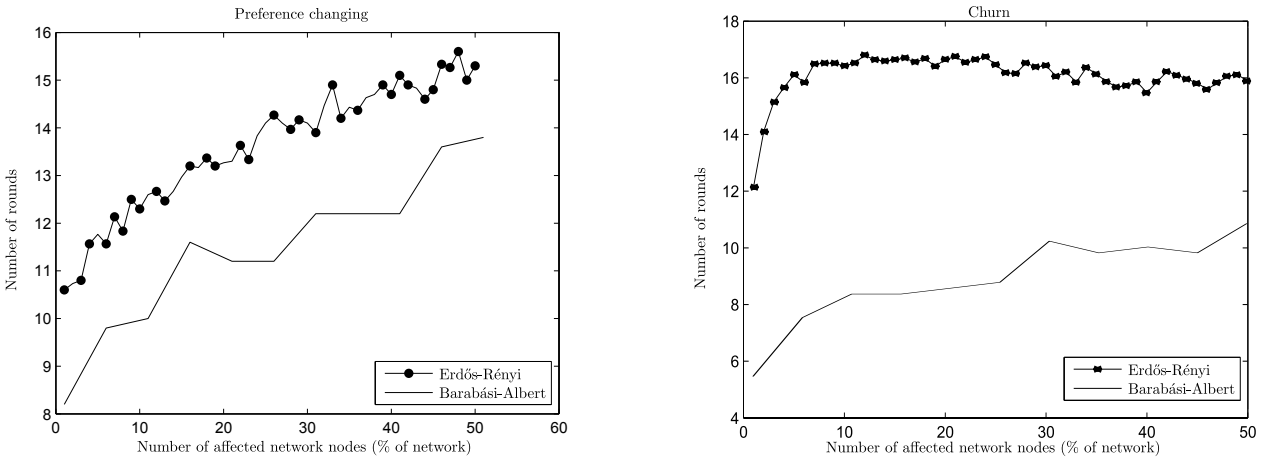
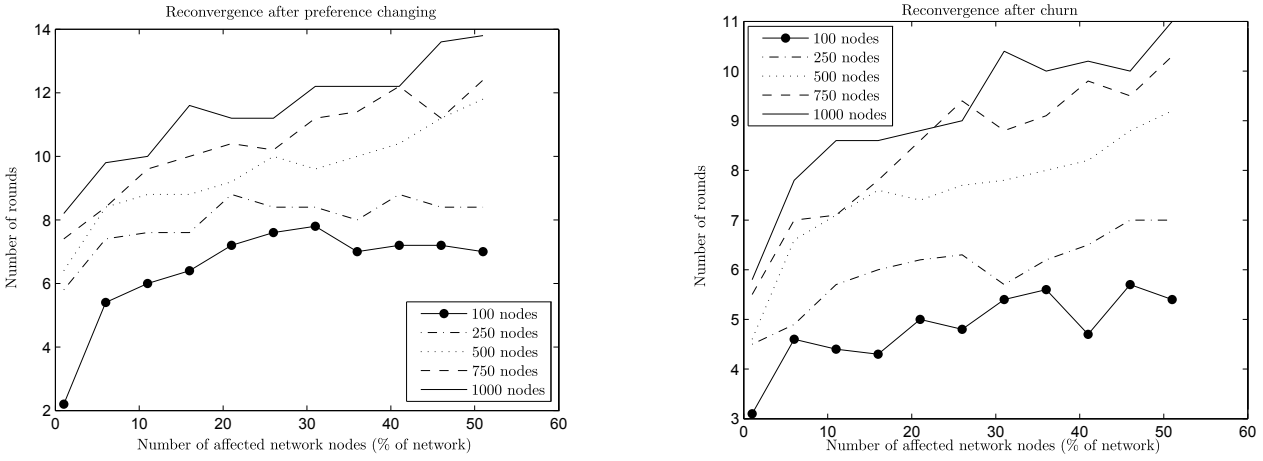
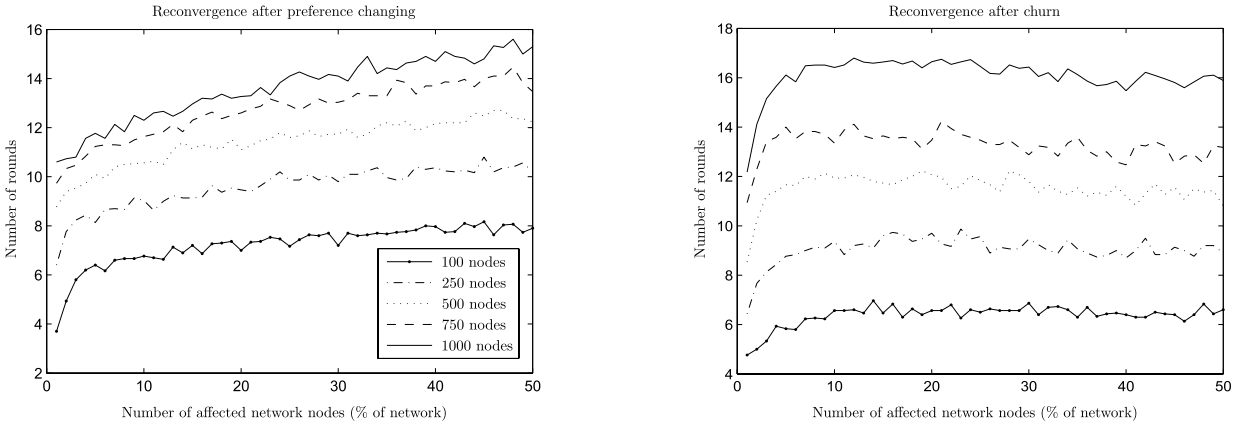


Figure 6. Reconvergence speed for preference change and churn (Barabási–Albert).



In both cases, though, by comparing the churn and preference change graphs in Figure 5, it is easy to see that, somewhat counterintuitively, it takes progressively more time for the ADAPTIVELID algorithm to reconverge when more nodes change preferences, but the reconvergence time stays more or less the same even for high values of churn, or it is consistently lower than preference change, as is the case in BA networks. Figures 6 and 7 show this phenomenon in more detail for all network sizes.

Figure 7. Reconvergence speed for preference change and churn (Erdős–Rényi).



The reason behind this behavior is that in churn situations, there are more parallel events taking place: a new, joining node that replaces a leaving one starts as an empty slate and sends an amount of PROP messages equal to the desired number of connections. On the other hand, a node that changes preferences might need to repair only some of its connections (which are now suboptimal) by sending appropriate PROP messages. However, in both cases, some responding nodes might decline, which will lead to additional PROP messages to be sent, and so on, until the issuing nodes are satisfied or no available nodes are left.

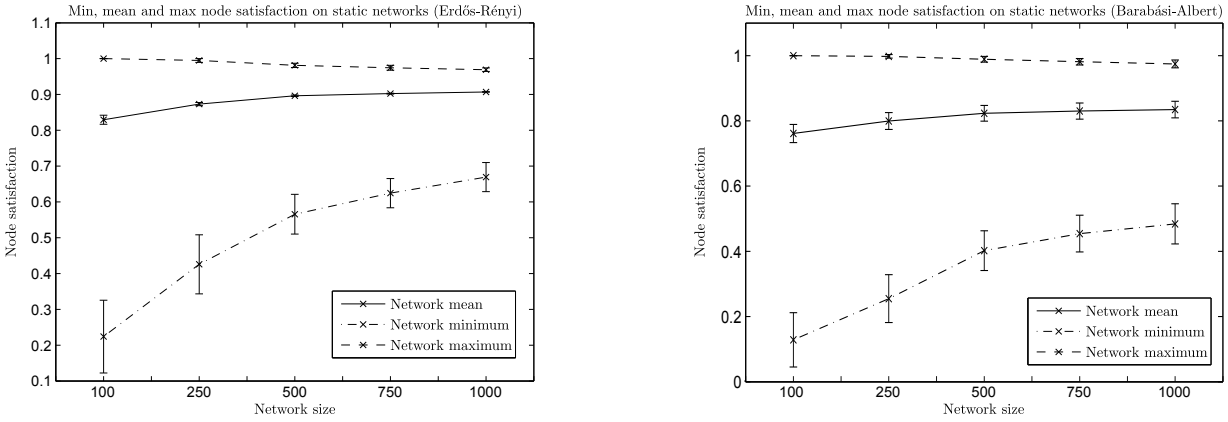
One may even wish to compare the two situations from the point of view of what is a desirable action by a node who changes preferences: to improve existing connections or to perform a leave and come back (thus contributing to churn). This is especially meaningful in the case of ER networks, since for BA networks churn is consistently cheaper in any case, due to their special structure and the parallelism mentioned above. In the case of ER networks, comparing the reconvergence times of the two situations, churn has the advantage over preference change in high values. This is only natural, since in that case more nodes start with no connections and all possibilities are explored in parallel. In contrast, having high values of preference change means that more nodes want to repair their connections, but other nodes have already connections that they want to maintain, leading to longer times of reconvergence. It could be useful in practical terms if there were a mechanism able to detect a high volume of preference changes in the network and to enforce a policy of pseudo-churn, with nodes dropping all connections when changing preferences. However, as is shown below, the amount of satisfaction under churn is far less than the satisfaction under preference change before reconvergence, which is a significant argument in favor of improving connections instead of dropping them and starting again.

6.4. Satisfaction

The mean satisfaction in the network achieved by both algorithms for a variety of network sizes can be found in Figure 8, along with the values of minimum and maximum satisfaction in the network. Note that satisfaction is slightly lower in the case of BA networks, due to differences in topology (*i.e.*, minimum satisfaction is lower, due to the large amount of low degree nodes), but it follows the same behavior as in the ER case. It is easy to see that the algorithms achieve consistently high satisfaction values, which are also increasing as network sizes increase. Of particular interest is that: (a) the minimum satisfaction

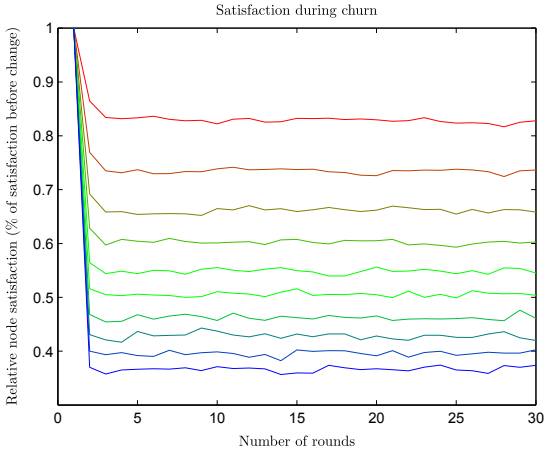
in the network is being increased, also, meaning that individual nodes enjoy high levels of satisfaction, too, implying asymptotically improved *fairness* properties, as well; and (b) the minimum satisfaction does not significantly affect the mean satisfaction, which implies that the number of nodes having low satisfaction is consistently very low compared to the size of the network.

Figure 8. Satisfaction per network size.



Even though the reconvergence results showed that the ADAPTIVELID algorithm can efficiently repair its solution once churn stops, it is interesting to see the levels of achieved satisfaction while churn is in progress. The relative satisfaction for ER networks under churn (to the one achieved before churn starts) can be found in Figure 9: the different graphs from top to bottom correspond to the relative satisfaction when churn affects 5% to 50% of the network's nodes (in steps of 5%), for a network of 100 nodes. It is obvious that the amount of satisfaction achieved remains fairly constant during churn and depends greatly on the amount of churn. However, even though churn is an intense operation, it is possible to retain a significant percentage of the original satisfaction, even for churn as high as 50% (*i.e.*, when half of the network is changing at every round).

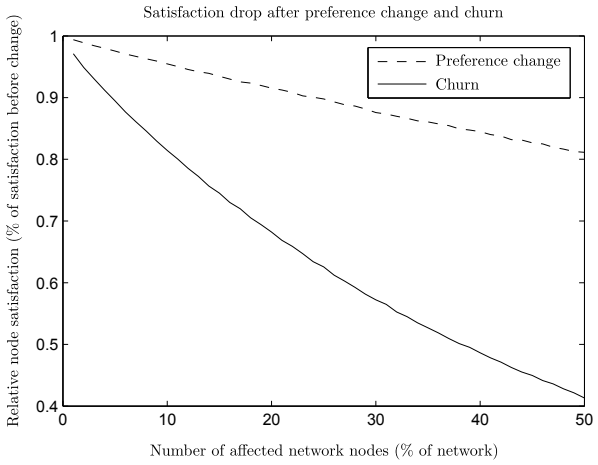
Figure 9. Satisfaction while churn is in progress, affecting 5% to 50% of the network's nodes (in steps of 5%, top to bottom).



On the other hand, the satisfaction drop is significant compared to the one caused by preference change. Figure 10 shows the relative satisfaction for both preference change and churn on ER networks,

right after the change happens, for various amounts of affected nodes, supporting our argument in favor of improving connections instead of rebuilding them from the beginning.

Figure 10. Satisfaction right after preference change or churn per amount of affected nodes.



7. Discussion

Locality Regarding the convergence complexity of the proposed algorithms, note that they achieve guaranteed approximation using only local information, and there is a well-known trade-off (as shown in [13]) between the amount of local information and the quality of the global solution for matching problems. In addition, by arguing as in the classic Chandy and Misra [17] Drinking Philosophers algorithm, we have the following:

Observation 2 *Following the main argument of the classic Chandy and Misra [17] Drinking Philosophers algorithm, the convergence complexity is bounded by the longest increasing edge weight path in the network.*

Symmetry Breaking The locally unique edge weights are assumed as a means to make sure that there are no cyclic dependencies. There may be other ways to break symmetry and perhaps achieve higher parallelism; we expect that it can be possible to build further on this paper's results, e.g., also via working with other, more elaborate algorithms for standard weighted matching as a starting point.

Other Related Work In general, matching is a quite well-studied problem in various contexts (e.g., [18]). In the simplest forms of matching, the maximum cardinality matching and the maximum weighted matching problems, the aim is to find a subset of edges in which no pair of them shares a common endpoint and which includes as many edges as possible or exhibits the maximum sum of edge weights (if they exist), respectively. Other popular variations, such as the stable marriages/roommates problems, assume that nodes have preference lists regarding their potential partner nodes and prefer to be matched to the ones of higher ranking in their preference list. In all of the above cases, the literature usually assumes a one-to-one matching of nodes, but even less studied variations of many-to-many

connections appear to be solvable in polynomial time by centralized algorithms [6,12,19,20], with the exception of particularly difficult subcategories of the stable marriages and roommates problems [1–3].

Of particular interest in distributed and overlay applications, it has been shown [13] that exact solutions of even simple matching problems cannot be derived locally in a distributed manner, and there is significant research interest for distributed approximation algorithms [8,21,22]. Prominent examples of this research area are the one-to-one weighted matching algorithms of Manne *et al.* [23] and Lotker *et al.* [21,24], with the former having proven self-stabilization properties and the latter having variants that can handle joins and leavings of nodes. However, whether it is possible to extend these techniques to many-to-many matchings remains an open research question. On the other hand, Koufogiannakis *et al.* [25] proposed a randomized δ-approximation distributed algorithm for maximum weighted b-matching in hypergraphs (with $\delta = 2$ for simple graphs); but it addresses only static graphs, and its elaborate nature makes its extension to support a dynamic setting non-trivial.

Additionally to the approaches above, there is an extensive research focus on many-to-many matchings with preferences [4,5,7,26]. First, Gai *et al.* in [4] proved that in the case of an acyclic preference system, there is always a stable configuration and also supplied examples of preference systems based on global or symmetric metrics. Mathieu in [5] introduced the measure of node satisfaction as a metric aimed to describe the quality of the proposed solutions. Previously, Lee [26] had used a similar *credit* metric in order to optimize the proposed solutions from their heuristic algorithms. Other approaches regarding the treatment of preferences include the notion of popular matchings, which is a relaxation of a maximum cardinality matching and where the aim is to maximize the amount of nodes that prefer a given matching over any other [27]. While particularly suitable for game theoretic studies, this approach does not offer a way to quantify the potential of possible connections and evaluate potential matchings. The algorithms proposed here are, to our knowledge, the first to address the b-matching with the preferences problem from an optimization-oriented perspective, suitable for overlay networks. Furthermore, they are the first to support dynamic operations for the same problem and offer convergence and approximation guarantees.

Proposal-Refusal Schemes In the literature, many of the algorithms for matching with preferences are inspired by the proposal-refusal algorithm of Gale and Shapley [20]. Here, we also utilize the proposal-refusal scheme, but in order to address a different problem with unique characteristics: while the Gale–Shapley algorithm is focused on absolute stability, we aim at solving an optimization problem with the maximum possible satisfaction. For example, in the case of the Gale-Shapley algorithm, it is important to guarantee that no cycles exist, since, given the distributed nature of the algorithm, a reply may not be possible to be given immediately by a node to another node's proposal. This is not necessary here, since any cycles in preference lists are broken by reducing the original problem to an acyclic weighted many-to-many matching, upon which the algorithm operates (*cf.* also Lemma 8). On the other hand, by focusing on optimization, we are able to develop both non-adaptive and adaptive variations of our algorithm in order to tolerate and work under changes in the underlying network (*cf.* Lemma 10).

The best mate initiative described in [5] has certain similarities to the way nodes propose connections to their neighbors according to the AdaptiveLID algorithm. In particular, according to the best mate initiative, a node knows at every moment which of its edges is the best blocking edge and chooses to

propose to that neighbor. By contrast, according to the ADAPTIVELID algorithm, a node proposes to its neighbors by descending order of ranking (related to their truncated satisfaction), but these edges might not be the best blocking edges at that point in time (*i.e.*, the neighbors might not reciprocate with a proposal of their own). However, by the termination of the ADAPTIVELID algorithm, it is indeed the case that every node would be connected to the endpoints of the best blocking edges related to the truncated satisfaction metric. Finally, the ADAPTIVELID algorithm needs only local information, does not utilize the notion of blocking pairs and can operate in a fully distributed setting.

Approximation In Theorem 3, we proved that the proposed distributed algorithm in both variations is a $\frac{1}{4}\left(1+b_{\max}^{-1}\left(1+\frac{1-b_{\max}^{-1}}{2s+b_{\max}^{-1}}\right)\right)$-approximation algorithm for the maximizing satisfaction b-matching problem. This approximation breaks down into: (a) $\frac{1}{2}\left(1+b_{\max}^{-1}\left(1+\frac{1-b_{\max}^{-1}}{2s+b_{\max}^{-1}}\right)\right)$-approximation between the original maximizing satisfaction b-matching problem and the modified version we are using (Theorem1); and (b) $\frac{1}{2}$-approximation of a distributed many-to-many weighted matching using the LID or ADAPTIVELID algorithms (see proof of Theorem 3). The first of these approximations is due to the satisfaction estimation in Lemma 1, and there is no indication whether it is a tight bound or not. The second term is due to the specific approximation algorithm used to compute a many-to-many weighted matching, in this case, the simple distributed algorithm, LID of Section 5, which is based on the one-to-one weighted matching algorithm found in [8]. Although there is a rather limited choice of algorithms for distributed many-to-many weighted matchings, it may be possible to convert some algorithms, such as the ones in [21,24] mentioned above for the simple (one-to-one) weighted matching. However, even the conversion of state-of-the-art algorithms, such as these, may give an approximation factor of $\left(\frac{1}{2}-\epsilon\right)$; so, further improvement of this term is the subject of future research. At the same time, this conversion, depending on whether the instances of the one-to-one matching algorithms can be executed in parallel or have to be synchronized in rounds, may hold potential implications for the computational complexity, and this, too, is the subject of future research.

8. Conclusion

In this paper, we studied the distributed b-matching problem and suggested a novel modeling that focuses on optimization of connections instead of strict matching stability (as defined in [12]). Following this modeling, we presented a distributed algorithm in two variants, an adaptive and a non-adaptive, which enables peers with preference lists to form an overlay network while collectively achieving a guaranteed level of quality for their requested connections. Each peer is free to form its preference list according to any suitability metric it chooses, based on, e.g., the peer's distance, interests, recommendations, transaction history or available resources. We showed that both variations of the algorithm provably terminate and succeed in maximizing the total satisfaction in the overlay with guaranteed approximation using only local information. In addition, the ADAPTIVELID variation is able to handle dynamicity, *i.e.*, joins/leaves of peers or changing preference lists.

Besides, an extensive experimental study of the proposed algorithms encompasses a variety of scenarios, including ones that put the adaptive algorithm under heavy stress and that have been previously used in the literature to simulate network attacks. In these scenarios, the adaptive algorithm succeeds

in maintaining a reduced but steady level of network service while under attack and resumes to normal service levels after the attack stops. Furthermore, both algorithms show attractive properties with respect to the satisfaction they can achieve and the convergence time (and hence overhead) needed. Regarding changes in particular, the experiments clearly strengthen the argument that it is preferable to improve connections and adapt to changes instead of rebuilding all the connections from the ground up.

To the best of our knowledge, the distributed algorithm presented here in both its adaptive and non-adaptive variations is the first method able to solve the b-matching with preferences problem with satisfaction and convergence guarantees. We expect that this contribution will be helpful for future work in the area, since the method can facilitate overlay construction with guarantees in a wide range of applications, from peer-to-peer resource sharing, to overlays in intelligent transportation systems and adaptive power grid environments. Finally, interesting paths of future research would be to develop variations of the proposed algorithm that can give minimum satisfaction guarantees individually to each collaborating peer or that can take into account scenarios in which malicious nodes actively try to disrupt the algorithm's execution.

References

1. Iwama, K.; Manlove, D.; Miyazaki, S.; Morita, Y. Stable Marriage with Incomplete Lists and Ties. In Proceedings of ICALP'99: The 26th International Colloquium on Automata, Languages and Programming, Prague, Czech Republic, 11–15 July, 1999; pp. 443–452.
2. Manlove, D.F.; Irving, R.W.; Iwama, K.; Miyazaki, S.; Morita, Y. Hard variants of stable marriage. *Theor. Comput. Sci.* **2002**, *276*, 261–279.
3. Ronn, E. NP-complete stable matching problems. *J. Algorithms* **1990**, *11*, 285–304.
4. Gai, A.T.; Lebedev, D.; Mathieu, F.; de Montgolfier, F.; Reynier, J.; Viennot, L. Acyclic Preference Systems in P2P Networks. In Proceedings of the 13th International Parallel Processing Conference (Euro-Par), Rennes, France,28–31 August, 2007; pp. 825–834.
5. Mathieu, F. Self-stabilization in preference-based systems. *Peer-to-Peer Netw. Appl.* **2008**, *1*, 104–121.
6. Irving, R.W.; Scott, S. The stable fixtures problem—A many-to-many extension of stable roommates. *Discrete Appl. Math.* **2007**, *155*, 2118–2129.
7. Cechlárová, K.; Fleiner, T. On a generalization of the stable roommates problem. *ACM Trans. Algorithms* **2005**, *1*, 143–156.
8. Hoepman, J.H. Simple Distributed Weighted Matchings. *CoRR* **2004**, cs.DC/0410047.
9. Preis, R. Linear Time $1/2$-Approximation Algorithm for Maximum Weighted Matching in General Graphs. *STACS 99* **1999**, 259–269.
10. Jacob, R.; Richa, A.; Scheideler, C.; Schmid, S.; Täubig, H. A Distributed Polylogarithmic Time Algorithm for Self-Stabilizing Skip Graphs. In Proceedings of the 28th ACM Symposium on Principles of Distributed Computing, Calgary, Alberta, Canada, 10–12 August, 2009; pp. 131–140.
11. Awerbuch, B.; Scheideler, C. Towards a scalable and robust DHT. *Theory Comput. Syst.* **2009**, *45*, 234–260.
12. Gusfield, D.; Irving, R.W. *The Stable Marriage Problem: Structure and Algorithms*; MIT Press: Cambridge, MA, USA, 1989.

13. Kuhn, F.; Moscibroda, T.; Wattenhofer, R. The Price of Being Near-Sighted. In SODA '06: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, Miami, Florida, USA, 22–26 January, 2006; pp. 980–989.
14. Barabási, A.L.; Albert, R. Emergence of scaling in random networks. *Science* **1999**, *286*, 509–512.
15. Bollobás, B. *Random Graphs*; Academic Press: New York, NY, USA, 1985.
16. Jelasity, M.; Montresor, A.; Jesi, G.P.; Voulgaris, S. The Peersim Simulator. Available online: http://peersim.sf.net (accessed on November 2013).
17. Chandy, K.M.; Misra, J. The drinking philosophers problem. *ACM Trans. Programm. Lang. Syst.* **1984**, *6*, 632–646.
18. Lovasz, L.; Plummer, M.D. *Matching Theory*; Number 121 in North-Holland Mathematical Studies/Number 29 in Annals of Discrete Mathematics; Publishing Co., Amsterdam, the Netherlands, 1986.
19. Edmonds, J. Paths, trees and flowers. *Can. J. Math.* **1965**, *17*, 449–467.
20. Gale, D.; Shapley, L.S. College admissions and the stability of marriage. *Am. Math. Mon.* **1962**, *69*, 9–15.
21. Lotker, Z.; Patt-Shamir, B.; Rosen, A. Distributed Approximate Matching. In Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, Portland, Oregon, USA, 12–15 August, 2007; pp. 167–174.
22. Wattenhofer, M.; Wattenhofer, R. Distributed Weighted Matching. In Proceedings of the 18th Annual Conference on Distributed Computing (DISC), Amsterdam, the Netherlands, 4–8 October, 2004; pp. 335–348.
23. Manne, F.; Mjelde, M. A Self-stabilizing Weighted Matching Algorithm. In *Stabilization, Safety, and Security of Distributed Systems*; Masuzawa, T., Tixeuil, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4838, pp. 383–393.
24. Lotker, Z.; Patt-Shamir, B.; Pettie, S. Improved Distributed Approximate Matching. In SPAA '08: Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures, Munich, Germany 14–16 June, 2008; pp. 129–136.
25. Koufogiannakis, C.; Young, N.E. Distributed Fractional Packing and Maximum Weighted B-Matching Via Tail-Recursive Duality. In Proceedings of the 23rd International Conference on Distributed Computing, Elche, Spain, 23–25 September, 2009; pp. 221–238.
26. Lee, H. Online stable matching as a means of allocating distributed resources. *J. Syst. Arch.* **1999**, *45*, 1345–1355.
27. Gärdenfors, P. Match making: Assignments based on bilateral preferences. *Behav. Sci.* **1975**, *20*, 166–73.