CHALMERS | GÖTEBORG UNIVERSITY

Technical Report No. 12-04

Adaptive distributed b-matching in overlays with preferences

GIORGOS GEORGIADIS
MARINA PAPATRIANTAFILOU

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY/
GOTEBORG UNIVERSITY
Goteborg, Sweden, 2012

Adaptive distributed b-matching in overlays with preferences[⋆]

Giorgos Georgiadis, Marina Papatriantafilou

Department of Computer Science and Engineering, Chalmers University of Technology, S-412 96 Göteborg, Sweden

`Email: {georgiog,ptrianta}@chalmers.se, Fax: +46-31-7723663`

Abstract. An important function of overlay networks is the facilitation of connection, interaction and resource sharing between peers. The peers may maintain some private notion of how a "desirable" peer should look like and they share their bounded resources with peers that they prefer better than others. Recent research proposed that this problem can be modeled and studied analytically as a many-to-many matching problem with preferences. The solutions suggested by the latter proposal guarantee both algorithmic convergence and stabilization, however they address static networks with specific properties, where no node joining or leaving is considered. In this paper we present an adaptive, distributed algorithm for the many-to-many matching problem with preferences that works over any network, provides a guaranteed approximation for the total satisfaction in the network and guarantees convergence. In addition, we provide a detailed experimental study of the algorithm that focuses on the levels of achieved satisfaction as well as convergence and reconvergence speed. Finally, we improve, both for static and dynamic networks, the previous known approximation ratio.

1 Introduction

Overlay networks play an increasingly important role in today's world: from social networks to ad hoc communication networks, people and machines connect, interact and share resources through novel overlay networks laid over Internet's infrastructure or other communication substrate. Unstructured overlays in particular aim at connecting peers with minimal assumptions on the protocols to be used, while addressing universal challenges: peers are willing to share both concrete resources, such as bandwidth, and abstract ones, such as attention span, but these resources are naturally limited and usually the contributors expect something in return. As an example, in a generic, fully distributed scenario each peer may rate its neighbors according to one or more individual metrics (e.g. distance, interests, available resources) but choose to connect to only a handful of them due to its own resource scarcity. The challenge in this scenario is to maintain a high level of the implemented service on a network scale, while at the same time adapt to and tolerate the high dynamicity commonly found in these networks, with peers leaving, joining or changing ratings about their neighbors at any time.

This kind of connection problem with limited resources and reciprocal relations between contributors gives rise to a natural modeling using undirected graphs where nodes have limited (but different) connection capacities. It is essentially a form of a *matching problem* in a graph, where nodes must be matched one to one with some neighbor of theirs in a maximal way on the graph level. The particular form of matching which is relevant here is *many-to-many matching with preferences* (commonly referred to as *stable fixtures* [1] or *b-matching with preferences* [2] problem), where each node maintains a preference list of its neighbors (rated from most to least preferable) and a total quota of desired connections b_i for each node i. The goal for each node in this setting is to be able to form the desired amount of connections with the highest quality (most preferred) neighbors.

Although current literature includes efficient algorithms for many-to-many matching with preferences that can produce a stable configuration if one exists, recent research showed [2] that the problem does not always admit stable solutions. Furthermore, most suitable algorithms from literature are centralized and cannot be deployed in a distributed setting such as the unstructured overlay

⋆ Work supported by the European Union 7th Framework Programme (FP7/2007-2013) under grant agreement No. 257007 and the Swedish Foundation for Strategic research (SSF), grant nr. IMF11-0051.

networks considered here. In addition to the above, none of the currently available distributed algorithms for the b-matching problem can handle the dynamic aspect of overlay networks (e.g. node arrivals/departures and preference changes). Concrete examples of previous work, along with their relation to the present study, can be found in the subsection below.

In this paper we focus on the problem of adaptive, distributed many-to-many matching problem with preferences. Using the metric of node *satisfaction* [2] that can be used for measuring the quality of a node's connections, we build on earlier work that modeled the problem from an optimization perspective. The contribution is threefold:

(i) We show an improved approximation ratio, that can also be applied to existing algorithms and imply improved bounds for their satisfaction guarantees.

(ii) We propose an adaptive, distributed algorithm for the problem, which guarantees that the calculated solution maximizes the total satisfaction in the network within the newly shown bound: an approximation of $\frac{1}{4}\left(1 + b_{\max}^{-1}\left(1 + \frac{1-b_{\max}^{-1}}{2s+b_{\max}^{-1}}\right)\right)$ in every case, 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. To the extend of our knowledge it is the first algorithm that can handle dynamicity while solving the many-to-many matching problem, with node joining, leaving and preference changes fully supported. Its key features include the use of only local information, a given approximation bound and a guaranteed convergence once the changes complete.

(iii) We also provide a 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 and reconvergence speed. Specifically, we show that the resulting satisfaction is high but also remains on high levels during and after reconvergence, while reconvergence is achieved in an efficient way under a variety of changes. Besides, motivated by [3,4] 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 it withstands the attacks while maintaining graceful satisfaction values throughout them.

1.1 Related work

Matching problems are well studied in their centralized form and an extensive literature exists, including solutions for the many-to-many variants (cf for example [1,5,6,7]). However, it has been shown [8] that exact solutions of even simple matching problems cannot be derived locally in a distributed manner, leading to a significant research interest for approximation distributed algorithms [9,10,11]. Prominent examples of this research area are the one-to-one weighted matching algorithms of Manne et al. [12] and Lotker et al. [10,13], 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. [14] 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 [2,15,16,17]. First Gai et al. in [16] 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 [2] introduced the measure of node satisfaction as a metric aimed to describe the quality of the proposed solutions. Georgiadis et al. in [18] modeled the b-matching with preferences problem as an optimization problem that uses satisfaction to achieve convergence. The authors showed an approximation is possible through the reduction of the original problem to a many-to-many weighted matching problem. However, the proposed algorithm is only suitable for static networks, since it cannot adapt to and guarantee convergence after changes in the topology of the network (joins/leavings) or nodes' preferences. Previously Lee [17] had used a similar *credit* metric in order to optimize the proposed solutions from their heuristic algorithms.

2 Problem definition and system model

In this paper we use standard terms and notions from the literature [1,2,13,18]) which we briefly describe here for self-containment. We represent an overlay 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 $\varGamma_i$[1]. 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 0 to its most desirable neighbor. Each node i wants to maintain at most b_i connections to the best possible nodes according to its preference list and rank function, and at no point it can exceed this number. 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* or *matched 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* [1] or *b-matching* [2]. We call *adaptive b-matching* the dynamic form of b-matching, where nodes can join, leave or change preferences at any time. In the remaining 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 a node i's efforts in establishing its b_i connections, we make use of the notion of *satisfaction* S_i (defined in [2] and analyzed in [18]) to be equivalent to the following:

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

where C_i (with $|C_i| = c_i \leq b_i$) is an ordered list of node i's established connections in decreasing preference and $Q_i\left(j\right)$ is the rank of node j in the connection list C_i of node i. According to the above formula, S_i takes values between 0 and 1, depending on how many and which connections a node has formed. A satisfaction value 1 is achieved by a node that has formed all b_i desired connections with its b_i most preferable neighbors, while a penalty is inserted for each non-optimal connection that it forms. Note that we can write formula 1 as:

$$S_i = \sum_{j \in C_i} \left(\frac{L_i - R_i\left(j\right)}{b_i L_i} \right) + \sum_{j \in C_i} \left(\frac{Q_i\left(j\right)}{b_i L_i} \right) = \sum_{j \in C_i} S^s_{i,j} + \sum_{j \in C_i} S^d_{i,j} = S^s_i + S^d_i \tag{2}$$

We refer to the quantities S^s_i and S^d_i as the *a priori part* and the *a posteriori part* of node i's satisfaction respectively, as the former summation terms are computable for each j regardless of whether it is matched with i or not, while the latter summation's terms are computable only for those j that are matched with i by the solution.

The truncatedS maximizing satisfaction b-matching problem Georgiadis et al. in [18] modeled the b-matching problem as an optimization problem that uses satisfaction to achieve convergence, and defined the problem of maximizing the total sum of node satisfaction as the *maximizing satisfaction b-matching* problem. That paper showed that an approximation to the original problem is possible by forming edge weights $w(i,j)$ using only both endpoints' marginal a priori part of satisfaction $S^s_{i,j}$ and $S^s_{j,i}$,

$$w\left(i,j\right) = S^s_{i,j} + S^s_{j,i} = \left(\frac{L_i - R_i\left(j\right)}{b_i L_i} \right) + \left(\frac{L_j - R_j\left(i\right)}{b_j L_j} \right). \tag{3}$$

When the above approximation is used for satisfaction calculations, the satisfaction for each node i is essentially computed by using only the a priori part of satisfaction in formula 2. The resulting problem,

[1] In the rest of the paper and when it is clear from context, we will use notation L_i to denote both the list and its length.

called here *truncatedS maximizing satisfaction b-matching* problem, has been proven equivalent to a *many-to-many weighted matching* problem with edge weights as defined in formula 3 [18]. Note here that a simple weighted matching problem is defined as the problem of finding a set of edges whose weight sum is maximized and which have no common endpoints between them. The many-to-many variant used here replaces the constraint on no common endpoints with node capacities that need to be respected, in this case the connection quotas b_i per node i.

In the analysis of the algorithm we will also use the notion of a *locally heaviest edge* [19]. Let E_{ij} be the set of edges having either of nodes i and j as an endpoint (but not both):

$$E_{ij} = \{(i, n_i)\,|n_i \in \mathit{\Gamma}_i \backslash j\} \cup \{(j, n_j)\,|n_j \in \mathit{\Gamma}_j \backslash i\} \tag{4}$$

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

$$w\,(i, j) > w\,(e)\,, e \in E_{ij} \tag{5}$$

3 Improved approximation ratio

Before proceeding to the presentation and analysis of the algorithm, we show an improved approximation ratio for the problem under study. As shown in the proof, the new ratio applies both for the static case of [18] and the dynamic case studied here.

Theorem 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 to study the ratio

$$r_i = \frac{S_i^s}{S_i^s + S_i^d} \tag{6}$$

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

$$S_{mod} = \sum_{i=1}^{n} S_i^s = \sum_{i=1}^{n} r_i\left(S_i^s + S_i^d\right) \geq r_{\min}\sum_{i=1}^{n}\left(S_i^s + S_i^d\right) = r_{\min} S_{orig} \tag{7}$$

where $n = |V|$ is the number of nodes in the graph.

Initially, consider the scenario where every node i is connected with its bottom-most b_i neighbors. Using the above notation, by lemma 1 of [18] we know that

$$S_{\text{ mod }} = \sum_{i=1}^{n} S_i^s = \sum_{i=1}^{n}\frac{b_i + 1}{2L_i} \tag{8}$$

$$S_{orig} = \sum_{i=1}^{n}\left(S_i^s + S_i^d\right) = \sum_{i=1}^{n}\frac{b_i}{L_i} \tag{9}$$

$$S_{\text{ mod }} \geq \frac{1}{2}\left(1 + \frac{1}{b_{\max}}\right) S_{orig}. \tag{10}$$

where $\frac{1}{2}\left(1 + \frac{1}{b_{\max}}\right)$ is the previous known approximation of the modified maximizing satisfaction b-matching problem.

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

$$S^s_{bc_i} = \frac{1 - \frac{L_{bc_i} - b_{bc_i}}{L_{bc_i}}}{b_{bc_i}} + \ldots + \frac{1 - \frac{L_{bc_i} - 1}{L_{bc_i}}}{b_{bc_i}} = \frac{b_{bc_i} + 1}{2L_{bc_i}} \tag{11}$$

$$S^d_{bc_i} = \frac{0}{b_{bc_i} L_{bc_i}} + \ldots + \frac{b_{bc_i} - 1}{b_{bc_i} L_{bc_i}} = \frac{b_{bc_i} - 1}{2L_{bc_i}} \tag{12}$$

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

$$S^s_{nb_i} = \frac{1 - \frac{L_{nb_i} - b_{nb_i} - 1}{L_{nb_i}}}{b_{nb_i}} + \ldots + \frac{1 - \frac{L_{nb_i} - 2}{L_{nb_i}}}{b_{nb_i}} = \frac{b_{nb_i} + 3}{2L_{nb_i}} \tag{13}$$

$$S^d_{nb_i} = \frac{0}{b_{nb_i} L_{nb_i}} + \ldots + \frac{b_{nb_i} - 1}{b_{nb_i} L_{nb_i}} = \frac{b_{nb_i} - 1}{2L_{nb_i}} \tag{14}$$

Using equations 8 to 14 we get:

$$\frac{S'_{\text{mod}}}{S'_{orig}} = \frac{\sum\limits_{i=1}^{n} S^s_i + \sum\limits_{i \in V/B} \frac{1}{L_i}}{\sum\limits_{i=1}^{n} \left(S^s_i + S^d_i\right) + \sum\limits_{i \in V/B} \frac{1}{L_i}} = \frac{S_{\text{mod}} + \sum\limits_{i \in V/B} \frac{1}{L_i}}{S_{orig} + \sum\limits_{i \in V/B} \frac{1}{L_i}}. \tag{15}$$

For reasons of clarity, in the remaining proof we are going to refer to quantities $\sum\limits_{i \in V/B} \frac{1}{L_i}$ and $\sum\limits_{i \in V/B} \frac{1}{L_i} \Big/ S_{orig}$ with Q and q respectively, as well as $q_{\min}$ for the minimum value of the latter. Using the above notation and equations 10 and 15 we get:

$$\frac{S'_{\text{mod}}}{S'_{orig}} \geq \frac{\frac{1}{2}\left(1 + \frac{1}{b_{\max}}\right) S_{orig} + Q}{S_{orig} + Q} \geq \frac{\frac{1}{2}\left(1 + \frac{1}{b_{\max}}\right) S_{orig} + q_{\min} \cdot S_{orig}}{S_{orig} + q_{\min} \cdot S_{orig}} = \frac{\frac{1}{2}\left(1 + \frac{1}{b_{\max}}\right) + q_{\min}}{1 + q_{\min}} \tag{16}$$

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

$$\sum_{i \in V/B} \frac{1}{L_i} \geq |V/B| \frac{1}{L_{\max}} \geq \min\left(|V/B|\right) \frac{1}{L_{\max}} = \left(n - \max|B|\right) \frac{1}{L_{\max}} \tag{17}$$

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

$$n = |B| + \sum_{i \in B} L_i - b_i \geq |B| + |B| \min_{i \in B} \left(L_i - b_i\right) = |B| + |B| \Rightarrow |B| \leq \frac{n}{2} \tag{18}$$

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

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 17,18 and the above definition we get:

$$q = \frac{\sum\limits_{i \in V/B} \frac{1}{L_i}}{S_{orig}} = \frac{\sum\limits_{i \in V/B} \frac{1}{L_i}}{\sum\limits_{i=1}^{n} \frac{b_i}{L_i}} = \frac{\sum\limits_{i \in V/B} \frac{1}{L_i}}{\sum\limits_{i=1}^{n} (1 - c_i)} \geq \frac{\left(n - \frac{n}{2}\right) \frac{1}{L_{\max}}}{n\left(1 - c_{\min}\right)} = \frac{1}{2L_{\max}\left(1 - c_{\min}\right)} \tag{19}$$

From equations 16,19 and using the observation that $1 - c_{\min} = \max \frac{b_i}{L_i} \leq \frac{b_{\max}}{L_{\min}}$, we get:

$$\begin{aligned}
\frac{S'_{\mathrm{mod}}}{S'_{orig}} &\geq \frac{\left(\frac{b_{\max}+1}{2b_{\max}}\right) + \frac{1}{2(1-c_{\min})L_{\max}}}{1 + \frac{1}{2(1-c_{\min})L_{\max}}} \\
&= \frac{\left(b_{\max}+1\right)\left(1-c_{\min}\right)L_{\max} + b_{\max}}{2b_{\max}\left(1-c_{\min}\right)L_{\max} + b_{\max}} \\
&= \frac{1}{2}\frac{\left(b_{\max}+1\right)\left(1-c_{\min}\right)L_{\max} + b_{\max}}{b_{\max}\left(1-c_{\min}\right)L_{\max} + \frac{1}{2}b_{\max}} \\
&= \frac{1}{2}\left(1 + \frac{\left(1-c_{\min}\right)L_{\max} + \frac{1}{2}b_{\max}}{b_{\max}\left(1-c_{\min}\right)L_{\max} + \frac{1}{2}b_{\max}}\right) \\
&= \frac{1}{2}\left(1 + \frac{1}{b_{\max}}\frac{\left(1-c_{\min}\right)L_{\max} + \frac{1}{2}b_{\max}}{\left(1-c_{\min}\right)L_{\max} + \frac{1}{2}}\right) \\
&= \frac{1}{2}\left(1 + \frac{1}{b_{\max}}\left(1 + \frac{b_{\max}-1}{2\left(1-c_{\min}\right)L_{\max}+1}\right)\right) \\
&\geq \frac{1}{2}\left(1 + \frac{1}{b_{\max}}\left(1 + \frac{b_{\max}-1}{2\frac{b_{\max}}{L_{\min}}L_{\max}+1}\right)\right) \\
&= \frac{1}{2}\left(1 + \frac{1}{b_{\max}}\left(1 + \frac{1-\frac{1}{b_{\max}}}{2\frac{L_{\max}}{L_{\min}}+\frac{1}{b_{\max}}}\right)\right) \\
&= \frac{1}{2}\left(1 + b_{\max}^{-1}\left(1 + \frac{1-b_{\max}^{-1}}{2s+b_{\max}^{-1}}\right)\right)
\end{aligned} \tag{20}$$

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.

The above theorem does not make any assumptions on whether the problem/graph is static or dynamic. Hence it leads to an improved approximation bound for the Local Information-based Distributed (LID) algorithm in [18] that solves the b-matching with preferences problem.

Corollary 1. *The* LID *algorithm solves the 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.*

4 Adaptive matching algorithm

The adaptive matching algorithm builds on the modeling of the Local Information-based Distributed (LID) algorithm [18] and utilizes the notion of node satisfaction to optimize the matching; for self-containment we summarize it here. The core idea of the LID algorithm is that every node maintains a preference list of all its neighbors and regards every potential connection as able to give a fraction of satisfaction, amounting to 1 for a full connection quota with top choices or less in the case of sub-optimal choices (0 for no connections). The network optimization goal we are considering is to maximize the total sum of individual node satisfaction while respecting individual node preferences and connection quotas. During initialization of the algorithm every node i exchanges approximated marginal a priori parts of satisfaction scores $S^s_{i,j}$ with its neighbors j and forms edge weights $w\left(i,j\right)$ using the scores it receives. Already at this point, the first approximation over the original maximizing satisfaction many-to-many matching is being employed by using the approximated form of marginal satisfaction. Using the resulting weights for the matching effectively converts the original problem into a maximum weight many-to-many matching problem, which the nodes proceed to solve by choosing greedily only locally heaviest edges. This second approximation (i.e. choosing locally heaviest edges instead of globally heaviest ones) along with the first one jointly lead to a $\frac{1}{4}\left(1+b_{\max}^{-1}\left(1+\frac{1-b_{\max}^{-1}}{2s+b_{\max}^{-1}}\right)\right)$-approximation solution of the original problem but also lead to valuable properties for the algorithm: on the one hand to a simple, fully distributed scheme and on the other to provable termination even when cycles are present among node preferences (cf lemma 5 in [18]).

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 instead of recomputing it from the beginning. 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 (resp. departing) node, neighboring nodes add (resp. delete) it to (resp. 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.

In literature, many of the algorithms for matching with preferences are inspired by the proposal-refusal algorithm of Gale and Shapley [6]. This also the case with AdaptiveLID but it addresses a different problem with unique characteristics: while the Gale-Shapley algorithm is focused on absolute stability, AdaptiveLID solves an optimization problem and aims for the maximum possible satisfaction. So, for example, it is important to guarantee that no cycles exist in the case of Gale-Shapley algorithm 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 in the case of AdaptiveLID since any cycles in preference orders are broken by reducing the original problem to an acyclic weighted many-to-many matching, upon which the algorithm operates (cf also lemma 3). On the other hand, it is important for the AdaptiveLID algorithm to tolerate and work under changes in the underlying network and by focusing on optimization it achieves exactly that (cf lemma 5).

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 in both endpoints. Set P_i stores the neighbors to which node i proposed with a PROP message, A_i stores the neighbors which 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 originated 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 1 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 Analysis

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 1. *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 maximum distance 2 and to a bounded amount of nodes (bounded by the size of distance 2 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 a *locally heaviest node* in the neighborhood of node i at some point in time if there are no available nodes that are endpoints of heavier edges. Note that when the endpoints of an edge consider simultaneously each other locally heaviest, the edge between them is a locally heaviest edge.

Lemma 2. *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 distance 1 (nodes that the joining/leaving node connects to/disconnects from or direct neighbors of a node that changes preferences) and possibly at distance 2 (neighbor's neighbors of a joining/leaving node). By lemma 1 the weight updates complete in a finite amount of time. When procedure 3 executes next in any node i at distance 1 or 2 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 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 lowest weight [2]. 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, and the same is true for every node in the network.

Lemma 3. *The* AdaptiveLID *algorithm terminates for every node $i \in V$ after changes complete.*

Proof. For the static case, lemma 5 of [18] applies and the algorithm terminates. For the dynamic case, a change may cause some proposals to be canceled and reissued on nodes at distance 1 or 2 (lemma 2). The only cases where 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 alternatingly way over time. By lemma 1 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

[2] Note that whether a node is considered locally heaviest or not may change during a single execution of procedure 3.

modulo operator we get

$$\begin{aligned}
&\sum_{i=0}^{k-1} w\left(n_{i \bmod k}, n_{(i+1) \bmod k}\right) > \\
&> \sum_{i=0}^{k-1} w\left(n_{i \bmod k}, n_{(i-1) \bmod k}\right) = \\
&= \sum_{i=0}^{k-1} w\left(n_{(i+1) \bmod k}, n_{i \bmod k}\right) = \\
&= \sum_{i=0}^{k-1} w\left(n_{i \bmod k}, n_{(i+1) \bmod k}\right) \qquad (21)
\end{aligned}$$

which is a contradiction.

For the second case, by lemma 3 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 4. *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 b_i of them that are heavier than any unchosen one.*

Proof. For a static network, lemma 4 of [18] is applicable. For a dynamic network, by lemma 2, after a finite amount of time every node i cancels all proposals to neighbors which are no longer locally heaviest after a change and proposes to 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 2 it would have to cancel the appropriate proposals and issue new ones towards locally heaviest neighbors.

Lemma 5. *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 3 we get that the AdaptiveLID algorithm terminates for every node and by lemma 4 we know that at termination it has chosen only locally heaviest edges of maximum weight. Since by lemma 4 of [18] the static algorithm also selects locally heaviest edges of maximum weight at termination, it follows that the two algorithms make the same choices for the same networks.

From lemma 5 and theorem 1, as well as lemma 2 and theorem 2 of [18], we get the following theorem about the approximation ratio of AdaptiveLID:

Theorem 2. *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.*

Observation 1 *Following the main argument of the classic Chandy and Misra [20] Drinking Philosophers algorithm, we observe that the convergence complexity is bounded by the longest increasing edge weight path in the network.*

6 Experimental study

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

– Behavior on different types of networks
– Behavior during different operations (joins, leaves, preference changes and churn)
– 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

The following experiments were conducted using the PeerSim [21] 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 the AdaptiveLID algorithm to converge. For every network instance used, a matching was calculated and 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. Each of these operations was conducted on networks of size $n = 100, 250, 500, 750$ and 1000 nodes, considering 30 network instances for each size, and the results presented here are mean values over these instances.

6.1 Network types

The networks used during the experiments were power-law and random networks, created with the Barabási-Albert (*BA*) [22] and Erdős-Rényi (*ER*) [23] procedures respectively. These networks were selected for their different node degree distributions: in power-law networks the vast majority of nodes has very low degree and few nodes have 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 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, both network types had node preferences formed uniformly at random since previous research [2,16] showed that (a) a strict matching solution can not always be found when they are used and (b) the measured satisfaction of unconverged instances can be relatively low. These characteristics make random preferences the challenging test case to evaluate the performance of the algorithm.

6.2 Convergence and reconvergence

The mean value and standard deviation of convergence speed for a variety of network sizes can be found in figure 1. It is easy to see that the convergence speed depends on the type of the network. For example, BA networks of size 1000 take almost twice the amount of time to converge than networks of size 100, while ER networks of size 1000 need less than twice the amount of time needed by networks of size 100 and only slightly higher amount of time than the networks of size 500 and 750.

Likewise, the time needed for reconvergence can be seen in figures 2 and 3, for networks of size 1000 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

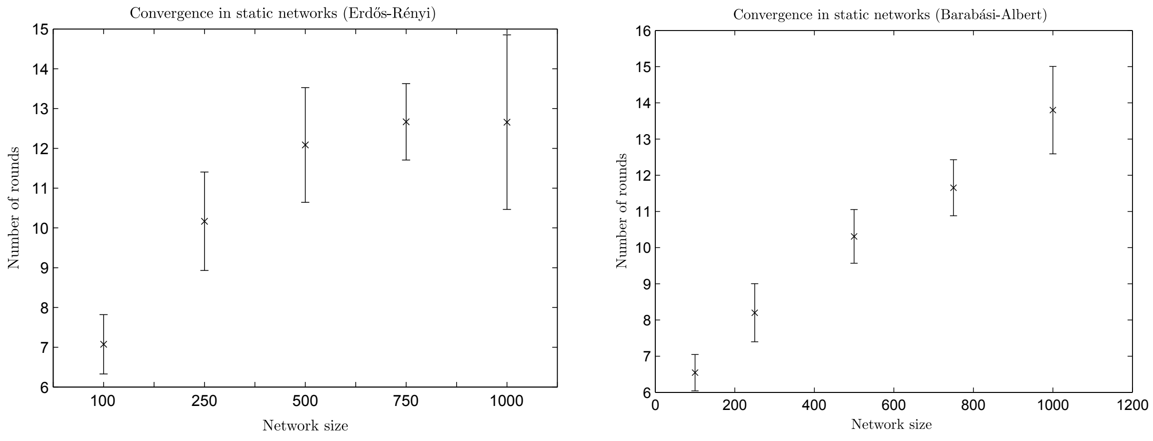


Fig. 1. Convergence speed per network size

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 is rounds.

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 they are neighbors with. 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 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 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 easier since they have more unformed connections than established ones.

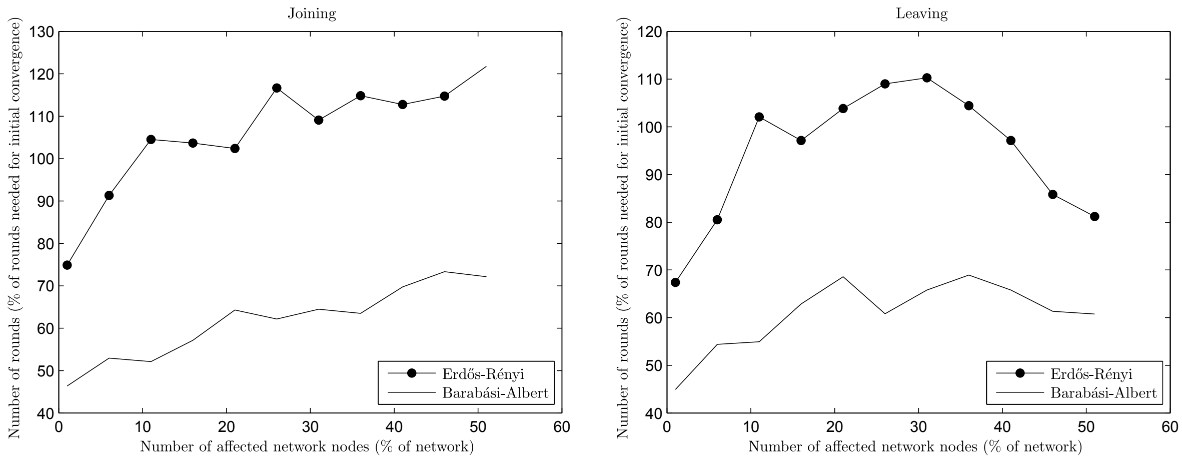


Fig. 2. Reconvergence speed per operation, joins/leaves, $n = 1000$

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 low degree, leading to an overall performance similar to the ER case.

The two network types show their differences more prominently under churn (figure 3). 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 low degree and the departing ones of 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 5): high churn for small networks is not particularly problematic (small slope on graph for 100 nodes) since degree variation is limited, but for networks of size 1000 it is quite detrimental, leading to higher reconvergence times (high slope on graph for 1000 nodes).

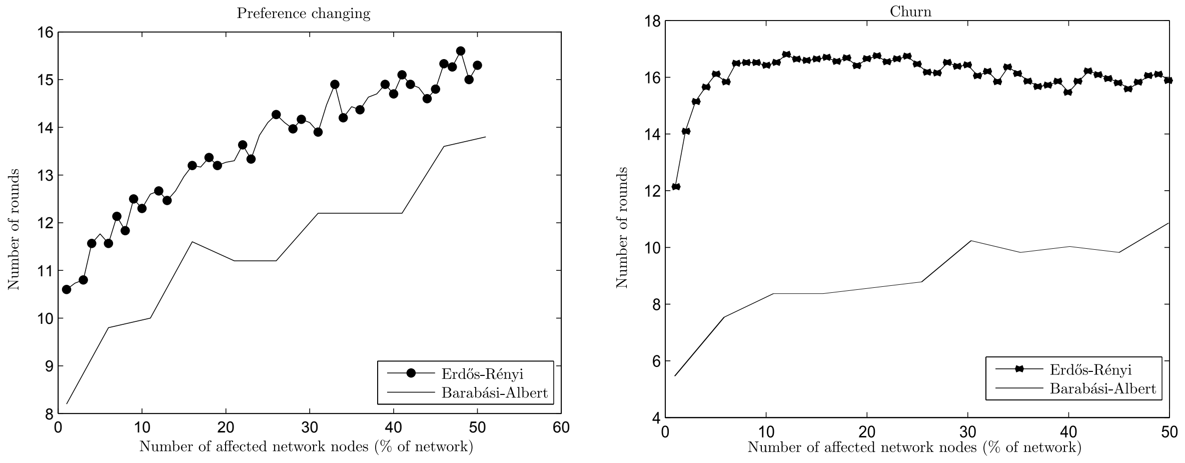


Fig. 3. Reconvergence speed per operation, preference change/churn, $n = 1000$

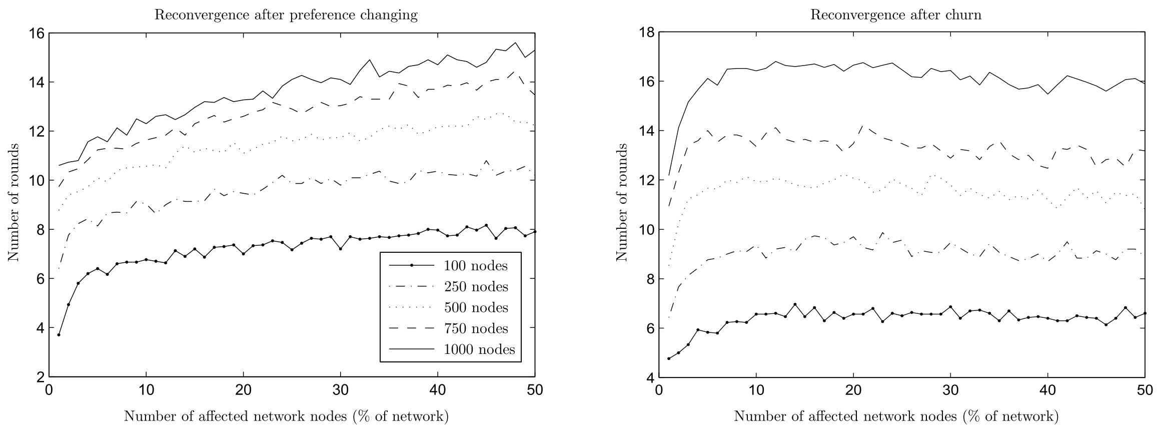


Fig. 4. Reconvergence speed for preference change and churn (Erdős-Rényi)

In both cases though, by comparing the churn and preference change graphs in figure 3 it is easy to see that, somewhat counterintuitively, it takes progressively more time for the algorithm to reconverge when more nodes change preferences but the reconvergence time stays more or less the

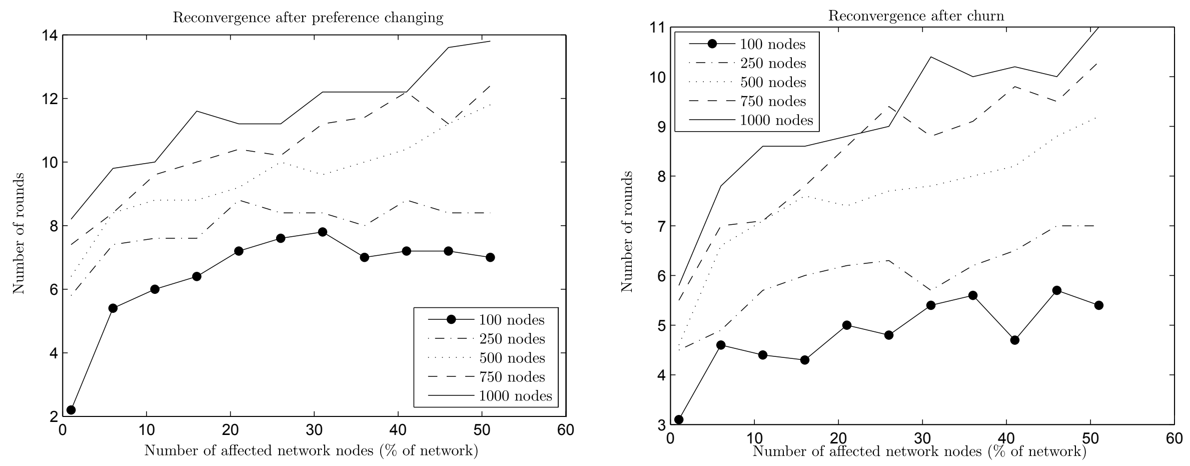


Fig. 5. Reconvergence speed for preference change and churn (Barabási-Albert)

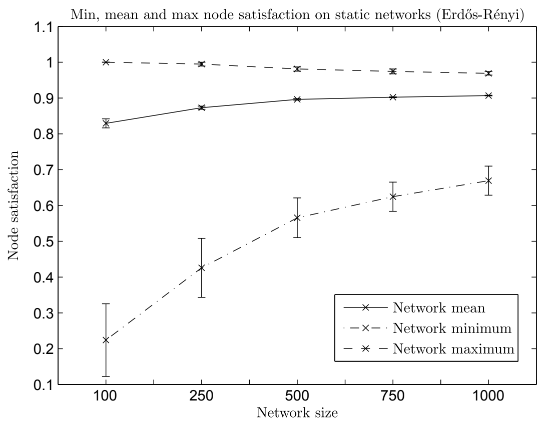
same even for high values of churn or it is consistently lower than preference change, as is the case in BA networks. Figures 4 and 5 shows this phenomenon in more detail for all network sizes.

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 send 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 was a mechanism able to detect high volume of preference changes in the network and enforce a policy of pseudo-churn, with nodes dropping all connections when changing preferences. However, as it 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.

6.3 Satisfaction

The mean satisfaction in the network achieved by the AdaptiveLID algorithm for a variety of network sizes can be found in figure 6, 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 AdaptiveLID algorithm achieves 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 affect significantly the mean satisfaction, which implies that the number of nodes having low satisfaction is consistently very low compared to the size of the network.



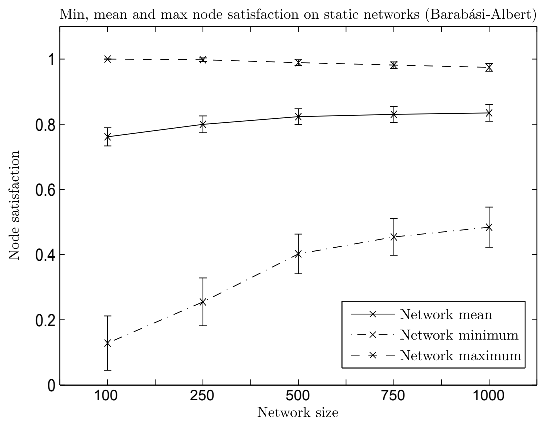


Fig. 6. Satisfaction per network size

Even though the reconvergence results showed that the 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 7: 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).

On the other hand the satisfaction drop is significant compared to the one caused by preference change. Figure 8 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.

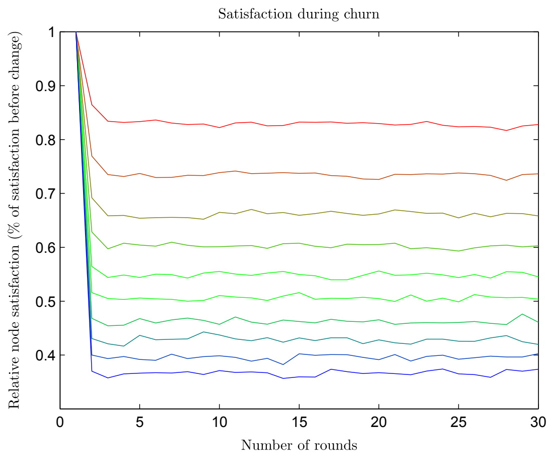


Fig. 7. Satisfaction while churn is in progress, affecting 5% to 50% of the network's nodes (in steps of 5%, top to bottom)

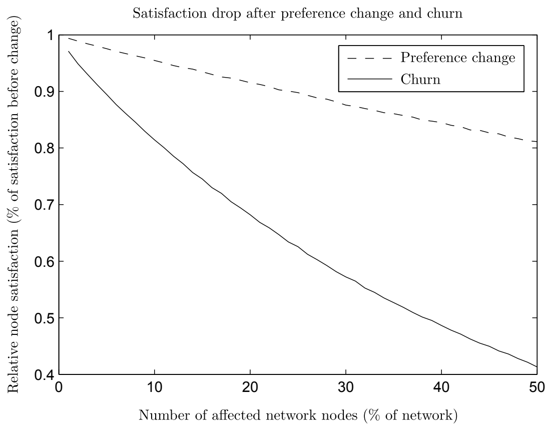


Fig. 8. Satisfaction right after preference change or churn per amount of affected nodes

7 Conclusions

The adaptive algorithm AdaptiveLID for distributed matching with preferences proposed in this paper provides a method to form overlays with preferences with guaranteed satisfaction and convergence, as shown in the analysis. The paper also shows an improved approximation ratio for the maximizing satisfaction problem, which holds both for static and dynamic networks.

Besides, an extensive experimental study of the proposed algorithm encompasses a variety of scenarios, including ones that put the algorithm under heavy stress and that have been previously used in literature to simulate network attacks. In these scenarios the 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, the algorithm shows attractive properties with respect to the satisfaction it can achieve and the convergence time (and hence overhead) it needs. 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 it is the first adaptive method with satisfaction and convergence guarantees for this problem. 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.

References

1. Irving, R.W., Scott, S.: The stable fixtures problem - a many-to-many extension of stable roommates. Discrete Appl. Math. **155**(16) (2007) 2118–2129
2. Mathieu, F.: Self-stabilization in preference-based systems. Peer-to-Peer Networking and Applications **1**(2) (sept 2008) 104–121
3. 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. PODC '09, ACM (2009) 131–140
4. Awerbuch, B., Scheideler, C.: Towards a scalable and robust dht. Theory of Computing Systems **45** (2009) 234–260 10.1007/s00224-008-9099-9.
5. Edmonds, J.: Paths, trees and flowers. Canadian Journal of Mathematics **17** (1965) 449–467
6. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. American Mathematical Monthly **69** (1962) 9–15
7. Gusfield, D., Irving, R.W.: The stable marriage problem: structure and algorithms. MIT Press, Cambridge, MA, USA (1989)
8. 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, ACM (2006) 980–989
9. Hoepman, J.H.: Simple distributed weighted matchings. CoRR **cs.DC/0410047** (2004)
10. Lotker, Z., Patt-Shamir, B., Rosen, A.: Distributed approximate matching. In: PODC '07: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing, ACM (2007) 167–174
11. Wattenhofer, M., Wattenhofer, R.: Distributed weighted matching. In: 18th Annual Conference on Distributed Computing (DISC). (2004) 335–348
12. Manne, F., Mjelde, M.: A self-stabilizing weighted matching algorithm. In: Stabilization, Safety, and Security of Distributed Systems. Volume 4838 of LNCS. Springer (2007) 383–393
13. 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, ACM (2008) 129–136
14. 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, Springer-Verlag (2009) 221–238
15. Cechlárová, K., Fleiner, T.: On a generalization of the stable roommates problem. ACM Trans. Algorithms **1**(1) (2005) 143–156
16. 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 (2007) 825–834

17. Lee, H.: Online stable matching as a means of allocating distributed resources. Journal of Systems Architecture **45**(15) (1999) 1345–1355
18. Georgiadis, G., Papatriantafilou, M.: Overlays with preferences: Approximation algorithms for matching with preference lists. In: Proceedings of 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS'10), IEEE Computer Society Press (April 2010)
19. Preis, R.: Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. STACS 99 (1999) 259–269
20. Chandy, K.M., Misra, J.: The drinking philosophers problem. ACM Transactions on Programming Languages and Systems **6**(4) (1984) 632–646
21. Jelasity, M., Montresor, A., Jesi, G.P., Voulgaris, S.: The Peersim simulator `http://peersim.sf.net`.
22. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. Science **286** (1999) 509–512
23. Bollobás, B.: Random Graphs. Academic Press, New York (1985)