CHALMERS | GÖTEBORG UNIVERSITY

Technical Report No. 11-13

Adaptive overlay construction through dynamic distributed matching with preferences

GIORGOS GEORGIADIS
MARINA PAPATRIANTAFILOU

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY/
GOTEBORG UNIVERSITY
Goteborg, Sweden, 2011

Adaptive overlay construction through dynamic distributed matching 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 with limited resources. It is usually the case that the peers maintain some private notion of how a "desirable" peer should look like and they share their limited resources with peers that they prefer better than others. Recent research proposed that this problem can be modeled as a many-to-many matching problem with preferences and studied analytically. However, the proposed solutions studied the problem on static networks, where no node joining or leaving is considered.
In this paper we present a dynamic, distributed algorithm for the many-to-many matching problem with preferences that utilizes a simple scheme, provides a guaranteed approximation for the total satisfaction in the network and guarantees the convergence after changes complete. We also provide a detailed experimental study of the algorithm that focuses on the levels of achieved satisfaction as well as convergence and reconvergence speed.

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. Unstructured overlays in specific aim at connecting peers with minimal assumptions on the protocols to be used but face universal challenges: peers are willing to share both concrete resources (i.e. bandwidth) and abstract ones (i.e. attention span) but these resources are naturally limited and usually the contributors expect something in return. For example, in a fully distributed scenario each peer may rate its neighbors according to one or more individual metrics (i.e. distance, interests, available resources) but chose 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 adapting and tolerating 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 resembles a *matching problem* in a graph, where nodes must be matched one to one with some neighbor of theirs in a maximal way on a 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* [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. 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. However, a recently proposed metric of node *satisfaction* [2] can be used to measure the quality of a node's connections, giving an optimization dimension into the many-to-many matching problem.

In this paper we present a dynamic, distributed algorithm for the many-to-many matching problem with preferences that utilizes a simple scheme, provides a guaranteed approximation for the total satisfaction in the network and guarantees the convergence after changes complete. 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. We also provide a detailed experimental study of the algorithm that focuses on the levels of achieved satisfaction as well as convergence and reconvergence speed. We show that the resulting satisfaction is high but also remains on high levels before reconvergence in many scenarios. Moreover, reconvergence is achieved in an efficient way under a variety of changes.

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 [3,4,5,1]). However, it has been shown [6] 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 [7,8,9]. Prominent examples of this research area are the one-to-one weighted matching algorithms of Manne et al [10] and Lotker et al [8], 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 [11] recently 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 [12,13,2,14]. First Gai et al in [13] 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, the same one we use here as an optimization metric. Previously Lee [14] 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 [1,2,15,16]) which we briefly describe them 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 Γ_i. 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 operations simply as *changes*.

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 which is

defined in [2] by the following formula:

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

where C_i (with $|C_i| = c_i \leq b_i$) is an *ordered list of node i's connections* in decreasing preference. Georgiadis et al in [15] modeled the b-matching problem as an optimizing 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, by forming edge weights $w(i,j)$ according to the formula:

$$w\left(i,j\right) = \overline{\Delta S_i^j} + \overline{\Delta S_j^i} = \left(\frac{1}{b_i} - \frac{R_i\left(j\right)}{b_i L_i}\right) + \left(\frac{1}{b_j} - \frac{R_j\left(i\right)}{b_j L_j}\right) \tag{2}$$

where $\overline{\Delta S_i^j}$ is an approximation of the marginal satisfaction that node i gleans by connecting to node j. A simple weighted matching problem is defined as 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 used here 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 also use the notion of a *locally heaviest edge* [17]. Let E_{ij} be the set of edges that have either node i or node j as an endpoint (but not both):

$$E_{ij} = \left\{\left(i, n_i\right) | n_i \in \Gamma_i \backslash j\right\} \cup \left\{\left(j, n_j\right) | n_j \in \Gamma_j \backslash i\right\} \tag{3}$$

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

$$w\left(i,j\right) > w\left(e\right), e \in E_{ij} \tag{4}$$

In this paper we will present AdaptiveLID, a distributed algorithm that solves the adaptive b-matching problem and provably converges to a solution. Through reduction to the adaptive many-to-many weighted matching problem we will also prove an approximation ratio of $\frac{1}{4}\left(1 + \frac{1}{b_{\max}}\right)$, where $b_{\max}$ is the maximum quota in the network. Later in the paper we evaluate experimentally the behavior of the AdaptiveLID algorithm by studying two performance metrics: convergence time and mean satisfaction. By a similar argument to the one found in the classic Chandy and

Misra [18] Drinking Philosophers algorithm, we expect the convergence complexity to be bounded by the longest increasing edge weight path in the network. In the following sections we will consider an asynchronous model for messages and we will not consider failures, i.e. messages arrive asynchronously but do not get lost.

3 Dynamic matching algorithm

In this section we present an adaptive algorithm based on a simple scheme that can find an optimized many-to-many matching in a dynamic network, where nodes can join, leave or change connection preferences at any time. Following earlier work on quality metrics [2], we are utilizing the notion of node satisfaction to optimize the matching: every node maintains a preference list of all its neighbors and regards every potential connection as able to give a fraction of satisfaction. This satisfaction can amount to 1 in the case of filling its connection quota with only top choices, otherwise the node suffers a penalty for each non-optimal (top) connection that can lower the total to 0 (case where no connections are made). 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. We achieve just that in a distributed way and under a variety of dynamicity conditions with approximation ratio $\frac{1}{4}\left(1+\frac{1}{b_{\max}}\right)$, where $b_{\max}$ is the maximum connection quota in the network.

A static many-to-many matching: the LID *algorithm* The new algorithm builds on the idea of the modeling in [15]; for self-containment we summarize it here. The core idea of the LID algorithm is to convert the above mentioned maximizing satisfaction many-to-many matching problem into a maximum weight many-to-many matching problem. First, every node calculates the marginal amount of satisfaction to be gleaned by every connection and exchanges this value with the corresponding one of its neighbors for all adjacent edges. Using this information, the newly formed edge weights are the sums of the adjacent endpoint marginal satisfaction and the resulting matching has a total weight which is the sum weight of all matched edges. Following the weight formation, each node selects locally heaviest edges for inclusion in the desired matching, that is edges that are heaviest in the neighborhoods of both their endpoints. These two approximations, converting the maximizing satisfaction to maximum weight many-to-many matching and selecting locally instead of globally heaviest edges, lead to the resulting $\frac{1}{4}\left(1+\frac{1}{b_{\max}}\right)$-approximation solution

of the original problem. However, the same approximation also results in valuable properties for the algorithm: on the one hand they lead to a simple, fully distributed algorithm and on the other to provable termination for the algorithm even when cycles are present among the node preferences (cf lemma 5 in [15]).

Dynamic many-to-many matching: challenges and the 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 instead of recomputing it from the beginning. It would also be advantageous to limit the repairing in the neighborhood of the operation, so that far enough nodes would remain unaffected. Note that the locally heaviest 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 dynamic algorithm presented here, all three cases of dynamicity mentioned above (join/leave/change) are supported and a node is free to join or depart the network or change its preferences at any time.

- **Join:** We address the join problem in the common way of having the new node contact a network node out of bound, get a list of potential neighbors and connect with them in some arbitrary way, which is outside the scope of the problem we study here. When the new node's neighbors are finalized, it sorts them (e.g. according to its own metric of preference) into a preference list from the best to the worst preferred neighbor. Symmetrically, neighboring nodes evaluate it according to their own criteria and insert it in the appropriate place in their preference lists.
- **Leave:** When a node wishes to depart from the network it notifies its neighbors about its intention and they simply remove it from their neighborhood and preference lists. Note here that the same result can be achieved in the case of an unexpected departure (i.e. due to a crashed node) if nodes search for and remove dead nodes in their neighborhood at regular intervals. After the departure, the nodes that were connected with the departed node have now a free slot and resume their regular run of the algorithm.
- **Preference change:** When a node changes its preferences there is no change in its neighborhood but its evaluation of neighboring nodes may change radically: unconnected nodes may be found more preferable than connected ones.

A common thread between these dynamicity 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 reevaluate their connections: if they are not locally heaviest any more, the nodes abandon the least weighted ones and try to select the locally heaviest. Note here that this method avoids the recalculation of the solution over the whole network, instead limiting it in a small 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. Since the resulting matching is comprised of locally heaviest edges, the previous approximation ratio and convergence are shown to hold here as well.

The AdaptiveLID algorithm (cf. Algorithm 1 in Appendix for pseudocode) uses at each node i five sets $(P_i, K_i, A_i, R_i, B_i)$ and a 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 with them. 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 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, B_i keeps 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.

4 Analysis

Lemma 1. *In a failure free execution, weight updates that are caused after changes complete in a finite amount of time.*[1]

We define as *available* with respect to node i, a node j in the neighborhood of node i that has not been proposed by node i or has 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 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.*

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

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.*

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.*

From lemma 10 and theorem 3 of [15], we get the following theorem about the approximation ratio of AdaptiveLID:

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

5 Experimental study

This section studies the achieved satisfaction and convergence times of the AdaptiveLID algorithm using random Erdős-Rényi networks [19] with preferences formed uniformly at random. The choice of random preferences is only natural since previous research [13,2] showed that a strict matching solution can not always be found when they are used, and the measured satisfaction of the unconverged instance can be relatively low.

[1] Please find the lemma proofs in the Appendix.

These characteristics make random preferences a challenging test case for the AdaptiveLID algorithm.

The following experiments were conducted using the PeerSim [20] platform in a synchronous way[2], in order to measure the time needed by the AdaptiveLID algorithm to converge. For every network used, a matching was calculated and the following operations performed:

- **Joining:** A varying amount of new nodes enter the network simultaneously and the network is left to reconverge.
- **Leaving:** A varying amount of existing nodes exit the network simultaneously and the network is left to reconverge.
- **Preference change:** A varying amount of existing nodes change the ranking of their neighbors in their preference lists simultaneously and the network is left to reconverge.
- **Churn:** A varying amount of existing nodes exit and an equal amount of new nodes enter the network simultaneously. This operation is repeated for several rounds and then the network is left to reconverge.

Each of these operations was conducted on networks of size $n = 100, 250, 500,$ 750 and 1000 nodes. For each network size we considered 30 network instances and the results presented here are mean values over these instances. Also, the amount of affected nodes for each operation varies from 1% to 50% of the network size, in increments of 1%.

5.1 Convergence and reconvergence

The convergence speed for a variety of network sizes can be found in figure 1 as a mean value of 30 instances for each network size, with the standard deviation also shown. It is easy to see that convergence of the AdaptiveLID is sublinear in practice, with networks of size 1000 needing 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 figure 3, for networks of size 1000 and for the four types of operations under consideration. For the join and leave operations, the time needed for the algorithm to reconverge is predictably increasing and decreasing respectively when the amount of affected nodes increases. This happens because the network size is increasing (resp. decreasing) in size (up to 1500 nodes for the

[2] i.e. round based, where each node in each round makes a receive-respond-process step, unless it has nothing to execute.

join operation and 500 nodes for the leave operation) and the algorithm needs more (resp. less) time to find a solution.

For the preference change and churn operations this looks different: 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). By comparing the appropriate 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 same even for high values of churn. Figure 4 shows this phenomenon in more detail for all network sizes.

The reason behind this behavior is that churn can take better advantage of the inherent parallelism in the AdaptiveLID algorithm: 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. Comparing the two situations, churn has the advantage in high values, where more nodes start with no connections and all possibilities are explored in parallel, over preference change where more nodes want to repair their connections but other nodes have already connections that they want to maintain. 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 (compared to the option of nodes with changed preferences to leave and come back, thus causing churn).

5.2 Satisfaction

The mean satisfaction achieved by the AdaptiveLID algorithm for a variety of network sizes can be found in figure 5. Shown also are the minimum and maximum satisfaction in the network, while the results are presented as mean values over 30 network instances along with the resulting standard deviation.

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 the fact that the minimum satisfaction in the network is being increased also, meaning that individual nodes enjoy high levels of satisfaction too, implying a fairness behavior as well.

Even though the reconvergence results showed that the algorithm can repair efficiently its solution once churn stops, it is interesting to see the levels of achieved satisfaction when churn is in progress. The relative satisfaction (to the achieved one before churn starts) under churn can be found in figure 6: 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 2 shows the relative satisfaction for both preference change and churn, 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.

6 Conclusions

The adaptive algorithm 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. Besides, its experimental performance study shows even more attractive properties with respect to the satisfaction it can achieve and the convergence time (and hence overhead) it needs.

To the best of our knowledge it is the first method with guarantees for this problem. We expect that this contribution will have an interesting impact, 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 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. Edmonds, J.: Paths, trees and flowers. Canadian Journal of Mathematics **17** (1965) 449–467
4. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. American Mathematical Monthly **69** (1962) 9–15
5. Gusfield, D., Irving, R.W.: The stable marriage problem: structure and algorithms. MIT Press, Cambridge, MA, USA (1989)
6. 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, New York, NY, USA, ACM (2006) 980–989
7. Hoepman, J.H.: Simple distributed weighted matchings. CoRR **cs.DC/0410047** (2004)
8. 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, New York, NY, USA, ACM (2007) 167–174
9. Wattenhofer, M., Wattenhofer, R.: Distributed weighted matching. In: 18th Annual Conference on Distributed Computing (DISC). (2004) 335–348
10. Manne, F., Mjelde, M.: A self-stabilizing weighted matching algorithm. In Masuzawa, T., Tixeuil, S., eds.: Stabilization, Safety, and Security of Distributed Systems. Volume 4838 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2007) 383–393
11. 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, Springer-Verlag (2009) 221–238
12. Cechlárová, K., Fleiner, T.: On a generalization of the stable roommates problem. ACM Trans. Algorithms **1**(1) (2005) 143–156
13. 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
14. Lee, H.: Online stable matching as a means of allocating distributed resources. Journal of Systems Architecture **45**(15) (1999) 1345–1355
15. 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), Atlanta, GA, IEEE Computer Society Press (April 2010)
16. 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, New York, NY, USA, ACM (2008) 129–136
17. Preis, R.: Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. STACS 99 (1999) 259–269
18. Chandy, K.M., Misra, J.: The drinking philosophers problem. ACM Transactions on Programming Languages and Systems **6**(4) (1984) 632–646
19. Bollobás, B.: Random Graphs. Academic Press, New York (1985)
20. Jelasity, M., Montresor, A., Jesi, G.P., Voulgaris, S.: The Peersim simulator `http://peersim.sf.net`.

7 Appendix

7.1 Proofs

Lemma 6. *In a failure free execution, weight updates that are caused after 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.

Lemma 7. *In a finite amount of time after a 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 6 the weight updates complete in a finite amount of time. When procedure Processing 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 [3]. In any case, at the end of Processing'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.

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

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

Proof. For the static case, lemma 5 of [15] 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 7). 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 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

$$\begin{aligned}\sum_{i=0}^{k-1} w\left(n_{i \bmod k}, n_{(i+1) \bmod k}\right) &> \sum_{i=0}^{k-1} w\left(n_{i \bmod k}, n_{(i-1) \bmod k}\right) = \\ &= \sum_{i=0}^{k-1} w\left(n_{(i+1) \bmod k}, n_{i \bmod k}\right) = \\ &= \sum_{i=0}^{k-1} w\left(n_{i \bmod k}, n_{(i+1) \bmod k}\right) \qquad (5)\end{aligned}$$

which is a contradiction.

For the second case, by lemma 7.1 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 b_i of them that are heavier than any unchosen one.*

Proof. For a static network, lemma 4 of [15] is applicable. For a dynamic network, by lemma 7, 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 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 7.1 we get that the AdaptiveLID algorithm terminates for every node and by lemma 7.1 we know that at termination it has chosen only locally heaviest edges of maximum weight. Since by lemma 4 of [15] 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.

7.2 Pseudocode of AdaptiveLID algorithm

Procedure ReceiveMessages

```
begin
    for msg ∈ queue_i do
        if msg.type = PROP then
            A_i ← A_i ∪ msg.sender
            B_i ← B_i − msg.sender
        if msg.type = REJ then
            B_i ← B_i ∪ msg.sender
            A_i ← A_i − msg.sender
            K_i ← K_i − msg.sender
            P_i ← P_i − msg.sender
        if msg.type = WAKE then
            B_i ← B_i − msg.sender
```

Procedure SendMessages

```
begin
    while (|P_i| < b_i) ∧ (|Γ_i − P_i − (B_i − R_i)| ≠ 0) do
        find heaviest edge neighbor c that belongs in
        (Γ_i − P_i − (B_i − R_i))
        if c ≠ null then
            if c ∈ B_i then
                send a WAKE message to c
                R_i ← R_i − c
            else
                send a PROP message to c
                P_i ← P_i ∪ c
                R_i ← R_i − c
```

Procedure Processing

begin

$T_i \leftarrow (P_i - K_i) \cap A_i$

if $|T_i| \neq 0$ **then**

$A_i \leftarrow A_i - T_i$

$K_i \leftarrow K_i \cap T_i$

`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**

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

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

while $(l \neq \mathit{null}) \wedge (h \neq \mathit{null})$ **do**

if $w(h,i) > w(l,i)$ **then**

if $h \in B_i$ **then**

`send a` WAKE `message to` h

$R_i \leftarrow R_i - h$

else

`send a` REJ `message to` l

$A_i \leftarrow A_i - l$

$R_i \leftarrow R_i \cap l$

$P_i \leftarrow P_i - l$

$K_i \leftarrow K_i - l$

`send a` PROP `message to` h

$P_i \leftarrow P_i \cap h$

$R_i \leftarrow R_i - h$

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

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

else if $P_i = K_i$ **then**

for $j \in (\Gamma_i - R_i - B_i^* + A_i^* - P_i)$ **do**

`send a` REJ `message to` j

$A_i \leftarrow A_i - lj$

$R_i \leftarrow R_i \cap j$

break

else

break

`unmatch node` i `from all nodes in` B_i

Algorithm 1: AdaptiveLID()

```
begin
    ReceiveMessages()
    SendMessages()
    Processing()
```

7.3 Figures

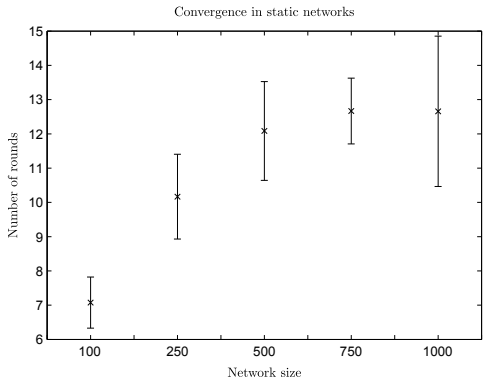

Fig. 1. Convergence speed per network size

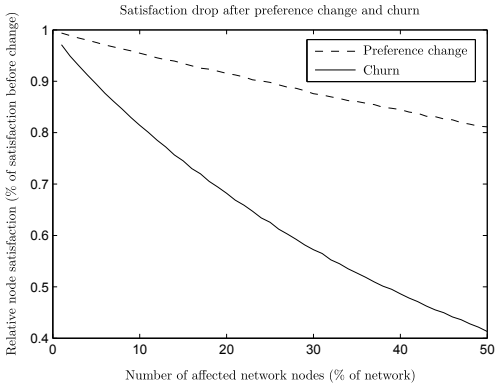

Fig. 2. Satisfaction right after preference change or churn per amount of affected nodes

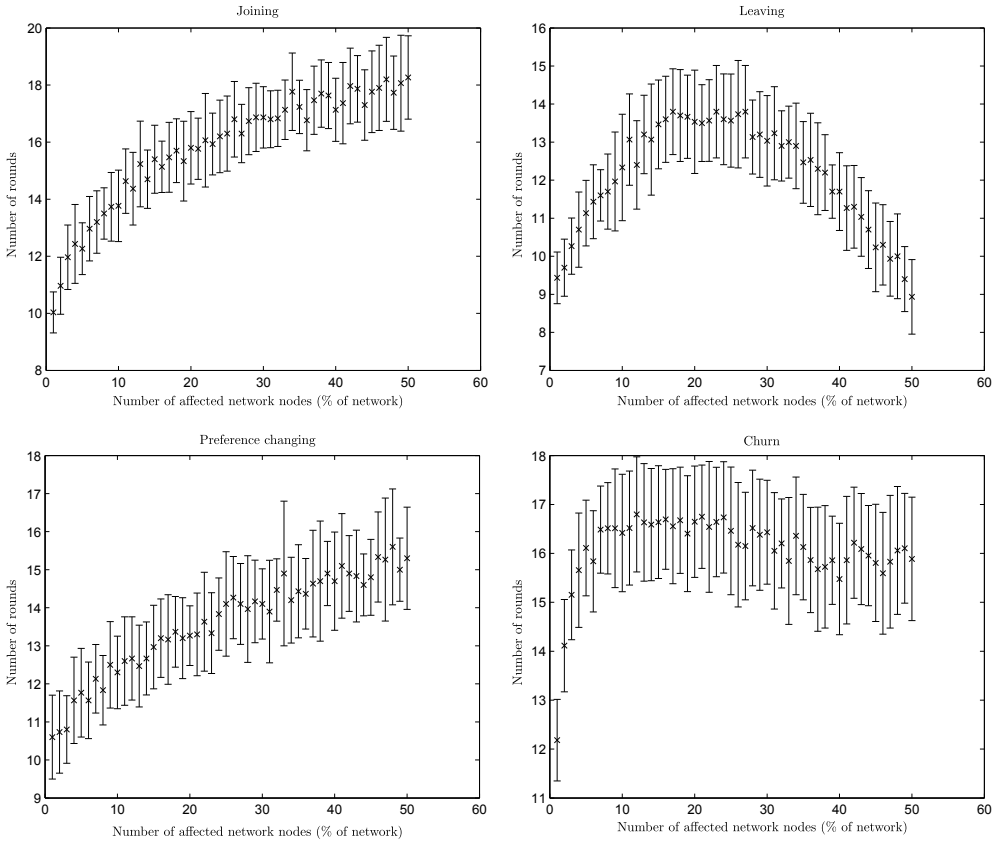

Fig. 3. Reconvergence speed per operation, $n = 1000$

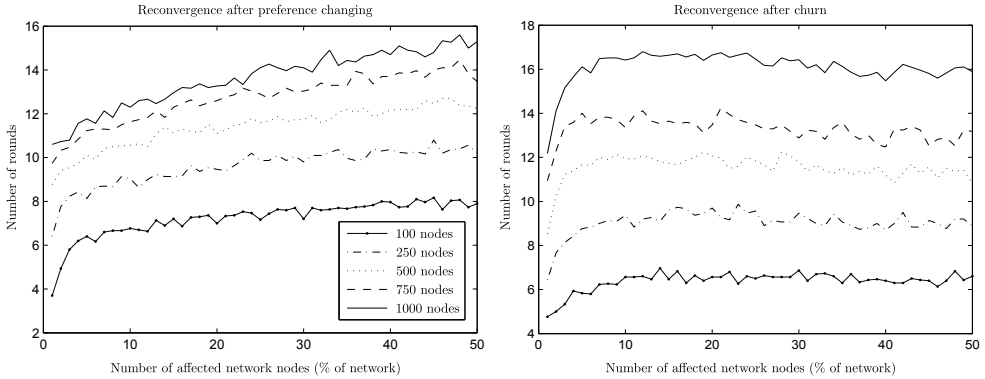

Fig. 4. Reconvergence speed for preference change and churn

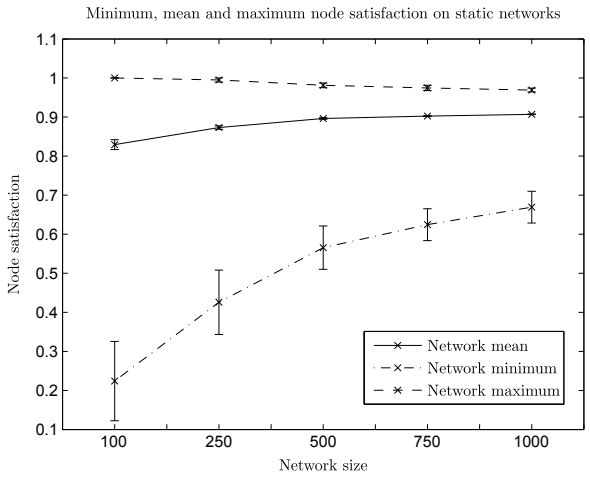

Fig. 5. Mean satisfaction per network size

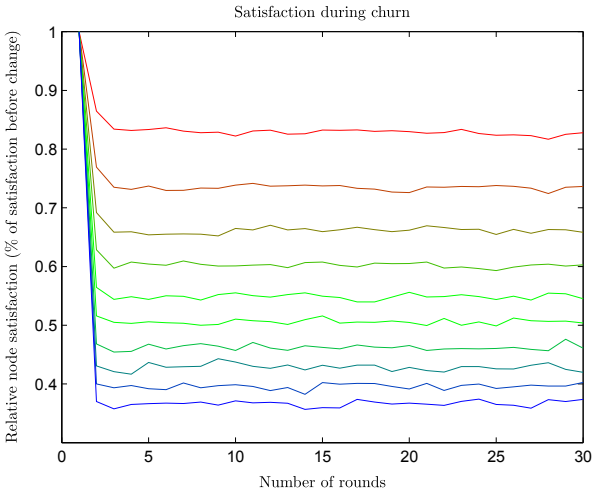

Fig. 6. Satisfaction while churn is in progress, affecting 5% to 50% of the network's nodes (in steps of 5%, top to bottom)