Chmielowiec *et al. Journal of Internet Services and Applications* 2014, **5**:12
http://www.jisajournal.com/content/5/1/12

Journal of Internet Services and Applications
a SpringerOpen Journal

RESEARCH Open Access

Decentralized group formation

Anna Chmielowiec, Spyros Voulgaris and Maarten van Steen*

Abstract

Imagine a network of entities, being it replica servers aiming to minimize the probability of data loss, players of online team-based games and tournaments, or companies that look into co-branding opportunities. The objective of each entity in any of these scenarios is to find a few suitable partners to help them achieve a shared goal: replication of the data for fault tolerance, winning the game, successful marketing campaign. All information attainable by the entities is limited to the profiles of other entities that can be used to assess the pairwise fitness. How can they create teams without help of any centralized component and without going into each other's way? We propose a decentralized algorithm that helps nodes in the network to form groups of a specific size. The protocol works by finding an approximation of a weighted *k*-clique matching of the underlying graph of assessments. We discuss the basic version of the protocol, and explain how dissemination via gossiping helps in improving its scalability. We evaluate its performance through extensive simulations.

Keywords: Web service composition; Distributed clique formation

1 Introduction

Ever since the Internet substantially facilitated the sharing of distant resources, researchers have been pondering on how to effectively and efficiently exploit the immense collection of these resources. A well-known example are Web pages as resources, which are crawled and copied to centralized databases so that they can be used for searching information. Likewise, with the advent of Web services it became much easier to realize large and heterogeneous infrastructures for large-scale computing, eventually leading to the OGSA architecture for grid computing [1]. These two examples illustrate two radically different approaches for structuring the use of resources. Conceptually, when crawling the Web, resources are copied and pulled into a centralized shared database for further processing. In contrast, for grid computing, resources stay in place (and often cannot even be moved), resulting in a decentralized solution for their usage.

In this paper, we concentrate on organizing resources under the assumption that they are dispersed across the Internet, yet it is impossible, or undesirable, to copy or move them to a centralized location. The specific organization that we are seeking is dividing a potentially very large group of resources into nonoverlapping subsets of the same size, also known as the *k*-clique matching problem.

As an example, consider the case where a resource is a replica server operating in the cloud, and we are seeking to partition a dataset and replicate its parts for fault tolerance. In practice, a high degree of fault tolerance can be achieved if a data item is replicated across three servers, provided that failures of those servers are mutually independent. This means that a data item should not be replicated in the same data center, but be spread across three different ones. We can model this problem by organizing the servers into an overlay network, and representing that network as an undirected graph. The weight of an edge represents the quality of the associated communication link, along with the level of failure independence of its end nodes. We then compute a weight of a 3-clique as the arithmetic (or sometimes the geometric) mean of its edges. Obviously, the higher this weight, the better. By finding a maximal 3-clique matching, we have essentially partitioned the set of N replica servers into roughly $N/3$ highly reliable (distributed) virtual servers, each of which can now be used to store one or several data items. Note that our partitioning should be optimal: ideally, there is no partitioning that will lead to a better set of more reliable virtual servers.

*Correspondence: m.r.van.steen@vu.nl
Department of Computer Sciences and The Network Institute, VU University, De Boelelaan 1081A, 1081HV Amsterdam, The Netherlands

As a completely different example, consider a multiplayer online game in which teams of k members each compete in a tournament. Forming teams such that we get the most challenging tournament again boils down to finding a maximal k-clique matching in a set of N players, where each of the roughly N/k cliques forms a team. A weight between two players reflects how appropriate it is to put them in the same team. So, typically, if two players indicate their preference to play the role of a goalkeeper in an online soccer competition, then typically the weight between the two should be very low.

In a previous version of this paper [2], we showed how a solution to the k-clique matching problem could be used for optimal *co-branding* across the Web. Another application is the optimal construction of composite Internet services, services which are now often available only as part of a package provided by ISPs [3].

As said, we assume in this paper that resources cannot be copied or moved to a central location, or that it is undesirable to do so, for example, because there is a shared distrust in the integrity of the computations performed by a third party. This implies that we need to solve the k-clique matching problem in a decentralized manner. Our main contribution is a fully decentralized algorithm for solving this problem. The algorithm has been partly described and evaluated in [2]. In this paper, we describe important improvements that speed up the convergence of the algorithm, allow us to handle cases where the set of resources is subject to churn, and help in overcoming the communication overhead of the protocol in its basic version. In particular, we introduce the following modification to our k-clique matching protocol: random-subset heuristic, pruning, partial views, and gossiping of clique weights.

The remainder of the paper begins with an overview of related work in Section 2. Following is Section 3 in which the system model for decentralized k-clique matching of resources is further detailed and the problem of finding $k-1$ partnering resources is formalized. The core of the paper is formed by the self-stabilizing distributed algorithm, which is described in Section 4. In Section 5, we present the optimizations that are useful in case of a large dynamically changing set of resources. Section 6 presents results of our extensive simulations followed by conclusions in Section 7.

2 Related work

Decentralized clustering based on weights between nodes has been studied in the context of self-organization for overlay construction in peer-to-peer (P2P) networks. Protocols such as T-Man [4] and Vicinity [5] assume weights between nodes. Nodes have a fixed outdegree of k neighbors, and periodically gossip with (some of) them to encounter new potential neighbors and replace ones with lower weight by ones having higher weight. When converged, each node has links to its k highest weight neighbors out of the whole network.

Although both this type of clustering and k-clique matching result in optimal weight links prevailing over suboptimal ones, they bear a fundamental difference. In the former, a node can serve as a preferred neighbor for *any* number of other nodes and formed links are not required to be reciprocated; in the latter each node can participate in exclusively *one* clique and all nodes must agree on participation in the same clique (symmetry constraint).

For $k = 2$ the k-clique matching problem reduces to the well-known and extensively studied problem of matching in graphs; a matching in a graph is defined as any subset of *nonadjacent* edges, which basically are cliques of size 2. In unweighted graphs, matching problems concentrate on finding a *maximum* matching (a matching with the largest number of edges). For this problem, sequential polynomial-time algorithms exist; for example, a maximum matching can be found in $O(\sqrt{|V|}|E|)$ time using one of the algorithms by Micali and Vazirani [6], Blum [7], or Gabow and Tarjan [8]. In our paper we focus solely on weighted graphs. In such graphs, for each matching its weight can be computed by summing the weights of the edges comprising the matching, and the standard matching problem is of finding a matching that is the *heaviest*, referred to as *maximum weighted matching*. Generalizations exist. One of them is the problem of finding in a graph the largest (the heaviest) subset of nonadjacent subgraphs, each of which is isomorphic to some given graph H. This type of problem is commonly referred to as an *H-packing* [9-11], an *H-matching* [12,13] or an *H*-partition [14]. In the special case when graph H is a clique of size k, we obtain a k-clique matching[a] problem that we address in this paper.

Whereas finding a maximum weighted matching can be done in polynomial time using, for instance, the algorithm by Gabow [15] that runs in $O(|V|(|E| + |V|log|V|))$ time, finding a maximum weighted k-clique matching for $k \geq 3$ becomes an NP-hard problem. This follows directly from work of Kirkpatrick and Hell in [14] which proves that finding a perfect H-partition of graph G is NP-complete if H contains a connected component with at least 3 vertices.

Moreover, when distributed algorithms are considered, there does not exist any algorithm that finds a maximum weighted matching. Yet, a few distributed approximation algorithms have been proposed, including the 1/2-approximation algorithm by Hoepman [16], the $O(log|V|)$-time $(1/2 - \epsilon)$-approximation algorithm by Lotker at al. [17], and a $(1 - \epsilon)$-approximation algorithm by Nieberg [18]. From our perspective, the most interesting are self-stabilizing algorithms for weighted matchings,

such as a 1/2-approximation algorithm by Manne and Mjelde [19], which we extend in this paper to solve the more general problem of weighted k-clique matching. Whether other distributed approximation algorithms for weighted matching can also be extended to solve weighted k-clique matching remains an open question.

The research on distributed algorithms for H-packing is even more scarce. For unweighted graphs, there exist a few approximation algorithms for solving packing problems, for example, distributed H-packing algorithms for unit-disk graphs by Czygrinow and Hańćkowiak [10] and planar graphs by Czygrinow et al. [11]. Yet, to our best knowledge, there is no work on distributed algorithms for packing in *weighted* graphs. There is also little work on sequential approximation algorithms for weighted packing problems; a few algorithms that exist deal with randomized finding of a triangle packing in weighted complete graphs [20,21] and do not appear to be easily converted into a distributed algorithm or adapted to finding k-clique packings for any $k \geq 3$. With respect to available research, our work is unique in attempting to tackle the weighted k-clique matching problem in a distributed fashion.

Table 1 summarizes the presented research on matching problem and its generalizations.

3 The *k*-clique matching problem

We consider a set of N *entities*, be they commercial brands, players in a multiplayer game, servers in a decentralized pool of computers, or generally any type of resources that we may want to group together. Each pair of entities is marked with a *weight*, indicating the benefit of combining these two entities.

Regarding weights, we make the following three assumptions. First, they are nonnegative real numbers. Second, they are a global function, in the sense that any entity can assess the weight between any two entities in the network. Third, weights are *symmetric*: the benefit perceived by entity A in being combined with entity B is the same as the one perceived by B in being combined with A. Satisfying these assumptions is not difficult. For example, in a co-branding case, each entity holds a profile that describes customer attitudes towards a given brand. The weight of the edge between any two entities can then be computed based on a symmetric function generally known by all entities.

The target is to group entities in cliques of k members in such a way that the aggregate clique weights are maximized. This problem can be formalized using terms from graph theory. We consider a graph $G = (V, E)$. Each entity corresponds to a single vertex in the graph. The edges connect only those vertices whose corresponding entities can be combined together. The weight of each edge tells how good this combination is.

In such a graph, we are mostly interested in *which* k-cliques can be created. A k-clique is a subgraph induced by k vertices in which an edge exists between every two vertices. Given the weights of the edges, it is possible to assess the weight of a k-clique. Some of the popular metrics include: sum of the edges, arithmetic or geometric mean, and the weight of the heaviest/lightest edge. We interpret the weight of the k-clique as an overall evaluation of suitability of the k entities for forming a k-group.

Each entity would like to be part of exactly one such group. Therefore, we want to partition the graph into disjoint k-cliques.

Definition 1. *(Weighted k-Clique Matching): Given a graph $G = (V, E)$ with nonnegative edge weights, a k-clique matching is a subgraph of G whose components are cliques of size k. The weight of the k-clique matching is defined as the sum of the weights of all its k-cliques.*

Table 1 Research on the matching problem and its generalizations

Problem	Sequential algorithms exact	Approximation	Distributed approximation algorithms
(2-clique) matching	unweighted: $O(\sqrt{V}E)$ [6-8], weighted: $O(V(E + VlogV))$ [8]	$O(E)$ 1/2-approx. [22], $O(ElogV)$ 1/2-approx. [23]	1/2-approx. [16,17,19], $(1-\epsilon)$-approx. [18]
k-clique matching	NP-hard [14]	randomized in complete graphs for $k = 3$ [20,21]	$1/k$-approx. in our previous paper [2]
H-matching	NP-hard [14] max snp-complete [12]	—	unweighted: - in unit-disk graphs [10], - in planar graphs [11]

The mentioned algorithms are for the *weighted* version of the problem unless stated otherwise.

For $k = 2$, the problem of finding a 2-clique matching in the graph is equivalent to finding a traditional matching (a set of independent edges) in a graph. Its weighted version (when the total weight of the 2-cliques is to be maximized) is solvable in polynomial time: $O(|V|(|E| + |V|log|V|))$ [15]. Yet for any $k \geq 3$, a weighted k-clique matching problem becomes NP-hard (see [14]).

Luckily, finding an optimal solution for the weighted k-clique matching problem might not be always necessary, or even desirable; finding an approximation might be completely satisfactory. We can argue, for example, that in the case of k-replication, each server is concerned only with the quality of the cluster it is going to be part of and has no interest in optimizing the quality of other clusters. Thus, entities can be seen as being egocentric, preoccupied only with their own welfare. Based on this observation, we devise a distributed algorithm for finding an approximation of the optimal k-clique matching.

4 The *k*-clique matching protocol

4.1 2-clique matching

Our protocol for creating a k-clique matching overlay is inspired by the self-stabilizing algorithm by Manne and Mjelde [19], which finds a matching which is a $1/2$-approximation of the optimal solution for the maximum weighted matching problem — the total weight of the matching found by this algorithm is at least $1/2$ of the optimal matching weight. In this algorithm, each node v uses two variables: the first, m_v, to store the id of the node it would like to be matched with, the second, $\mathbf{w}_v$, to store the weight of the edge connecting it to that node. Every node tries to find the heaviest incident edge (by pointing with m_v to the other end of that edge), and the only rule that nodes have to obey is that a node v cannot link to a neighbor u if the value of $\mathbf{w}_u$ is higher than the weight of the edge joining v and u. This rule is meant to prevent nodes from bound-to-fail attempts to match with neighbors that have found heavier edges for matching. The final matching M is composed of those edges whose ends point to each other ($m_v = u$ and $m_u = v$) and is achieved in at most $2|M| + 1$ rounds under a fair scheduler, where each node has a chance to execute its step at least once per round.

In this section, we show that the algorithm from [19] can be easily generalized to find a weighted k-clique matching (for any $k \geq 2$) that is at most a factor k off from the maximum. Before we provide pseudo-code for this algorithm, we first present a sequential algorithm by Preis [22] that computes a $1/2$-approximation of the weighted matching. A high-level explanation of this algorithm will help us gain intuition about how the self-stabilizing algorithms for weighted matching and weighted k-clique matching work.

Preis's sequential greedy algorithm is based on the observation that selecting locally heaviest edges produces the aforementioned approximation within a factor $1/2$. Its running time is $O(|E|)$, which is faster than the running time of another sequential algorithm which creates a matching by adding the remaining globally heaviest edge (described in [23], with $O(|E| \cdot log|V|)$ running time). The locally heaviest edge is defined as an edge whose weight is at least as high as the weight of any incident edge. The matching is constructed by iteratively adding to the matching some locally heaviest edge from all the edges remaining in E and removing this edge and all edges incident to it from E until E becomes empty (see Figure 1).

Hoepman [16] describes how Preis's algorithm can be distributed deterministically. But we can also look at the algorithm from [19] as a self-stabilizing variant of Preis's algorithm. Some node v, by choosing one of its neighbors, u, and setting the weight of the edge $\langle v, u \rangle$ to variable $\mathbf{w}_v$, eliminates from the matching all other edges that have v as one of the ends and a weight smaller than $\mathbf{w}_v$. As a result other neighbors of v are forced to look for a matching node among their set of neighbors that does not include v. If an edge is locally the heaviest, i.e., there is no available edge of higher weight incident to it, then the nodes at the ends of this edge would point to each other and as a consequence this edge will become a part of the matching.

We can use the same reasoning to compute a weighted k-clique matching by selecting locally heaviest k-cliques and achieving an approximation factor of k.

4.2 *k*-clique matching

We denote by $N(v)$ the set of all neighbors of v and by $w(U)$ the weight of the subgraph induced by the nodes in U for any subset of nodes $U \subseteq V$; in particular, $w(\{v, u\})$ denotes the weight of an edge between two adjacent nodes u and v.

To accommodate the algorithm from [19] for solving the weighted k-clique matching problem, we start by changing the variables stored by each node. Instead of variable m_v, which kept track of the single neighbor's id that node v would like to be matched with, each node v will now store in $\mathbf{C}_v$ a set of $k-1$ ids of those neighbors with which v wants to create a k-clique. Therefore, node v will now store in variable $\mathbf{w}_v$ the weight of the clique composed of v and its $k-1$ neighbors from $\mathbf{C}_v$. This modification has an obvious effect on which cliques we regard as

```
1: M ← ∅
2: while E ≠ ∅ do
3:     e ← locally heaviest edge from E
4:     M ← M + {e}
5:     E ← E − {e} − {e′ ∈ E : e′ is incident to e}
6: end while
```

Figure 1 Sequential greedy algorithm for weighted matching.

matched: a clique induced by nodes $v_1, v_2, \ldots, v_k$ is considered as *matched* only if for each v_i ($1 \leq i \leq k$) variable $\mathbf{C}_{v_i}$ contains ids of the $k-1$ remaining nodes, i.e. $\mathbf{C}_{v_i} = \{v_1, v_2, \ldots, v_k\} - \{v_i\}$. Thus, in a stable state each node v that is part of some clique should have all other nodes from that clique stored in its set $\mathbf{C}_v$. Moreover, if in a stable state there is some node u that is not part of any clique, its set $\mathbf{C}_u$ should be empty.

Because the algorithm differentiates between cliques based solely on their weights, we need to guarantee that the weight of each k-clique in the graph is unique and that a total ordering of clique weights can be imposed. Otherwise, the protocol may be unable to converge to a correct k-clique matching due to some nodes becoming stuck in livelock or deadlock situations, such as shown in Figure 2. Here we have two adjacent 3-cliques $\triangle v_1, v_2, v_3$ and $\triangle v_1, v_3, v_4$ of equal weight. For nodes v_1 and v_3 either of these cliques is equally attractive, while nodes v_2 and v_4 can be part of only one clique each, $\triangle v_1, v_2, v_3$ or $\triangle v_1, v_3, v_4$ respectively. This leads to four possible configurations generated by choices made by v_1 and v_3: (a) both v_1 and v_3 choose $\triangle v_1, v_2, v_3$, (b) both v_1 and v_3 choose $\triangle v_1, v_3, v_4$, (c) v_1 chooses $\triangle v_1, v_2, v_3$, while v_3 chooses $\triangle v_1, v_3, v_4$, and (d) v_1 chooses $\triangle v_1, v_3, v_4$, while v_3 chooses $\triangle v_1, v_2, v_3$. Moreover, they can keep changing their decision, as either choice is equally good. None of the nodes in the graph can determine in which configuration the system is at the moment based solely on the weights of the cliques chosen by nodes. Thus, v_2 and v_4 are stuck in their choice of v_1 and v_3 for their clique partners, and at least one of them is not in the correctly matched clique according to our definition from the previous paragraph. As a result, none of these configurations leads to a correct 3-clique matching.

To avoid such problems, we assume that each node v has a unique identifier (without the loss of coherence we denote v's identifier simply as v) and that a total ordering is imposed on these identifiers. With that assumption, realization of uniqueness and total ordering of clique weights is straightforward: the weight of each clique is extended with a sorted tuple of the ids of its nodes and a lexicographical ordering is applied to these new weights.

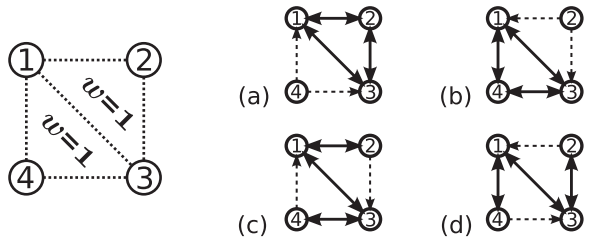

Figure 2 Two adjacent 3-cliques of equal weight impeding the correct convergence of the protocol and 4 possible configurations (a)–(d) that do not create a correct 3-clique matching: the contents of $\mathbf{C}_v$'s are visualized by outgoing arrows.

Moreover, we assume that each node has readily available information on the weights of any edge between itself and its neighbor and also between any pair of its neighbors. This information is necessary but also sufficient in order for a node to compute the weights of any cliques it can be part of. We abstract here from how the weights of these edges are obtained by each node. One feasible solution, is that the weights can be computed from the information a node has about its neighbors. The effects of using different weights, but also different weight distributions, is discussed in [24].

The pseudo-code of our weighted k-clique matching protocol is presented in Figure 3. In an infinite loop each node in the network looks for the most attractive k-clique that it can become part of (we will formalize attractiveness shortly). In order to discover such a clique, node v considers all $\binom{|N(v)|}{k-1}$ subsets of $k-1$ neighboring nodes (line 3) and keeps the most attractive one.

To assess the attractiveness $attr_v(U)$ of a clique formed with nodes from set U, node v has to ensure that none of these nodes is currently involved in a heavier clique, because such a node would not be interested in joining a clique of a smaller weight. To this end, we call a set $U = \{u_1, \ldots, u_{k-1}\}$ of $k-1$ neighbors *proper* if and only if for each neighbor u_i this set is heavier than any clique found by that neighbor so far, i.e.,

$$\forall u_i \in U : w(\{v, u_1, \ldots, u_{k-1}\}) \geq \mathbf{w}_{u_i}$$

We denote the fact that U is proper through the predicate $proper(v, U)$. Only cliques constructed with proper combinations of neighbors can be considered as admissible.

Subsequently, to compare any two sets of $k-1$ neighbors, nodes follow two straightforward rules. From the perspective of node v, subset C' is better than subset C if:

- C' is *proper* and C is not, **or**
- both C' and C are *proper* and $w(\{v\}+C') > w(\{v\}+C)$.

We can express it in a more concise way by, firstly, defining the function $attr_v()$:

$$attr_v(C) = \begin{cases} w(\{v\} + C) & \text{if } proper(v, C) \\ 0 & \text{otherwise} \end{cases}$$

In other words, the *attractiveness* of a set C for node v is expressed as the weight of C after adding v, provided C is indeed admissible, otherwise C should be ignored. We can now check whether C' is better than C by evaluating the expression $attr_v(C') > attr_v(C)$. By executing lines 3–7, node v chooses the most attractive clique, setting $\mathbf{C}_v$ and $\mathbf{w}_v$ accordingly.

Finally, node v sends the new value of $\mathbf{w}_v$ to all of its neighbors (line 10), allowing each neighbor to update

Variables:
$\mathbf{C}_v$ set of $k-1$ neighbors with which v wants to create a clique
$\mathbf{w}_v$ the weight $w(\{v\} + \mathbf{C}_v)$

```
1: loop
2:    C ← {}
3:    for all U ≡ {u_1, ..., u_{k-1}} ⊆ N(v) do
4:       if attr_v(U) > attr_v(C) then
5:          C ← U
6:       end if
7:    end for
8:    C_v ← C
9:    w_v ← w({v} + C_v)
10:   send w_v to all u ∈ N(v)
11: end loop
```

Figure 3 Self-stabilizing weighted *k*-clique matching protocol (executed by node *v*).

information on the clique it was (hoping to) participate in. We need not assume that communication is reliable: message loss will merely slow down the algorithm, but does not affect its correctness.

To sum up, what every node is doing in each round boils down to solving a version of the heaviest k-subgraph problem in a graph induced by the node itself and all its neighbors. What is different from the classical k-subgraph problem is that: (a) we are interested only in k-subgraphs that are cliques, (b) one of the nodes from the resulting k-clique is fixed — the node itself must be a part of the solution, (c) each of the neighbors imposes a constraint on the minimum weight of a clique it can be part of.

5 Optimizations

In some applications of the algorithm it may happen that the average size of a node's neighborhood is relatively high, for example, it is restricted only by the size of the network, meaning that any k nodes in the network can, in principle, create a clique. This may lead to problems of applying the algorithm in its current form for several reasons.

High computational complexity of a single round. In each round every node has to examine $\binom{|N(v)|}{k-1}$ combinations of $k-1$ neighbors with which it could potentially create a k-clique. If the number of neighbors is substantial then the cost of executing the protocol from Figure 3 may be prohibitively high even for small values of k.

Difficulty for a node to keep track of all other nodes in the network. In the case of a dynamic network, in which nodes may join and leave at any point in time, each node needs to be aware of all these changes. Failing to do so may prevent nodes from finding the best possible cliques. For example, if nodes u, v, and w are the best candidates to form a 3-clique but each of them is unaware of the existence of at least one of the other two nodes, they would never be able to form a clique together and would be forced to settle down for lesser options.

Messaging overload. For the protocol to converge in a timely manner and even to work correctly, it is necessary that all nodes have constantly updated information about the clique weights pursued by their neighbors. Therefore, in each round every node v sends out to all its neighbors its current value of $\mathbf{w}_v$, which translates to broadcasting to each node in each round the number of messages relative to the size of the network.

5.1 Heuristics for finding the heaviest *k*-clique

To circumvent consideration of all possible $k-1$ neighbor combinations, and thus, to reduce the high computational complexity of a single round, a node can use a heuristic to find its most attractive k-clique. The obvious candidates for heuristics to be used are the heuristics that perform well when trying to solve the heaviest k-subgraph problem, as that is in principle what each node is doing in every round. The choice of the heuristic requires careful consideration, as a wrong decision may result in suboptimal solutions.

For example, consider a simple greedy heuristic that constructs the k-clique for node v in the following manner. First, node v becomes the first node of a new clique: $U \leftarrow \{v\}$. Then $k-1$ times, node u from $N(v) - U$ is chosen such that the sum of weights of edges between u and nodes from U is maximal, and added to the new clique. The $(k-1)$-th node is chosen from $N(v) - U$ with the additional condition that $U - \{v\}$ is proper. The biggest disadvantage of using this heuristic is that it is simply too deterministic: the first $k-2$ neighbors are always fixed

(due to fixed edge weights), and in the choice of the $(k-1)$-th neighbor lies the node's only chance to find a promising clique. The portion of the explored solution space is therefore hugely restricted for the node; part of it will never be explored, and this can lead to nodes settling on suboptimal cliques or even not being able to find any clique to be part of. As is so often the case, it is much better to explicitly randomize how nodes select candidate nodes to team up with.

When incorporating a heuristic into the protocol, the biggest challenge is not to lose the convergence along the way. If the chosen heuristic was simply used to replace the lines 3–7 in the algorithm from Figure 3, in each round a node would most probably arrive at a different combination of neighbors to be set as $\mathbf{C}_v$, quite often this new combination would be also worse than the one from the previous round. This was not the case in the original protocol from Figure 3, in which we had the guarantee that in each round a subset of neighbors chosen by a node for a clique would be reconsidered, simply because all of the possible combinations are evaluated anew in every single round. Therefore, to avoid discarding a good clique candidate found in the previous round when a heuristic is used, in the modified protocol (see Figure 4) this clique is re-evaluated (lines 3–5). If this clique is still proper, it will be kept if no better clique is found.

Another modification is that nodes send out not only the information about the weight of their currently pursued clique but include in their message also the ids of the nodes from this clique (line 18). This information is mostly beneficial to the nodes that are the owners of the aforementioned ids. These nodes are now able to improve their own clique choice (lines 6–10). What happens here can be interpreted as a parallelization of the heuristic computation, which greatly improves the convergence speed.

Lastly, in line 11, the heuristic algorithm is executed. As a starting point for its execution, the value of C computed in lines 3–10 may be utilized; or a randomly generated value can be used. The best solution found is saved in $\mathbf{C}_v$ and its weight in $\mathbf{w}_v$. The value of $\mathbf{w}_v$ is then sent to all the neighbors of node v. Moreover, nodes from $\mathbf{C}_v$ receive also the information about the contents of $\mathbf{C}_v$. Afterwards, a new execution of the loop begins.

The fact that in each round a node repeats the heuristic search, results in our protocol being transformed into a multi-start version of the chosen heuristic. Each round means the re-execution of the heuristic procedure with a new (randomly generated) solution as a starting point. When choosing a heuristic it is worth to take into account this multi-start property together with the implicit parallelization achieved by collaboration of nodes towards finding the best clique, mentioned in the previous paragraph.

```
1:  loop
2:      C ← {}
3:      if proper(v, C_v) then
4:          C ← C_v
5:      end if
6:      for all u ∈ N(v) do
7:          if v ∈ C_u and
            attr_v(C_u + {u} − {v}) > attr_v(C) then
8:              C ← C_u + {u} − {v}
9:          end if
10:     end for
11:     C' ← HEURISTIC(v, C)
12:     if attr_v(C') > attr_v(C) then
13:         C ← C'
14:     end if
15:     C_v ← C
16:     w_v ← w({v} + C_v)
17:     send w_v to all u ∈ N(v)
18:     send C_v to all u ∈ C_v
19: end loop
```

Figure 4 ***k*-Clique matching protocol with a heuristic.**

5.1.1 VNS heuristic

We tested the modified protocol using variable neighborhood search (VNS) as a heuristic. The choice of this particular heuristic for our protocol is motivated by the work of Brimberg et al. who report in [25] that VNS consistently achieved the best results in solving the heaviest k-subgraph problem. The heuristics against which Brimberg et al. tested VNS include [26]: multi-start local search, tabu search, and scatter search. Each of these heuristics could also be applied to or adopted by our protocol, but investigating the differences of the protocol's performance under different heuristics falls out of scope of this paper.

VNS is a meta-heuristic that found its application in a vast range of combinatorial and global optimization problems [27] including various graph problems, knapsack and packing problems, scheduling problems, and data-mining problems. Its name stems from the term used to describe the subset of the solution space that, according to some predefined metric, is close to a particular point of this solution space. In our problem the solution space consists of all possible $(k-1)$-element combinations of a node's neighbors:

$$S_v = \{C | C \subseteq N(v) \textbf{ with } |C| = k-1\}.$$

A natural metric that can be imposed on this solution space defines the distance between two combinations as the number of nodes by which they differ, i.e.:

$$\delta(C, C') = |C - C'|.$$

Then the d-th *neighborhood structure* of a certain combination C would consist of all combinations that differ by at most d nodes from C:

$$NS_d(C) = \{C' | C' \in S_v \textbf{ with } \delta(C, C') \leq d\}.$$

Note that the neighborhood structure is not identical with the neighborhood of node v. v's neighborhood consists of all other nodes that are adjacent to the node by some edge, while the neighborhood structure consists of $(k-1)$-node subsets from v's neighborhood.

The VNS algorithm is composed of two functions which are executed in turns. First, the *shake* function modifies the current solution C by randomly changing a few nodes such that a new solution C' belongs to the d-th neighborhood structure of C. The goal of this step is to avoid getting stuck in a local optimum. Second, the *local search function* improves the new solution by trying to find a better one in the 1st neighborhood structure of C' (differing by a single node). This function can either return the first improvement found over C' or the best improvement (in our simulations the first improvement node was used). In case no improvement is found d is increased, otherwise it is set to some default value. The VNS algorithm executes until a certain halting condition is reached, for example, when the execution time exceeds some limit. For more details on how the VNS was adopted to our protocol, we refer to [2].

Applying the VNS heuristic proved to speed up the convergence of the protocol. This can be accounted to the fact that nodes inform each other not only about the weight of the clique they are trying to create but also about which nodes they are trying to create a clique with. This way nodes can learn quickly about good cliques without the need to search the entire solution space themselves. Yet, the assumption that each node has full (up-to-date) knowledge about the entire network (especially if the network is large, changes in time, nodes come and go, etc) is unrealistic. In Section 5.3 we present how this assumption can be dropped.

5.1.2 Random subset heuristic

Another heuristic is based on a simple trick: if considering all possible combinations of neighbors is computationally too expensive, a node can evaluate only a subset of neighbors in each round. The size of this subset can be set in such a way that for the given value of k, a node has enough resources to fully explore all $k-1$ combinations of nodes from the subset. To construct a subset, node v can simply pick randomly the necessary number of neighbors. Moreover, v can supplement the subset by adding $k-1$ neighbors that are in its current clique $\mathbf{C}_v$. Once the subset is created, the node executes the for-loop from the basic protocol (see Figure 3 lines 3–7) and returns the most attractive clique found.

The big advantage of this heuristic is that it is possible to implement the construction of the subsets in a completely distributed fashion, which eliminates the necessity of knowing all the nodes in the network by every node. We expand on this matter in Section 5.3.

5.2 Pruning

Completely independently from the heuristics, nodes can try to exploit their knowledge about the possible values of edge weights. For example, if the weights of the edges are bounded, node v can judge whether neighbor u has any chances to become a clique partner prior to considering any possible cliques this neighbor can be part of. To do that, v computes the best possible weight of the clique that it can create with u using the weight of the edge $\{v, u\}$ and the maximum possible value for the weights of remaining edges w_{max}. Thus, when the weight of a clique is computed as an arithmetic mean of the respective edges, the best possible weight of the clique with nodes v and u is:

$$w' = \frac{w\{v,u\} + \left(\binom{k}{2} - 1\right) w_{max}}{\binom{k}{2}}$$

If w' is smaller than either $\mathbf{w}_v$ or $\mathbf{w}_u$, node v can safely ignore this neighbor for two reasons. First, when $w' < \mathbf{w}_v$, node v has already found a better clique than any possible clique with u could be. Second, if $w' < \mathbf{w}_u$, node v has no chances in creating a clique with u since u has a better clique. Of course, because cliques pursued by nodes can change, in each round a node has to re-evaluate which nodes can be pruned. Nonetheless, once the set of neighbors is pruned, node v can apply any heuristic to this reduced set, which can greatly improve the convergence speed. Yet, one has to keep in mind that with the increase of k the impact of a single edge on the total weight of the cliques diminishes. A single edge weight contributes on average only $2/k(k-1)$ of the total clique's weight. As a consequence, pruning will bring the best results for small values of k.

5.3 Partial views with pruning

When any two nodes in the network can be each other's neighbors, then a list of any node's neighbors is limited only by the size of the network. In large dynamic networks maintaining such hefty lists by each node might be an expensive task. This is due not only to the sheer size of these neighbor lists but also to their volatility. Our goal is to remove the necessity of each node having complete information about the entire network, keeping a node's chances high of finding the best clique.

To this end, we allow nodes to maintain fixed-sized lists that contain only a relatively small number of all neighbors. We refer to such a list as a partial view of a node. Nodes present in the partial view of node v would be the

only ones among which v would be able to look for the most promising clique in each round. Thus, we could look at this type of the protocol as a variation of the random subset heuristic for finding the best k-clique — in each round a node has a small subset of all neighbors available, in this subset v must find $k-1$ neighbors with which it can create the best clique. This means that nodes can simply follow the protocol from Figure 4, and the efficiency of the clique matching will largely depend on the policy for modifying partial views.

As our algorithm is decentralized we would prefer to implement the partial views maintenance functionality also in a fully decentralized way. For this purpose we use a gossiping/epidemic protocol. Gossiping [28] is a simple, lightweight and robust type of peer-to-peer protocol that found its application in information dissemination, peer sampling, resource management, and most importantly from our perspective, topology construction. The framework of a gossiping protocol used for creation and maintenance of various overlays is summarized in Figure 5. Here, each node keeps a list of nodes of a specific small size c. The partial views are modified when nodes exchange among each other portions (of size b) of their views.

Different rules incorporated into the three functions from the framework will result in different overlays. In this paper, we consider two implementations: *Random* and *Pruning*, described in Figure 6. The goal of the *Random* exchanges is to provide enough variety of the partial view contents for the subsequent executions of clique matching protocol. This enables nodes to discover all nodes in the network and explore all possible neighbor combinations. At the same time, the drawback of this approach is that good candidates for clique members might be discarded too easily and be replaced by other random nodes. Conversely, the idea behind *Pruning* exchanges is that nodes send out portions of their partial views that are most "promising" from the recipient's perspective. Moreover, nodes keep in their partial view only the "promising" neighbors and we use the pruning measure described in previous subsection to separate "promising" neighbors from the rest. The risk of using *Pruning* lies in the possibility of partitioning the overlay network induced by partial views if pruning turns out to be too greedy. Therefore, we also investigate a third approach which is the combination of the *Random* and *Pruning* approaches, in which separate gossiping protocols implementing the former two approaches are stacked on top of each other (see Figure 7), similarly to the Vicinity protocol described in [5].

Active thread loop:

1: $u \leftarrow$ `selectNode()`
2: $buffer_{out} \leftarrow$ `selectItemsToSend()`
3: **send** $buffer_{out}$ **to** u
4: **receive** $buffer_{in}$ **from** u
5: `selectItemsToKeep()`

Passive thread loop:

1: **receive** $buffer_{in}$ **from** v
2: $buffer_{out} \leftarrow$ `selectItemsToSend()`
3: **send** $buffer_{out}$ **to** v
4: `selectItemsToKeep()`

Figure 5 Gossiping protocol framework.

Another advantage of using gossiping to implement partial views is that it provides a natural mechanism for tackling changes to the network by disseminating information about newly joined nodes or by gradually removing from partial views nodes that left the system. This is a desirable feature even considering that we are dealing with servers (and not end-user systems) and thus that it is fair to assume that churn will be very low. As further reported in our previous work (see, e.g., [29]), gossiping is ideal for effectively and efficiently handling the adverse effects of churn. Further investigations about effects of churn on our k-clique matching protocol with partial views is left for future research.

5.4 Gossiping updates

The timely convergence of the k-clique matching protocol is contingent on nodes having up-to-date information about the cliques pursued by their neighbors. Therefore, in all versions of the protocol presented so far, each node v after computing a new value of $\mathbf{C}_v$, sends to all its neighbors its new value of $\mathbf{w}_v$, which afterwards is used by v's neighbors to determine if they can conceivably propose a better clique to v. However, this means that in each round the number of messages sent through the network can reach the order of $|V|^2$.

Instead of such an expensive broadcast mechanism, we can use a gossiping protocol for this purpose. In the previous section we have shown how a gossiping algorithm can be used to maintain and modify partial views of nodes. The same mechanism can be also used to spread the values of clique weights pursued by each node. To this end, a separate gossiping protocol can run in the background.

This protocol follows the same framework from Figure 5 but instead of exchanging only the contact information to nodes in the network, each item, stored in the view and exchanged between nodes, consists of an *id* of some node v, its contact information, the value of $\mathbf{w}_v$ and a counter indicating the *age* of this information. These age counters are increased at the beginning of each loop iteration in the active thread of the gossiping protocol, as well as upon an exchange of information between nodes. If node v finds itself to have two items of the same node id (e.g., as a result

Random
`selectItemsToSend()`
active thread: randomly select $b-1$ nodes from the protocol's view and add yourself
passive thread: randomly select b nodes from the protocol's view
`selectItemsToKeep()`
keep all b received nodes, removing (if necessary) the nodes selected to send (and in case of *active thread*, the node returned by selectRandomNode())
Pruning
`selectItemsToSend()`
select b nodes whose weights are best from the point of the receiving node
`selectItemsToKeep()`
keep c best out of all nodes from the received buffer and current view

Figure 6 Implementation details of *Random* and *Pruning* protocols.

of a gossip exchange) it keeps the item with the smaller age. This way, older information about the clique weights is seamlessly discarded.

The efficiency of the clique matching protocol will largely depend on the speed at which the gossiping protocol propagates current information on clique weights through the network. In Figure 8 we present details of possible rules to incorporate into the three core functions of the gossiping protocol. In Section 6, we will compare the impact of various combinations of these rules on the convergence of the clique matching protocol. We will also examine other factors, such as the gossiping communication frequency.

Although we have mentioned that this protocol can run in parallel, completely independently from other protocols, there is also a potential for consolidating the dissemination of $\mathbf{w}_v$ values with the dissemination of contact information and the maintenance of partial views. We leave the evaluation of these dissemination ideas to future research.

6 Performance evaluation

6.1 Simulation setup

In the following simulations, we examine the efficiency of the presented protocols in partitioning the network into k-cliques. We focus on measuring convergence of various versions of our k-clique matching algorithm. Note that measuring the quality of created k-clique matchings as compared to the optimal solution is not possible as it is computationally infeasible to derive the latter.

The execution of simulations is based on the notion of rounds. In each round, every node executes the entire loop body exactly once. The ordering in which the nodes execute their protocols in each round is entirely random.

The number of elapsed rounds is not always the best time metric as it does not reflect the computational costs incurred by nodes during a single round when different versions of the protocol are used or even when comparing the same protocol for different values of k. Therefore, when needed we adopt the average number of cliques considered by nodes since simulation start. This metric

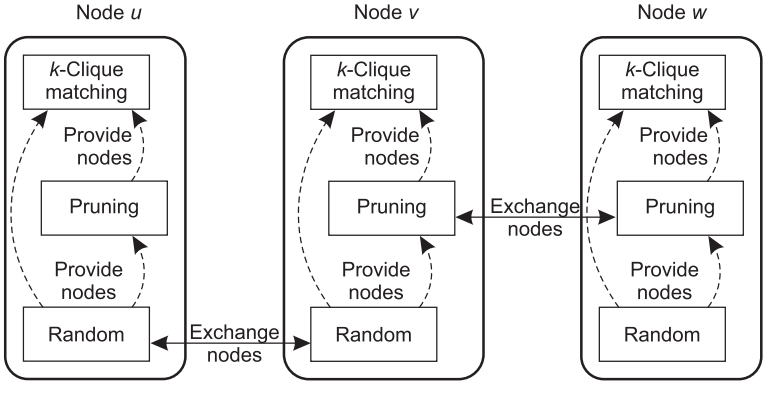

Figure 7 The layered framework.

`selectNode()`
random: randomly select node from the view
oldest: select node whose item has the highest age
youngest: select node whose item has the lowest age

`selectItemsToSend()`
create an item with node's own clique weight info and fill the remaining $b-1$ places in the buffer with:
random: randomly selected items from the view
oldest: oldest items from the view
youngest: youngest items from the view

`selectItemsToKeep()`
if a received item has its counterpart (item with the same node's id) in the view, keep the item with the smaller age, otherwise simply add the received item to the view

Figure 8 Implementation details of gossiping protocol for clique weights dissemination.

reflects more closely the differences between the computational complexities of the protocols.

All of our simulations are conducted on complete graphs, which means that for each node v the neighbor set $N(v)$ is equal to $V-\{v\}$. As a result, any k nodes can potentially form a clique. This guarantees that, irrespectively of the distribution of edge weights, it is always possible to find a k-clique matching that consists of $\lfloor n/k \rfloor$ cliques.

Each edge is assigned a weight drawn uniformly at random from the interval $(0,1)$. The weight of a clique is computed as the arithmetic mean of the respective edge weights in the clique. The objective of the nodes in a network is then to maximize the weight of the clique they are going to be part of.

All simulations presented here were conducted using PeerSim [30], an open-source simulator for peer-to-peer protocols. Each presented curve is an average of 5 simulations executed with the same parameters but different random number generator seeds. In each simulation, all nodes start without any clique chosen.

6.2 Performance of *k*-clique matching protocol

First, let us focus on the convergence speed of the basic k-clique matching protocol from Figure 3. Figure 9(a) depicts the percentage of nodes in cliques against the number of elapsed rounds. In a network of 300 nodes, the number of rounds necessary for all nodes to form disjoint cliques increases with the size of the cliques. Nonetheless, even for a clique size of $k = 5$, the time needed for full convergence does not exceed 20 rounds.

Although the clique size, k, does not have a dramatic effect on the protocol performance in terms of convergence rounds, it does affect it with respect to the amount of computation required by each node. This is depicted in Figure 9(b), which plots the same metric (percentage of nodes in cliques) as a function of the number of cliques considered on average by each node. Here, the discrepancies of the computational load under different values of k become clearly pronounced (notice the logarithmic scale of the horizontal axis). In fact, the exponential increase in computational load comes as no surprise, as each node has to evaluate all $\binom{|N(v)|}{k-1}$ possible cliques in each round, as specified by the basic protocol from Figure 3.

The next pair of graphs shows the differences in the convergence speed of the basic protocol for different sizes of networks. In Figure 10(a) we do not observe significant discrepancies in the number of rounds needed by all nodes to find their cliques. Yet again, as we take into account the average number of cliques evaluated by each node, we can observe that with each twofold increase in the size of the network, the convergence time grows by a factor of 2^{k-1}. Again, this is the direct consequence of the fact that in the basic algorithm every node evaluates $\binom{|N(v)|}{k-1}$ possible cliques.

Note here that in case of a sparse graph the computational cost (as well as the communication one) of the basic protocol would be much lower. For example, if the size of each node's neighborhood was in the order of the logarithm of the network size, $O(log|V|)$, the computational cost of a single round would be only in the order of $O(|V|)$ instead of $O\left(|V|^k\right)$ (and each node would send only $O(log|V|)$ messages per round). For such graphs, our basic protocol would be really efficient and would scale gracefully with increase in clique and/or network size. Yet, for graphs where neighborhoods of nodes are much larger, we need other strategies to lower the costs and preserve the performance.

6.3 Performance with heuristics

To assess the performance of the k-clique matching protocol that uses heuristics, we compare it against the basic version of the protocol. Figure 11 shows the convergence (explained below) as a function of the average number of

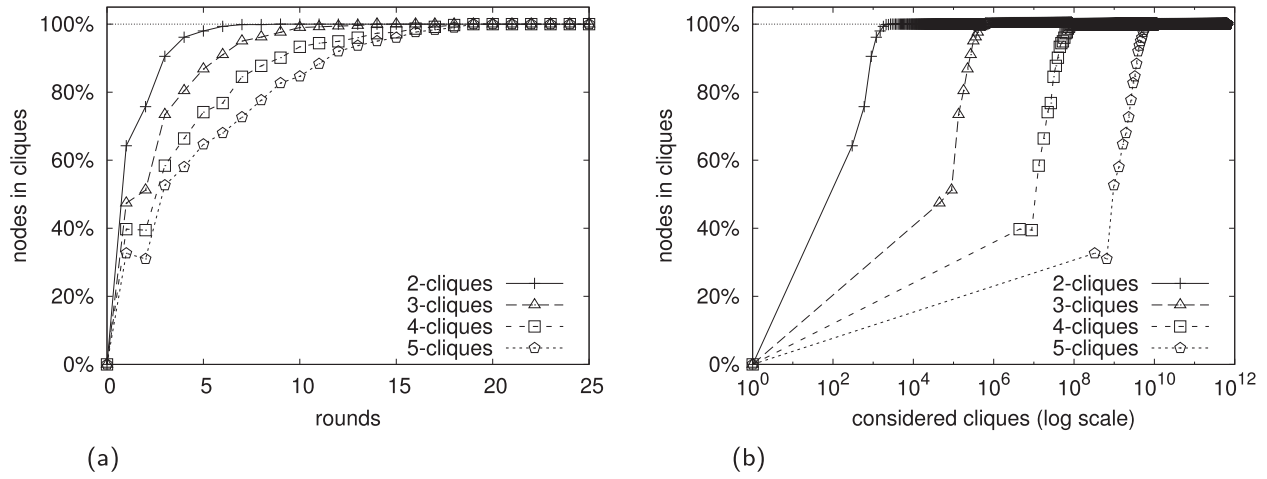

Figure 9 Performance of the basic *k*-clique matching protocol; percentages of nodes matched into cliques for various values of *k* plotted against: (a) number of rounds, (b) average number of cliques evaluated by a single node since the beginning of a simulation.

cliques considered per node, for 5 different versions of the *k*-clique matching protocol: (1) the basic one, (2) using the VNS heuristic, (3) using pruning and the VNS heuristic, (4) using the random subset heuristic (RS), (5) using pruning in combination with RS. The convergence of the protocols is not expressed by the percentage of nodes in cliques at a given point, but by the percentage of nodes in cliques that has been reached at a given point and remains above this level till the end of experiment. This results in curves being monotonically increasing and is done only for the sake of legibility. The size of the network is 300 nodes and $k = 4$. The size of the random subset is set to 40. The halting condition of the VNS protocol is based on the number of cliques considered by a node. This number is chosen in such a way that it provides a fair comparison between VNS and random subset heuristics. Given that with the random subset heuristic a node will consider in a single round not more than $\binom{40+k-1}{k-1}$ cliques, the number of cliques that a node using the VNS heuristic can check in a single round is also set to $\binom{40+k-1}{k-1}$.

In Figure 11, we see that both the VNS and the RS heuristics outperform the basic *k*-clique matching protocol. This can be attributed to the fact that nodes make much better use of the cliques formed by other nodes. Moreover, adding pruning mechanisms speeds up the convergence of each of the heuristics further. We see that the difference between using VNS and the RS heuristics is minimal, which means that we can often use the simpler and cheaper RS heuristics to achieve the same results. Apparently, having only a relatively small subset of

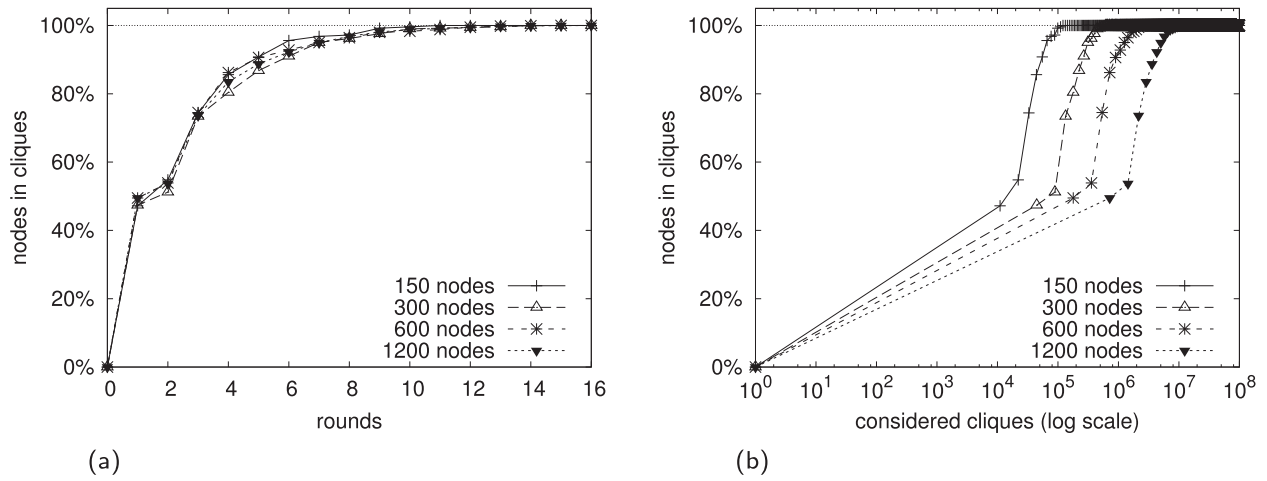

Figure 10 Performance of the basic *k*-clique matching protocol; percentages of nodes matched into 3-cliques for various network sizes plotted against: (a) number of rounds, (b) average number of cliques evaluated by a single node since the beginning of a simulation.

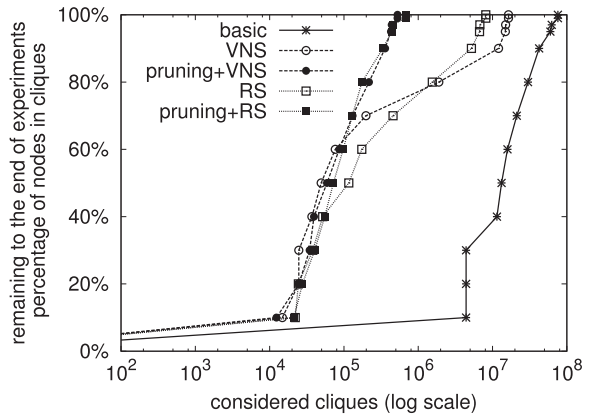

Figure 11 Comparison between performance of the basic *k*-clique matching protocol and the *k*-clique matching protocol employing heuristics and pruning; plotted against the average number of cliques evaluated by a single node since the beginning of an experiment.

neighbors already achieves the major effect also attained by VNS. A more comprehensive comparison between the basic protocol, VNS with various halting conditions and RS with various random subset sizes is presented in [24].

Figure 12 focuses on the performance of the VNS heuristic for clique size 2, 3, and 4, and for networks of 300, 600, 1200, and 2400 nodes. The halting condition for "VNS (M)" is set to $\binom{M+k-1}{k-1}$ considered cliques. This means that "VNS (40)" ("VNS (60)") considers almost 2^{k-1} $\left(3^{k-1}\right)$ cliques more in each round compared to "VNS (20)". Firstly, we can observe that pruning significantly speeds up the convergence of the protocol. Further, we can observe that the convergence times get slightly longer when the size of the network increases, but the increase in the size of the clique has a much bigger negative impact. This can be attributed to the increase in the solution space of the best clique for a given node, which is equal to $\binom{N-1}{k-1}$. When the size of the network grows by a factor of two, the solution space increases by a factor of circa 2^{k-1}; when the size of the clique increases by just 1, the solution space increases by a factor of N/k. For small values of k, the latter will be significantly larger than the former. The increasing differences between results for "VNS (20)", "VNS (40)", "VNS (60)" with pruning can be explained in a similar manner.

We see a similar regularity in performance of the RS heuristic (see Figure 13). Moreover, when compared to VNS, the random subset heuristic seems to converge slightly faster. This is especially encouraging in light of our next simulations regarding partial views.

6.4 Performance with partial views

Interesting results have been obtained for the performance of the k-clique matching protocol that uses partial views. In Figure 14 we can see the comparison between the partial views implementing *Random*, *Pruning*, and layered *Random+Pruning* approaches. We can observe that layered *Random+Pruning* outperforms *Random* implementation for all clique and network sizes investigated, which suggests that using pruning in combination with some randomness to maintain partial views does help in the convergence of our k-clique matching protocol.

At the same time, we can see curious results for the *Pruning* implementation. For $k = 2$ it performs worse than both *Random* and *Random+Pruning*; for $k = 3$ it starts to perform better than *Random* across all network sizes but is still worse than *Random+Pruning*; finally, for $k = 4$ it starts to outperform also *Random+Pruning*. The explanation lies in the close dependency between the size of the clique and the strength of pruning, as hinted already in Section 5.2. For $k = 2$, pruning is very efficient but this is exactly the source of the problem. At the beginning of a simulation, the situation in the network is very unstable and many cliques are created and broken from round to round. Pruning acts then over-zealously, discarding from partial views nodes that are potentially very attractive but for this brief moment are unavailable or uninteresting. With the increase of the clique size, pruning becomes more and more relaxed and its performance improves. Nonetheless, to prevent hyperactivity of pruning, one might consider disabling it in the early stages of algorithm execution and enable it only once the system starts to settle down. We leave for future research investigation of how nodes could recognize the right moment to switch on pruning.

When we compare the convergence times of the simulations with partial views with *Random* implementation (see Figure 14) and the simulations of random subset heuristic *without* pruning, we do not observe any major discrepancies. To the contrary, the results for partial views with *Pruning* differ highly from the results for random subset heuristic *with* pruning. In all setups, adding pruning to random subset heuristic improved the convergence of the protocol significantly. This was not the case for partial views with *Pruning*. This differences can be explained by the fact that in case of the random subset heuristic nodes have the knowledge of all other nodes in the network and apply pruning to the full list of nodes anew at the beginning of each round, thus removing from a consideration a node has the effect only for duration of a single round. Conversely, if node decides to replace in its partial view one node with another, it must reckon that it might take some time before it will stumble upon this node again.

6.5 Performance when gossiping clique weights

We start our evaluation of the gossiping protocol for clique-weight dissemination by comparing the effectiveness of various implementations of the two core methods

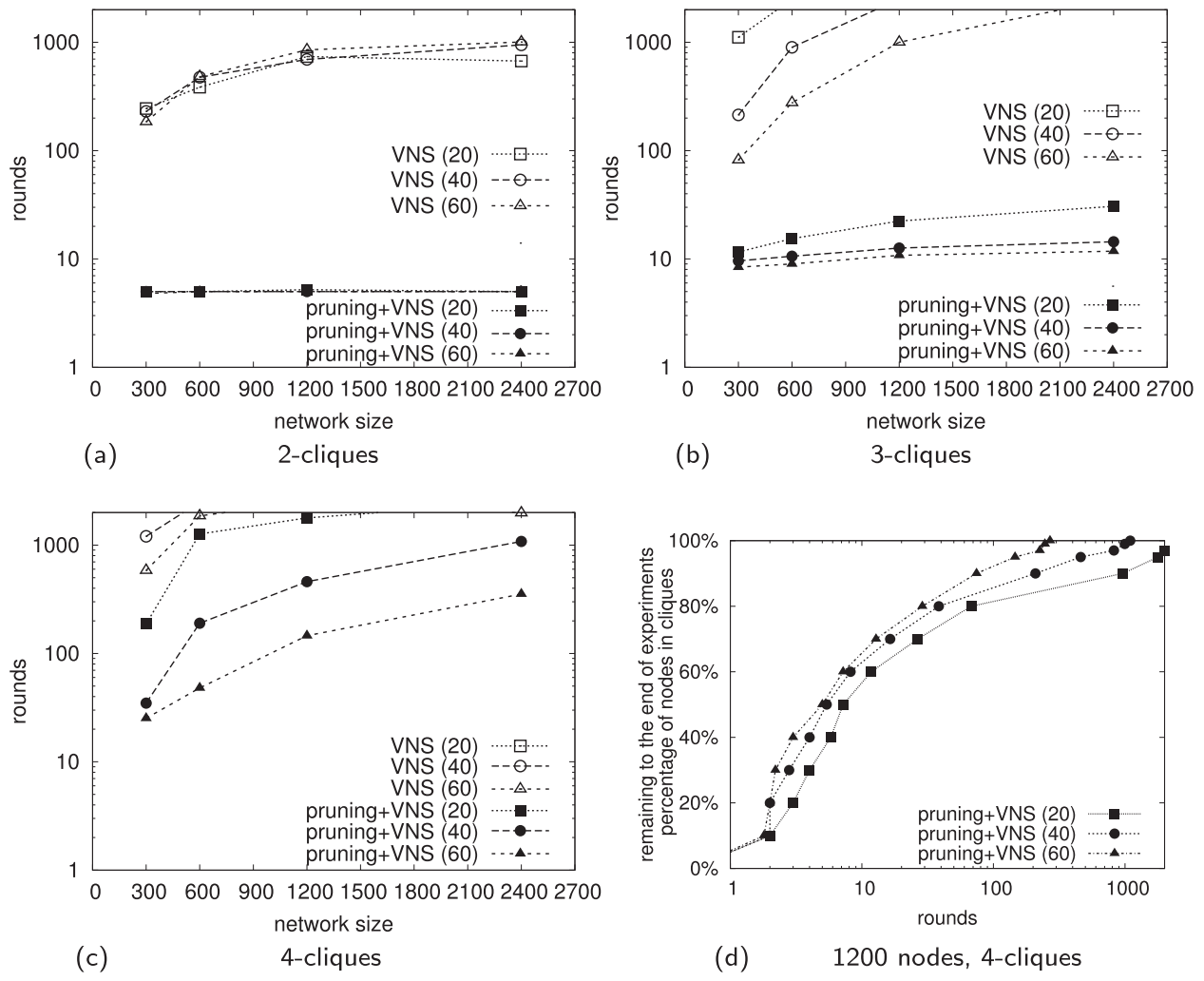

Figure 12 Number of rounds needed by *k*-clique matching using VNS to reach and remain over the level of 95% of nodes in cliques over different network sizes (a)–(c) and an example of convergence (d).

`selectNode()` and `selectItemsToSend`. In Figure 15 we can see simulation results of the basic *k*-clique matching protocol that, instead of broadcasting new clique weight values, uses a gossiping protocol to disseminate this information. Each iteration of the *k*-clique matching protocol loop is followed by a single execution of the active thread loop of the gossiping protocol. The simulations have been performed on a network of 300 nodes and for $k = 3$. The size of the buffer for the gossiping protocol has been set to 10. Each curve corresponds to a different combination of `selectNode()` and `selectItemsToSend` implementations, as detailed in Section 5.4.

In Figure 15 we can distinguish four different groups of curves. The fastest convergence times have been achieved, when `selectItemsToSend()` chooses the *youngest* items from the view, and `selectNode()` returns either the oldest or a random node. For the same two implementations of `selectNode()` but with `selectItemsToSend()` returning *random* items, the convergence of our *k*-clique matching protocol slows down almost twofold. Moreover, when `selectItemsToSend()` returns the oldest items, the protocol converges even more slowly and does not manage to achieve 100% in the first 1000 rounds. Yet, the worst results have been obtained for `selectNode()` that returns the node from the youngest item, for any of the implementations of `selectItemsToSend()`; the *k*-clique matching protocol gets stuck at 30%.

These results can be explained by considering the role of the age counter attached to every item. This counter coincides with the freshness of the clique weight information. Therefore, when nodes choose the youngest items in `selectItemsToSend()`, they contribute to the dissemination of the most up-to-date information about other nodes' clique weights. On the other hand, when a

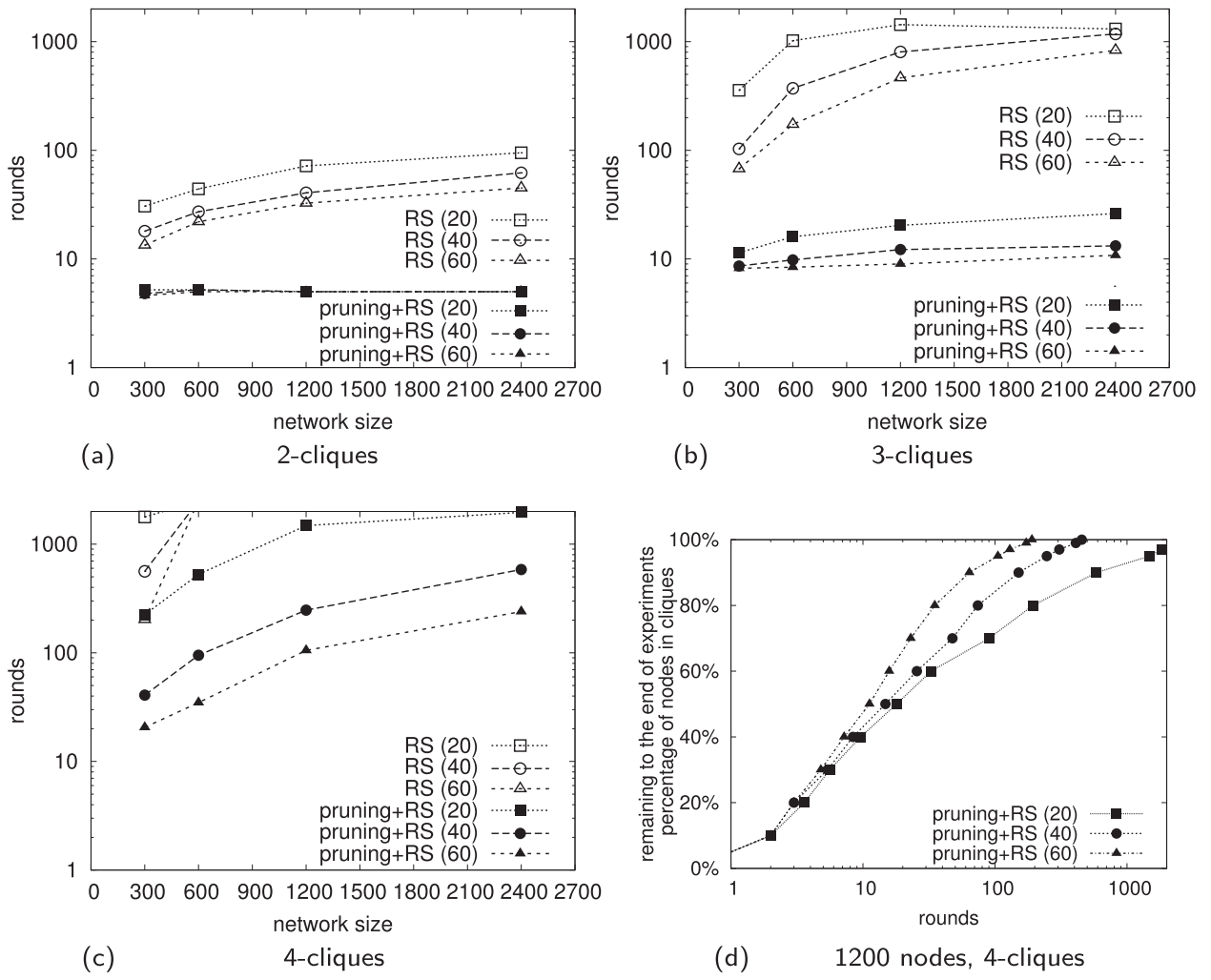

Figure 13 Number of rounds needed by *k*-clique matching using random subset heuristic to reach and remain over the level of 95% of nodes in cliques over different network sizes (a)–(c) and and example of convergence (d).

node selects the node from the youngest item as the next node to communicate with (via `selectNode()`) it falls in to the trap of exchanging information with the same node over and over again; when a node selects items to send, it always adds an item with its own clique weight; this item is going to become the youngest item in the view of the recipient.

Even with the most efficient combination of `selectNode()` and `selectItemsToSend()` implementations, *k*-clique matching using gossiping to disseminate clique weight information is significantly (over 30 times) slower than its counterpart that uses broadcast. Nonetheless, the convergence can be improved by increasing the number of items exchanged by nodes between any two executions of *k*-clique matching protocol loop. This can be done by either increasing the number of items exchanged in every gossiping communication or by increasing the frequency of the gossiping protocol relatively to the *k*-clique matching protocol. Here we present the results for the latter.

First, note that when broadcast is used, a node has to send out $N - 1$ messages with its new clique weight in every round. At the same time, when gossiping is used in every round each node initiates only one exchange and, thus, on average it is also contacted by one other node. As a result, each node sends out the number of items equal to twice the size of the gossiping buffer. If the frequency of gossiping relatively to the *k*-clique matching protocol is increased, the average number of items sent out will also increase linearly to the frequency increase. The differences between the message load of broadcast and gossiping with different frequencies is depicted in Figure 16(a): the network size equals 300 and the size of the gossiping buffer is set to 10. Furthermore, in

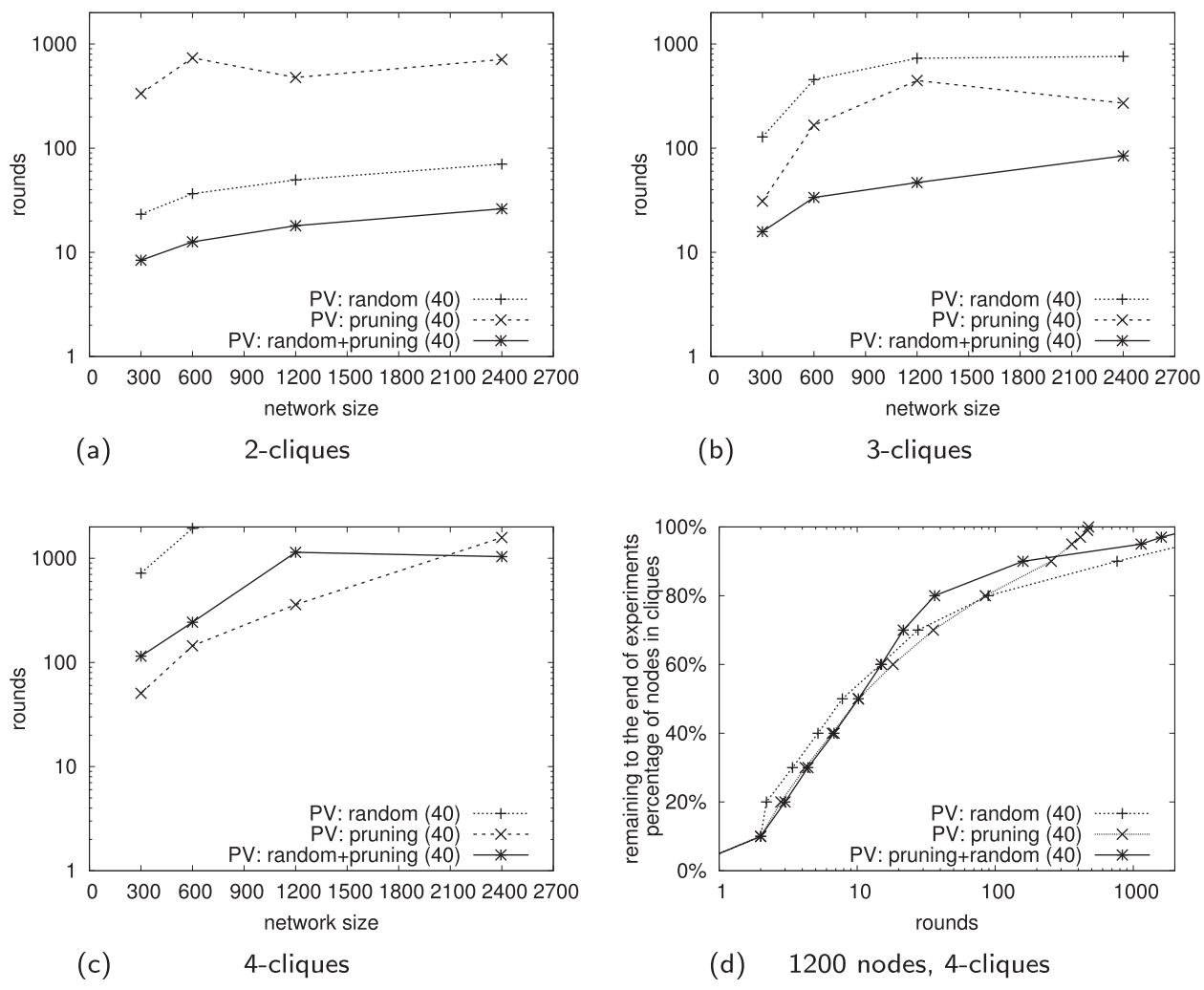

Figure 14 Number of rounds needed by *k*-clique matching using partial views to reach and remain over the level of 95% of nodes in cliques over different network sizes (a)–(c) and and example of convergence (d).

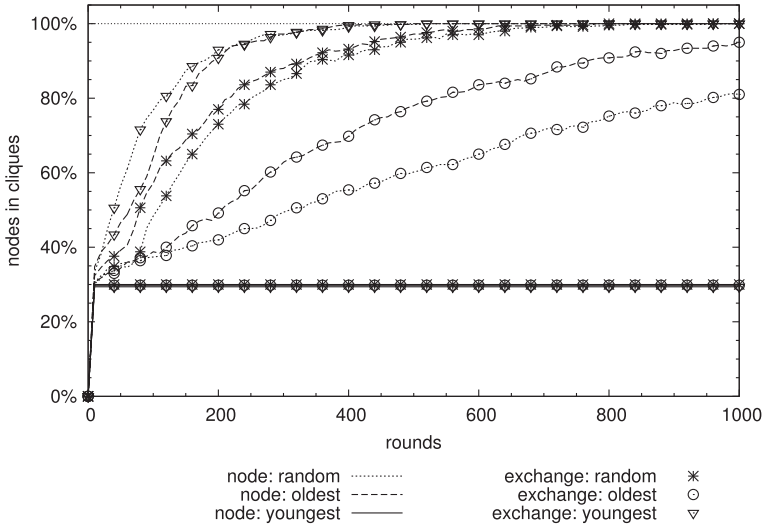

Figure 15 Comparison between various implementations of clique weight gossiping core functions.

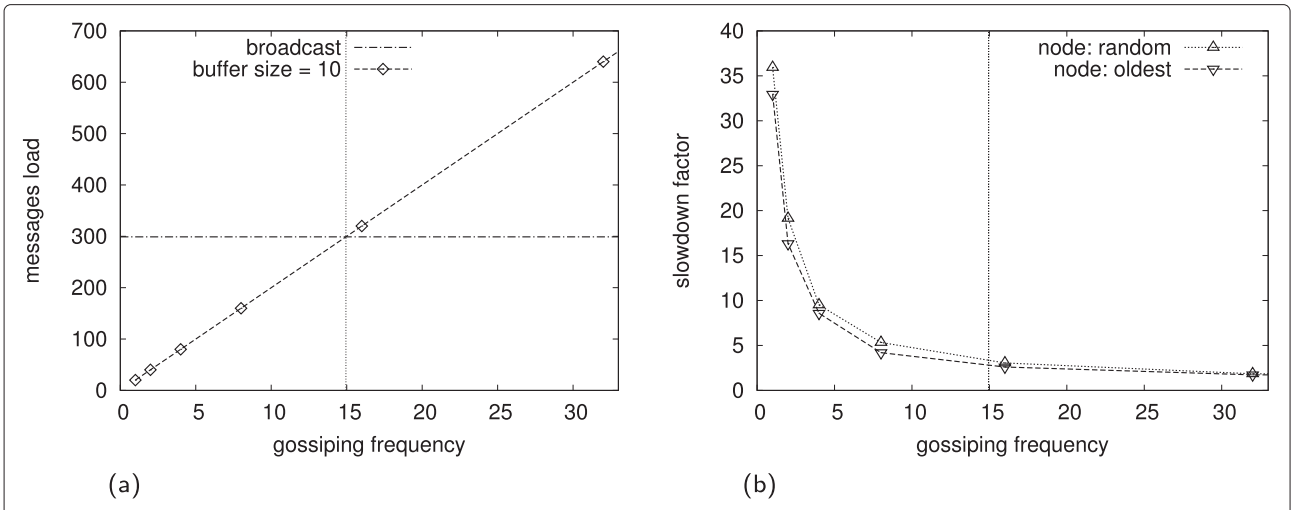

Figure 16 Comparison between various implementations of clique weight gossiping core functions: message load (a) and slowdown factor (b) versus gossiping frequency.

Figure 16(b) we can see that the convergence speed of the k-clique matching protocol with gossiping of clique weights improves exponentially. This shows that gossiping is a viable alternative to the brute force broadcast of clique weights.

7 Conclusions

In this paper we presented a protocol that finds an approximation of a k-clique matching in the network. The protocol can be used to divide entities in the network into fixed size groups, based on the pairwise assessments between nodes. The final partitioning emerges from nodes' local decision on which neighbors to link to.

Although the convergence of the protocol in its basic form in terms of rounds looks very attractive, there are hidden costs of the protocol imposed on each of the nodes in terms of computational complexity related to the evaluation of all possible combinations of neighbors performed in each round. To elevate these costs we proposed that nodes use a heuristic to compute their new choice of $k-1$ neighbors instead of performing a full search over the solution space. The two heuristics that we used were (1) variable neighborhood search heuristic that was reported to outperform other heuristics for finding heaviest k-clique and (2) a simple random subset heuristic. As our simulations show, the protocol using any of these two heuristics achieves a converged state faster than the original protocol. We attribute these results to the fact that the nodes share their best clique with its other members, which significantly speeds up the search for the best clique by implicit parallelization of computations. In contrast, in the basic version on the protocol, nodes perform their computations fully on their own and than only agree on the clique's weight.

To further improve the performance of the protocol, we employed a pruning mechanism that temporarily (for the period of a single round) removes from consideration those neighbors that have no prospects for participating in a clique better than the current one. We validated by simulations that combining heuristics with pruning performs better than applying heuristics alone.

The fact that the random sample heuristic performed equally well as a more elaborate variable neighborhood search heuristic, led us to exploring a variation of the protocol where each node keeps only a subset of all its neighbors in its local memory (creating a partial view of the full neighbor set). To discover new neighbors nodes periodically exchange parts of their partial views. Thus in each round, similarly to the random subset heuristic case, they consider a different subset of neighboring nodes. This approach is especially attractive for the scenarios in which the set of neighbors is equivalent with the set of nodes in the network and the assumption that nodes know all their neighbors would be too strict. The simulations confirmed that the partial views approach provides performance indistinguishable from the random subset heuristic.

Independently from these improvements, we have also investigated to what extent the broadcast communication between the nodes in the protocol can be replaced by other methods of information dissemination through the network. The initial results of the use of gossiping protocol instead of broadcast show the viability of this approach and its tradeoff in terms of slower convergence times.

In our future work we plan to focus on extending the protocol to support formation of cliques of different sizes and relaxing the constraint that each node can be part of

only one clique, by allowing nodes to set their own limits on the number of cliques they are interested to participate in.

Endnote

[a]We adopted the terminology of the k-clique matching from [12] where Kann uses the term H-matching to describe a set of disjoint subgraphs in a given graph where each of these subgraphs is isomorphic to H. This term is also used by Crescenzi and Kann in the highly cited [13]. After a more thorough search of related work we observe that the term H-packing is more widely used to describe this notion. Nonetheless, to stay consistent with our previous paper, we keep our terminology of k-clique matching.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
AC is the main contributor of this work, which was undertaken as part of her Ph.D. studies. She is also the author of a first version of this paper. SV has been daily supervisor for AC, with main contributions to the design of the algorithms, notably concerning extensions including gossiping. MvS has been supervisor for AC, contributing to all work reported in this paper. He is the main author of the revised version of this paper when submitting it to JISA. All authors have read and approved the final manuscript.

Received: 29 May 2014 Accepted: 15 September 2014
Published online: 09 October 2014

References

1. Foster I, Berry D, Djaoui A, Grimshaw A, Horn B, Kishimoto H, Maciel F, Savva A, Siebenlist F, Subramaniam R, Treadwell J, Von Reich J (2006) The Open Grid Services Architecture, Version 1.5. GGF Informational Document GFD-I.080
2. Chmielowiec A, van Steen M (2010) Optimal decentralized formation of k-member partnerships. In: SASO 2010 proceedings of the 4th, IEEE international conference on self-adaptive and self-organizing systems. IEEE Computer Society, Washington DC. pp 154–163. doi:10.1109/SASO.2010.14
3. Chmielowiec A, Pierre G, Gordijn J, van Steen M (2008) Technical challenges in market-driven automated service provisioning. In: MW4SOC '08: proceedings of the 3rd workshop on middleware for service oriented computing. ACM, New York. pp 25–30. doi:10.1145/1462802.1462807
4. Jelasity M, Montresor A, Babaoglu O (2009) T-man: gossip-based fast overlay topology construction. Comput Netw 53(13):2321–2339. doi:10.1016/j.comnet.2009.03.013
5. Voulgaris S (2006) Epidemic-based self-organization in peer-to-peer systems. PhD thesis. Vrije Universiteit, Amsterdam, The Netherlands
6. Micali S, Vazirani VV (1980) An $o(\sqrt{(}|v|)|e|)$ algoithm for finding maximum matching in general graphs. In: SFCS '80: proceedings of the 21st annual symposium on foundations of computer science. IEEE Computer Society, Washington, DC. pp 17–27. doi:10.1109/SFCS.1980.12
7. Blum N (1990) A new approach to maximum matching in general graphs. Automata Languages Program 443:586–597. doi:10.1007/BFb0032060
8. Gabow HN, Tarjan RE (1991) Faster scaling algorithms for general graph matching problems. J ACM 38(4):815–853. doi:10.1145/115234.115366
9. Hell P, Klein S, Nogueira LT, Protti F (2005) Packing r-cliques in weighted chordal graphs. Ann Oper Res 138:179–187
10. Czygrinow A, Hańćkowiak M (2007) Distributed approximations for packing in unit-disk graphs. Distr Comput 4731:152–164. doi:10.1007/978-3-540-75142-7_14
11. Czygrinow A, Hańćkowiak M, Wawrzyniak W (2008) Distributed packing in planar graphs. In: Proceedings of the twentieth annual symposium on parallelism in algorithms and architectures. SPAA '08. ACM, New York. pp 55–61. doi:10.1145/1378533.1378541
12. Kann V (1994) Maximum bounded h-matching is max snp-complete. Inform Process Lett 49(6):309–318. doi:10.1016/0020-0190(94)90105-8
13. Crescenzi P, Kann V (1998) A compedium of NP optimization problems. http://www.nada.kth.se/~viggo/problemlist/compendium.html
14. Kirkpatrick DG, Hell P (1978) On the completeness of a generalized matching problem. In: STOC '78 proceedings of the tenth annual ACM symposium on theory of computing. pp 240–245
15. Gabow HN (1990) Data structures for weighted matching and nearest common ancestors with linking. In: SODA '90: proceedings of the first annual, ACM-SIAM symposium on discrete algorithms. Society for Industrial and Applied Mathematics, Philadelphia. pp 434–443
16. Hoepman J-H (2004) Simple Distributed Weighted Matchings. http://arxiv.org/abs/cs/0410047
17. Lotker Z, Patt-Shamir B, Pettie S (2008) Improved distributed approximate matching. In: SPAA '08: proceedings of the twentieth annual symposium on parallelism in algorithms and architectures. ACM, New York. pp 129–136. doi:10.1145/1378533.1378558
18. Nieberg T (2008) Local, distributed weighted matching on general and wireless topologies. ACM, New York. doi:10.1145/1400863.1400880
19. Manne F, Mjelde M (2007) A self-stabilizing weighted matching algorithm. In: Stabilization, safety, and security of distributed systems. LNCS, vol 4838/2007. pp 383–393. doi:10.1007/978-3-540-76627-8_29
20. Hassin R, Rubinstein S (2006) An approximation algorithm for maximum triangle packing. Discrete Appl Math 154(6):971–979
21. Chen Z-Z, Tanahashi R, Wang L (2009) An improved randomized approximation algorithm for maximum triangle packing. Discrete Appl Math 157(7):1640–1646. doi:10.1016/j.dam.2008.11.009
22. Preis R (1999) Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In: In STACS 99: proceedings symposium on theoretical aspects of computer science. Springer, Berlin. pp 259–269. doi:10.1007/3-540-49116-3_24
23. Avis D (1983) A survey of heuristics for the weighted matching problem. Networks 13:475–493. doi:10.1002/net.3230130404
24. Chmielowiec A (2014) Decentralized k-clique matching. PhD thesis. Vrije Universiteit Amsterdam
25. Brimberg J, Mladenovic N, Urosevic D, Ngai E (2009) Variable neighborhood search for the heaviest k-subgraph. Comput Oper Res 36(11):2885–2891. doi:10.1016/j.cor.2008.12.020
26. Gendreau M, Potvin J-Y(eds.) (2010) Handbook of metaheuristicsInternational, Series in Operations Research & Management Science, vol. 146. Springer, New York. doi:10.1007/978-1-4419-1665-5
27. Hansen P, Mladenović N, Moreno Pérez JA (2010) Variable neighbourhood search: methods and applications. Ann Oper Res 175(1):367–407. doi:10.1007/s10479-009-0657-6
28. Kermarrec A-M, van Steen M (2007) Gossiping in distributed systems. SIGOPS Oper Syst Rev 41(5):2–7. doi:10.1145/1317379.1317381
29. Jelasity M, Voulgaris S, Guerraoui R, Kermarrec A-M, van Steen M (2007) Gossip-based peer sampling. ACM Trans Comput Syst 25(3):8. doi:10.1145/1275517.1275520
30. Jelasity M, Montresor A, Jesi GP, Voulgaris S The Peersim Simulator. http://peersim.sourceforge.net/

doi:10.1186/s13174-014-0012-2
Cite this article as: Chmielowiec *et al.*: **Decentralized group formation.** *Journal of Internet Services and Applications* 2014 **5**:12.